FLÁVIO S. TRUZZI
MARCELO A. PITA

SISTEMA DE DISTRIBUIÇÃO DE EXPERIMENTOS DE MODELOS DE DISTRIBUIÇÃO DE ESPÉCIES.

São Paulo
2011

FLÁVIO S. TRUZZI
MARCELO A. PITTA

SISTEMA DE DISTRIBUIÇÃO DE EXPERIMENTOS DE MODELOS DE DISTRIBUIÇÃO DE ESPÉCIES.

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do diploma de conclusão de curso.

São Paulo
2011

FLÁVIO S. TRUZZI
MARCELO A. PITA

SISTEMA DE DISTRIBUIÇÃO DE EXPERIMENTOS DE MODELOS DE DISTRIBUIÇÃO DE ESPÉCIES.

Monografia apresentada á Escola Politécnica da Universidade de São Paulo para obtenção do diploma de conclusão de curso.

Área de Concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Pedro Luiz Pizzigatti Corrêa
Co-orientadora:
Prof. Dra. Karla Donato Fook

São Paulo
2011

Aos nossos familiares e amigos.

AGRADECIMENTOS

Agradecemos

Ao nosso orientador, Pedro Luiz Pizzigatti Correa.
A nossa coorientadora Karla Donato Fook, pelo auxílio prestado.
Aos amigos, pelo suporte.
A nossas famílias, pelo apoio e compreensão.

"In the long history of humankind (and animal kind, too)
those who learned to collaborate
and improvise most effectively have prevailed."
— **Charles Darwin**

"If I have seen further it is only by
standing on the shoulders of giants." — **Isaac Newton**

RESUMO

A busca pelo crescimento sustentável passa indiscutivelmente pela preservação de espécies. Pesquisas nesta área estão cada vez mais sendo suportadas pela tecnologia. Uma dessas tecnologias, o openModeller Desktop, auxilia cientistas e institutos na criação de modelos de distribuição de espécie. Estes modelos são criados a partir da distribuição geográfica observada da espécie, camadas ambientais (relevo, pluviosidade, vegetação, clima etc) e algoritmos computacionais. Com eles é possível prever localizações geográficas com nichos propícios para a preservação da espécie estudada. Porém, o openModeller Desktop não possui formas triviais de compartilhamento de tais modelos. Este projeto visa justamente prover uma forma simples e rápida de compartilhamento de experimentos de biodiversidade. Para isso, foi desenvolvido um Plugin para o openModeller Desktop e um Servidor, onde os experimentos serão armazenados e disponibilizados a outros usuários. O projeto e desenvolvimento do Plugin e do Servidor será descrito neste documento.

Palavras-chave: Sistemas Distribuídos, Meio Ambiente (Experimentos), SOA

ABSTRACT

The pursuit of the sustainable growth passes indisputably through the preservation of species. Nowadays, researches in this area are being more supported by technology. One of this technologies, the openModeller Desktop, assists scientists and institutes in the creation of species distribution models. These models are created according to geographic species distribution observation, environmental layers (relief, pluviosity, vegetation, weather, etc) and computational algorithms. With them it is possible to predict the geographic locations with propitious niches to preserve the studied species. Nevertheless, the openModeller Desktop do not possess trivial sharing forms of such models. This project aims to promote a fast and simplified biodiversity's experiments sharing. Therefore, a Plugin was developed for the openModeller Desktop and a Server, where the experiments will be stored and available for the other users. The project and the development of the Plugin and the Server will be described in this document.

Keywords: Distributed Systems, Environment (Experiments), SOA

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

USP	Universidade de São Paulo
PCS	Departamento de Engenharia de Computação e Sistemas Digitais
W3C	*World Wide Consortium*
WS	*Web Service*
SW	Serviço Web
JAX-WS	*Java API for XML Web Services*
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
WSDL	*Web Service Description Language*
WBCMS	*Web Biodiversity Collaborative Modelling Service*
XML	*Extensible Markup Language*
SAX	*Simple API for XML*
DOM	*Document Object Model*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
UI	*User Interface*
GUI	*Graphical User Interface*
CRIA	*Centro de Referência em Informação Ambiental*
SGDB	Sistema Gerenciador de Banco de Dados

IUCN	*International Union for Conservation of Nature*
INPE	Instituto Nacional de Pesquisas Espaciais
FAPESP	Fundação de Amparo à Pesquisa do Estado de São Paulo
IDE	*Integrated Development Environment*
API	*Application programming interface*
CCE	Centro de Computação Eletrônica

SUMÁRIO

1 INTRODUÇÃO

A tecnologia de Informática na Biodiversidade (*Biodiversity Informatics*) vem se desenvolvendo rapidamente como um resultado da crescente necessidade de preservar nichos ecológicos que é de importância inegável ao desenvolvimento econômico sustentável. Entretanto, para realizar avanços significativos nesta área é preciso empregar técnicas computacionais avançadas de forma a gerar modelos e conceitos condizentes com a realidade em constante transformação, tanto do ponto de vista tecnológico como ambiental.

Estes recursos exigem cada vez mais capacidade de processamento e armazenamento de dados por parte do aparato físico empregado, bem como um alicerce em software capaz de lidar com todo esse poder. Surgem, neste contexto, ferramentas para modelagem como o framework openModeller Desktop, que reúne um conjunto de funcionalidades para desenvolver estes modelos, além de sua visualização e armazenamento locais. Contudo, o compartilhamento dos dados gerados ainda é deficiente, o que leva a uma repetição supérflua da criação de modelos iguais ou dificuldade de troca de informações entre diferentes instituições e pesquisadores.

Nosso projeto de formatura busca suprir as necessidades computacionais dessa área, fornecendo uma plataforma online para o armazenamento e compartilhamento de modelos de biodiversidade, evitando a redundância de modelos, economizando assim, recursos computacionais, tempo, e dinheiro.

A modelagem de distribuição de espécies é uma ferramenta indispensável no mapeamento de regiões de risco de perda de biodiversidade ou áreas de interesse comercial, principalmente no que se refere à conservação e plantio de culturas que tragam bom retorno ou que permitam impulsionar o desenvolvimento econômico e social em regiões outrora pouco aproveitadas. (SIQUEIRA, 2005)

Para tanto é necessário dispor de recursos que facilitem a obtenção dos dados necessários à modelagem, bem como meios para processar estes dados, apresentá-los e armazená-los para consultas futuras. Neste contexto surgiram diversas ferramentas que agrupam estas funcionalidades e as disponibilizam através de interfaces gráficas amigáveis.

Dentre estas devemos destacar o openModeller Desktop, um arcabouço completo para construção de modelos de distribuição espacial de espécies. A partir de dados ambientais organizados em camadas contendo as informações necessárias pode-se executar algoritmos para obtenção dos resultados.(MUÑOZ et al., 2011)

Entretanto, a demanda computacional, tanto em termos de recursos de processamento como espaço para armazenamento, pode atingir néveis proibitivos, principalmente considerando o volume de dados a serem analisados. Embora inúmeras iniciativas de estudo de aperfeiçoamento da eficiência destes sistemas estejam em andamento, há casos em que não é possível reduzir a quantidade de informações de entrada sem prejudicar o modelo gerado.

Uma alternativa bastante simples e ao mesmo tempo sofisticada para reduzir essa exigência de desempenho é persistir (GONÇALVEZ, 2007), isto é, armazenar os modelos já construídos em um banco de dados de fácil acesso que centralize e ofereça recursos de busca e manipulação da informação ali armazenada. Através deste projeto pretende-se implementar, seguindo conceitos abordados por (FOOK, 2009), este sistema de persistência de modelos sob os alicerces sólidos do openModeller Desktop, reduzindo portanto a demanda computacional e inte-

grando pesquisas e desenvolvimentos realizados pelas instituições que o utilizem.

1.1 Objetivos

Desenvolver sistema de persistência de modelos de distribuição geográfica de espécies visando aprimorar o framework openModeller Desktop e facilitar o compartilhamento de informações entre pesquisadores e instituições em geral, utilizando-se como base conceitos e idéias contidas no WBCMS desenvolvido por Karla Donato Fook durante sua tese de doutorado.

1.2 Objetivos Específicos

Incorporação de um plugin no framework openModeller Desktop, que seja capaz de se comunicar via web service a um banco de dados distribuído em cluster, que também está implementado pelo projeto.

Procuramos implementar as funcionalidades de escrita, consulta, modificação, e organização dos meta dados. Todas as funcionalidades necessitarão de controle de acesso de usuários.

1.3 Motivação

A União Internacional da Conservação da Natureza (IUCN) possui estimativas sobre as espécies que possuem risco de extinção, na figura 1 as legendas significam, respectivamente: LC - Menor Preocupação, NT - Quase ameaçadas, VU - vulneráveis, EN - ameaçadas de extinção, CR - criticamente em perigo, EW - extinta na natureza, EX - extinta.

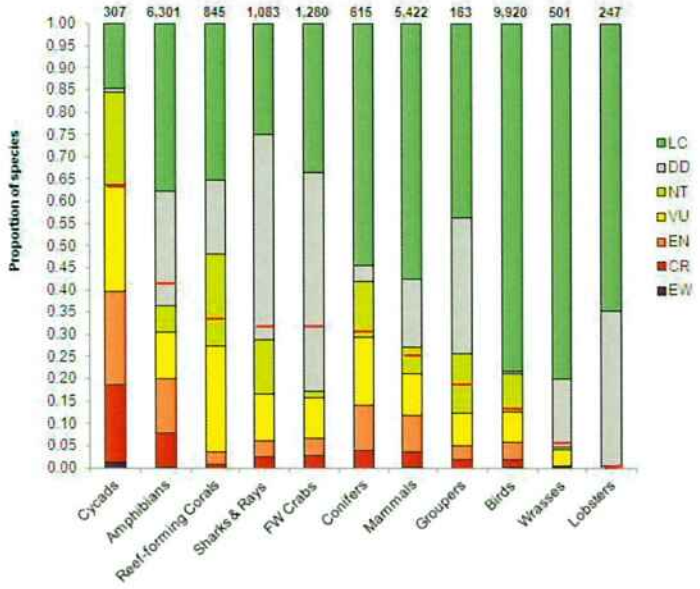

Figura 1: Proporção de espécies existentes (incluindo extintas) - *The IUCN Red List of Threatened Species. Version 2011.2.*

Fonte: IUCN

Conservar a biodiversidade do planeta é um grande desafio, devido às mudanças climáticas naturais ou decorrentes da ação humana, como práticas na agricultura, pecuária, indústria e serviços.

Pesquisadores analisam dados e algoritmos a fim de descobrir as probabilidades de ocorrência de espécies para poder, assim, focar os esforços de pesquisa e conservação.

Os modelos de distribuição de biodiversidade criados são essenciais para esse processo de conservação e pesquisa, no entanto, o compartilhamento desses modelos não é uma tarefa simples. Atualmente a distribuição ocorre através do compartilhamento dos dados e dos resultados, não mostrando o algoritmo utilizado.

Autores já apresentaram uma arquitetura orientada a serviços (SOA) que suportam o compartilhamento de experimentos de distribuição de espécies (re-

sultados, modelagem e informação) através de Web Services.(FOOK, 2009)

O openModeller é um arcabouço completo para a análise e desenvolvimento de modelos de biodiversidade. Dentro deste contexto, propomos um plugin para o openModeller capaz de compartilhar os experimentos desenvolvidos através de Web Services, armazenando os dados em um banco de dados distribuído em cluster.

1.4 Organização

O restante deste trabalho está dividido da seguinte forma:

O Capítulo 2, Revisão da literatura, apresenta o resultado da pesquisa bibliográfica realizada sobre os conceitos e tecnologias envolvidas na produção deste trabalho. O capítulo 3 apresenta as características funcionais, digramas em blocos e principais características do software, que possui o codinome: BioRep. No capítulo 4 é explicado a metodologia empregada no trabalho, envolvendo todas as etapas (concepção do projeto, implementação e testes). No Capítulo 5 é detalhado a implementação realizada, destacando o previsto e o efetivamente realizado. No capítulo 6 indicamos qual a metodologia empregada para atingir os objetivos estabelecidos no trabalho. E, por fim, o capítulo 7 consiste em nossas considerações finais, discutindo os resultados atingidos e não atingidos com suas devidas justificativas, contribuições e perspectivas de continuidade, além de um comentário individual de cada componente do grupo.

2 ASPECTOS CONCEITUAIS

Nesta sessão são explicados os aspectos conceituais, tecnologias envolvidas no projeto e implementação do projeto. Embora nosso foco não seja a modelagem de espécies, precisamos também entender o problema para que possamos contribuir com a pesquisa nessa área, provendo a capacidade de compartilhamento de experimentos.

2.1 openModeller

O openModeller é um framework que visa prover um ambiente flexível, amigável, multiplataforma, onde se pode conduzir todo o processo de modelagem de nichos. Foi desenvolvido pelo Centro de Referência em Informação Ambiental (CRIA), Escola Politécnica da USP (Poli) e pelo Instituto Nacional de Pesquisas Espaciais (INPE) como uma ferramenta *open-source* sendo financiada pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP). Possuiu apoio do projeto BDWorld - da Universidade de Reading e da Universidade do Kansas - Museu de História Natural, e do Centro de Pesquisas de Biodiversidade (*Biodiversity Research Center*).

2.1.1 Modelo de distribuição de Espécies

Um modelo de distribuição de espécies é um modelo que utiliza uma distribuição observada de espécies e/ou características biológicas para predizer o seu

estado atual (ou potencial). Definição de conceitos:

1. Observação: Faixa geográfica em que a espécie é vista.

2. Distribuição atual: distribuição da espécie atual.

3. Distribuição Potencial: Faixa geográfica em que uma espécie pode ocorrer

Estes termos são comuns e amplamente utilizados, porém a distribuição atual é na realidade a distribuição que está sendo modelada, e não a real distribuição das espécies.

2.1.2 Conceito de nicho ecológico

A distribuição de espécies, ou mais precisamente habitat, podem ser ligados ao conceito de nicho ecológico criado por Grinnell em 1917, que definiu nicho ecológico como sendo todas as localizações onde organismos de uma espécie podem viver. Trinta anos depois em 1959, Hutchinson propôs uma nova formulação à definição de Grinnell, como sendo uma região em um espaço multi-dimensional de fatores ambientais (hipervolume $n - dimensional$) que afetam o bem-estar de uma espécie. Finalmente, em 1987 Rosenzwig definiu um nicho ecológico como sendo um conjunto de fatores que permitem uma espécie sobreviver em um determinado ambiente. Deve se compreender que todas as definições citadas são importantes para compreender distribuições de espécies.(CHUINE, 2010)

2.1.3 Modelagem de distribuição de espécies

A modelagem de distribuição de espécies pode ser realizada com a utilização de dois tipos de dados:

1. Dados de ocorrência: Pontos geográficos que representam a ocorrência ou ausência de uma determinada espécie.

2. Dados de ambiente: Dados do nicho ecológico (e.g. temperatura, precipitação, altitude, etc). Neste trabalho, utilizamos esse conceito para definir o conceito de *layer* que consiste em uma camada digital das variáveis ambientais.

A partir destes dados podem-se criar, através de algoritmos (que se baseiam em encontrar correlação ou deduzir através dos *layers*), cenários potenciais da distribuição da espécie.

Por exemplo, caso se utilize *layers* que exprimam as condições atuais do ambiente pode-se obter um cenário potêncial desta distribuição, caso se utilize *layers* que exprimam condições do passado pode-se obter um cenário desta distribuição no passado, e caso se utilize *layers* que representem a expectativa das mudanças no ambiente pode-se inferir estas expectativas em um cenário futuro.

O processo de modelagem pode ser visto no diagrama da figura 2.

