Instituto de Matemática e Estatística - USP

Trabalho de Formatura Supervisionado

Projeto e implementação de serviços analíticos para o DataUSP-PósGrad

Autor:
Bruno Padilha

Orientador:
João Eduardo Ferreira

30 de janeiro de 2014

Resumo

Padilha, B. Projeto e Implementação de Serviços Analíticos para o DataUSP-PósGrad.

Monografia - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2013.

O sistema DataUSP-PósGrad é composto de um conjunto de serviços analíticos, construídos sobre a arquitetura *RESTful web services*, que tem como principal objetivo auxiliar a tomada de decisões da Pró-reitoria de Pós-Graduação da Universidade de São Paulo. Esses serviços analíticos são implementados: por meio de relatórios dinâmicos e estáticos necessários para o acompanhamento gerencial da pós graduação; pelas análises da produção intelectual dos docentes; como também pela análise da coleta de dados enviados à Capes. A fonte de dados para a alimentação desses serviços analíticos é o data warehouse da Pós-graduação da USP.

Sumário

Capítulo 1

Introdução e Contexto

A capacidade de armazenar dados em meios eletrônicos cresce exponencialmente desde os primórdios da computação sustentada pelos avanços em pesquisa e tecnologia.

Os programas de computador, valendo-se desses avanços, também evoluíram e passaram a gerar e manipular um volume cada vez maior de dados que, para serem processados e analisados de modo rápido, eficiente e consistente, passaram a ser armazenados em sistemas especializados denominados Bancos de Dados.

Os Sistemas de Bancos de Dados viabilizaram a manutenção do histórico dos dados, que além de armazenar os dados e seus respectivos processos de captação, permitem realizar análises estatísticas poderosas como, por exemplo, a detecção de anomalias, agrupamento e predição.

Em geral quanto mais dados se obtém do ciclo de vida de um determinado processo mais precisas serão as análises realizadas sobre os mesmos, porém com um tempo de processamento maior. Esse tempo consumido para a realização das análises de dados muitas vezes é crítico para se tomar uma decisão. O tempo de processamento está diretamente relacionado a estrutura de armazenamento e de relacionamentos dos dados dos sistemas computacionais.

Um sistema computacional cuja principal finalidade é a analise de dados é denominado sistema analítico, enquanto que um sistema que insere, atualiza, remove, ou seja, modifica constantemente e pontualmente os dados é denominado sistema transacional. Os sistemas analíticos têm como principal característica a consulta a uma grande quantidade de dados com a finalidade de sintetizar ou descobrir informações.

Se um sistema analítico concorre com um sistema transacional ao acessar os dados, o banco pode sobrecarregar e comprometer a eficiência de ambos os sistemas. Os efeitos colaterais dessa concorrência podem por um lado pode causar a indisponibilidade do sistema transacional e por outro a causar uma lentidão indesejada do sistema analítico. A replicação dos dados em vários níveis é a solução para evitar o problema da concorrência entre o ambiente analítico e o transacional. Essa replicação com alternativas para novas estruturas, tratamento, integração e otimização de acesso aos dados é denominada Data Warehouse.

Para atender a uma importante demanda de análise de dados da Universidade de São Paulo foi criado o projeto denominado DataUSP-PosGrad. Esse projeto propôs a criação de um Data Warehouse com dados de todos os programas da Pós-Graduação e um conjunto de serviços analíticos com o propósito de gerar relatórios sob demanda com análises de tais dados. Esses relatórios incluem análises estatísticas dos programas, áreas e pessoas ligadas à Pós-Graduação, como por exemplo, o número de titulações em uma área, o tempo médio de titulação em um programa, o número de teses dos docentes em uma área ou programa além de muitas outras, publicações e citações.

O principal objetivo deste Trabalho de Conclusão de Curso é o de descrever os principais desafios encontrados e as soluções encontradas para o desenvolvimento de serviços analíticos do Projeto DataUSP-PosGrad.

1.1 Problema e Hipótese de solução

O DataUSP-PósGrad precisa ser um sistema escalável, no qual novas funcionalidades podem ser agregadas de maneira simples, rápida e sem comprometer o seu desempenho. Sua arquitetura deve ser leve e robusta, capaz de manipular um grande volume de dados eficientemente e devolver os resultados ao usuário no menor intervalo de tempo possível, além de ser capaz de fornecer dados a outros sistemas computacionais autonomamente utilizando uma interface web. Para atender a esses requisitos, uma possível solução é construir um conjunto de Web Services sobre a arquitetura RESTful.

1.2 Evidências da solução RESTful

A arquitetura REST separa a implementação de um sistema em cliente e servidor. Essa separação aumenta a escalabilidade do servidor, já que a implementação dos serviços é transparente ao usuário e não mantém estado entre requisições, o que também contribui para um melhor desempenho do sistema como um todo e facilita a incorporação de novos recursos. A interface de acesso aos serviços é homogênea e os recursos disponibilizados são acessados por identificadores únicos (endereços de rede). A arquitetura REST também utiliza o HTTP como protocolo de acesso e não apenas para transporte de dados. Essa interface de acesso utilizará um subconjunto dos métodos do HTTP (GET, POST, PUT, DELETE, etc) para disponibilizar os recursos, simplificando a interação cliente-servidor.

Um Serviço Web, de forma resumida, é uma aplicação que disponibiliza seus recursos por meio de um endereço de rede. Assim qualquer usuário do serviço, seja ele humano ou um programa de computador que conheça o protocolo de comunicação, pode interagir com o mesmo, consumindo seus recursos por meio de troca de mensagens de texto estruturadas. Um Web Service que implementa os conceitos da arquitetura REST sobre o protocolo HTTP é chamado RESTful Web Service.

1.3 A arquitetura REST no DataUSP-PósGrad

A construção do DataUSP-PósGrad sobre a arquitetura REST permite que novas funcionalidades (em geral a geração de novos relatórios) sejam implementadas de modo simples e pragmático. O uso do HTTP como protocolo de acesso garante a uniformidade da interface de acesso aos seus recursos, e a aplicação cliente pode ser desenvolvida em diferentes plataformas e sistemas operacionais. Como exemplo de implementação dessa aplicação cliente, pode-se destacar: o uso de dispositivos móveis ou navegadores de internet garantindo ao sistema uma maior portabilidade a abrangência de uso. Além disso, por haver uma divisão específica entre cliente e servidor parte do processamento pode ser feito do lado cliente, como por exemplo manipular os dados para facilitar a visualização por meio de gráficos ou tabelas, aliviando a carga do servidor e melhorando o desempenho assim como a fluidez geral do sistema.

1.4 Descrição dos próximos capítulos

No capítulo 2 serão apresentados os fundamentos e conceitos tecnológicos utilizados, tanto na elaboração quanto na construção do sistema DataUSP-PósGrad. O capítulo 3 descreve as atividades realizadas e as metodologias utilizadas no decorrer do projeto. Já no capítulo 4 são apresentados os resultados obtidos, com o lançamento oficial do sistema, e os testes de desempenho. No capítulo 5 constam as conclusões e há uma descrição subjetiva do trabalho no apêncice A.

Capítulo 2

Conceitos e tecnologias estudadas

Esse capítulo apresenta um panorama geral das tecnologias e conceitos gerais utilizados na elaboração e concepção do DataUSP-PósGrad. Serão abordados: os conceitos de Data Warehouse e algumas ferramentas relacionadas como o Pentaho e o Saiku; Serviços Web e as arquiteturas REST e SOAP; protocolos de comunicação como o HTTP; as linguagens de programação utilizadas no projeto (Java, Python e JavaScript); a técnica de Web Crawling para a obtenção dados de fontes externas aos bancos de dados da USP.

2.1 Data Warehouse

Data Warehouse [1] é um grande repositório de dados, atuais e históricos, com a finalidade de fazer análises e gerar relatórios. Oriundos de um ou mais bancos de dados, os dados unificados, validados e normalizados, geralmente por meio de processos automatizados (ETL 2.11), que também tem a função de manter o Data Warehouse atualizado.

2.1.1 Modelo transacional vs multidimensional

No mundo de banco de dados podemos dividir os sistemas computacionais em duas grandes áreas: os sistemas transacionais, e os sistemas analíticos.

Os sistemas transacionais (OLTP) alteram o banco de dados constantemente, modificando e inserindo novas informações e realizando consultas pontuais. Já os sistemas analíticos (OLAP) agregam e analisam um grande

volume de dados por meio de consultas complexas, ou seja, custosas computacionalmente, com o objetivo de detectar tendências, anomalias em um contexto semântico dos dados e, também, realizar análises estatísticas.

Para evitar o problema de concorrência entre um sistema transacional e o sistema analítico associado a nível de banco de dados, os dados são replicados de um ou mais bancos dos sistemas transacionais e transformados em um Data Warehouse, que é onde o sistema analítico irá operar.

Essa replicação permite ainda que os dados sejam armazenados em estruturas apropriadas ao tipo de consulta incidente. No Data Warehouse, por exemplo, os dados podem ser organizados em um estrutura denominada *estrela*, onde há uma tabela central denominada *fato* que armazena as referências das outras tabelas denominadas *dimensões*. Esse modelo de dados é pode ser armazenado em um banco de dados multidimensional [4], que é otimizado para realizar as consultas típicas de um sistema analítico.

Um Data Warehouse pode armazenar seus dados em um modelo normalizado(NDS) ou multidimensional. Os dados do modelo multidimensional são denominados *cubo MOLAP*.

2.1.2 Arquitetura de dados no DataUSP-PósGrad

A arquitetura de dados do DataUSP-PósGrad pode ser subdividida em níveis (ou camadas), conforme a figura 2.1. São eles:

- No nível 0 estão as bases transacionais dos diversos sistemas administrativos, como por exemplo o Janus, o Júpiter e etc. O nível 1 é a replicação do nível 0, porém aqui os dados são normalizados, validados e possivelmente transformados para permitir uma construção consistente do modelo multidimensional, recebe o nome de NDS (Normalized Data Store).
- O nível 2 é modelo multidimensional, porém em uma base relacional (conhecido pelo nome de base de dados ROLAP[1]). Aqui os dados do nível 1 são transformados por meio de ETL, que nada mais são do que scripts para fazer a carga no modelo multidimensional, desnormalizando os dados conforme necessário. [1]

[1]ROLAP significa Relational OLAP, ou seja é uma representação relacional do modelo multidimensional

- No nível 3 está a base multidimensional MOLAP. Aqui os dados estão armazenados em um *cubo OLAP*, no qual as consultas são altamente eficientes pela estrutura do mesmo.
- O nível 4 é a camada de visualização do DataUSP-PósGrad, onde são gerados os relatórios sob demanda (adhoc) e também onde são compilados os cubos offline para uso com o MS Excel.