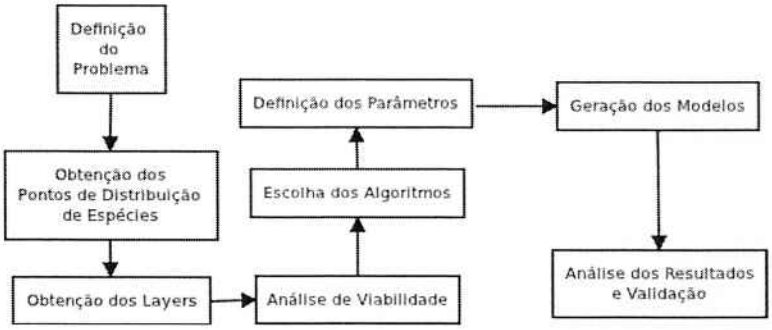

Figura 2: Processo de modelagem de distribuição de espécies.

Fonte: Adaptado de (CORRÊA; RODRIGUES; RODRIGUES, 2010).

2.1.4 openModeller Desktop

O openModeller Desktop é uma GUI (*Graphical User Interface*) que provê acesso à biblioteca do openModeller, facilitando a utilização da ferramenta openModeller aos pesquisadores em geral, que podem não ter um conhecimento em informática suficientemente adequado para a utilização de um programa em linha de comando. Atualmente ele suporta duas plataformas, a saber: Windows (9x, ME, 2k, NT, XP, Vista, 7) e GNU/Linux.

Nosso projeto pode ser compreendido como um plugin ao openModeller Desktop, provendo, dentro da própria ferramenta, a capacidade de compartilhamento de experimentos de modelos de distribuição de espécies.

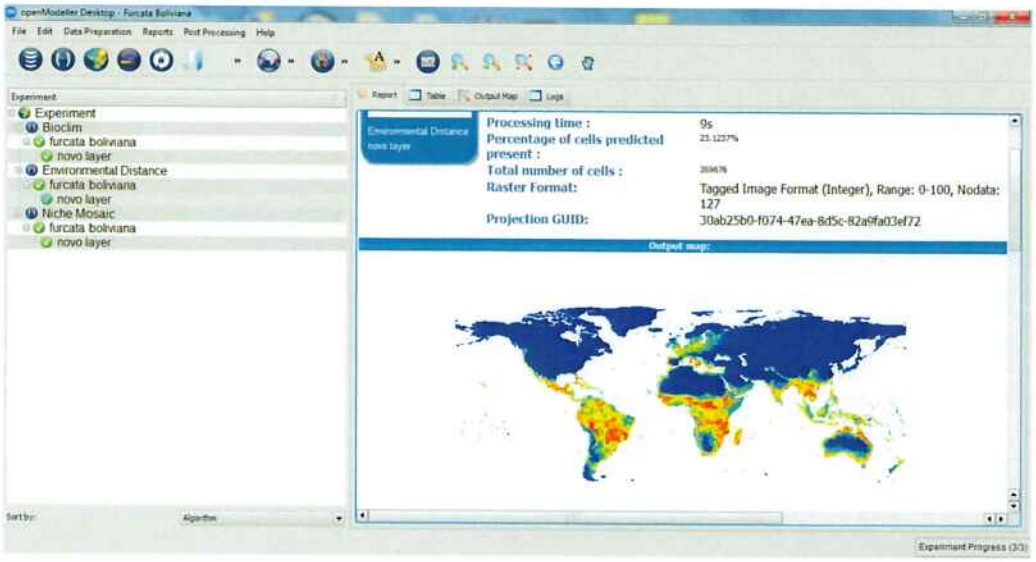

Figura 3: Processo de modelagem de distribuição de espécies.

Fonte: Autores.

2.2 WBCMS

O WBCMS (*Web Biodiversity Collaborative Modelling Services*) é um conjunto de Web Services que visa apoiar o compartilhamento de resultados de mo-

delos de distribuição de espécies via Web. Estes serviços também permitem incluir comentários e a fonte das informações. Os protocolos do WBCMS utilizam a idéia de instâncias de modelos, que descrevem o experimento como um todo, incluindo os dados e metadados relacionados aos modelos, resultados e algoritmos. Um pesquisador ao inspecionar uma instância de modelo consegue extrair informações de como os resultados foram produzidos, podendo fazer comparações e utilizar estas informações para suas próprias modelagens.(FOOK, 2009)

2.2.1 Instância de modelo

Uma instância de um modelo no WBCMS consiste em três partes: descrição do objeto, geração do modelo e resultados, além de conter seus próprios metadados como nome, título, descrição, autor, entre outras.(FOOK et al., 2009)

A primeira parte: descrição do objeto guarda as informações sobre as espécies que estão sendo modeladas. A segunda parte: geração do modelo inclui dados e métodos utilizados no modelo de distribuição de espécies, incluindo informações como: localização das espécies e seus metadados, camadas ambientais e algoritmos utilizados. A terceira parte consiste dos principais resultados do modelo, um conjunto de dados georeferenciados.(FOOK, 2009)

2.2.2 Arquitetura

O WBCMS considera que os pesquisadores desejam compartilhar seus experimentos através da Web. Ele recebe os resultados da modelagem através de uma aplicação cliente, e acessa dados remotos e web services para criar as instâncias de modelo, além de inserir este modelo em um repositório para torná-lo acessivel.

Sua arquitetura é consideravelmente complexa, sendo composta por três principais componentes: *Catalogue processor*, *Access Processor* e *Model Processor*.

O *Catalogue processor* consiste de quatro serviços, que trabalham em conjunto com o objetivo de compor modelos de instância. O *Access processor* funciona como uma interface de acesso, onde pesquisadores podem procurar e visualizar instâncias de modelos. O *Model processor* utiliza dois outros web services para permitir que modelos possam ser executados remotamente, permitindo aos usuários modificar parâmetros dos algoritmos e executar modelos reutilizando dados catalogados.(FOOK, 2009)

2.2.3 Principais diferenças

O estudo do WBCMS foi de grande valia para este projeto. No entanto, ao invés de focar na distribuição de resultados, nosso sistema faz o compartilhamento do experimento, permitindo que os pesquisadores modifiquem qualquer informação dentro do openModeller Desktop e gerem seus próprios resultados.

2.3 SOA

Segundo o W3C, SOA (*Service-Oriented Architecture* ou Arquitetura Orientada a serviços) é uma forma de arquitetura de sistemas distribuídos que pode ser caracterizada por conter as seguintes propriedades(BOOTH et al., 2004; ENDREI et al., 2004):

- Visão lógica: O serviço é uma abstração (dos programas, bancos de dados, processos de negócio, entre outros), sendo definido em termos do que ele faz, tipicamente realizando uma operação a nível de negócio.
- Orientado a mensagens: O serviço é formalmente definido de acordo com as mensagens trocadas entre os agentes provedores de serviço e os agentes requisitantes, e não sobre as propriedades dos agentes envolvidos. A estrutura interna destes agentes, incluindo suas características como linguagem de im-

plementação, estrutura dos processos e até mesmo sua estrutura de banco de dados são abstraídos, ao se utilizar a arquitetura SOA não é necessário saber como um agente que fornece um serviço é construído.

- Orientado a descrição: Um serviço é descrito como um meta-dado que pode ser processável por uma máquina. A descrição pública do SOA deve conter apenas o que é importante para o uso do serviço, a semântica de um serviço deve ser documentada, direta ou indiretamente por sua descrição.

- Glanularidade: Serviços tendem a possuir um pequeno número de operações, no entanto utilizar grandes e complexas mensagens.

- Orientado a rede: Serviços tendem a ser orientados para seu uso sobre uma rede, embora isto não seja um requisito.

- Independência de plataforma: As mensagens enviadas são independente de plataforma, seguindo um padrão através das interfaces.

2.3.1 Web Service

Web Service é definido pela W3C como "um sistema de software projetado para suportar a interoperabilidade entre máquinas em uma rede", a sua base consiste em dois padrões: XML (eXtensible Markup Language) e SOAP (Simple Object Access Protocol).(BOOTH et al., 2004)

XML é uma linguagem simples e muito flexível, fornecendo a descrição, armazenamento e o formato da transmissão para a troca de dados através dos Web Services. SOAP por sua vez, é o protocolo de comunicação, que utiliza o XML como formato da mensagem e HTTP ou HTTPS para a comunicação. Os serviços são definidos utilizando a linguagem WSDL (Web Service Description Language), uma linguagem padronizada em formato XML.(ENDREI et al., 2004; BOOTH et al., 2004)

Web Service é uma abstração que deve ser implementada por um agente. O agente é algo concreto, software ou hardware que envia e recebe mensagens. Deve-se notar que o agente pode ser implementado em qualquer linguagem, podendo inclusive mudar de linguagem ou tecnologias, no entanto se ele implementa a mesma funcionalidade o Web Service ainda é o mesmo.

2.3.1.1 WSDL

WSDL (*Web Service Description Language*) é a linguagem de descrição de um Web Service que se baseia em XML, esta linguagem funciona como um contrato de serviço, descrevendo o serviço, como acessá-lo e quais as operações ou métodos disponíveis. Ele descreve os Web Services como sendo um conjunto de pontos de entrada aos serviços (*endpoint*) que operam sobre mensagens.(BOOTH et al., 2004)

2.3.1.2 SOAP

SOAP (*Simple Object Access Protocol*(BOOTH et al., 2004) ou em português, Protocolo Simples de Acesso a Objetos), utiliza XML para sua formatação e algum protocolo de transporte para a troca de mensagens (e.g. RPC, HTTP). Este protocolo consiste em três partes:

- *Envelope*: Define o que está na mensagem e como processá-la.
- *Header*: Elemento que contém informações específicas da aplicação, como método de autenticação, pagamento, etc.
- *Body*: Este elemento contém a informação que deve ser transmitida para o Web Service.

Essa estrutura pode ser vista na figura 4.

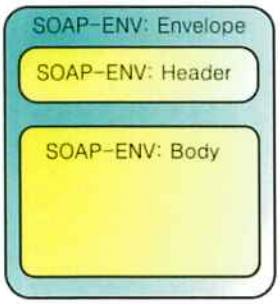

Figura 4: Estrutura do Envelope SOAP.

Fonte: Wikipedia.

2.4 gSoap

É um conjunto de ferramentas opensource para C e C++ com intuito de auxiliar o desenvolvimento de Web Services que utilizam SOAP e XML. Ele analiza WSDLs e esquemas XML (*XML schemas*) e mapeia os esquemas XML e o protocolo SOAP em um código C/C++ eficiente e simples de utilizar. Esta ferramenta é utilizado pelas maiores 5 empresas de tecnologia do mundo.(GSOAP..., 2011)

De posse dos WSDLs do servidor, utilizamos esta ferramenta para criar o cliente Web Service, gerando código C++ automaticamente. Desta forma alavancamos o desenvolvimento, e abstraímos todo o desenvolvimento de controle do protocolo, podendo focar todo o nosso esforço na concepção da interface e melhoria dos serviços no servidor.

A figura 5 ilustra o processo de criação, em que se fornece um arquivo WSDL, e automaticamente é gerado código C++

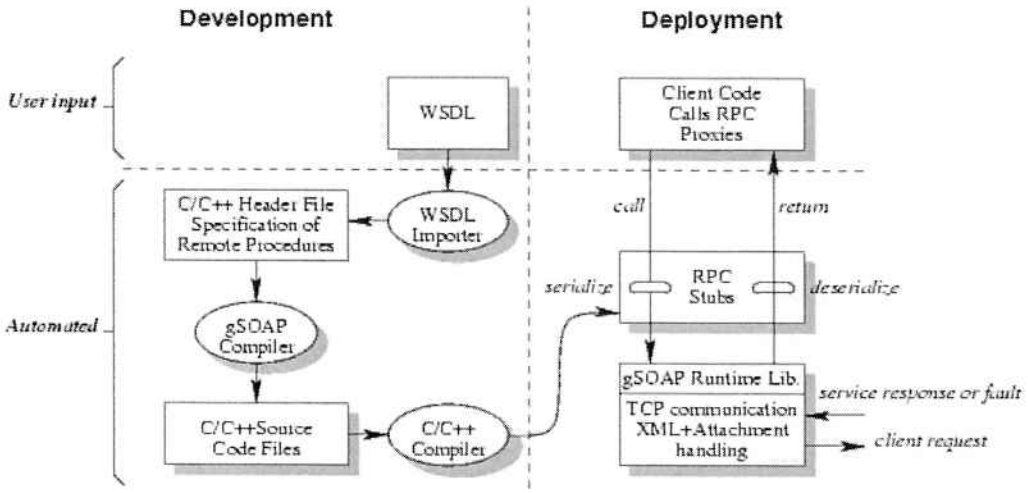

Figura 5: Processo de criação do cliente Web Service.

Fonte: Adaptado da documentação do gSoap.

2.5 Framework Qt

Trata-se de um framework multiplataforma, tendo suporte aos principais Sistemas Operacionais, como Windows, Linux e Mac/OS. Sua arquitetura pode ser vista na figura 6.(ONLINE..., 2011)

Possui uma poderosa biblioteca de interface gráfica escrita em C++, e uma IDE própria que facilita a programação, pois é possível criar UI's (User Interface) iterativamente, havendo geração automática de código.

O Qt foi desenvolvido por uma empresa norueguesa, a Trolltech, que foi adquirida pela Nokia. Atualmente, um setor da Nokia (Nokia's Qt Development Frameworks) mantém o projeto.

O grande diferencial do framework é que, apesar do ser escrito em C++, o Qt faz uso extensivo de geração automática de código, utilizando para isso um

poderoso gerador de código chamado Meta Object Compiler, ou "moc".

A biblioteca gráfica do Qt faz grande uso de macros, o que possibilita maior produtividade.

Além da parte gráfica, Qt possui outras características interessantes aos desenvolvedores, como monitoração de aplicações multithread, acesso a bancos de dados SQL, suporte a XML, suporte à utilização de outras linguagens, como Java entre outras.

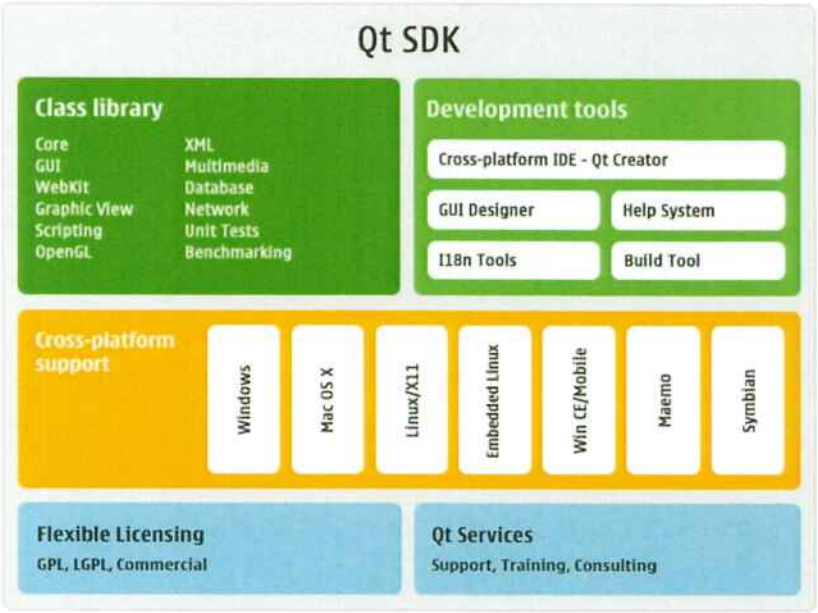

Figura 6: Qt SDK - Arquitetura de desenvolvimento.

Fonte: Qt Project.

2.5.1 A utilização do Qt no nosso projeto

A ferramenta de geração de modelos de distribuição geográfica de espécies, o openModeller, foi desenvolvido utilizando Qt.

Como o escopo de nosso projeto está totalmente ligado à esta ferramenta, é preciso ter um conhecimento aprofundado sobre o framework e seu ambiente de desenvolvimento. Assim, o estudo e utilização de Qt é prioridade na fase inicial do projeto, para criarmos uma base de conhecimento para a implementação do plugin do openModeller.

A facilidade que o Qt oferece na criação de interfaces, criando automaticamente os objetos, sua disposição na janela, seus parâmetros e configurações iniciais foram de suma importância para focarmos na lógica do cliente, otimizando o tempo de projeto.

2.6 PostgreSQL

Utilizamos em nosso projeto o SGBD PostgreSQL, um poderoso banco de dados opensource, que possui um mapeamento objeto-relacional, suportando diversas plataformas, inclusive o cluster que estamos utilizando, que já possuía uma versão instalada em produção.(GROUP, 2010)

PostgreSQL foi criado na Universidade da Califórnia de Berkeley pelo professor Michael Stonebraker. Inicialmente chamado de Postgres, o projeto foi iniciado em 1986. A intenção do projeto era criar um banco de dados que explorasse os novos conceitos de tecnologias objeto-relacionais.(GROUP, 2010)

Utilizamos, ainda, o pgAdminIII, uma poderosa plataforma de administração e desenvolvimento para PostgreSQL, usada tanto para escrita e execução de simples queries SQL como para desenvolvimento de bases de dados complexas.

2.6.1 PostgreSQL no projeto

PostgreSQL foi utilizado no Servidor provedor dos Serviços Web. Foi utilizado para armazenar os usuários cadastrados, os layers referenciados e os experimentos.

Os layers, por serem arquivos grandes, foram armazenados em disco e seus caminhos referenciados por um campo (path) da tabela de layers. O arquivo xml que representa o experimento é armazenado no próprio banco, sendo que alguns campos-chave são extraídos e armazenados na tabela de experimentos para otimizar queries de busca.

2.7 Java

A linguagem Java foi criada por um pequeno grupo de engenheiros da Sun chamados de *Green Team*. Java é uma linguagem planamente difundida, que utiliza o paradigma da orientação a objetos.

Hoje em dia, Java está presente em praticamente todos os dispositivos, desde telefones celulares portáteis a servidores.

2.7.1 Java no projeto

Utilizamos Java na versão 1.7 para produzir os Web Services que ficarão hospedados no servidor. Escolhemos essa linguagem por ser amplamente difundida, fácil de manter, e por possuir uma infinidade de bibliotecas de apoio. Aproveitamos também a infra-estrutura da IDE Netbeans 7.0, de forma a alavancar o processo de desenvolvimento, abstraindo a camada SOAP, projetando as interfaces e criando código java para implementar os métodos, gerando de forma automática o WSDL, além de poder automatizar todo o tratamento da camada de transporte utilizando servidores java como Tomcat ou Glassfish.

Algumas bibliotecas e técnicas de programação foram fundamentais para agilizar a fase de desenvolvimento e fornecer segurança no acesso ao banco de dados.

2.7.1.1 JDOM

JDOM é uma API Java que tem por objetivo oferecer manipulação de XML de maneira simples, intuitiva e otimizada. A facilidade de uso do JDOM foi essencial para o desenvolvimento do projeto, já que a manipulação de arquivos XML foi recorrente. Por analisarmos arquivos XML relativamente grandes, termos um manipulador XML com algoritmo otimizado foi de suma importância para não termos *overheads* desnecessários.(HAROLD, 2002)

As classes de suporte à manipulação de XML são:

- Document: representa um documento XML. Através do *SAXParser* é possível construir Documents tanto a partir de arquivos XML como de *Strings* que contêm um código XML. Possui um elemento raiz a partir do qual é possível manipular todo o arquivo.

- Element: representa um elemento XML. Pode possuir atributos, conteúdo na forma de texto ou outros elementos. Permite recuperar e alterar atributos e conteúdo. Possui métodos que acessam seus elementos filhos. Assim, partindo-se do elemento raiz, é possível percorrer e manipular todo o documento XML através de seus filhos.

2.7.1.2 SAX

Utilizamos SAX, uma API para manipulação de XML, que oferece um mecanismo para ler dados de um documento XML, provendo uma alternativa ao DOM (Document Object Model). Eles operam de maneira diferente, o SAX lê o XML parte a parte sequencialmente, oposto ao DOM que trabalha com o documento inteiro. Isto possibilita uma diminuição significativa da quantidade de memória utilizada, além de ser mais rápido.(HAROLD, 2002)

2.7.1.3 *Singleton*

Uma classe *singleton* permite a criação de apenas um objeto ativo. É interessante ter classes *singletons* que acessem o banco de dados, pois assim o número de conexões é limitado (apenas uma conexão), não permitindo que erros de inconsistência ocorram.(GAMMA et al., 1995)

2.8 Cluster

Cluster: Cluster é um conjunto de máquinas conectadas, normalmente através de redes locais de alta velocidade. Possui diversas aplicações, como oferecer alta performance computacional ou oferecer suporte a aplicações com requisitos prioritários de disponibilidade.(VRENIOS, 2002)

Quando bem gerenciado, clusters podem distribuir de forma inteligente a execução de uma única aplicação. Logicamente, é necessário que o Sistema Operacional realize esta complexa tarefa de distribuição de processamento e troca de dados entre as máquinas.

Os principais Sistemas Operacionais possuem suporte ao gerenciamento de clusters, como o Linux-HA ou o LanderCluster, que pode rodar nas plataformas do Windows, Linux e UNIX.

Utilizamos um cluster existente no CCE da USP. Ele posssui a seguinte infraestrutura de Hardware: SGI Altix XE 1300 - nó de entrada Altix XE 210, dois processadores Xeon quad Core, 2GHz, 8GB RAM, HD 500 GB, 10 nós Altix XE 310, cada um com dois processadores Xeon quad Core, 2GHz, 8GB RAM e HD 250 GB, totalizando 80 cores. Sua infraestrutura de Software é: Condor: software especializado em gestão de carga em cluster para computação intensiva, LAM (Local Area Multicomputer) e MPI: implementação do padrão Message Passing Interface (MPI).

3 ESPECIFICAÇÃO DO PROJETO DE FORMATURA

Este capítulo apresenta a especificação do software desenvolvido, iniciando pela análise de requisitos, casos de uso, arquitetura e projeto de interface.

3.1 Análise de requisitos

Foram considerados, além de nossas idéias originais as opiniões dos seguintes stakeholders: desenvolvedores, usuários, orientador, coorientador. Os requisitos levandatos estão expostos na tabela 1.

Tabela 1: Requisitos do sistema

Nº	**Nome**	**Tipo**
1	Arnazenamento	Funcional
2	Recuperação	Funcional
3	Alteração	Funcional
4	Versionamento	Funcional
5	Controle de Usuários	Funcional
6	Controle de Licensas	Funcional
7	Interface entre Banco de Dados e Plugin	Não funcional
8	Desenvolvimento do Plugin	Não funcional
9	Banco de Dados	Não funcional
10	Integração	Não funcional
11	Visualização de Experimento Armazenado	Funcional
12	Integridade de Dados	Não Funcional
13	Tratamento de Erros	Não Funcional
14	Atomicidade do Banco de Dados	Não Funcional
15	Consistência do Banco de Dados	Não Funcional
16	Isolamento do Banco de Dados	Não Funcional
17	Durabilidade do Banco de Dados	Não Funcional

3.1.1 Requisitos Funcionais

A descrição dos requisitos funcionais estão apresentados na tabela 2. O requisito número 4 não foi aceito, pois consideramos que cada experimento postado é um novo experimento, e não uma nova versão de um experimento anterior.

Tabela 2: Requisitos Funcionais do sistema

Nº	Descrição	Status
1	O sistema deve permitir o armazenamento de experimentos de modelos de distribuição de espécies produzidos no openModeller Desktop.	Aceito
2	O sistema deve permitir a recuperação de experimentos de modelos de distribuição de espécies produzidos no openModeller Desktop.	Aceito
3	O sistema deve permitir a alteração de experimentos de modelos de distribuição de espécies produzidos no openModeller Desktop.	Aceito
4	O sistema deve permitir o versionamento de experimentos de modelos de distribuição de espécies produzidos no openModeller Desktop.	Não aceito
5	O Sistema deve fazer o controle de usuários, dando permissões de acordo com sua característica.	Aceito
6	O Sistema deve respeitar as licenças e direitos autorais em que os experimentos foram distribuídos, promovendo a segurança dos dados e dos resultados.	Aceito

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais estão apresentados na tabela 3.

Tabela 3: Requisitos Não Funcionais do sistema

Nº	Descrição	Status
7	O Sistema deve utilizar Web Services para fazer a interface entre o banco de dados e o plugin do openModeller Desktop.	Aceito
8	O Plugin deve ser desenvolvido utilizando a linguagem C−+, utilizando a biblioteca Qt para o desenvolvimento das interfaces.	Aceito
9	O banco de dados deve estar distribuído em um cluster e ser capaz de armazenar dados geográficos.	Aceito
10	O sistema deve integrar o plugin desenvolvido no openModeller com os Web Services desenvolvidos.	Aceito
11	O sistema deve permitir que o usuário visualize um experimento armazenado junto com sua respectiva descrição.	Aceito
12	Os dados de um experimento não podem ser modificados a não ser pelo seu criador original.	Aceito
13	Caso ocorra erros de conexão, comunicação, ou outros tipos o sistema não deve exibir erros ou travar, deixando o funcionamento transparente para o usuário.	Aceito
14	As transações devem ser atômicas, sendo que etapas parciais não devem influir no estado do banco de dados.	Aceito
15	A integridade dos dados deve assegurada, ou seja, as transações não podem quebrar as regras do Banco de Dados.	Aceito
16	Nenhuma transação pode ocorrer concomitantemente se essas transações alterarem o mesmo dado no banco de dados.	Aceito
17	Os efeitos de uma transação em caso de sucesso são permanentes, mesmo em presença de falhas	Aceito

3.2 Casos de Uso

Esta sessão descreve os casos de Uso do sistema, inicialmente são listados todos os casos de uso divididos por contexto. A tabela 5 descreve os casos de uso referentes a manipulação de usuários, a tabela 6 descreve os casos de uso referentes a manipulação de experimentos, e a tabela 7 descreve outros casos de uso. A descrição de cada caso de uso pode ser encontrada no apêndice A.

Tabela 5: Manipulação de Usuários

Nº	**Nome**	**Descrição**	**Atores**
1.1	Cadastro de usuário	Este caso de uso descreve o processo de cadastro de usuários no sistema.	Novo Usuário
1.2	Exclusão de conta de usuários	Este caso de uso descreve a remoção de um usuário existente.	Usuário
1.3	Alteração de usuários	Este caso de uso descreve a alteração dos dados de um usuário existente.	Usuário
1.4	Recuperação de usuário	Este caso de uso descreve a consulta aos dados de um usuário existente.	Usuário

Tabela 6: Manipulação de Experimentos

Nº	**Nome**	**Descrição**	**Atores**
2.1	Cadastro de Experimento	Este caso de uso descreve o processo de cadastro de novos experimentos no sistema.	Usuário
2.2	Recuperação de Experimento	Este caso de uso descreve o processo de obtenção de experimentos já cadastrados no sistema.	Usuário
2.3	Consulta Experimento	Este caso de uso descreve o processo de consulta do acervo de experimentos.	Usuário

Tabela 7: Outros Casos de Uso

Nº	Nome	Descrição	Atores
3	Login	Este caso de uso descreve o processo de login de usuários no sistema.	Usuário
4	Logoff	Este caso de uso descreve o processo de logoff de usuários no sistema.	Usuário

3.3 Arquitetura

A arquitetura do sistema pode ser vista na figura 7, mostrando como se ocorre a integração das partes, o bloco BioRep Plugin é o plugin que foi incorporado ao openModeller Desktop, apenas a título informativo foi destacado duas partes do openModeller Desktop, que é a interface em Qt e também a biblioteca openModeller. O Módulo BioRep Plugin se comunica através de mensagens utilizando Web Services com o BioRep Server que esta distribuido em cluster.

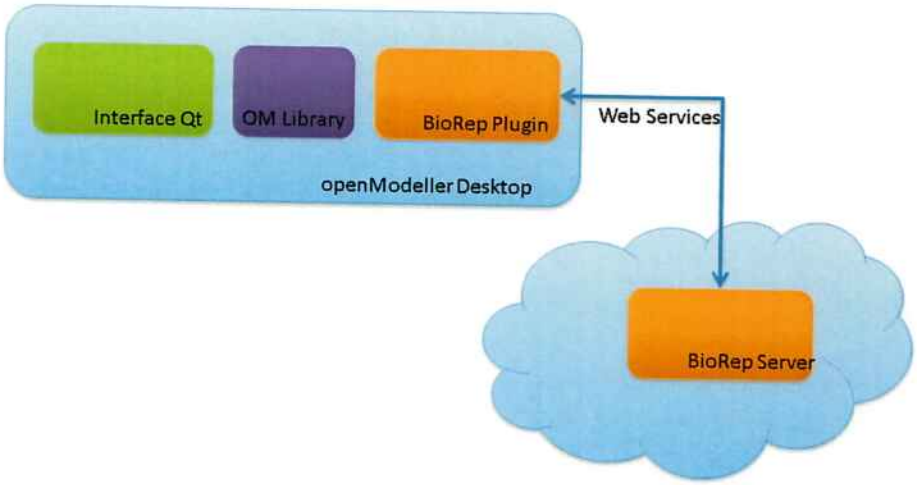

Figura 7: Arquitetura do Sistema

Fonte: Autores.

3.3.1 BioRep Plugin

Na figura 8 são destacados algumas das tecnologias e conceitos envolvidos na criação do plugin. A interface foi programada utilizando o framework Qt na linguagem C++.

Este módulo foi incorporado ao openModeller Desktop, através da criação de um menu na sua interface principal, com um item que inicia o plugin BioRep. Este faz a comunicação com o servidor BioRep através de um cliente Web Service, que foi criado baseando-se nos arquivos de descrição dos serviços (WSDL) utilizando-se da ferramenta gSoap. O método de criação deste cliente será explicado no capítulo 4.

Figura 8: Componentes do Plugin

Fonte: Autores

3.3.2 BioRep Server

Na figura 9 estão destacadas algumas das tecnologias e conceitos envolvidos na criação do servidor. Optou-se por utilizar-se a linguagem Java para a criação deste agente. Embora qualquer servidor Java pudesse ser utilizado, optamos pelo Glassfish 3.0.1. Este servidor foi implantado em um cluster, e seu banco de dados distribuído neste mesmo cluster. Os detalhes das escolhas das tecnologias estarão descritas no capítulo 4, e sua implementação descrita no capítulo 5.

Figura 9: Componentes do Plugin

Fonte: Autores.

3.4 Diagrama de Classes

Diagramas de Classes especificam a estrutura das classes e suas relações. A figura 10 representa o diagrama de classes do lado servidor.