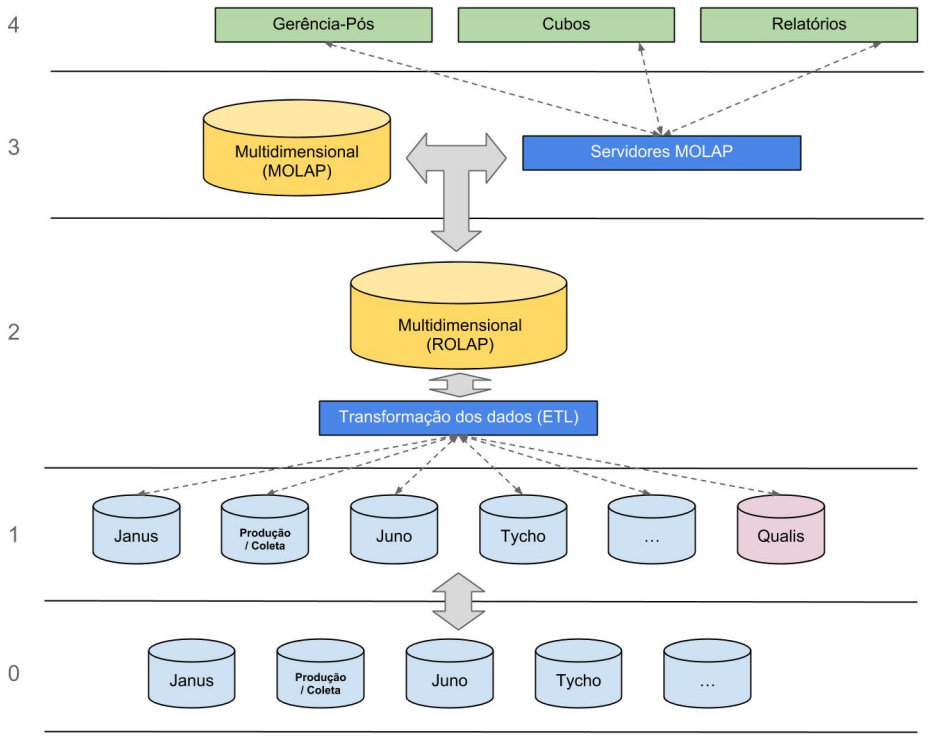


Figura 2.1: O mundo dos dados no DataUSP-PósGrad

2.2 Java

Java[2] é uma linguagem de programação orientada a objetos [12] muito popular, usada e aceita comercialmente. Originalmente desenvolvida pela *Sun Microsystems*, em meados de 1995, hoje é mantida pela *Oracle*[3].

[2]Java - http://docs.oracle.com/javase/tutorial/java/javaOO/index.html

[3]História da tecnologia Java - http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html

Mais do que uma linguagem, Java é uma plataforma de desenvolvimento composta de bibliotecas pré-compiladas (APIs) e também pela máquina virtual java (JVM). Um programa de computador escrito em Java é compilado em um código específico para ser interpretado pela JVM e, sendo assim, qualquer dispositivo capaz executá-la, independentemente da plataforma utilizada (por exemplo o microsoft windows, linux, mac, android, etc ..), pode executar código Java.

As unidades fundamentais de um programa Java são os Objetos, que são porções de código auto contidos com características similares e com um objetivo em comum. Podem ser usados para descrever objetos abstratos, como um objeto matemático, ou um objeto do mundo real, como por exemplo um carro com seus atributos (cor, modelo, potencia, etc) e também suas funcionalidades (ligar, andar, abrir portas, etc).

Os Objetos em Java são instâncias de uma Classe, um protótipo do Objeto contendo sua descrição por meio de Atributos e Métodos. Os Atributos são os parâmetros e estruturas de dados da Classe, enquanto que os Métodos são as funções da classe, utilizadas para realizar as tarefas pertinentes ao respectivo objeto instanciado.

Valendo-se do conceito de encapsulamento [12], existem Métodos especiais para manipular os atributos de uma Classe Java: os Métodos *getters*, utilizados para obter o valor de um atributo; os Métodos *setters*, para atribuir um valor ao atributo.

Toda Classe contém um Método especial chamado de Construtor Primário. Podendo receber parâmetros externos, instancia um Objeto a partir dessa Classe fazendo as inicializações necessárias.

Existe ainda um conceito de Classe em Java chamada de Bean. Essa Classe contém apenas Atributos e Métodos *getters* e Métodos *setters*. É usada geralmente para agrupar um conjunto de objetos, como por exemplo os dados cadastrais de um cliente com nome, cpf, data de nascimento e etc. Seu construtor primário não recebe argumentos (em geral não executa ação alguma além de instanciar o Objeto), e assim sua inicialização fica a cargo de quem a instancia.

Uma outra funcionalidade de Java, que é muito usada no projeto DataUSP-PósGrad, é o conceito de Anotações. As Anotações são meta dados adicionados às Classes, Atributos e Métodos com a finalidade de indicar funcionalidades extras. Não alteram o comportamento do código e servem apenas como referência. São usadas no DataUSP-PósGrad para mapear os serviços

e seus recursos em seus respectivos endereços de rede.

2.3 Jersey

Os RESTful Web Services em java foram apresentados na JSR-311. Uma JSR[4] (Java Specification Request) é uma solicitação formal para se incluir novas especificações na plataforma Java.

Como produto final da JSR-311 nasceu a JAX-RS, uma api[5] baseada em anotações para implementar RESTful Web Services em Java. Porém a JAX-RS é apenas uma especificação e sua implementação de referência fica a cargo do framework Jersey.[6]

O Jersey é responsável por intermediar uma requisição HTTP e instanciar a respectiva classe Java que representa o recurso solicitado, por meio do uso de anotações. 2.2

2.4 HTTP

O HTTP[7] (Hyper Text Transfer Protocol) é um protocolo de comunicação em redes redes de computadores, fundamentado no modelo requisição resposta, utilizado na transferência de hipermídia[8] entre aplicações. É o principal protocolo de comunicação da internet. Seu uso mais comum é a interação de um navegador de internet(cliente) com um servidor web.

O HTTP implementa oito métodos de requisição que definem a semântica da troca de mensagens, ou seja, informam ao servidor a operação a ser realizada ao receber uma requisição do cliente. São eles: GET; POST; PUT; DELETE; HEAD; CONNECT; TRACE; OPTIONS.

Os servidores que implementam o HTTP devem obrigatoriamente implementar dois de seus métodos: HEAD e GET. Além desses, geralmente, são

[4] JSR - `http://jcp.org/aboutJava/communityprocess/final/jsr311/`

[5] Application Programming Interface - é uma especificação de como um conjunto de recursos deve ser utilizado, incluindo protocolos, modelo de dados regras de negócio e etc

[6] Jersey - `https://jersey.java.net/`

[7] HTTP 1.1 - `http://tools.ietf.org/html/rfc2616`

[8] Hipermídia é uma extensão do hipertexto, onde os blocos de informação interconectados por meio de *links* podem ser, além de textos, imagens, sons, vídeos, arquivos, etc.

implementados o POST, o PUT e o DELETE. Semanticamente suas funções são:

- GET - Cujo objetivo é pedir dados ao servidor;
- POST - Usado principalmente para enviar dados ao servidor (formulários, arquivos, ...);
- PUT - Também usado para enviar dados, mas com o intuito de modificar uma entrada já existente;
- DELETE - Para fazer uma requisição de remoção de algum dado.

Um servidor web pode também receber dados via método GET, porém de modo não seguro uma vez que os dados são enviados explicitamente, concatenados com o endereço da requisição. A ocorrência do símbolo "?"no endereço indica que daí em diante são dados, no formato *parâmetro=valor*, separados pelo símbolo "&". Exemplo:

http://www.webmail.com.br/?username=usuario&password=senha

Toda resposta a uma requisição HTTP contém um código numérico de três dígitos. O servidor utiliza esse código para informar ao cliente sobre o estado da requisição. São eles:

- 1xx - São os códigos para um resposta provisória, apenas informativa. Por exemplo o código 100 indica que o cliente pode prosseguir com a conexão;
- 2xx - Indicam que a requisição foi recebida, entendida e aceita pelo servidor. O mais famoso exemplo é o código 200, que indica que a requisição foi processada com sucesso;
- 3xx - São utilizados para redirecionar uma requisição a um outro servidor, por exemplo em balanceamento de carga entre múltiplos servidores de um mesmo serviço;
- 4xx - Códigos de erros por parte do cliente. O código 404 informa ao cliente que o recurso solicitado não foi encontrado no servidor;
- 5xx - Códigos de erros por parte do servidor. O código 500 indica que o servidor não pôde atender à solicitação de um recurso por um erro interno.

2.5 Web Services

De acordo com a definição do W3C[9] (órgão responsável por regulamentar e sugerir novos padrões para a web), um Serviço Web é uma aplicação projetada para interagir com outras aplicações em redes de computadores, ou seja, um Serviço Web disponibiliza seus recursos mapeados em endereços de rede por meio de uma interface de acesso. Esse acesso ao serviço é feito tipicamente utilizando-se mensagens SOAP , transportadas por meio do protocolo de rede HTTP em formato de um documento XML serializado.

Existem duas arquiteturas principais de Serviços Web, classificados de acordo com a sua interface de acesso: SimpleObject Access Protocol (SOAP) e RepresentationalStateTransfer (REST). O SOAP é um padrão mais antigo e amplamente adotado por empresas e na indústria, enquanto que o REST é uma arquitetura proposta mais recentemente para simplificar o acesso e a implementação de Serviços Web.

Os Serviços Web que implementam a arquitetura REST são denominados RESTFul Web Services.

2.6 SOAP

O padrão SOAP foi inicialmente concebido para facilitar a comunicação em aplicações protegidas por firewalls. É um protocolo bem definido, atualmente mantido pela W3C[10], baseado na troca de mensagens em formato XML. A troca de mensagens pode ser implementada utilizando-se qualquer protocolo de transporte, como por exemplo HTTP, SMTP ou ainda diretamente via TCP ou UDP, cujo propósito é transmitir a mensagem XML serializada.

Uma mensagem SOAP é um documento XML com a seguinte estrutura:

[9]Web Services Architecture - http://www.w3.org/TR/ws-arch/#whatis
[10]SOAP specification - http://www.w3.org/TR/soap/

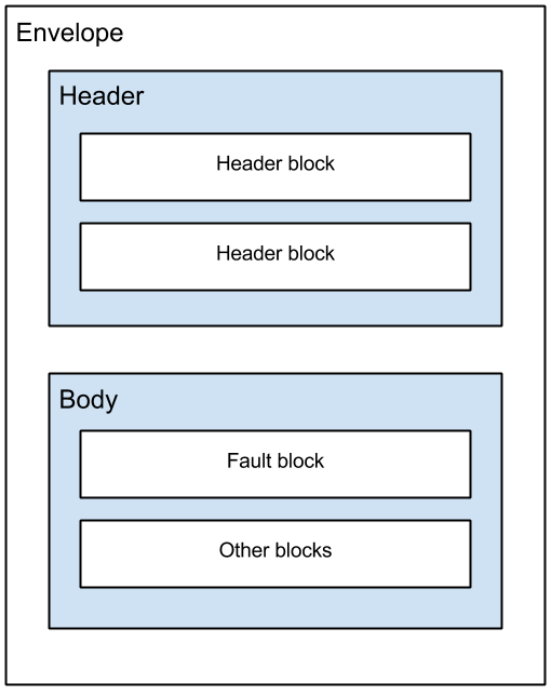


Figura 2.2: Diagrama de uma mensagem SOAP

Onde:

- <Envelope> é o elemento principal e contém outros 2 elementos, o <Header> e o <Body>;
- <Header> é opcional e quando presente precisa ser o primeiro descendente do <Envelope>. Contém informações específicas que serão processadas por nós intermediários, antes da mensagem atingir o destinatário final;
- <Body> sessão destinada à requisição de um recurso ao Serviço Web. Contém o endereço especifico do recurso solicitado, nome e os dados;
- <Fault> subelemento do <body> destinado a mensagens de erro.