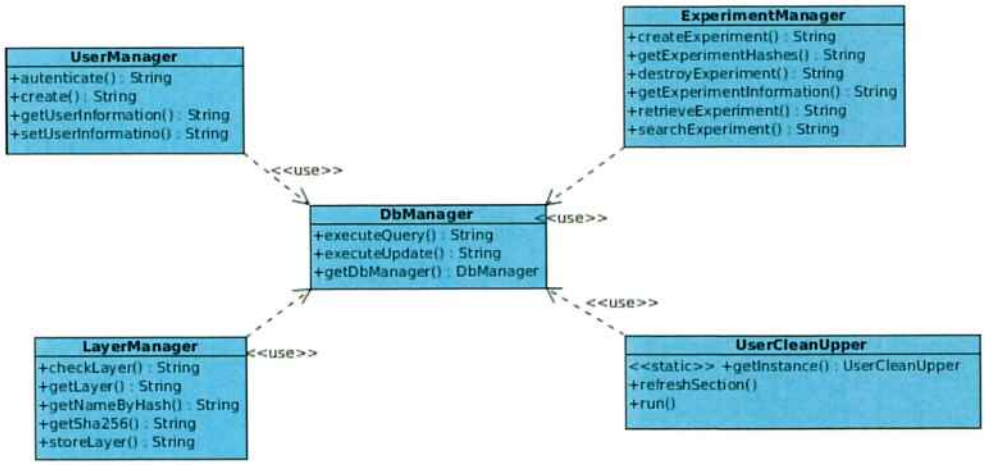

Figura 10: Diagrama de Classes - Servidor

Fonte: Autores.

DbManager é a única classe que faz acesso ao banco de dados. Para controle de conexões com o banco de dados, fizemos dessa classe um singleton

(descrito no capítulo 2). O objeto desse singleton é acessado através do método *getDbManager()*.

A classe *UserCleanUpper* é utilizada para validar a sessão de um usuário. As outras classes implementam os *Web Services*: *UserManager* oferece métodos web de manipulação de usuários, *LayerManager* de manipulação de Layers e *ExperimentManager* de manipulação de experimentos. Os *Web Methods* utilizados nessas classes serão descritos no capítulo 5.

3.5 Diagramas de Sequência

Diagrama de Sequência é uma das ferramentas UML usadas para representar interações entre objetos de um cenário, realizadas através de chamadas de métodos. Este diagrama é construído a partir dos casos de uso. Abaixo, alguns diagramas importantes que foram utilizados nas funcionalidades mais críticas do sistema. Os diagramas de sequência foram baseados no modelo MVC (Model View Controller), não representando as classes reais do sistema, funcionando apenas como uma abstração de camadas.

3.5.1 Create Experiment

Experimentos referenciam uma série de *Layers*, que são arquivos que representam camadas ambientais. Baseado nessas camadas, o algoritmo gera uma distribuição da espécie em estudo. O processo de criação de um experimento no Web Service pode ser visto na figura 11.

Assim, para persistir um experimento no Servidor, é preciso que todos os *Layers* referenciados sejam persistidos também. Para persistir um experimento, então, inicialmente é preciso checar se os *Layers* já existem no Servidor. Se não existirem, se faz necessário o cliente enviar tais Layers.

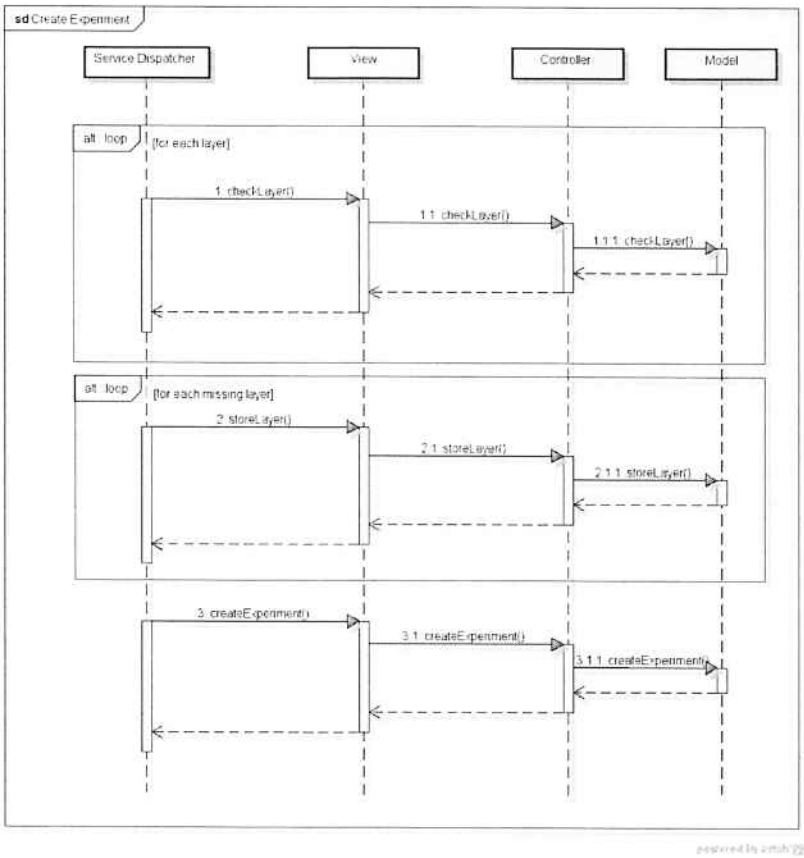

Figura 11: Create Experiment

Fonte: Autores.

O algoritmo para envio de experimento se inicia com a chamada do método *checkLayer()*, que passa como parâmetro o *hash* do *Layer* em questão. Isso é feito para todos os *Layers* referenciados no experimento. Ao final do *loop*, temos uma lista de todos os *Layers* ausêntes no servidor.

É realizado outro *loop* que envia os *Layers* faltantes no servidor através do *Web Method storeLayer()*. Os arquivos dos *Layers* são relativamente grande, podendo este método demorar.

Com todos os *Layers* do experimento já no servidor, pode-se mandar o XML

que representa o experimento.

3.5.2 Retrieve Experiment

Assim como no caso anterior (Create Experiment), não se sabe se os *Layers* referenciados no experimento que se deseja baixar existem na máquina local. Novamente, então, será preciso checar se os *Layers* já existem na máquina local, e em caso negativo, deve-se baixar o *Layer*. O diagrama de sequência está representado na figura 12.

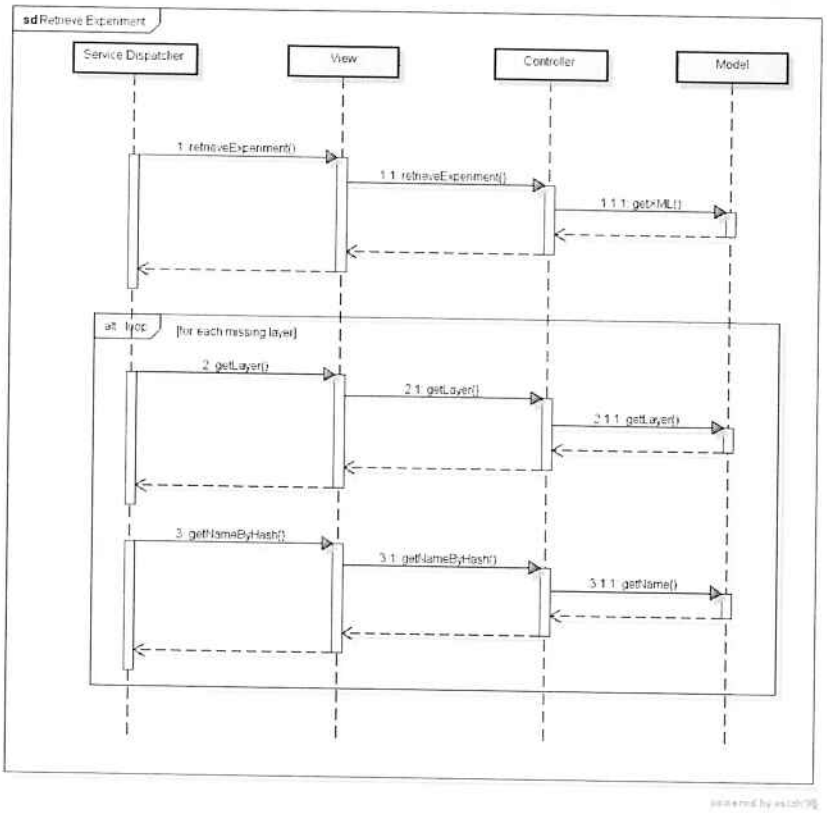

Figura 12: Retrieve Experiment

Fonte: Autores.

Deve-se, inicialmente, baixar o XML do experimento. Pela lógica utili-

zada, as referências aos *Layers* nesse XML foram substituídas pelos respectivos *hashes*. Assim, para cada *hash* encontrado no XML, deve-se comparar com os *hashes* dos *Layers* que já estão armazenados na máquina cliente.

Aqueles layers que não foram encontrados na máquina cliente devem ser baixados. Para isso, envia-se o *Web Method getLayer()* ao servidor, passando como parâmetro o *hash* do *Layer*. Os nomes dos arquivos desses *Layers* são desconhecidos pelo cliente, já que no lugar do nome do *Layer* no arquivo XML, havia o seu *hash*. Portanto, para recuperar o nome do *Layer*, foi criado o *Web Method getNameByHash()*, em que se passa como parâmetro o *hash* e é retornado o nome do *Layer*.

3.5.3 Search Experiment

Este diagrama de sequência retrata o caso de uso busca de experimentos. Os campos disponibilizados para busca são *Name, Date, Author, Specie, Algorithm*. Seu diagrama pode ser visto na figura 13

Após a confirmação da busca (devem ser preenchidos um ou mais campos), o *Web Method searchExperiment()* é enviado ao servidor, contendo os campos da busca. Uma *query SQL* é formada e o banco de dados retorna os *ID's* dos experimentos que se enquadram na busca. Esses *ID's* são retornados ao cliente.

No plugin do openModeller, então, aparece uma lista com os experimentos que satisfizeram a busca. Assim que o usuário selecionar um dos experimentos, é preciso que as informações deste sejam disponibilizadas na interface do plugin. Para isso, chama-se o método *getExperimentInformation()* passando como parâmetro o *ID* do experimento selecionado.

O servidor recupera as informações necessárias fazendo acesso ao banco de dados e as organiza em um documento XML, retornando-o ao cliente, que exibe

as informações na interface.

Este caso de uso pode ser seguido pelo *Retrieve Experiment* descrito acima. Basta o usuário selecionar o botão de *Download* da interface.

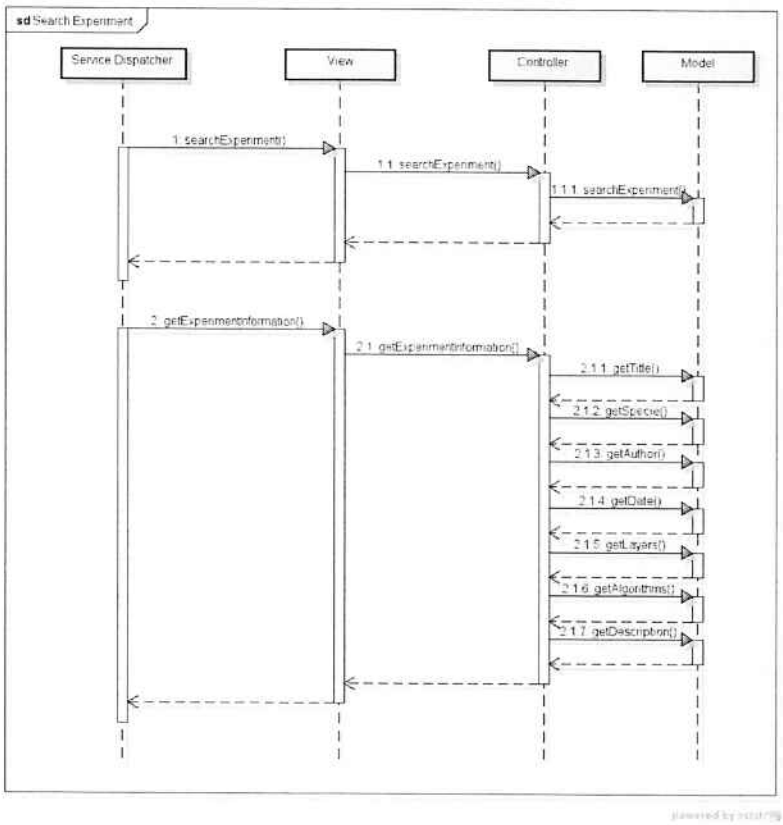

Figura 13: Search Experiment.

Fonte: Autores.

3.6 Projeto de Interface

Optou-se por uma abordagem minimalista, em que todas as funções pudessem ser acessadas em poucos clicks. A interface utilizada pode ser vista nas figuras 14, 15, 16, 17, 18 e 19.