2.7 REST

O REST , mais recentemente proposto [7], não é exatamente um protocolo mas sim uma arquitetura proposta para simplificar o acesso e a implementação de Serviços Web.

Não há um modelo padrão para uma mensagem REST, podendo-se utilizar diversos tipos de documentos ou formatos de dados, incluindo o próprio XML ou texto plano. Isso permite adequar o modelo de dados adotado na requisição de cada recurso fornecido pelo Serviço Web.

A troca de mensagens, assim como o acesso aos recursos de um Serviço Web REST, são realizados utilizando-se exclusivamente o protocolo HTTP , por meio de seus métodos nativos, como por exemplo GET, POST, DELETE, e etc. Os recursos disponibilizados são identificados por meio de endereços únicos de rede (URI), e não no interior da mensagem como no caso do SOAP.

Um Serviço Web REST não mantém o estado de dados (contexto) entre requisições, ou seja, cada requisição por um recurso do Serviço Web contém todos os dados necessários para a solicitação na mensagem.

2.7.1 REST vs SOAP

As principais diferenças entre REST e SOAP estão no tipo de mensagem utilizada na comunicação, e no modo como essas mensagens são transportadas.

Uma mensagem SOAP tem um formato especifico e bem definido, denominado envelope SOAP. Os dados enviados a um Serviço Web SOAP precisam necessariamente estar contidos nesse envelope que, além dos dados, contém outras informações relevantes para se obter acesso a um recurso, como seu nome, seus parâmetros, seu endereço específico dentre outras informações.

Já uma mensagem REST não possui um formato de dados específico prédefinido, no qual qualquer tipo de hipertexto pode ser usado, por exemplo texto plano, o próprio XML, JSON e inúmeros outros. Em um mesmo Serviço Web pode haver recursos que utilizam mensagens de tipos distintos, aumentando a flexibilidade e melhor adequando o modelo de dados ao tipo de aplicação. Além disso, ao utilizar-se o formato JSON[11], por exemplo, ao invés do XML, uma mesma mensagem pode ficar muito mais compacta , diminuindo o fluxo de dados na rede, e mais simples de interpretar por quem

[11] JSON, Java Script Object Notation - http://www.json.org/

solicita tal recurso.

Quanto ao modo de transporte de mensagens, o SOAP teoricamente pode utilizar qualquer protocolo de transporte de rede, porém na prática o mais utilizado é o HTTP, assim como o REST. A diferença é que apesar de ambos empregarem o HTTP, o SOAP o utiliza apenas como protocolo de transporte, enquanto que o REST o utiliza também para controle.

Os recursos de um Serviço Web RESTful são acessados por meio dos métodos de controle do HTTP 2.4, juntamente com um endereço único que identifica cada recurso fornecido pelo Serviço Web. Já em um Serviço Web SOAP o método HTTP utilizado na requisição de um recurso geralmente é ignorado, ficando a cargo do serviço fazer o controle da transação de acordo com o recurso solicitado descrito no envelope SOAP. Por exemplo, haverá um recurso getDados() para retornar alguma informação, e um outro recurso sendDados() para receber dados, no qual o Serviço Web terá a competência de resolver a requisição.

Utilizar diretamente os métodos do HTTP, que já é também responsável por transportar os dados na rede, tem a vantagem de simplificar a implementação da interface de acesso aos recursos de um Serviço Web, aumentando assim sua escalabilidade, que além de mantê-la mais homogênea ainda facilita o consumo dos recursos pelos usuários do serviço.

As figuras 2.3 e 2.4 ilustram, respectivamente, as mensagens de requisição e resposta de um mesmo recurso em um Serviço Web REST, e de sua variante SOAP:

requisição

```
http://nhagaexemplo.usp.br/servicos/almoco/central

GET servicos/almoco/central HTTP/1.1
Accept: application/json, text/javascript, */*;
Content-Length: nnn

```

resposta

```
HTTP/1.1 200 OK
Content-Type: application/json

{"Mistura":"Carne Moida","Guarnicao":"Batata Gratinada",
 "Sobremesa":"Melancia","Suco":"Uva",
 "Base1":"Arroz","Base2":"Feijão"}
```

Figura 2.3: Requisição e resposta de um recurso de um Serviço Web REST

requisição

```
POST /Cardapio HTTP/1.1
Host: www.nhagaexemplo.usp.br
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
    <m:dateAndTime>2013-09-29T13:20:00.000-05:00</m:dateAndTime>
</soap:Header>

<soap:Body xmlns:m="http://www.nhagaexemplo.usp.br/cardapio">
    <m:GetAlmoco>
        <m:Restaurante>Central</m:Restaurante>
    </m:GetAlmoco>
</soap:Body>

</soap:Envelope>
```

resposta

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
    <m:status>1</m:status>
    <m:restaurante>central</m:restaurante>
</soap:Header>

<soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetAlmocoResponse>
        <m:Mistura>Carne Moida</m:Mistura>
            <m:Guarnicao>Batata Gratinada</m:Guarnicao>
            <m:Sobremesa>Melancia</m:Sobremesa>
            <m:Suco>Uva</m:Suco>
            <m:Base1>Arroz</m:Base1>
            <m:Base2>Feijão</m:Base2>
    </m:GetAlmocoResponse>
</soap:Body>

</soap:Envelope>
```

Figura 2.4: Requisição e resposta de um recurso de um Serviço Web SOAP

2.7.2 RESTFul web services

A popularidade dos Serviços Web RESTful tem aumentado devido, principalmente, a maior simplicidade de acesso aos seus recursos, uma vez que as requisições são realizadas por meio dos métodos nativos do protocolo HTTP , que também é utilizado como protocolo de transporte de dados. Além disso, a não obrigatoriedade de utilizar-se mensagens XML para a troca de informações, permite o uso de modelos de dados mais simples, como por exemplo o JSON, o que diminui a sobrecarga de dados (metadados) na rede.

Os Serviços Web são amplamente utilizados na Internet, por esse motivo o desempenho de tais aplicações é um fator muito importante. Conforme o trabalho apresentado por Snehal Mumbaikar e Puja Padiya [8], um Serviço Web RESTFul é uma melhor alternativa para implementação de aplicações na Web, pois reduz a quantidade de dados na rede, melhorando o tráfego de informação útil. Além do mais reduz a latência de acesso com mensagem mais compactas em tamanho de bytes. Os Serviços Web RESTful são mais leves e fáceis de implementar que os Serviços Web SOAP pois são auto descritivos com maior flexibilidade e menor sobrecarga. Como o objetivo do Sistema DataUSP-PosGrad é o de gerar relatórios de modo eficiente e rápido com o mínimo de impacto no tráfego de dados na rede, além de permitir seu uso por outros sistemas corporativos da USP e viabilizar seu acesso por dispositivos móveis, como tablets e smartfone, foi adotada a arquitetura REST para sua implementação e projeto.

2.8 SQL

SQL ou Structured Query Language ([5],cp.4) é uma linguagem declarativa usada para acessar os dados de um SGBD[12] relacional. Suas cláusulas podem ser divididas em duas categorias: a DDL e a DML.

A DML (Data Manipulation Language) é usada para fazer consultas, modificar e inserir dados no banco. As principais cláusulas são o SELECT, INSERT, UPDATE e DELETE. Exemplo:

[12]SGBD - Sistema Gerenciador de Banco de Dados

```
SELECT <coluna1,coluna2,...> FROM <tabela>
       WHERE <condicao1 AND condicao2 ...>

INSERT INTO <tabela> (coluna1,coluna2,...)
       VALUES (valor1,valor2,...)

UPDATE <tabela>
SET <coluna1>=<valor1>, <coluna2>=<valor2>, ...
WHERE <condicao>

DELETE FROM <tabela> WHERE <condicao>
```

Figura 2.5: Exemplos de consultas SQL

A DDL (Data Definition Language) descreve e modifica a estrutura das tabelas e de outros objetos do SGBD, como usuários e permissões de acesso. Exemplo:

```
CREATE TABLE `Usuarios` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `email` varchar(255) COLLATE utf8_bin NOT NULL,
  `nome` varchar(255) COLLATE utf8_bin NOT NULL,
  `instituicao` varchar(255) COLLATE utf8_bin NOT NULL,
  `cargo` varchar(255) COLLATE utf8_bin NOT NULL,
  `associacao_id` int(11) NOT NULL,
  `modified` datetime NOT NULL,
  PRIMARY KEY (`id`),
  FOREIGN KEY (associacao_id) REFERENCES Associacoes(id)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT
  CHARSET=utf8 COLLATE=utf8_bin;
```

Figura 2.6: Exemplo de criação de tabela em SQL

2.9 JavaScript

Javascript é a linguagem de script mais popular da web, presente na maioria das paginas HTML atuais. Tem tipagem fraca , type-safety [11] e é capaz

de representar múltiplos paradigmas como orientação a objetos, funcional e imperativo. Foi formalizada no padrão ECMA-262.[13]

É uma linguagem amplamente usada para processar dados do lado cliente de uma aplicação web, por exemplo:

- modificando a página dinamicamente e interagindo com o usuário;
- tratando os dados recebidos da página antes de enviar ao servidor;
- fazendo chamadas assíncronas ao servidor sem recarregar a página.

2.9.1 JQuery

Jquery[14] é uma biblioteca de JavaScript que facilita e simplifica a manipulação de documentos HTML e o tratamento de eventos. Por exemplo, o trecho de código abaixo troca a cor de todos os links em uma página, utilizando JQuery e JavaScript puro respectivamente:

Jquery

```
1 \$("a").css('color','red');
```

javaScript

```
1 var l = document.getElementsByTagName("a");
2 for(var i=0; i < l.length; i++){
3   l[i].style.color = "red";
4 }
```

O seletor $() em JQuery utiliza a mesma sintaxe da notação de estilos css[15] para selecionar um elemento.Exemplo:

- *$("#id")* para selecionar um id;
- *$(".cl")* para selecionar elementos que possuem a classe "cl";
- *$("tag")* para selecionar uma tag qualquer de HTML.

[13] ECMA-262 - http://www.ecma-international.org/ecma-262/5.1/
[14] JQuery - http://jquery.com/
[15] CSS - http://www.w3.org/TR/CSS21/cover.html#minitoc

Enquanto que em JavaScript puro há um método específico para cada tipo de elemento:

- *document.getElementById("id")* para selecionar um id;
- *document.getElementsByClassName("cl")* para selecionar um elemento que possui a classe "cl";
- *document.getElementsByTagName("tag")* para selecionar uma tag qualquer.

Jquery possui ainda diversos plugins para prover funcionalidades úteis, como preencher datas, auto completar campos de busca, mostrar uma sequência de imagens com efeitos de transição, dentre muitas outras. Isso tudo permite escrever um código menor, mais legível e de fácil manutenção.

2.9.2 AJAX

Ajax [13] (Asynchronous JavaScript and XML) foi um termo cunhado por Jesse Jamers Garret para designar um conjunto de tecnologias para fazer requisições assíncronas a um Serviço Web, isto é, adicionar ou modificar dados em uma página web(site) dinamicamente, sem a necessidade de recarregá-la completamente. Esse tipo de requisição é feita usando-se JavaScript, ou por meio de JQuery. Apesar do nome é possível usar JSON como formato de dados ao invés de XML.

Uma requisição Ajax por intermédio de JQuery tem o seguinte formato:

```
\$.ajax({
    url : "/datausppg/recursos/listagem",
    type: "POST",
    data : formData,
    success: function(data, textStatus, jqXHR)
    {
        //data - dados retornados pelo servidor
    },
    error: function (jqXHR, textStatus, errorThrown)
    {

    }
});
```

Onde:

- *url*: endereço do recurso solicitado;
- *type*: tipo de requisição (GET, POST, ...);
- *data*: dados a serem enviados de acordo com o tipo de requisição;
- *success*: função executada caso o servidor retorne um código HTTP de sucesso (possivelmente enviando dados);
- *error*: função executada quando o servidor responde com um código HTTP de erro;

2.9.3 HTML5

HTML (HyperText Markup Language) é a linguagem usada pelos navegadores de internet (Web Browsers) para interpretar os documentos (páginas) da web. Sua estrutura é composta de tags, que são marcadores que identificam os elementos que compõe um documento HTML.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8">
5          <title>Titulo do documento</title>
6      </head>
7      <body>
8          <!-- Conteudo do documento -->
9      </body>
10 </html>
```

Figura 2.7: Estrutura básica de um documento HTML

Sua versão mais recente é o HTML5 [16], cujo um dos propósitos é diminuir a dependência de plugins externos e scripts (com o advento de novas tags para renderização de áudio, video e animações). Além disso, passa a suportar novas funcionalidades que visam melhorar a padronização do código, assim como sua portabilidade entre diferentes plataformas e navegadores.

[16] HTML - `http://www.w3.org/community/webed/wiki/HTML`

2.9.4 Gráficos - FusionCharts

Fusion Charts[17] é uma biblioteca javaScript para construir gráficos. Usada conjuntamente com HTML5, os gráficos são renderizados unicamente por meio de javaScript, criando modelos animados em três dimensões sem a necessidade de se utilizar algum plugin externo (como o Adobe Flashplayer por exemplo), reduzindo a carga no servidor e melhorando o desempenho do lado cliente. A figura 2.8 ilustra alguns modelos de gŕaficos construidos com o Fusion Charts.

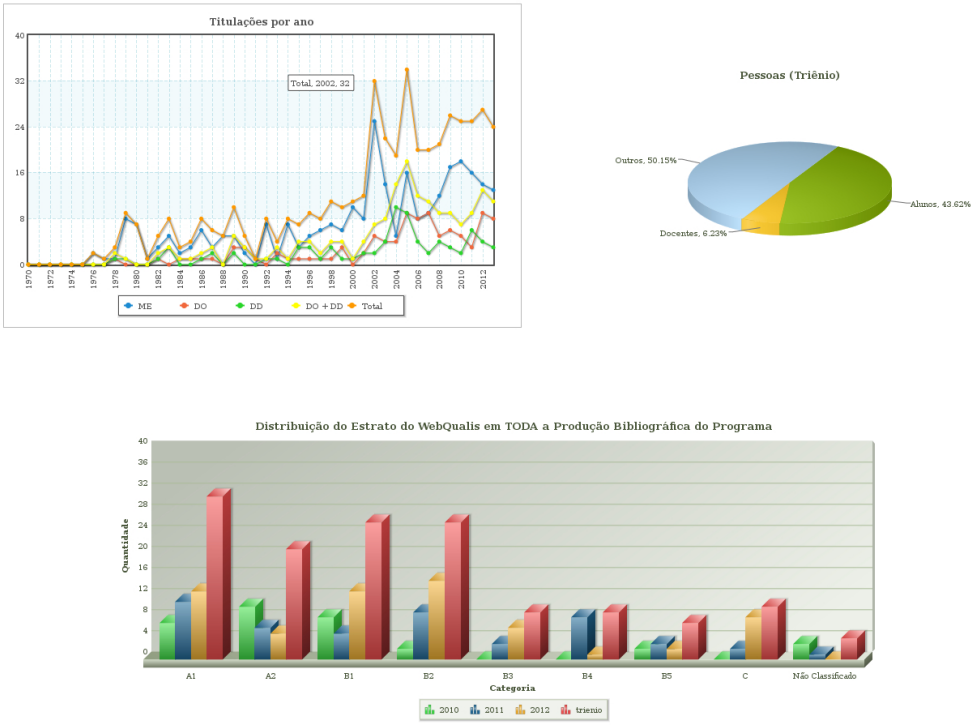


Figura 2.8: Alguns modelos de gráficos disponíveis no Fusion Charts

[17] Fusion Charts - `http://www.fusioncharts.com/`

2.10 Web Crawler

Web Crawler [14] é um programa de computador que navega na internet de modo autônomo, visitando páginas e lendo seu conteúdo, geralmente com a finalidade de indexação.

Duas das principais ferramentas para interagir com a web em linux são o curl e o wget. Enquanto que o wget é apenas um comando o curl usa a biblioteca libcurl[18], disponível em múltiplas plataformas e suportada por diversas linguagens de script, dentre elas Python.

A biblioteca libcurl trabalha com diversos protocolos incluindo o HTTP, permitindo simular o comportamento de um navegador de internet, inclusive utilizando cookies[19], enviando dados de formulários, fazendo autenticação e etc. Além disso pode-se configurar o cabeçalho de uma requisição para incluir dados adicionais e até mesmo se passar por algum navegador específico, como o Mozilla Firefox por exemplo (Algumas páginas fazem redirecionamentos específicos e até mesmo bloqueiam conteúdo dependendo do navegador usado).

Por meio da bibliotea libcurl é possível construir uma aplicação que navegue na internet, extraindo dados de páginas específicas afim de utlilizá-los como uma fonte externa de informações, armazenando-os em um banco de dados.

2.10.1 Python

Python[20] é uma linguagem de programação interpretada que vem ganhando notoriedade por sua versatilidade e facilidade de usar. É uma linguagem multiparadigma com uma variedade extensa de bibliotecas para as mais diversas aplicações (por exemplo o framework mvc django[21] e a biblioteca de ferramentas científicas SciPy[22]).

Utilizando as bibliotecas pycurl e pyquery é fácil construir um script para navegar na internet. A primeira é a implementação em python da libcurl

[18]libcurl - `http://curl.haxx.se/libcurl/`

[19]HTTP Cookie - É um arquivo de texto salvo pelo navegador do lado cliente usado pelo servidor para obter dados anteriores da navegação de seus usuários

[20]Python - `http://www.python.org/`

[21]Django - `https://www.djangoproject.com/`

[22]SciPy - `http://www.scipy.org/`

e a segunda permite percorrer um documento HTML de modo similar ao JQuery.

2.11 Pentaho

Pentaho Business Analytics[23] é um conjunto de ferramentas de BI (Business Inteligence) para fazer análise, visualização e transformação de dados por meio de interfaces amigáveis e intuitivas. O pacote tem licença GPL, ou seja é open source, apesar de possuir também uma versão comercial.

Uma das ferramentas do Pentaho é o Kettle, utilizada para fazer acesso e transformação de dados (ETL [4]) entre diferentes fontes, que vão desde arquivos de texto às mais diversas bases de dados. Muito utilizada na construção e manutenção de um Data Warehouse.

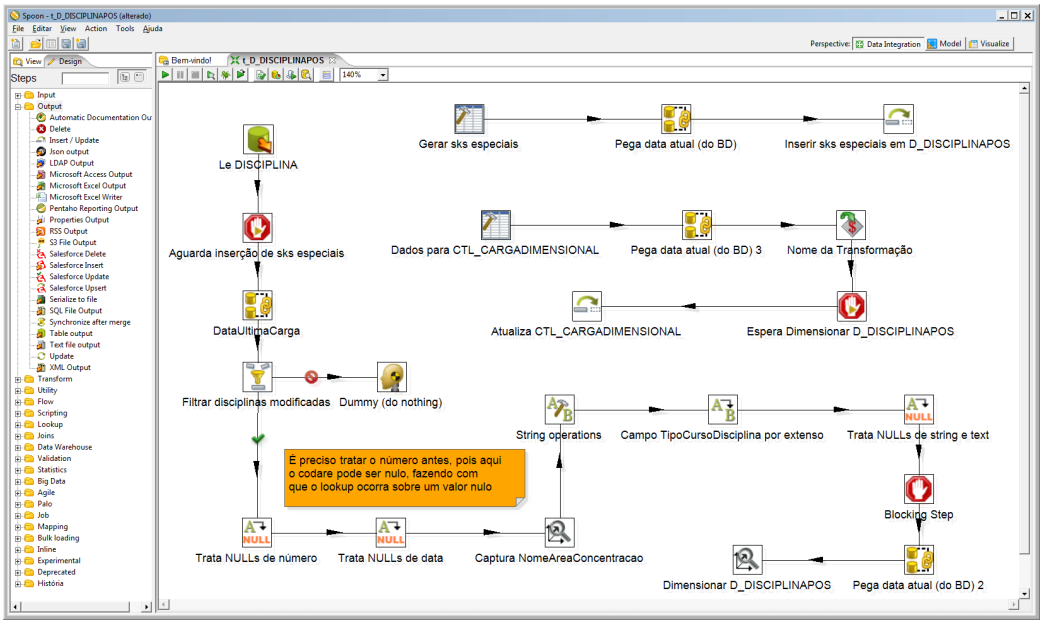


Figura 2.9: Ferramenta Kettle do pacote Pentaho

[23] Pentaho - `http://sourceforge.net/projects/pentaho/`

Capítulo 3

Atividades realizadas

3.1 Modelagem

O projeto DataUSP-PosGrad iniciou-se na Pró-reitoria de Pós-graduação da Universidade de São Paulo (USP) em 2012, com a modelagem dos dados para a criação de um Data Warehouse. Com o domínio dos dados bem definido, iniciou-se o processo de modelagem do sistema e a escolha das tecnologias que seriam utilizadas em sua construção. O Departamento de Tecnologia da Informação da USP, que é a equipe responsável por fazer a manutenção e desenvolver novos sistemas para a universidade, utiliza por padrão a linguagem de programação Java juntamente com o sistema gerenciador de banco de dados Sybase ASE.