Figura 14: Interface - Sign In

Fonte: Autores

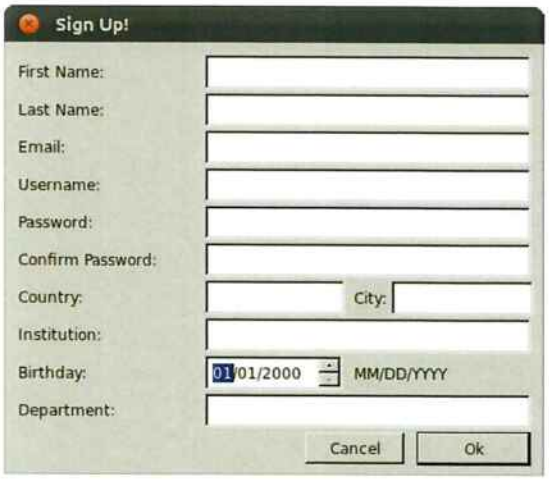

Figura 15: Interface - Sign Up

Fonte: Autores

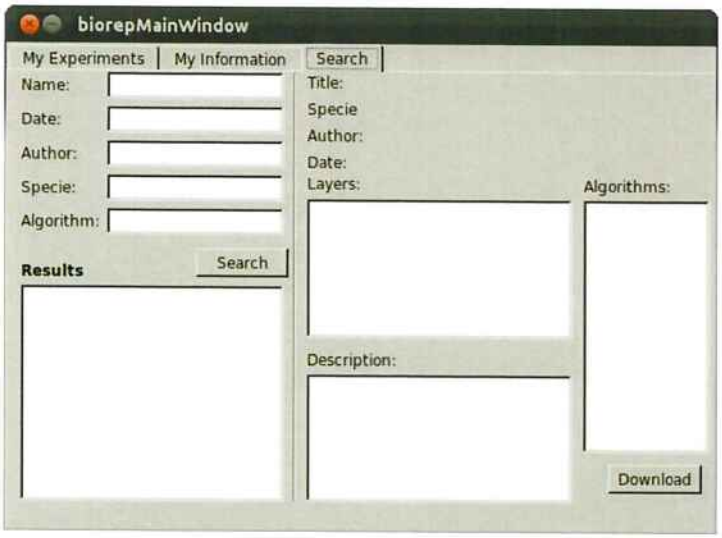

Figura 16: Interface - Search Tab

Fonte: Autores

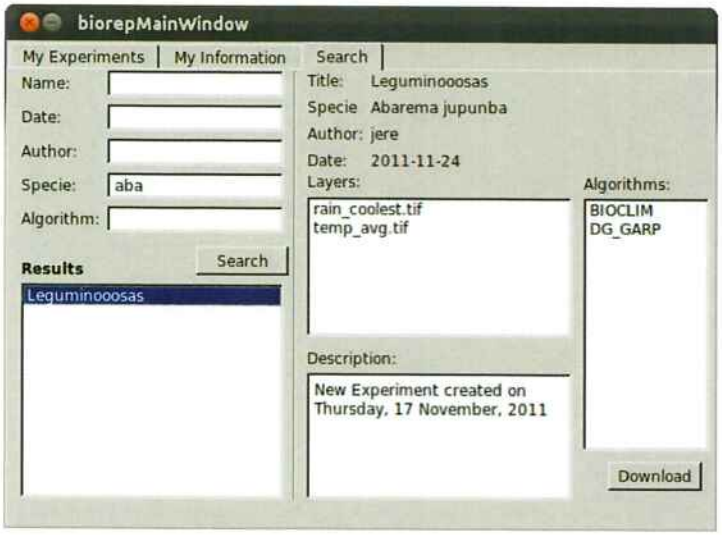

Figura 17: Interface - Search Tab

Fonte: Autores

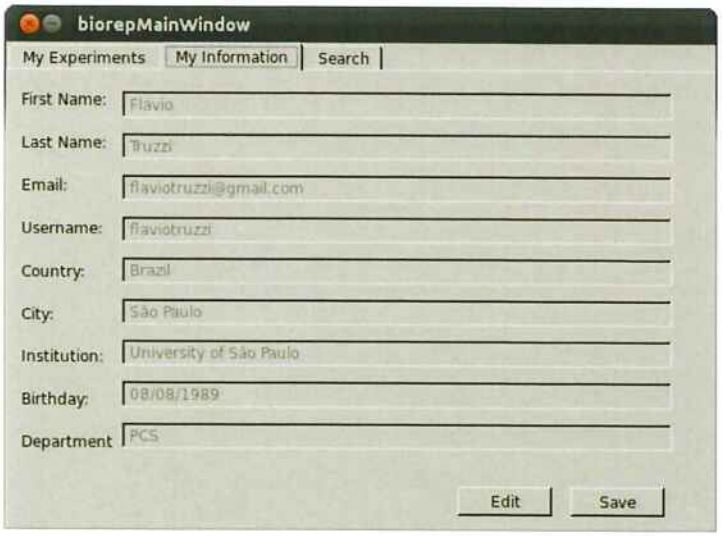

Figura 18: Interface - User Information Tab

Fonte: Autores

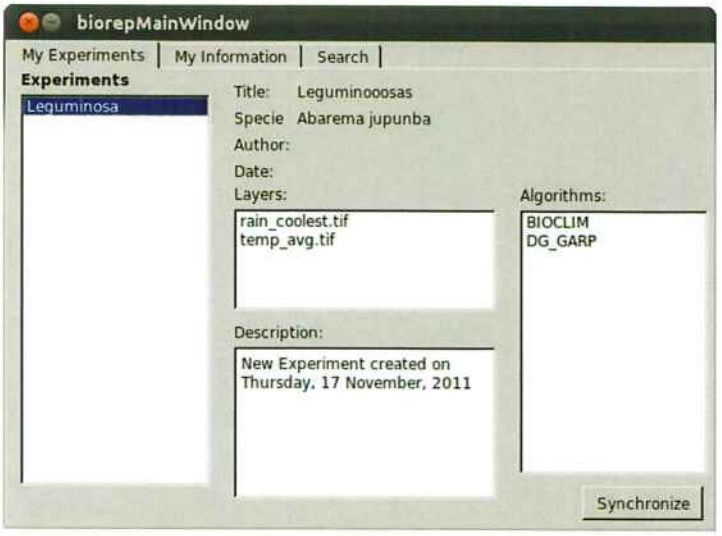

Figura 19: Interface - Sign Up

Fonte: Autores

3.7 Banco de Dados

3.7.1 Tabelas

Foram criadas três tabelas: *Users, Layers* e *Experiments.* Cada uma das tabelas é descrita abaixo. O diagrama de entidade relacionamento pode ser visto na figura 20

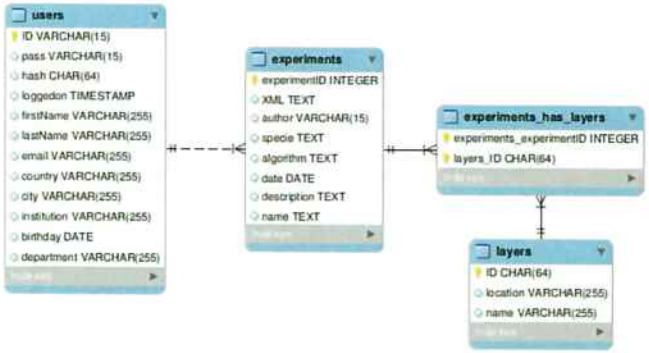

Figura 20: Diagrama entidade relacionamento do banco de dados.

Fonte: Autores.

3.7.1.1 *Users*

Tabela que guarda os usuários cadastrados no Plugin do openModeller. Abaixo seus campos e descrições.

- **id**: é o login do usuário. É a chave primária da tabela.
- **pass**: password do usuário.
- **hash**: espécie de senha de sessão do usuário. Assim que faz o login, o usuário recebe seu hash e, na chamada de qualquer método, passa esse hash como parâmetro, que é comparado com este campo do banco de dados.
- **loggedon**: *timestamp* que registra o último acesso do usuário.
- **firstname**: nome do usuário.

- **lastname**: sobrenome do usuário.
- **email**: e-mail do usuário.
- **country**: país do usuário.
- **city**: cidade do usuário.
- **institution**: instituição do usuário.
- **birthday**: aniversário do usuário.
- **department**: departamento em que o usuário trabalha.

3.7.1.2 *Layers*

Tabela que guarda os *Layers* cadastrados no Servidor. Abaixo seus campos e descrições.

- **id**: guarda o hash do *Layer*. É a chave primária da tabela.
- **location**: os *Layers* são armazenados em disco. Este campo guarda o caminho do arquivo do *Layer*.
- **name**: nome do *Layer*.

3.7.1.3 *Experiments*

Tabela que guarda os experimentos cadastrados no Servidor. Abaixo seus campos e descrições.

- **experimentId**: guarda um ID único do experimento. É chave primária da tabela.

- **xml**: guarda o XML que representa o experimento. Todas as informações abaixo estão também dentro deste XML, porém, como são alvo de buscas e de descrições de experimentos, optamos por usá-los como campos.
- **autor**: autor do experimento. É chave externa.
- **specie**: espécies analisadas no experimento. Este campo pode conter mais de uma espécie.
- **algorithm**: algoritmos utilizados no experimento. Este campo pode conter mais de um algoritmo.
- **date**: data de criação do experimento.
- **name**: nome do experimento.
- **description**: descrição do experimento.

4 METODOLOGIA

Durante a concepção decidiu-se dividir o esforço em cinco grandes grupos: revisão bibliográfica, projeto, lado servidor, lado cliente e monografia. Foi adotada essa mesma ordem para o desenvolvimento. A figura 21 ilustra as etapas do projeto.

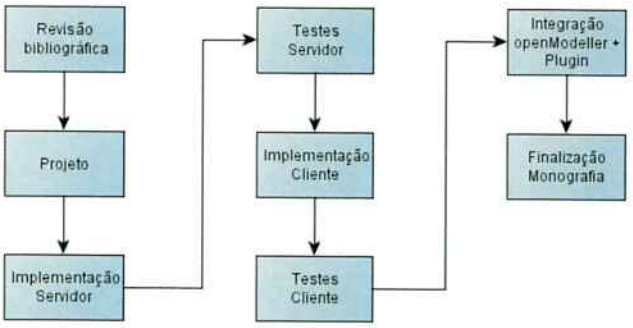

Figura 21: Fluxograma do projeto.

Fonte: Autores.

4.0.2 Revisão Bibliográfica

A revisão bibliográfica foi um processo longo durante o projeto, sendo o mais longo em carga horária. Algumas tecnologias utilizadas foram estudadas apenas quando seu uso foi necessário, embora, a maioria delas já estivesse prevista durante o decorrer do projeto.

Foram analisados artigos e teses relacionados a modelagem de distribuição de espécies, em seguida, trabalhos relacionados como (FOOK, 2009), e por fim o

openModeller.

Durante a etapa de projeto, novas tecnologias foram avaliadas e incluídas, então uma nova etapa de revisão bibliográfica foi necessária.

Ao passar à implementação tanto do lado servidor como do lado cliente do sistema, algumas tecnologias foram adicionadas e outras substituídas, e com isso um novo levantamento bibliográfico foi realizado.

Grande parte desse levantamento bibliográfico foi resumido no capítulo 2, as referências completas podem ser encontradas no final deste documento.

4.0.3 Projeto

Esta etapa se iniciou com o levantamento de requisitos do projeto, e após termos todos os requisitos partimos para a concepção do projeto que seria implementado.

Deve-se notar que não utilizamos nenhum processo definido, mas sim um processo pessoal baseado na experiência adquirida na formação de engenheiros dada pela Escola Politécnica.

O lado servidor foi idealizado baseando-se nos serviços que lhe seriam atribuídos, essencialmente prover a capacidade de compartilhar experimentos de modelos de distribuição de espécies. Para isso utilizamos o conhecimento adquirido durante a revisão bibliográfica, inicialmente selecionando quais informações eram necessárias para poder realizar um experimento desta categoria na ferramenta openModeller, depois verificando quais os meta-dados e controles necessários para a plena funcionalidade. Dessa forma iniciamos a criação das assinaturas dos métodos web, e as classes necessárias para sua execução.

Paralelamente, desenvolvemos o banco de dados que armazenaria estes modelos, utilizando uma abordagem híbrida, não armazenamos todas as informações

em bancos de dados, alguns conjuntos de dados como os *layers* foram armazenados em arquivos e apenas sua referência em banco de dados. Por outro lado, um experimento em xml é armazenado diretamente no banco de dados, isto se deve a capacidade do gerenciador de banco de dados utilizado de fazer buscas dentro de campos da tabela, que possuia o tipo XML.

De posse das assinaturas, que foram codificados na linguagem java utilizando a ferramenta Netbeans, foram gerados automaticamente os Web Services. A cada criação de um método web de um web service testavamos utilizando a interface de testes que é gerada automaticamente através da ferramenta Netbeans. Um exemplo para o web service userManager pode ser vista na figura 22.

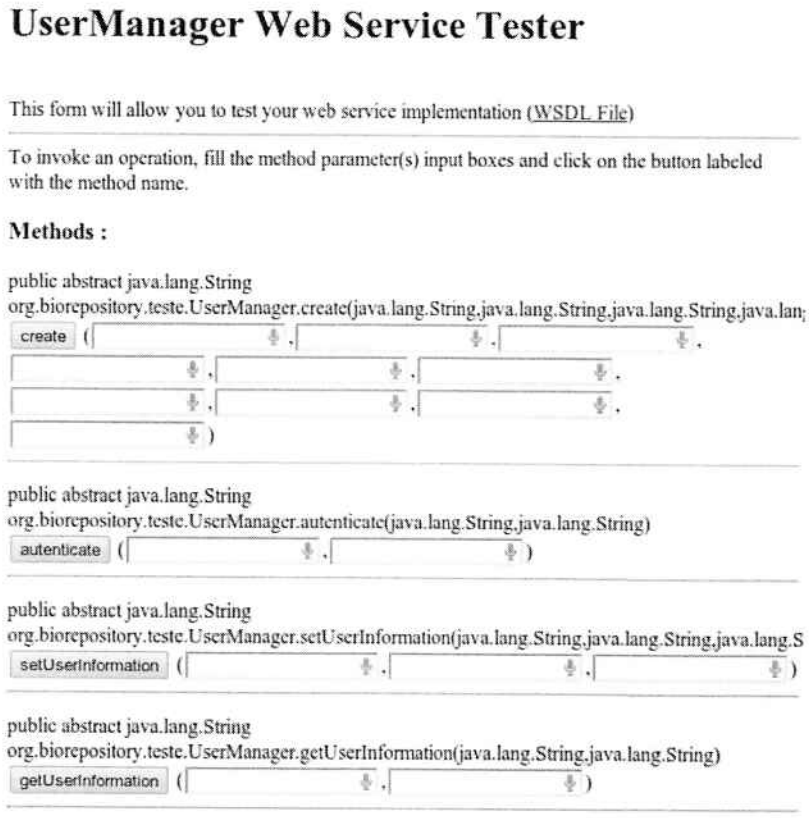

Figura 22: Interface de testes automática

Fonte: Autores.

Para o desenvolvimento do cliente projetamos inicialmente a interface de forma a contemplar todas as funcionalidades previstas nos requisitos, utilizando um design minimalista. Feito isto, utilizamos os WSDLs gerados no servidor e com auxílio da ferramenta gSoap geramos código C++ para acesso aos métodos web criando uma biblioteca chamada internamente de Client. Utilizamos essa biblioteca através das interfaces Qt para ter acesso aos Web Services.

Depois de criado o cliente partiu-se para a integração do cliente ao openModeller Desktop. Deve-se notar que ele funciona tanto como plugin dentro do openModeller Desktop como sendo um aplicativo a parte. Partiu-se então aos testes que visavam testar a integração com o openModeller Desktop e também se o conjunto estava funcionando como um todo.

Por fim a monografia, que foi a última parte do projeto. Ela foi criada agregando-se todos os artefatos produzidos nas etapas anteriores, resumindo e reescrevendo textos que foram entregues na disciplina de projeto de formatura I e II, formando, assim, este documento.

5 PROJETO E IMPLEMENTAÇÃO

5.1 Implementação

A implementação foi realizada em duas partes: na primeira focamos no servidor e nos serviços web que disponibilizaríamos para o plugin do openModeller, na segunda implementamos o plugin (cliente) utilizando os métodos web do servidor.

5.1.1 Servidor

A implementação dos Web Services que darão apoio ao plugin do openModeller foi desenvolvido em Java, com ajuda de ferramentas do Netbeans que automatizam parte do desenvolvimento de Aplicações Web que implementem Web Services. Assim, a camada SOAP é transparente, facilitando o desenvolvimento.

Abaixo serão descritos todos os *Web Methods* disponíveis no servidor de cada uma das classes.

5.1.1.1 *UserManager*

Autentica um usuário, retornando um hash gerado por SHA-1 que será a identificação do usuário na sessão. Este método é utilizado no login de usuário.

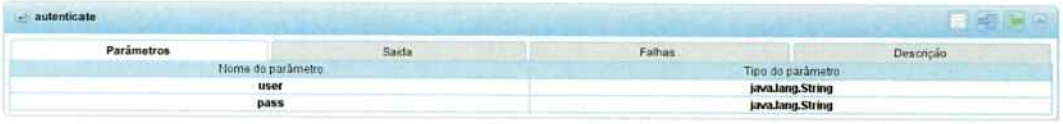

Figura 23: Método *autenticate()*

Fonte: Autores.

Parâmetros:

- user: login do usuário.
- pass: senha do usuário.

Tenta criar um usuário, retornando se foi possível ou não a criação do usuário. É utilizado na interface *signUp* do cliente.

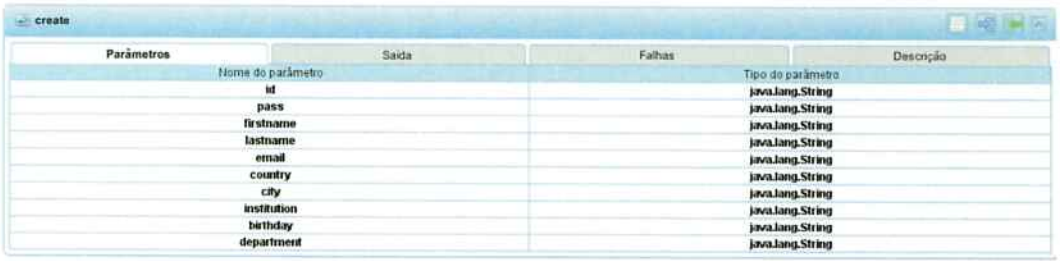

Figura 24: Método *create()*

Fonte: Autores.

Parâmetros:

- id: login do usuário.
- pass: senha do usuário.
- firsName: nome.
- lastName: sobrenome.

- email: e-mail.
- country: país.
- city: cidade.
- institution: instituto.
- birthday: data de nascimento.
- department: departamento.

Atualiza um dado do usuário. Retorna se a alteração foi bem sucedida ou não.