Deste modo os requisitos básicos do sistema foram definidos como:

1. Linguagem de programação Java;
2. Interface web para interação com o usuário;
3. Orientado a serviços (Serviço Web);
4. Banco de dados Sybase ASE;

Por ser orientado a serviços o sistema foi então dividido em duas grandes componentes: O servidor de recursos e uma interface web(aplicação cliente) para a exibição dos dados.

3.1.1 Servidor de recursos

Um Serviço Web pode interagir com outros serviços e não exclusivamente com uma única aplicação. Portanto ele deve dispor de um mecanismo de requisição de seus recursos, ou seja, uma interface de acesso. Para facilitar a escalabilidade e a manutenção do sistema foi decidido então que o DataUSP-PosGrad seria implementado seguindo as premissas do REST 2.7 por meio do framework Jersey 2.3. Além disso, o formato de dados JSON (JavaScript Object Notation) foi adotado como padrão para a troca de mensagens.

O JSON é um formato texto para a troca de dados completamente independente de linguagem. Utiliza convenções que são familiares às linguagens C , C++, C#, Java, JavaScript, Perl, Python dentre outras. É uma estrutura de dados simples, leve e fácil de ser interpretada por humanos e também por computadores. Sua estrutura é composta por um conjunto de pares chave e valor do com a seguinte sintaxe: chave1 : valor1, chave2 : valor2, ... As chaves são cadeias de caracteres e os valores podem ser diversos tipos de objetos, dependendo da linguagem utilizada, como listas, dicionários, números, ou até mesmo outros JSON.

Como uma interface web será o principal consumidor dos recursos do servidor, o formato JSON é adequado para a comunicação, uma vez que quem a executa é um navegador de internet. Os navegadores de internet, também conhecidos por Web Browsers (por exemplo o Mozilla Firefox, Google Chrome, Internet Explorer e etc), têm motores específicos para executar código JavaScript e por conseguinte interpretam o formato JSON nativamente.

3.1.2 Interface Web

A Interface Web é um cliente que utiliza os Serviços Web no DataUSP-PósGrad. Sua função é requisitar os dados ao servidor de recursos, de acordo com os parâmetros fornecidos pelo usuário do sistema,e construir relatórios Ad-Hoc (relatórios sob demanda), por meio de gráficos e tabelas. Além disso a Interface Web deve ser capaz de receber e enviar arquivos ao servidor, como também enviar informações de autenticação do usuário. Desse modo, a seguinte lista de requisitos deve ser atendida em sua implementação:

1. Integração com o login único da USP de autenticação;
2. Exibição de gráficos em três dimensões interativos e renderizados em tempo real;

3. Utilizar a biblioteca JQuery 2.9.1 para a manipulação dos dados;

4. 4. Interagir com o servidor de recursos por meio de requisições assíncronas AJAX 2.9.2.

A biblioteca FusionCharts 2.9.4, utilizada para renderização gráfica, foi escolhida por atender os requisitos do sistema e por sua diversidade de modelos de gráficos. Nessa etapa de modelagem da Interface Web também foram definidos seu leiaute, cores e padrões de desenho.

3.1.3 Consultas

Definida a arquitetura do sistema, iniciou-se o processo de modelagem dos relatórios analíticos, que são basicamente as consultas SQL 2.8 responsáveis por buscar os dados no DataWarehouse. Além das consultas para gerar relatórios foram definidas ainda as consultas de controle e as consultas de filtragem.

As consultas de controle buscam dados relativos às permissões de acesso dos usuários, como a unidade a que pertencem, a quais programas e áreas ele tem acesso e quais os relatórios que eles podem gerar. Já as consultas de filtragem permitem que o usuário selecione dados específicos dentro de seu escopo de visualização, filtrando as informações mostradas nos relatórios.

3.1.4 Usuários

Os usuários foram classificados em grupos de acordo com a permissão de acesso dentro do sistema. São eles:

- *Nível 1* - Grupo responsável pelo sistema, não tem restrição de acesso (usado apenas para desenvolvimento);
- *Nível 2* - É o nível máximo que tem permissão total de acesso (Reitor e vices, Pró-reitor de pós-graduação);
- *Nível 3* - Usuário comum que tem permissão apenas aos seus programas e áreas específicos (Coordenadores e presidentes de CCP e CPG, assim como seus suplentes);

3.2 Construção do sistema

Uma vez determinadas as tecnologias a serem utilizadas, iniciou-se o processo de construção de um arcabouço, afim de testar as funcionalidades de Serviços Web do Jersey 2.3. Os detalhes técnicos e arquiteturais da linguagem Java apresentados nesta seção, encontram-se na seção 2.2.

Esse esqueleto do DataUSP-PósGrad tem a seguinte estrutura:

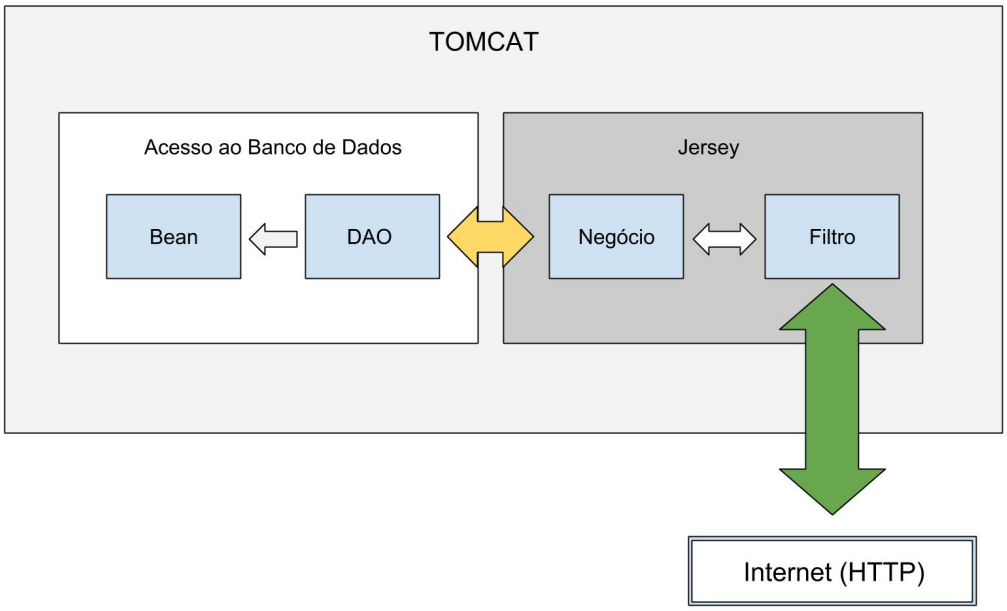


Figura 3.1: Esqueleto do DataUSP-PósGrad

1. **Tomcat**O servidor Apache Tomcat é um contêiner Web de código aberto. Foi desenvolvido para executar aplicações Web que utilizam as tecnologias Java Servlet e JavaServer Pages. Ele é responsável por instanciar as classes Java de acordo com as requisições web recebidas;

2. **Filtro** - Todas as requisições atendidas pelo Tomcat seguem por um filtro do Jersey, responsável por validar e autenticar o acesso a todos os recursos disponíveis no Web Service;

3. **Negócio** - Classe de negócio contém a lógica responsável por tratar as requisições. Os métodos públicos dessa classe são a implementação dos recursos do Serviço Web (que é a própria classe Java). Cada um desses

métodos públicos é identificado por meio de anotações, que nada mais são do que o identificador único de cada recurso, ou seja, seu endereço na rede;

4. **Data Access Object (DAO)** - Classe Java responsável por abstrair a interação das classes de negócio com o banco de dados. O DAO gerencia a conexão, executa as consultas SQL ou procedimentos armazenados na base de dados (Stored Procedures) e, retorna os dados à classe de negócio;

5. **Bean** - Essa classe encapsula os atributos que representam as colunas de uma tabela no banco de dados. É usada pelo DAO para retornar o resultado de uma consulta.

3.2.1 Funcionamento

Ao receber uma requisição por um recurso, o Tomcat instancia o filtro e a classe Java de Negócio referente ao Serviço Web solicitado. Isso é feito utilizando-se o endereço único de tal recurso por meio das Anotações em Java.

O Filtro então autentica o usuário do serviço e transfere a solicitação para o Negócio que, por sua vez, executa seu método público referente ao recurso requisitado (novamente por meio do uso de anotações em Java). Os métodos públicos (recursos) das classes de Negócio (Serviços Web), contém ainda outras anotações referentes ao método HTTP de acesso(@GET, @POST, etc) e também aos formatos de dados recebidos(@Consumes) e enviados (@Produces).

O método público correspondente ao recurso solicitado instancia um DAO, que busca os dados no banco e os retorna uma lista de BEAN. Cada BEAN nessa lista corresponde a uma linha da tabela gerada pela consulta sql executada no banco. Esses dados então são processados para gerar a resposta enviada ao usuário do recurso, por meio do protocolo HTTP.

3.2.2 Exemplo de funcionamento

Por meio da Interface Web, um usuário do DataUSP-PósGrad pode, por exemplo, gerar um relatório com a quantidade de titulações do programa de

Ciência da Computação do Instituto de Matemática e Estatística, em um intervalo de anos:

- O cliente (Interface Web) solicita o recurso "número de títulos"por meio do seguinte endereço:

  http://uspdigital.usp.br/datausppg/servicos/relatorio/numero_titulos/45/45005/45134/1998/2013

  Note que após ...numero_titulos/ os números significam:

  - 45 -> Código da unidade;
  - 45005 -> Código do programa;
  - 45134 -> Código da área;
  - 1998 -> Ano inicial;
  - 2013 -> Ano final.

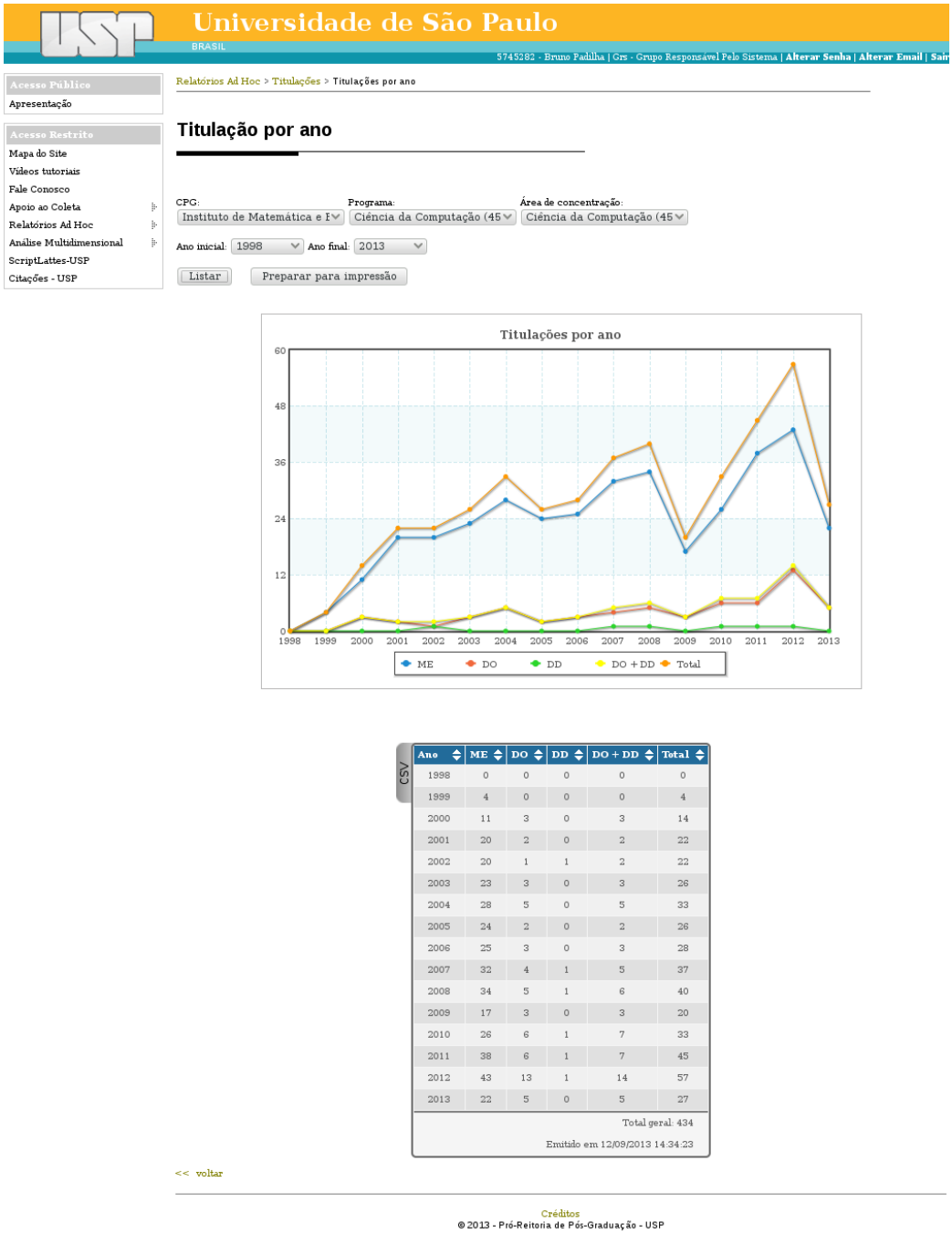
- O servidor atende a requisição 3.2.1 e retorna os dados no formato json 3.1.1:

  {"anoInicial":1998,"anoFinal":2013,
  "totalME":[0,4,11,20,20,23,28,24,25,32,34,17,26,38,43,22],
  "totalDO":[0,0,3,2,1,3,5,2,3,4,5,3,6,6,13,5],
  "totalDD":[0,0,0,0,1,0,0,0,0,1,1,0,1,1,1,0],
  "totalDODD":[0,0,3,2,2,3,5,2,3,5,6,3,7,7,14,5],
  "totalGeral":[0,4,14,22,22,26,33,26,28,37,40,20,33,45,57,27]}

  Esse json contem quatro listas referentes à quantidade de títulos nos cursos de Mestrado (totalME), Doutorado (totalDO), Doutorado Direto (totalDD) e a soma das três anteriores (totalGeral), no intervalo de "anoInicial"a "anoFinal".

A Interface Web então constrói o relatório e exibe os dados ao usuário como na figura 3.2.

Figura 3.2: Interface Web do DataUSP-PósGrad



3.3 Relatórios Ad-hoc

Os relatórios Ad-hoc são gerados dinamicamente e, em geral, exibem os dados na forma de um gráfico interativo e também em uma tabela, que pode ser exportada como uma planilha. São classificados em quatro grupos principais conforme o interesse da Universidade:

1. **Titulações** - Estatísticas quantitativas dos docentes, programas e áreas da pós-graduação;
2. **Egressos** - Situação dos alunos egressos;
3. **Internacionalização** - Estrangeiros em cursos de pós-graduação da USP;
4. **Produção Coleta** - Produção intelectual dos orientadores e orientandos.

3.3.1 Titulações

Titulações por ano

Esse relatório exibe a quantidade de títulos de mestrado, doutorado e doutorado direto para cada área da pós-graduação na USP, desde o ano de 1970. Foi o primeiro relatório do DataUSP-PósGrad, serviu para consolidar o modelo 3.1, fazer alguns ajustes no Data Warehouse e explorar as funcionalidades do Fusion Charts 2.9.4(animações, tipos de gráficos, cores e etc). Veja a figura 3.2

Tempo médio de titulação por programa

Exibe o tempo médio para se obter um título de mestrado, doutorado e doutorado direto em meses. Pode-se mostrar todas as áreas de um dado programa no mesmo gráfico.

Tempo médio de titulação por docente

Exibe o tempo médio em meses de titulação dos orientandos por docente. Ao escolher uma área será exibida uma lista com todos os orientadores ca-

dastrados, onde pode-se selecionar um ou mais para exibir os dados em um mesmo gráfico.

Teses por programa

Numero total de teses dividido pelo numero de docentes de uma área.

Teses por docente

Do mesmo modo que em 3.3.1, um lista de docentes é exibida mas será gerado um gráfico para cada um dos selecionados, com o seu número de teses de mestrado, doutorado e doutorado direto no intervalo escolhido.

Downloads de teses por docente

Mostra uma lista com a quantidade de downloads da produção intelectual dos docentes na Biblioteca Digital de Teses e Dissertações [1]. Atualmente esta funcionalidade está desativada por um erro de inconsistência nos dados informados pela BDTD.

3.3.2 Egressos

Após conseguir um título de pós-graduação, os alunos egressos podem responder um questionário que tem por objetivo traçar um perfil de sua atuação profissional. Com a devida validação desses dados é possível saber, por exemplo a quantidade de egressos que atuam em empresas privadas, em cargos públicos, na área acadêmica e etc.

Alguns relatório de egressos foram desativados, até que se possa validar os dados dos questionários de modo mais rigoroso, para evitar erros e inconsistências.

Distribuição de respostas por colegiado

Calcula a quantidade de questionários respondidos por cada unidade da USP.

[1] BDTD - `http://bdtd.ibict.br/`

Total de respostas por conclusões

Como no relatório anterior porém relativamente ao numero total de egressos por unidade.

3.3.3 Internacionalização

Alunos por País

Exibe a distribuição dos alunos estrangeiros por país em cursos de pós-graduação na USP.

Estrangeiros em estágio de média/longa duração

Alunos estrangeiros em cursos de pós-graduação na USP que estão realizando estágio com mais de 3 meses de duração.

Estrangeiros em estágio de curta duração

Como no relatório anterior porém para estágio com menos de 3 meses de duração.

Alunos de duplo diploma

Alunos brasileiros ou estrangeiros que fizeram parte do curso na USP e parte em universidade estrangeira, obtendo dois diplomas.

3.3.4 Produção Coleta

São relatórios da produção intelectual da pós-graduação, de acordo com os dados que são enviados anualmente à Capes para análise.

Produção por Estrato

Relatório com a classificação Qualis[2] da produção dos programas de pós-graduação. Veja a figura 3.3

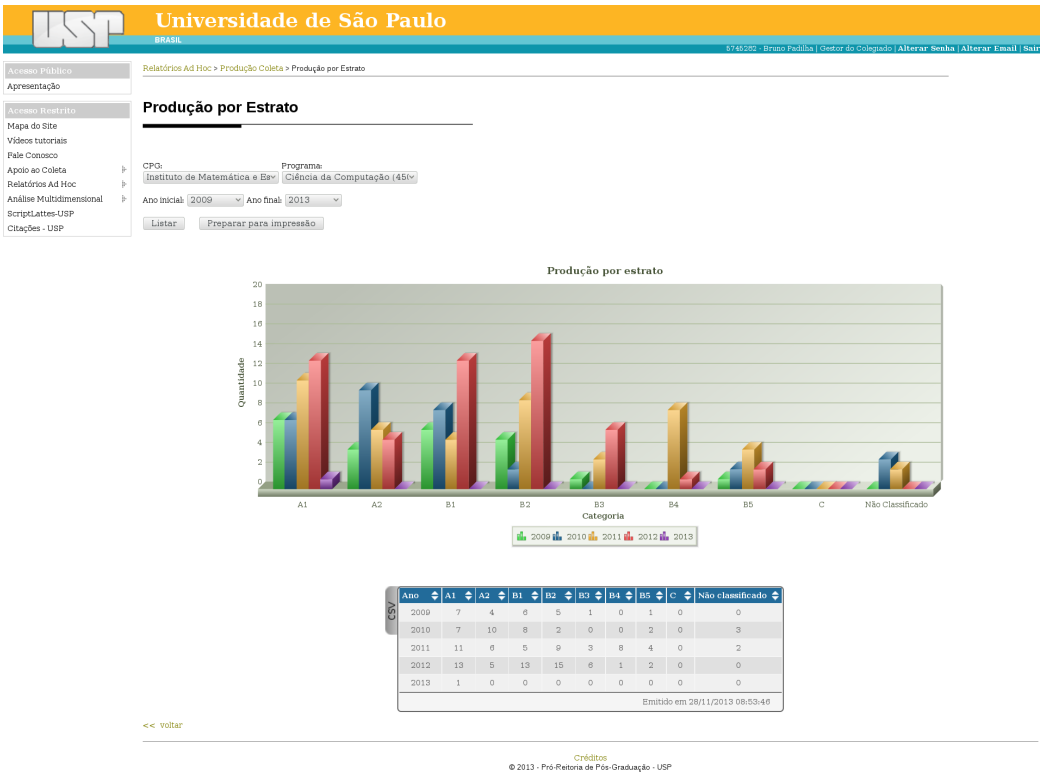


Figura 3.3: Produção intelectual por extrato

[2]Avaliação Qualis - `http://www.capes.gov.br/avaliacao/qualis`

Produção Total

Classificação do tipo de produção por programa em um dado intervalo de tempo. Alguns exemplos:

- Apresentação de trabalho;
- Artigo em jornal ou revista;
- Artigo em periódico;
- Livro
- Trabalho em anais

3.4 Análise Multidimensional

Principalmente por se tratar de um sistema analítico, o DataUSP-PosGrad é intrinsecamente orientado a dados. Uma estrutura multidimensional de dados, o cubo MOLAP 2.1.1, é essencial para a flexibilidade e eficiência das consultas que, basicamente, navegam no cubo em busca de uma informação específica.