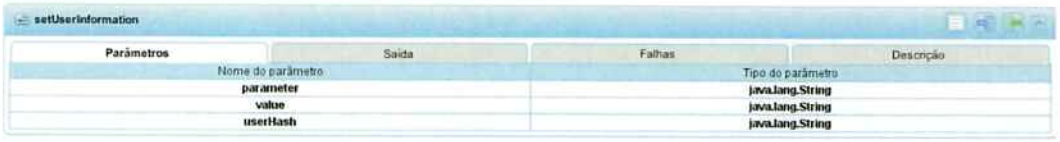

Figura 25: Método *setUserInformation()*

Fonte: Autores.

Parâmetros:

- parameter: nome do campo que se deseja alterar.
- value: novo valor do campo.
- userHash: ID de sessão do usuário.

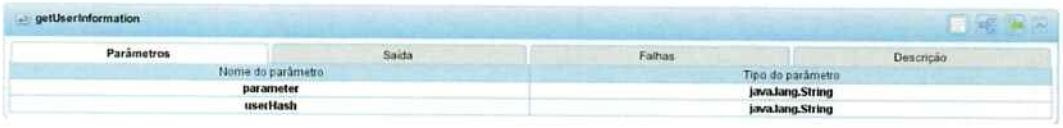

Figura 26: Método *getUserInformation()*

Fonte: Autores.

Retorna informação sobre o usuário.

Parâmetros:

- parameter: nome do campo que se deseja recuperar.
- userHash: ID de sessão do usuário.

5.1.1.2 *LayerManager*

Avalia se o layer utilizado no experimento encontra-se disponível no banco de dados. a checagem é feita através de hash SHA-1. Retorna *true* caso o Layer já exista no repositório, caso contrário retorna *false*.

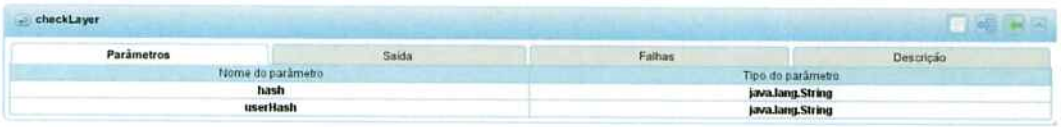

Figura 27: Método *checkLayer()*

Fonte: Autores.

Parâmetros:

- hash: hash do Layer.
- userHash: ID de sessão do usuário.

Faz o download de um Layer que não se encontra na máquina do usuário. O arquivo é transferido via conexão http, codificado em base64, muito parecido à uma transferência de arquivo no protocolo pop3/smtp.

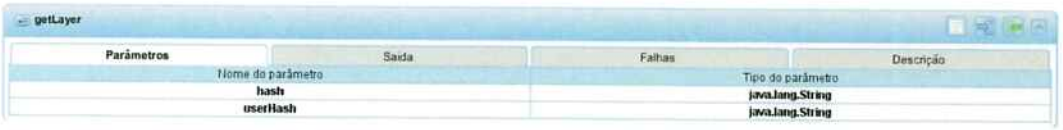

Figura 28: Método *getLayer()*

Fonte: Autores.

Parâmetros:

- hash: hash do Layer a ser baixado.
- userHash: ID de sessão do usuário.

Envia um layer para o servidor, seguindo o mesmo modo de transferência do método acima. Retorna se a operação foi bem sucedida ou não.

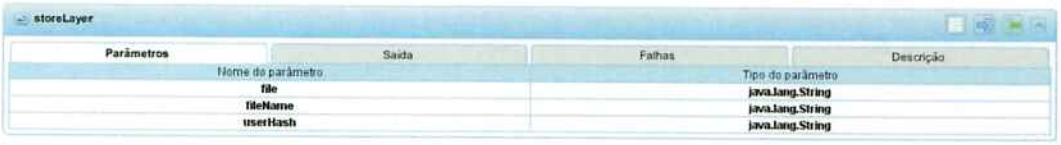

Figura 29: Método *storeLayer()*

Fonte: Autores.

Parâmetros:

- file: arquivo do Layer a ser armazenado no servidor.

- fileName: nome do Layer.
- userHash: ID de sessão do usuário.

Retorna o nome do Layer cujo hash é passado como parâmetro. É utilizado quando o cliente vai armazenar um Layer que ainda não existia em sua máquina local.

Figura 30: Método *getNameByHash()*

Fonte: Autores.

Parâmetros:

- hash: hash do Layer que se deseja saber o nome.
- userHash: ID de sessão do usuário.

Retorna uma *message digest* resultante da aplicação do algorítmo gerador de hash *SHA-256*. Não foi pedido userHash como parâmetro neste método por enterdermos que pode ser um serviço web útil a quem quiser utilizá-lo.

Parâmetros:

- file: String que se deseja saber o hash pelo algoritmo *SHA-256*.

Figura 31: Método *getSha256()*

Fonte: Autores.

5.1.1.3 *ExperimentManager*

Este método deleta da base de dados o experimento cujo ID é passado como parâmetro. É necessário que o usuário que chama o método tenha permissão para deletar o experimento. Isto é verificado pelo método privado *hasPermission()*. O método retorna o status da operação.

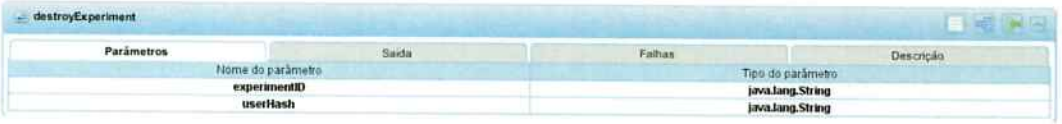

Figura 32: Método *destroyExperiment()*

Fonte: Autores.

Parâmetros:

- experimentID: ID do experimento a ser excluído.
- userHash: ID de sessão do usuário. O userHash é também utilizado, neste método, para reconhecer o usuário que está chamando o método para verificar se este possui permissão para excluir o experimento.

Recupera o experimento cujo ID é passado como parâmetro. As referências aos caminhos dos layers, por não serem os caminhos reais que serão utilizados no cliente, são substituídos por seus hashes. Assim, é necessário que no lado do

cliente haja um tratamento deste campo, devendo os hashes serem substituidos pelos respectivos paths locais. Retorna o XML necessário para se reproduzir o experimento.

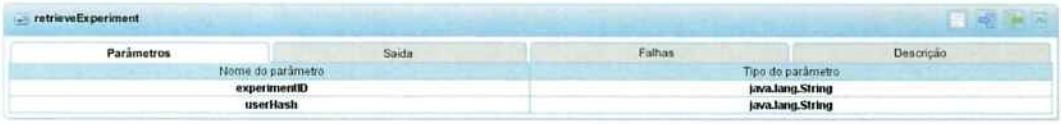

Figura 33: Método *retrieveExperiment()*

Fonte: Autores.

Parâmetros:

- experimentID: ID do experimento a ser recuperado.
- userHash: ID de sessão do usuário.

Realiza uma busca entre os experimentos armazenados no Banco de Dados com os parâmetros especificados. Se um ou mais parâmetros não forem especificados, estes não são considerados na busca. Retorna uma *String* contendo todos os IDs dos experimentos que satisfizeram a condição de busca.

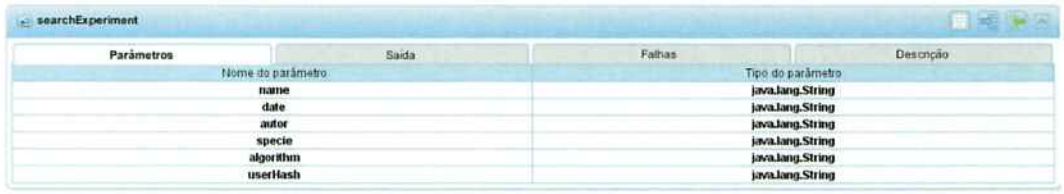

Figura 34: Método *searchExperiment()*

Fonte: Autores.

Parâmetros:

- name: contém o campo *name* de busca.
- date: contém o campo *date* de busca.
- autor: contém o campo *author* de busca.
- specie: contém o campo *specie* de busca.
- algorithm: contém o campo *algorithm* de busca.
- userHash: ID de sessão do usuário.

Retorna informações sobre o experimento cujo ID foi passado como parâmetro. Estas informações são organizadas em um documento XML cuja raíz é "experiment"e seus filhos são informações acerca do experimento. Este método é utilizado quando um usuário seleciona algum nome de experimento resultado de uma busca.

Figura 35: Método *getExperimentInformation()*

Fonte: Autores.

Parâmetros:

- experimentID: ID do experimento cujas informações devem ser retornadas.
- userHash: ID de sessão do usuário.

WebMethod que insere um novo experimento na Base de Dados do Servidor. Retorna o status da operação.

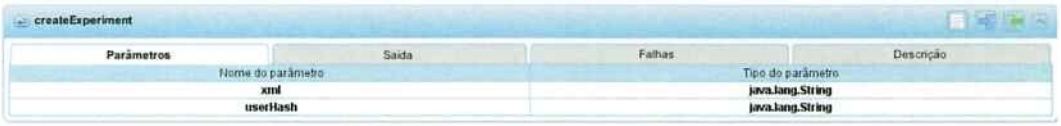

Figura 36: Método *createExperiment()*

Fonte: Autores.

Parâmetros:

- xml: *String* com o conteúdo do arquivo XML que representa um experimento.
- userHash: ID de sessão do usuário.

5.1.2 Cliente

A interface gráfica em Qt se relaciona com a lógica do programa através de sinais emitidos por alguma ação do usuário (apertar um botão, selecionar uma aba, editar um campo) e dos *slots* aos quais os sinais são conectados. Assim, os *slots* representam a reação a alguma ação do usuário.

Não são os *slots* que fazem a comunicação com os *Web Services* do servidor. Para isso, fazem uso de uma classe chamada *Client*. Esta utiliza-se do código gerado pelo *gSoap* e realiza a comunicação com o servidor, recebendo as respostas dos *Web Methods* e repassando-as aos *slots*.

A assinatura dos *slots* sugerem a que sinal eles estão relacionados, facilitando a documentação do programa. As assinaturas seguem a seguinte regra:

void on_<*tipo de widget*><*nome da aba*><*texto na interface*>_<*ação*>();

Abaixo são listadas as assinaturas dos *slots* implementados nas respectivas classes.

5.1.2.1 LoginWindow

Esta classe trata os sinais referentes à figura 14. Abaixo as assinaturas dos *slots* utilizados.

void *on_pushButtonSignIn_clicked();*

Este método é chamado assim que o usuário confirma o Login. Então, lê-se os campos de *login* e *password*, chama-se o método web *autenticate()* (figura 23) e, em caso de login inválido, um alerta é enviado ao usuário. Caso o login seja válido, o sinal *loginOk()* é emitido. Este sinal será tratado na classe *BioRepMainWindow* pelo *slot on_validLogin()*.

---

void *on_pushButtonCancel_clicked();*

Fecha a janela de login.

---

void *on_pushButtonSignUp_clicked();*

Usuários que desejem se cadastrar utilizam este botão. O *slot* abre a janela de *signUp*.

5.1.2.2 SignUp

Esta classe trata os sinais referentes à figura 15. Abaixo as assinaturas dos *slots* utilizados.

void *on_pushButtonCancel_clicked();*

Fecha a janela de *signUp*.

---

void *on_pushButtonOk_clicked();*

Confirma a criação de novo usuário. Se algum campo não estiver correto ou o nome do usuário já existir no sistema, gera-se um alerta informando o problema ao usuário. Caso tudo esteja correto, o usuário novo é criado.

5.1.2.3 BioRepMainWindow

Esta classe trata os sinais referentes às figuras 16, 17, 18, 19. Abaixo as assinaturas dos *slots* utilizados.

void *on_pushButtonMyExperimentsSynchronize_clicked();*

Executado quando há um experimento local selecionado na aba *My Experiments* e o botão *Synchronize* é clicado. Deve buscar todas as informações necessárias para se persistir um experimento no servidor, como o arquivo XML do experimento e os *Layers* faltantes no servidor (e apenas os faltantes), e enviá-los através dos serviços web já descritos.

---

void *on_pushButtonMyInformationEdit_clicked();*

Executado quanto o usuário está na aba *My Information* e deseja editar algum campo de suas informações pessoais.

---

void *on_pushButtonMyInformationSave_clicked();*

Executado quando o usuário está na aba *My Information* e deseja salvar as informações editadas.

---

void *on_pushButtonSearchSearch_clicked();*

Executado quando o usuário está na aba *Search*, preencheu os campos de busca e deseja obter os resultados. Os experimentos que satisfazem a busca são listados.

---

void *on_pushButtonSearchDownload_clicked();*

Executado quando o usuário está na aba *Search* e pressiona o botão *Download* com algum experimento da lista de resultados de busca selecionado.

O método deve buscar o XML do experimento, bem como os *Layers* referenciados que estão faltando na máquina local, criar uma nova pasta com o nome do experimento e salvar o arquivo XML nela, já com as referências aos *paths* dos *Layers* atualizadas.

---

void *on_validLogin(QString hash);*

Slot conectado ao sinal emitido assim que um login é considerado válido. Deve exibir a janela principal (*bioRepMainWindow*) para que o usuário possa manipular experimentos, editar seu perfil, fazer buscas etc.

O *hash* de sessão do usuário é passado como parâmetro. Ele deve ser armazenado em um atributo da classe e utilizado sempre que houver necessidade de se chamar o servidor.

---

void *on_itemClicked(QModelIndex modelIndex);*

Executado quando o usuário seleciona, na aba *My Experiment*, um de seus experimentos locais listados. Deve exibir informações como os layers utilizados, os algoritmos, as espécies analisadas, o autor e uma descrição (feita pelo próprio autor) do experimento.

O parâmetro *modelIndem* serve para descobrir o item da lista que foi selecionado.

---

void *on_currentTabChanged(int index);*

Executado quando o usuário seleciona uma nova aba. É utilizado para atualizar as informações pessoais do usuário quando selecionada a aba *My Information*.

O parâmetro *index* informa a aba selecionada.

void *on_itemSearchClicked(QModelIndex modelIndex);*

Executado quando o usuário seleciona, na aba *Search*, um dos experimentos resultados da busca. Deve exibir informações como os layers utilizados, os algoritmos, as espécies analisadas, o autor e uma descrição do experimento. É parecido com o emphon_itemClicked(QModelIndex modelIndex), porém neste as informações sobre o experimento são acessadas a partir do XML local, já em *on_itemSearchClicked(QModelIndex modelIndex)* as informações são acessadas via *Web Services*, que retorna um XML com as informações do experimento.

5.1.3 Comunicação Cliente-Servidor

Inicialmente utilizamos uma biblioteca chamada QtSoap, no entanto obtivemos inúmeras dificuldades em sua utilização, principalmente por utilizar uma versão antiga do protocolo SOAP. Decidimos então utilizar o gSoap para produzir o cliente.

Para a sua utilização, pegamos os arquivos *WSDL* que descrevem nossos serviços que são gerados automaticamente pela ferramenta Netbeans, assim como os arquivos *xsds* que descrevem as regras de validação necessárias para a utilização das interfaces dos serviços. Com posse dessas informações geramos código automaticamente utilizando a ferramenta gSoap.

O código gerado consiste em toda a implementação do protocolo Soap, estruturas de dados para requisição e recebimento das respostas do servidor. As assinaturas das funções geradas estão disponíveis no apêndice B.

De posse dessas funções definimos uma classe chamada Client que possui métodos para fazer a interface entre as funções produzidas pelo gSoap e pelas

chamadas da interface gráfica do cliente. Ela basicamente possui um método para cada função gerada pelo gSoap, e faz a conversão dos tipos de interface (e.g. QString para std::string) para elementos da biblioteca padrão STL, cria as estruturas preenchendo seus valores, e finalmente faz a requisição e conversão de volta para tipos de interface e retorna essas respostas.

O código de um dos métodos descritos está disponibilizado no apêndice C, na figura 47.

No total, o código produzido e gerado pelo *gSoap* para o cliente possui cerca de 19000 linhas de código.