Para ampliar a cobertura dos relatórios Ad-hoc e também para permitir que cada unidade da USP possa gerar suas próprias análises, o DataUSP-PosGrad fornece acesso direto ao cubo MOLAP, segmentado por unidade da USP, por meio de um visualizador chamado Saiku[3]. No Saiku o usuário pode selecionar as dimensões do cubo pertinentes e filtrar os resultados para obter os dados em forma de uma tabela. Veja um exemplo na figura 3.4

[3]Saiku - http://meteorite.bi/saiku

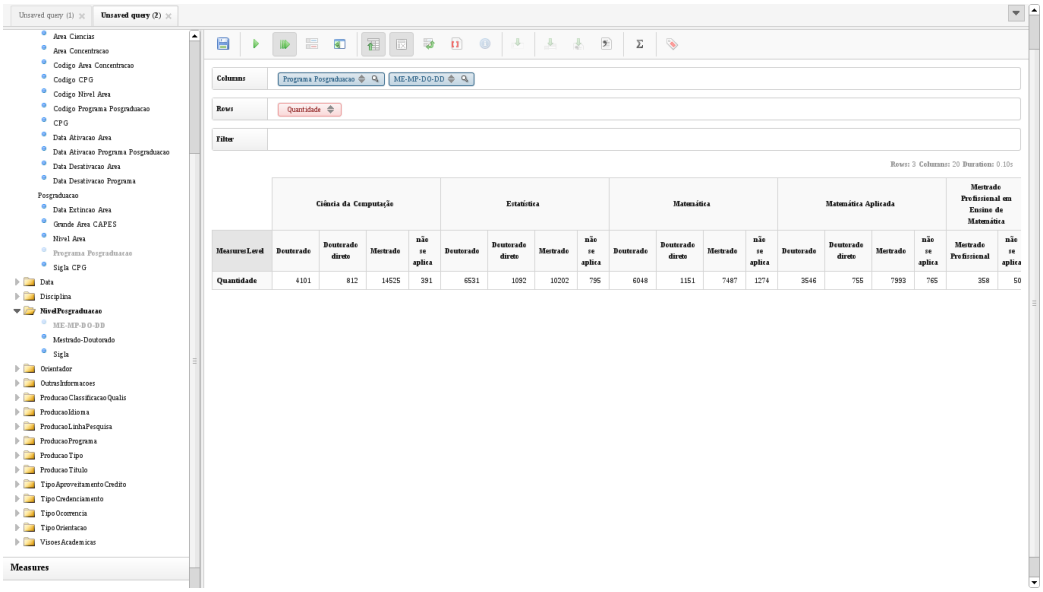


Figura 3.4: Exemplo de consulta dinâmica no cubo MOLAP usando o visualizador Saiku

Cada usuário pode baixar uma versão *offline* do cubo MOLAP da unidade USP a qual esteja vinculado, que pode ser visualizado com o Microsoft Excel.

Um exemplo da versatilidade de um Serviço Web RESTful é o recurso de download de arquivos no DataUSP-PósGrad. Uma solicitação de download responde com o próprio arquivo e não com o caminho onde o mesmo se encontra, ou seja, o formato de dados da resposta do recurso de download são os bytes conjuntamente com as metainformações de tamanho, nome e tipo do arquivo. Isso é feito modificando-se o cabeçalho da requisição por meio do protocolo HTTP, que já é usado como interface de acesso ao serviço. Quem solicita esse recurso tem então a capacidade de reconstruir o arquivo.

Essa alternativa de controle de download aumenta a segurança de acesso aos arquivos, uma vez que o caminho onde os mesmos se encontram no servidor nunca é revelado a quem consome tal recurso.

3.5 Citações-USP

Ao publicar um artigo, periódico, livro ou algum outro trabalho científico, o autor geralmente acaba citando outros trabalhos utilizados em sua pesquisa.

O impacto da produção de um autor em uma dada literatura pode ser medida por meio da quantidade de citações de seus trabalhos.

A quantidade de publicações e de suas respectivas citações definem uma métrica quantitativa denominada *índice-h* [15]. O índice-h é calculado como o número *ncit* de publicações de um autor que tenham ao menos *ncit* citações cada.

As principais fontes que contabilizam o número de citações são o *Google Scholar*[4], o *Scopus*[5] e o *Web of Science*. O DataUSP-PósGrad, por meio da técnica de *Web Crawling* 2.10, varre o Google Scholar e também o Scopus em busca de dados de publicações dos docentes da USP. Futuramente serão inclusos também os dados do Web of Science.

A figura 3.5 ilustra um exemplo de visualização dos dados de citações.

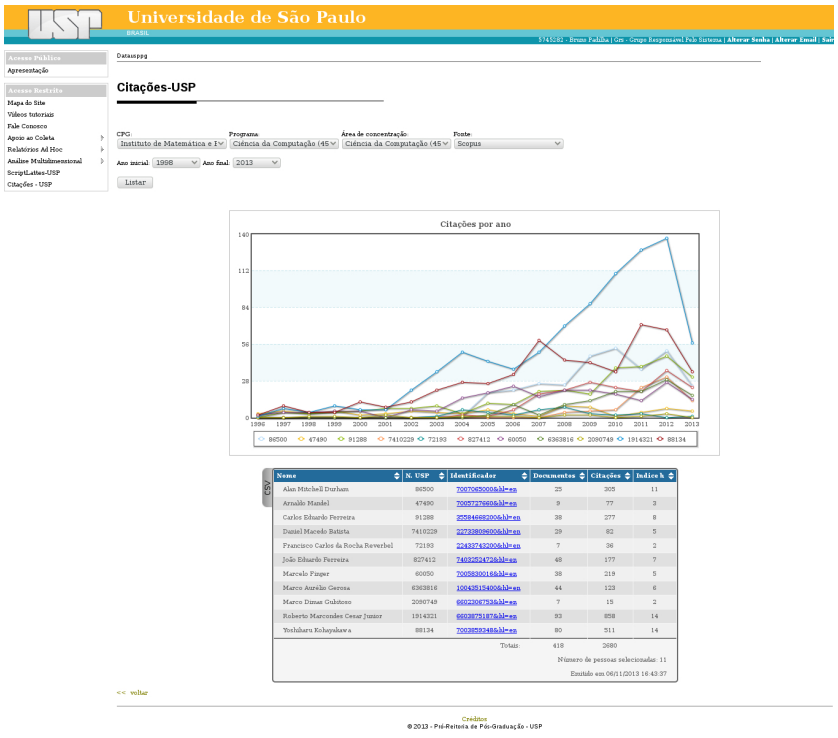


Figura 3.5: Gráfico comparativo com o número de citações de alguns docentes do IME

[4]Google Acadêmico - `http://scholar.google.com.br/`
[5]Scopus - `http://www.elsevier.com/online-tools/scopus`

3.5.1 Funcionamento do Web Crawler

Como essa parte do sistema foi escrita em Python, a interface com os bancos de dados (leitura e escrita) é feita por meio da ferramenta Kettle do Pentaho 2.11, já que não foi possível fazer com que o Web Crawler interagisse diretamente com os bancos de dados Sybase ASE, utilizados na USP.

Inicialmente, um script Kettle busca os dados dos docentes no banco (nome, número usp, programa a que pertence, etc) e gera um arquivo com essas informações. O Web Crawler então processa esse arquivo de entrada e, para cada docente, varre as páginas do Google Scholar e também do Scopus em busca dos dados de citações. Os dados gerados pelo Web Crawler são armazenados em um outro arquivo que, finalmente, é processado por um outro script Kettle que os insere no banco de dados.

Cada docente cadastrado no Google Scholar ou no Scopus possui um respectivo identificador único. Como a principio esses identificadores não estão armazenados nos bancos de dados da USP, a primeira tarefa do Web Crawler é fazer a associação desses identificadores aos docentes. A busca pelos identificadores é feita pelo nome do docente, considerando alguns outros parâmetros de similaridade como o nome da universidade, instituto e endereço de email. Ainda assim o nome do docente cadastrado na USP pode não ser exatamente igual ao buscado nas páginas, portanto o casamento de padrão no nome é feito heuristicamente.

O passo anterior apenas é executado para docentes que ainda não possuem seus identificadores Scopus ou Scholar associados ao seu número usp. O processo de atualização é similar, porém a busca é feita diretamente pelo identificador, o que torna o processo consideravelmente mais ágil e preciso.

Por se tratar de um processo heurístico, nem sempre os números usp e identificadores são associados corretamente. Para suprir essa deficiência, na Interface Web do DataUSP-PósGrad, o usuário tem a opção de corrigir um identificador, ou ainda informar um outro que não tenha sido encontrado pelo Web Crawler. Assim a tendência é que os dados se consolidem pelos próprios usuários.

3.6 Outros Serviços

Dois outros sistemas já existentes foram incorporados ao DataUSP-PósGrad: o *ScriptLattes* [16], no formato de uma ferramenta ; um outro sistema administrativo da pró-reitoria de pós-graduação da USP, anteriormente conhecido por Gerência-Pós, em forma de serviços.

3.6.1 ScriptLattes

O ScriptLattes é uma ferramenta para fazer análises de dados obtidos da plataforma *Lattes*[6]. Foi desenvolvido em Python e utiliza um arquivo de configuração onde, por meio do identificador lattes [7], são descritos os autores dos quais se deseja baixar os currículos.

exemplo.list

```
# Lista de professores (linhas de comentarios comecam com '#')
4575931307749163,Carlos Hitoshi Morimoto, 1999-HOJE, Professor
0131770792108992,Joao Eduardo Ferreira, 1999-2006 & 2006-HOJE, Professor
0362417828475021,Junior Barrera, 1992-2008, Professor
0926213060635986,Marcel Parolin Jackowski, 2006-HOJE, Professor
0644408634493034,Nina Sumiko Tomita Hirata,,Professor # membro sem periodo
1647118503085126,Roberto Hirata Junior,, Professor # membro sem periodo
2240951178648368,Roberto Marcondes Cesar Junior, 1998-HOJE, Professor
9283304583756076,Ronaldo Fumio Hashimoto,, Professor # membro sem periodo

# lista de colaboradores
4727357182510680,Jesus P. Mena-Chalco, 1995-1999 & 2005-HOJE, Colaborador

# lista de alunos
2837012019824386,Andrea Britto Mattos,, Aluno# membro sem periodo
```

Figura 3.6: Arquivo de configuração do ScriptLattes que contém a lista de pesquisadores

Um outro arquivo de configuração com os parâmetros da análise, contendo o caminho dos arquivos (por exemplo o caminho do arquivo 3.6 e configurações gerais do script, também é necessário.

Além de o usuário precisar conhecer previamente os identificadores lattes das pessoas em análise, requer-se um certo conhecimento sobre o ambiente onde o ScriptLattes será executado (instalar dependências, configurar permissões, criar os arquivos de configuração do script e etc).

[6] CNPQ Lattes - `http://lattes.cnpq.br/`

[7] O identificador lattes é um número como *"0131770792108992"* que identifica unicamente um pesquisador cadastrado na plataforma

Com o propósito de potencializar seu uso, o ScriptLattes foi integrado ao DataUSP-PósGrad. Na Interface Web, uma lista de pesquisadores é exibida (no formato dos relatórios ad-hoc 3.2) onde o usuário seleciona um grupo, inclusive filtrando por unidade ou área. Em seguida são gerados os arquivos de configuração automaticamente e o ScriptLattes é executado no Servidor de Recursos 3.1.1, e o resultado da análise é exibido na Interface Web.