6 TESTES E AVALIAÇÃO

6.1 Testes do Servidor e Banco de Dados

A primeira parte do projeto desenvolvida foi o Servidor. Os testes no Servidor foram realizados por uma ferramenta do NetBeans que permite chamadas dos serviços via interface web (figura 22). Assim foi feito com todos os *Web Services* que foram desenvolvidos.

Os testes e depuração do acesso ao Banco de Dados foi realizado juntamente com o teste dos *Web Services*, já que estes fazem o acesso ao banco. Terminadas as correções, passamos ao desenvolvimento do Plugin no cliente.

6.2 Testes no Cliente

Os testes no cliente foram realizados em paralelo ao seu desenvolvimento. Suas funcionalidades são independentes, então foi possível testá-las separadamente. Os testes eram quase que inteiramente depenentes dos *Web Services*, então dependíamos destes funcionando perfeitamente. Alguns erros foram detectados no Servidor, o que atrasou um pouco o desenvolvimento do cliente e seus testes, porém nada que comprometesse a fundo nosso cronograma.

6.3 Teste completo de Integração

Este teste visa explicar todo o processo de utilização do sistema. Inicialmente é necessário logar-se no sistema, no entanto, um usuário que não possua o cadastro deve faze-lo. Esse procedimento é ilustrado nas figuras 37, 38, 39 e 40.

Figura 37: Tela inicial do sistema.

Fonte: Autores.

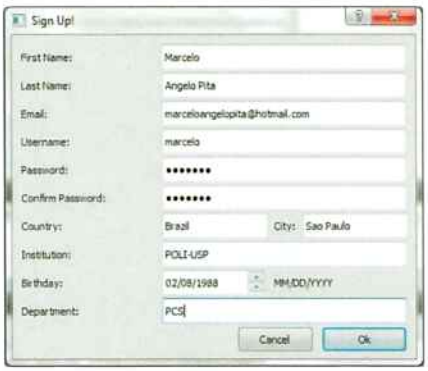

Figura 38: Tela de cadastro.

Fonte: Autores.

Figura 39: Alerta de resultado de operação de cadastro.

Fonte: Autores.

Figura 40: Exemplo de usuário acessando o sistema.

Fonte: Autores.

Uma vez no sistema, o usuário possui algumas abas de navegação: *My Experiments*, *My Information* e *Search*. Não será abordado aqui a utilização da aba *My Information*, pois é extremamente intuitivo. Na figura 41 está representado a tela *My Experiments*, nela estão mostrados todos os experimentos desenvolvidos no openModeller Desktop que o usuário possui localmente. Ao clicar em um dos experimentos as informações deste experimento é mostrado ao usuário. Ao clicar no botão *Synchronize* o experimento é enviado para o servidor.

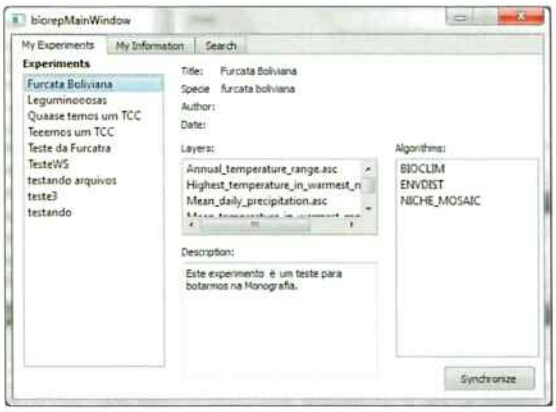

Figura 41: Aba *My Experiments* do sistema.

Fonte: Autores.

A aba *Search* esta demonstrado na figura 42, nela está sendo realizado uma busca por nome de experimento, que foi previamente cadastrado. Ao clicar no botão *Download* o experimento é baixado e fica disponível para acesso dentro do openModeller Desktop. A figura 43 mostra o experimento aberto dentro do openModeller Desktop, nela estamos setando o experimento para seu estado ini-

cial, isso deve ao fato de que nosso foco é permitir a mudança na modelagem, compartilhar o experimento e não apenas os resultados.

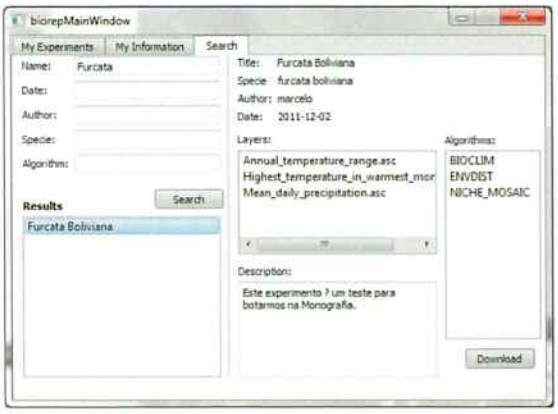

Figura 42: Aba *Search* do sistema.

Fonte: Autores.

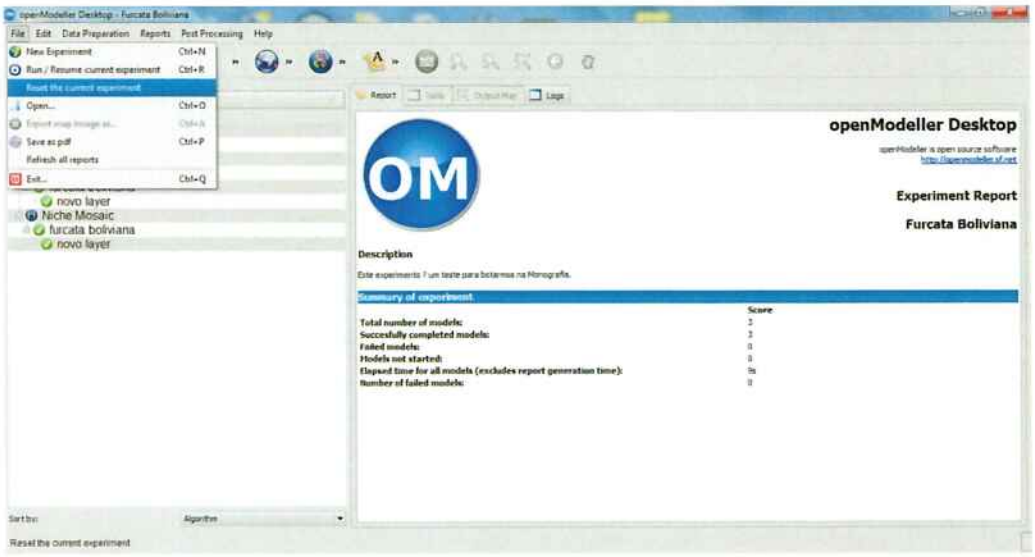

Figura 43: openModeller Desktop - Reiniciando Experimento.

Fonte: Autores.

As figuras 44, 45 e 46 mostram, respectivamente, o início da execução do experimento, o experimento sendo executado e, por fim, resultado final que é igual ao experimento inicialmente idealizado pelo seu autor.

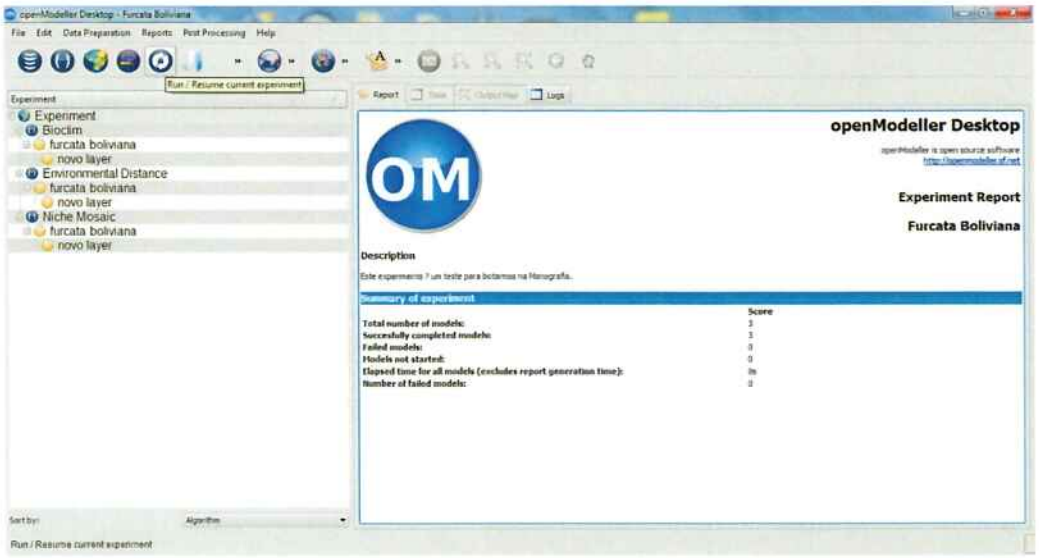

Figura 44: openModeller Desktop - Experimento inicializado.

Fonte: Autores.

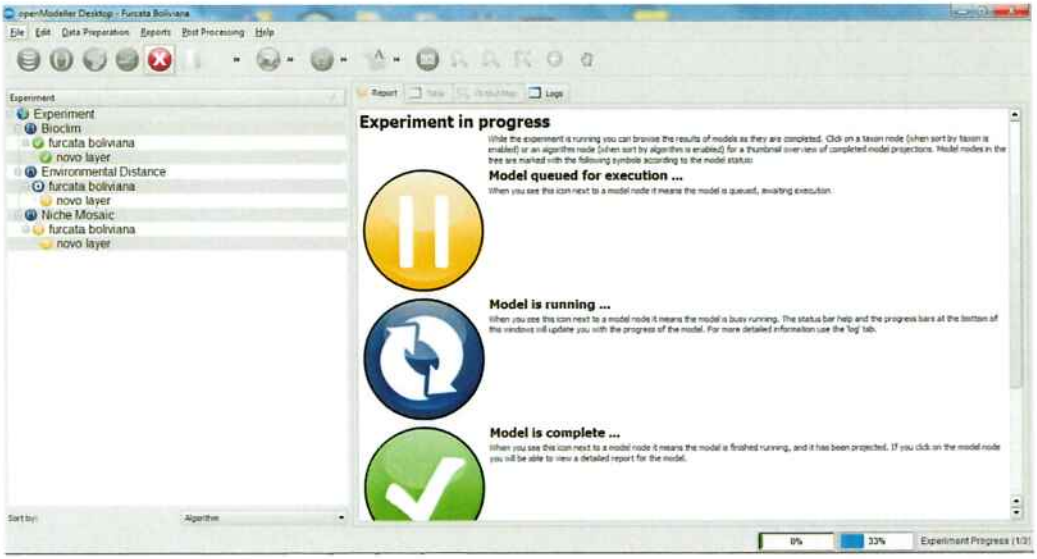

Figura 45: openModeller Desktop - Experimento em execução.

Fonte: Autores.

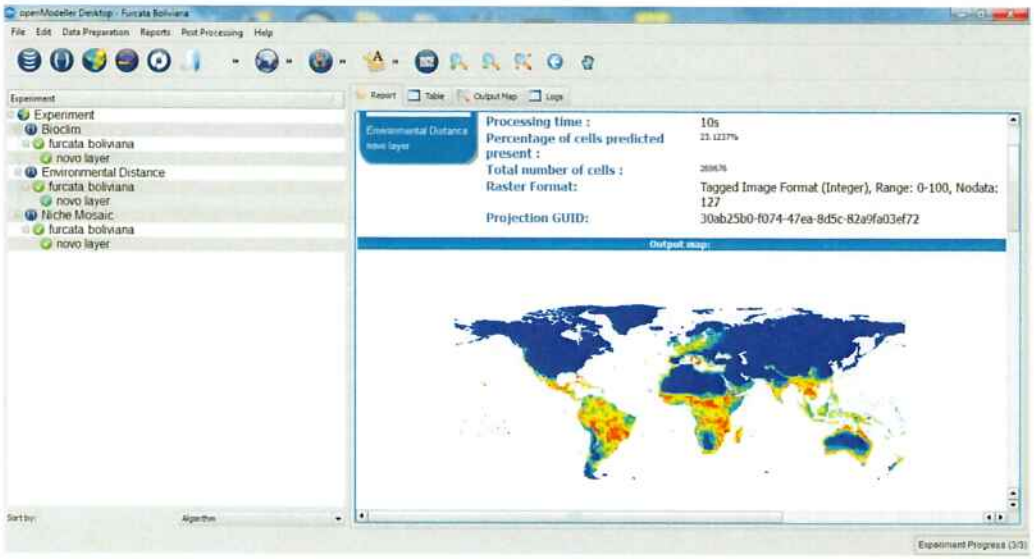

Figura 46: openModeller Desktop - Experimento completado.

Fonte: Autores.

7 CONSIDERAÇÕES FINAIS

Os objetivos iniciais do projeto, descritos nas seções 1.1 e 1.2, foram alcançados satisfatoriamente. Entregamos um projeto com o potencial de agregar em pesquisas de cientistas e institutos na busca pela preservação da biodiversidade do planeta.

Abaixo são listados os objetivos realizados, idéias para a contínua evolução dos sistemas desenvolvidos e considerações finais dos dois integrantes responsáveis pelo projeto.

7.1 Objetivos alcançados

Acreditamos que uma ferramenta como o openModeller, que faz predições sobre a distribuição de espécies, é fundamental para avanços na área de preservação. Nosso trabalho agrega funcionalidades à esta ferramenta, oferecendo compartilhamento de experimentos, evitando desperdício de tempo e a ineficiência que seria a recriação de experimentos que já foram realizados por outros pesquisadores.

Como já descrito na seção 1.3 sobre nossas motivações para o trabalho e como a figura 1 descreve, muitas espécies seguem ameaçadas de extinção e preservá-las é imprescindível para o crescimento sustentável. Com nosso projeto estamos, de alguma forma, contribuindo para isso.

Nosso trabalho também contribui para a interdisciplinaridade, tão necessária

nos dias de hoje. Engenheiros da computação estudando, entendendo e propondo soluções tecnológicas para problemas de biodiversidade e preservação de espécies. Acreditamos que, cada vez mais, as áreas de estudo devem interagir para enriquecer pesquisas e soluções de projetos.

7.2 Perspectivas e continuidade

O que fizemos foi uma versão inicial do Plugin, que pode e deve ser continuamente melhorado. A versão que fizemos não está livre de falhas. Os casos testados funcionaram como planejado, mas temos consciência de que uma série de casos não puderam ser testados e, mesmo tentando prever todas essas adversidades, alguns erros poderão não estar sendo tratados. Com a continuidade do projeto pelo meio científico, esperamos que estes casos possam ser diminuídos.

Além dos possíveis erros imprevistos, temos também consciência de algumas limitações e capacidade de expansão do projeto. Abaixo são listadas algumas idéias para a continuidade do projeto:

No cliente:

- Criar algum meio de usuários reportarem erros ocorridos durante utilização do sistema.
- Atualmente, os dados exibidos de algum experimento são gerais do experimento como um todo. Porém, um experimento é constituido de vários modelos, e podem haver diferentes *Layersets* para os diferentes modelos, bem como diferentes espécies. Assim, uma melhoria útil seria, escolhido um experimento, dar a opção ao usuário de observar os modelos separadamente. Tivemos esta idéia tardiamente e priorizamos outras tarefas, porém a funcionalidade seria interessante.

- Maior integração com o código fonte do openModeller: desenvolvemos o Plugin separadamente e da maneira mais desacoplada possível do código fonte do openModeller. O desacoplamento facilita a manutenção do Plugin e evita inserirmos erros no código do openModeller. Porém algumas vantagens seriam interessantes, como acesso às variáveis de pastas padrões de experimentos, *layersets* e outras referências que estão atualmente engessadas no Plugin.

No servidor:

- Otimização de código: a manipulação de arquivos XML não está otimizada e, dependendo do crescimento do projeto e restrições de desempenho no servidor, seria vantajoso realizar uma otimização de código.

- Melhorar o projeto do banco de dados: o banco de dados foi se moldando à medida que o projeto evoluía e entendíamos as reais necessidades de tabelas, campos e tipos. Por termos pouco tempo e apenas dois integrantes no grupo tivemos que priorizar os requisitos do sistema. O banco de dados seria mais elegante com novas tabelas e relações, porém vale ressaltar que a funcionalidade seria a mesma.

7.3 Comentários

Flávio S. Truzzi: Acredito que nosso projeto terá um grande impacto na comunidade de pesquisadores que utilizam a ferramenta openModeller Desktop, de acordo com minhas conversas com a comunidade de usuários o projeto está sendo aguardado com expectativa e deve ser incorporado no projeto principal, o que é muito gratificante.

Do ponto de vista tecnológico foi uma maratona, envolvendo diversas tecnolo-

gias que ainda não tinha tido oportunidade de utilizar. A etapa de integração do plugin com o openModeller Desktop foi interessante já que nunca havia inserido um programa completo dentro de outro, fazendo-os interoperar.

Outro fator importante foi o tema. Ele garantiu um projeto multidisciplinar ajudando a ampliar meus horizontes, além de me informar sobre um tema que não conhecia e não acompanhava.

Portanto, mesmo com os pequenos defeitos que ainda devam existir no sistema, acredito que no geral, o projeto foi um sucesso no que tange a minha capacidade como engenheiro de especificar, construir e avaliar um sistema completo de engenharia.

Marcelo Angelo Pita: O projeto serviu de grande aprendizado em minha área de atuação. Trabalhamos enfaticamente em todas as etapas de um projeto de software: levantamento de requisitos, estudo de conceitos e tecnologias, modelagem, implementação, depuração, testes e documentação. Em particular, o estudo de diversas tecnologias e protocolos enriqueceram meus conhecimentos na área.

O trabalho em grupo foi também muito importante. A divisão de tarefa apropriada e a superação de obstáculos dão um ar ainda mais especial em finalizar este projeto.

REFERÊNCIAS

BOOTH, D. et al. (Ed.). *Web Services Architecture* . [S.l.], 2004. Disponível em: <http://www.w3.org/TR/ws-arch/>.

CHUINE, I. Why does phenology drive species distribution? *Philosophical Transactions of the Royal Society B: Biological Sciences*, v. 365, n. 1555, p. 3149–3160, 2010. Disponível em: <http://rstb.royalsocietypublishing.org-/content/365/1555/3149.abstract>.

CORRÊA, P. L. P.; RODRIGUES, F. A.; RODRIGUES, E. S. C. Computational systems for management and modeling of species geographic distribution. *5º Congresso Brasileiro de Mastozoologia*, 2010.

ENDREI, M. et al. *Patterns: Service-Oriented Architecture and Web Services*. IBM Redbooks, 2004. Disponível em: <http://www.redbooks.ibm.com-/redbooks/pdfs/sg246303.pdf>.

FOOK, K. D. *WBCMS - a service oriented Web architecture for enhancing collaboration in biodiversity: the case of species distribution modelling community*. 92 p. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2009. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m18@80%-/2009/03.13.21.32>.

FOOK, K. D. et al. Geoweb services for sharing modelling results in biodiversity networks. *Transactions in GIS*, p. 379–399, 2009.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995. ISBN 978-0-201-63361-0.

GONÇALVEZ, E. *Desenvolvendo Aplicações Web com JSP, Servlets, JavaServer Faces, Hibernate, EJB 3 Persistence e Ajax*. [S.l.]: Ciência Moderna, 2007. ISBN 9788573935721.

GROUP, T. P. G. D. *PostgreSQL 9.0 Reference Manual - Volume 3 Server Administration Guide*. Network Theory Ltd, 2010. Disponível em: <http://www.network-theory.co.uk/postgresql9/vol3/>.

GSOAP Tutorial. [S.l.], 2011. Disponível em: <http://www.cs.fsu.edu/~engelen-/soap.html>.

HAROLD, E. R. *Processing Xml with Java*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0201771861.

IUCN. *IUCN Red List of Threatened Species*. [S.l.], 2011. Disponível em: <http://www.iucnredlist.org/>.

MUÑOZ, M. de S. et al. openModeller: a generic approach to species' potential distribution modelling. *GeoInformatica*, Springer Netherlands, v. 15, n. 1, p. 111–135, jan. 2011. ISSN 1384-6175. Disponível em: <http://dx.doi.org/10-.1007/s10707-009-0090-7>.

ONLINE Reference Documentation. [S.l.], 2011. Disponível em: <http://doc.qt-.nokia.com/>.

SIQUEIRA, M. F. *Uso de modelagem de nicho fundamental na avaliação do padrão de distribuição geográfica de espécies vegetais*. Tese (Doutorado) — Ciências da Engenharia Ambiental, Escola de Engenharia de São Carlos, Universidade de São Paulo, Brasil, 2005.

VRENIOS, A. *Linux Cluster Architecture*. Indianapolis, IN, USA: Sams, 2002. ISBN 0672323680.

APÊNDICE A – DESCRIÇÃO DOS CASOS DE USO

Este apêndice descreve os casos de uso de forma completa, ou seja, eventos iniciadores, atores, pré-condição, e sua sequência de eventos.

Descrição	Descreve o processo de cadastro de usuários no sistema.
Evento iniciador:	Solicitação de cadastro do usuário.
Atores	Usuário.
Pré-condição	Nenhuma.
Atores	Usuário.
Sequência de eventos:	
1	Usuário seleciona, através do clique a opção de cadastro de usuário.
2	Sistema exibe os campos necessários para cadastro de usuário.
3	Usuário insere os dados do usuário e confirma cadastro.
4	Sistema cadastra o usuário.
Pós-condição	Usuário foi cadastrado no sistema.
Exceções:	
5	Usuário a ser cadastrado já existe: sistema apresenta mensagem de erro e retorna ao passo 2 (passo 4).
6	Dados fornecidos para o usuário não consistentes: sistema apresenta mensagem de erro e retorna ao passo 2 (passo 4).
Inclusões:	Não se aplica.

Tabela 8: 1.1 Cadastro de usuário

Descrição	Descreve a remoção de um usuário existente.
Evento iniciador:	Solicitação de remoção de usuário.
Atores	Usuário.
Pré-condição	Usuário autenticado.
Atores	Usuário.
Sequência de eventos:	
1	Usuário solicita operação de remoção de usuário.
2	Sistema solicita a identificação do usuário a ser removido.
3	Usuário se identifica.
4	Sistema remove o usuário e exibe o resultado.
Pós-condição	Usuário e seus experimentos privativos são removidos.
Exceções:	
5	Falha de identificação: sistema bloqueia a exclusão por 24h.(passo 4).
Inclusões:	Caso de uso de recuperação de usuário.

Tabela 9: 1.2 Remoção de usuário

Descrição	Descreve a alteração dos dados de um usuário existente.
Evento iniciador:	Solicitação de alteração de usuário.
Atores	Usuário.
Pré-condição	Usuário autenticado.
Atores	Usuário.
Sequência de eventos:	
1	Usuário solicita operação de alteração de usuário.
2	Sistema disponibiliza campos que podem ser alterados.
3	Usuário altera os campos que deseja e confirma.
4	Sistema altera os dados do usuário.
Pós-condição	Dados do usuário são alterados.
Exceções:	
5	Campos Inválidos: Exibe erro e retorna ao passo 2. (passo 4).
Inclusões:	Caso de uso de recuperação de usuário.

Tabela 10: 1.3 Alteração de um usuário

Descrição	Descreve a consulta aos dados de um usuário existente.
Evento iniciador:	Solicitação de consulta de usuário.
Atores	Usuário.
Pré-condição	Usuário autenticado.
Atores	Usuário.
Sequência de eventos:	
1	Usuário solicita operação de consulta de usuário.
2	Sistema apresenta os dados do usuário.
Pós-condição	Dados do usuário são exibidos.
Exceções	Não há.
Inclusões:	Caso de uso de recuperação de usuário.

Tabela 11: 1.4 Recuperação de usuário

Descrição	Descreve o processo de cadastro de novo experimento no sistema.
Evento iniciador:	Solicitação de cadastro de novo experimento no sistema.
Atores	Usuário.
Pré-condição	Usuário autenticado.
Atores	Usuário.
Sequência de eventos:	
1	Usuário solicita o cadastro de experimento no sistema.
2	Sistema lista os experimentos locais.
3	Usuário seleciona um experimento e solicita seu cadastro.
4	Sistema cadastra experimento.
Pós-condição	Experimento está cadastrado no sitema.
Exceções	
5	Falha no envio de arquivo: sistema informa a falha ao usuário e cancela a transação (Passo 4).
Inclusões	Não se aplica

Tabela 12: 2.1 Cadastro de Experimento

Descrição	Descreve o processo de obtenção de experimentos já cadastrados no sistema.
Evento iniciador:	Solicitação de recuperação de experimento do sistema.
Atores	Usuário.
Pré-condição	Usuário autenticado.
Atores	Usuário.
Sequência de eventos:	
1	Usuário solicita recuperação de experimento no sistema.
2	Sistema lista os experimentos cadastrados.
3	Usuário seleciona experimento que deseja recuperar.
4	Sistema exibe as informações do experimento.
5	Usuário confirma recuperação de experimento.
6	Sistema envia experimento para usuário.
Pós-condição	Experimento é recuperado pelo usuário.
Exceções	
7	Falha no envio de arquivo: sistema informa a falha ao usuário e cancela a transação (Passo 6).
Inclusões	Não se aplica

Tabela 13: 2.2 Recuperação de Experimento

Descrição	Descreve o processo de consulta do acervo de experimentos.
Evento iniciador:	Solicitação de consulta de experimento do sistema.
Atores	Usuário.
Pré-condição	Usuário autenticado.
Atores	Usuário.
Sequência de eventos:	
1	Usuário solicita consulta de experimentos no sistema.
2	Sistema apresenta os campos a serem preenchidos para a busca.
3	Usuário preenche um ou mais campos para a busca.
4	Usuário confirma busca.
5	Sistema apresenta os resultados.
Pós-condição	Usuário realizou a busca de experimentos.
Exceções	
6	Falha no envio de dados: sistema informa a falha ao usuário e cancela a transação (Passo 5).
Inclusões	Pode-se seguir a este caso a Recuperação de experimento.

Tabela 14: 2.3 Consulta Experimento

Descrição	Descreve o processo de login no sistema.
Evento iniciador:	Abertura do plugin do openModeller.
Atores	Usuário.
Pré-condição	Não há
Atores	Usuário.
Sequência de eventos:	
1	Usuário solicita abertura do plugin do openModeller.
2	Sistema apresenta os campos a serem preenchidos para o login.
3	Usuário preenche seu login e sua senha.
4	Usuário confirma login.
5	Sistema valida o login e abre a interface principal.
Pós-condição	Usuário está autenticado no sistema.
Exceções	
6	Falha na autenticação: usuário ou senha inválidos (Passo 5). Sistema informa a falha na autenticação.
7	Falha de conexão: não há conexão com a Internet (Passo 5). Sistema informa a falha na conexão.
8	Falha de comunicação: não há comunicação com o servidor (Passo 5). Sistema informa a falha na comunicação.
Inclusões	Não se aplica.

Tabela 15: 3 Login

Descrição	Descreve o processo de logoff no sistema.
Evento iniciador:	Fechamento do plugin no openModeller Desktop
Atores	Usuário.
Pré-condição	Não há
Atores	Usuário.
Sequência de eventos:	
1	Usuário fecha plugin no openModeller Desktop
Pós-condição	Usuário não autenticado no sistema.
Exceções	Não há
Inclusões	Não se aplica.

Tabela 16: 4 Logoff

APÊNDICE B – ASSINATURAS GERADAS PELO GSOAP

Abaixo as assinaturas geradas pelo *gSoap*.

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__autenticate(struct soap *soap, const char *soap_endpoint, const char *soap_action, ns1__autenticate *ns1__autenticate_, ns1__autenticateResponse *ns1__autenticateResponse_);

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__setUserInformation(struct soap *soap, const char *soap_endpoint, const char *soap_action, ns1__setUserInformation *ns1__setUserInformation_, ns1__setUserInformationResponse *ns1__setUserInformationResponse_);

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__getUserInformation(struct soap *soap, const char *soap_endpoint, const char *soap_action, ns1__getUserInformation *ns1__getUserInformation_, ns1__getUserInformationResponse *ns1__getUserInformationResponse_);

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__getUserInformation(struct soap *soap, const char *soap_endpoint, const

char *soap_action, ns1__getUserInformation *ns1__getUserInformation_, ns1__getUserInformationResponse *ns1__getUserInformationResponse_);

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__destroyExperiment(struct soap *soap, const char *soap_endpoint, const char *soap_action, ns1__destroyExperiment *ns1__destroyExperiment_, ns1__destroyExperimentResponse *ns1__destroyExperimentResponse_);

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__getExperimentHashes(struct soap *soap, const char *soap_endpoint, const char *soap_action, ns1__getExperimentHashes *ns1__getExperimentHashes_, ns1__getExperimentHashesResponse *ns1__getExperimentHashesResponse_);

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__retrieveExperiment(struct soap *soap, const char *soap_endpoint, const char *soap_action, ns1__retrieveExperiment *ns1__retrieveExperiment_, ns1__retrieveExperimentResponse *ns1__retrieveExperimentResponse_);

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__searchExperiment(struct soap *soap, const char *soap_endpoint, const char *soap_action, ns1__searchExperiment *ns1__searchExperiment_, ns1__searchExperimentResponse *ns1__searchExperimentResponse_);

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__getExperimentInformation(str soap *soap, const char *soap_endpoint, const char *soap_action, ns1__getExperimentInformation *ns1__getExperimentInformation_, ns1__getExperimentInformationResponse *ns1__getExperimentInformationResponse_

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__createExperiment(struct soap *soap, const char *soap_endpoint, const char *soap_action, ns1__createExperiment *ns1__createExperiment_, ns1__createExperimentResponse *ns1__createExperimentResponse_);

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__checkLayer(struct soap *soap, const char *soap_endpoint, const char *soap_action, ns1__checkLayer *ns1__checkLayer_, ns1__checkLayerResponse *ns1__checkLayerResponse_);

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__storeLayer(struct soap *soap, const char *soap_endpoint, const char *soap_action, ns1__storeLayer *ns1__storeLayer_, ns1__storeLayerResponse *ns1__storeLayerResponse_);

•SOAP_FMAC5 int SOAP_FMAC6 soap_call___ns1__getLayer(struct soap *soap, const char *soap_endpoint, const char *soap_action, ns1__getLayer *ns1__getLayer_, ns1__getLayerResponse *ns1__getLayerResponse_);

APÊNDICE C – MÉTODO DA CLASSE CLIENT

Método exemplo da classe *Client* que utiliza as assinaturas geradas pelo *gSoap*, disponíveis no apêndice *assinaturasSoap*.

```
QString Client::Login(QString user, QString pass) {

    auth = new ns1__autenticate;
    authResponse = new ns1__autenticateResponse;

    string *stduser = new string(user.toStdString());
    string *stdpass = new string(pass.toStdString());

    auth->user = stduser;
    auth->pass = stdpass;

    QString resp("");

    if (soap_call___ns1__autenticate(soap, url, NULL, auth, authResponse) == SOAP_OK) {
        resp.append((*authResponse->return_).data());
    } else {
        resp.append("Server Error");
    }


    delete stduser;
    delete stdpass;

    delete auth;
    delete authResponse;

    return resp;
}
```

Figura 47: Código de método da classe Client

Fonte: Autores.