Essa abstração da execução do ScriptLattes facilita o uso da ferramenta para os usuários do DataUSP-PósGrad, que tem ainda a possibilidade de receber os resultados da análise por email.

A figura 3.7 ilustra o resultado de uma análise realizada pelo ScriptLattes dentro do DataUSP-PósGrad.

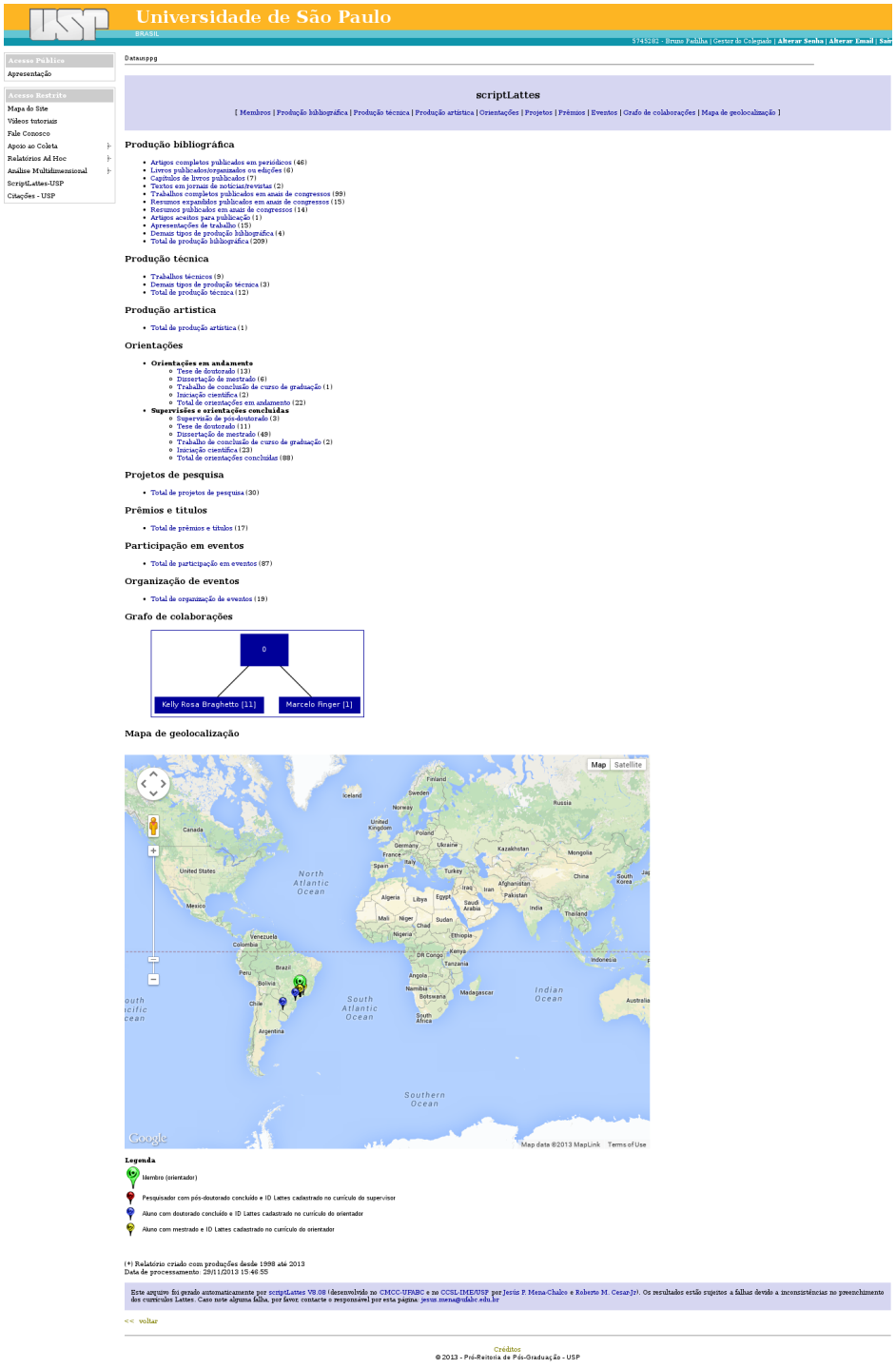


Figura 3.7: ScriptLattes dentro do DataUSP-PósGrad

3.6.2 Gerência-Pós (Apoio ao coleta)

O Gerência-Pós é um conjunto de ferramentas para extração e análise dos dados enviados à Capes anualmente para a avaliação dos programas de pós-graduação. Agiliza o processo de coleta dos dados automatizando o pre-enchimento de formulários, além de gerar um relatório histórico do *extrato WebQualis*.

Sua principais ferramentas foram transformadas em serviços e incorporadas ao DataUSP-PósGrad (por exemplo os relatórios 3.3.4 e 3.3.4) com o nome de *Apoio ao Coleta*.

No menu do apoio ao coleta, dentro do DataUSP-PósGrad, é possível enviar o arquivo no formato coleta capes de um programa para fazer a análise em tempo real. Isso é feito de modo similar ao download dos cubos offline 3.4, mas agora há um recurso no servidor que recebe o arquivo em formato binário. Por parte da interface web o envio de arquivos é feito de maneira assíncrona via Ajax 2.9.2 e HTML5 2.9.3, e o resultado da análise é retornado em seguida[8].

Outra funcionalidade disponível no mesmo menu é fazer a análise anterior automaticamente, porém para o triênio (últimos 3 anos, figura 3.8) e sem a necessidade de envio de arquivos.

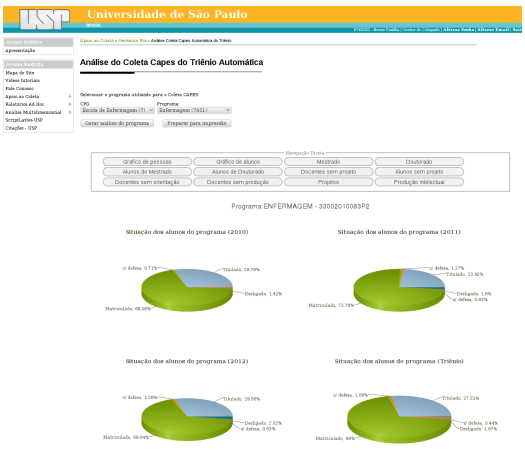


Figura 3.8: Imagem parcial do relatório de análise do triênio

[8] O uso de HTML5 é necessário para extrair o conteúdo binário do arquivo selecionado em um formulário, e também para exibir o progresso do upload ao usuário, a exemplo do que é utilizado no Gmail, o serviço de emails do Google, para envio de arquivos anexos

Capítulo 4

Resultados e produtos obtidos

4.1 Software final

O conjunto de ferramentas analíticas desenvolvidas nesse trabalho para auxiliar a tomada de decisões da pró-reitoria de pós-graduação da USP, recebeu o nome de DataUSP-PósGrad.

Sua primeira versão foi lançada oficialmente no dia 05 de junho de 2013, bem recebida pela comunidade de gestores dos programas de pós-graduação da USP. Ganhou uma matéria na revista *Espaço Aberto* da USP, nas edições 152 e 153[1].

Para capacitar os usuários e apresentar as ferramentas em detalhes, foram realizadas duas oficinas em todos os campus da USP, uma logo após o lançamento [2] e a segunda na última semana de novembro de 2013.

O sistema pode ser acessado por meio do portal *uspdigital* no endereço *http://uspdigital.usp.br/datausppg*. Por disponibilizar informações sensíveis dos programas e dos docentes da USP, o sistema pode ser acessado apenas por um grupo restrito de usuários (Dirigentes de unidades e seus secretários, Reitor, Pró-reitores e alguns outros). Além disso as bases de dados são acessíveis somente de alguns endereços ip específicos, administrados pelo Departamento de Informática da USP.

[1] DataUSP-PósGrad na revista Espaço Aberto - `http://espaber.uspnet.usp.br/espaber/?materia=usp-desenvolve-sistema-analitico-de-dados`

[2] 1ª oficina DataUSP-PósGrad - `http://iptv.usp.br/portal/home.jsp?tipo=0&_InstanceIdentifier=0&_EntityIdentifier=uspJ3kF6s9CQwQPddjpvikTCs_sbQicdp54HCw6bnX8bJs.&idRepositorio=0&modelo=0`

Pelos motivos descritos acima, e também por direitos de produção intelectual, o código do DataUSP-PósGrad pertencente à Pró-Reitoria de Pós-Graduação. O software produzido está em fase de registro pela Diretoria de Informática da Universidade de São Paulo e, uma vez registrado, deverá ser disponibilizado publicamente em forma de serviços.

4.2 Teste de Carga

Algumas consultas aos bancos de dados para gerar os relatórios no DataUSP-PósGrad são expressivamente caras computacionalmente, ou seja, um grande volume de dados precisa ser processado para se obter o resultado. Isso se traduz em tempo de processamento e, por consequência, o usuário do sistema precisa esperar alguns segundos para poder visualizar os dados. Com isso foram elaborados alguns teste de carga, com a finalidade de aferir o tempo de resposta do sistema simulando uma certa quantidade de usuários simultâneos e, poder avaliar a usabilidade como também a sensação de fluidez de DataUSP-PósGrad em uma situação de alta demanda.

4.2.1 Metodologia de teste

Os testes foram realizados com o relatório de Titulações por Ano 3.3.1(Ilustrado no exemplo 3.2.2). Para cada usuário simulado é sorteada aleatoriamente uma área da pós-graduação para a qual deseja-se exibir o relatório do numero de títulos por ano, no intervalo de 1970 a 2013, do mesmo modo que seria feito utilizando-se a Interface Web. Os resultados são uma média de três repetições de cada teste.

Foram utilizados dois bancos de dados distintos: Sybase ASE versão 13, que é utilizado em produção pelo DataUSP-PósGrad e também pelos demais sistemas administrativos da USP; Microsoft SQL Server versão 11, utilizado para armazenar o modelo multi-dimensional 2.1.1 e gerar os cubos off-line.

Para simular a carga de usuários foi utilizado o software JMeter [3]. Além disso, tanto o Servidor de Recursos quanto a Interface Web do DataUSP-PósGrad foram executados na mesma máquina em todos os testes.

O JMeter (figura 4.1) é um software livre da fundação Apache escrito em Java especialmente para realizar testes de carga, aom suporte a diversos tipos

[3] Apache JMeter - http://jmeter.apache.org/

de protocolos de rede e dentre eles o HTTP. No caso do DataUSP-PósGrad, o JMeter funciona executando um script que simula as ações de um usuário interagindo com a Interface Web. Esse script pode então ser executado em paralelo simulando um determinado número de usuários. O resultado é um gráfico que exibe a vazão de dados, média, mediana e desvio padrão do tempo de resposta das requisições ao Servidor de Recursos.

O objetivo desse teste é determinar o impacto da quantidade de usuários simultâneos no desempenho geral do sistema. Atualmente o DataUSP-PósGrad possui cerca de 2000 usuários cadastrados.

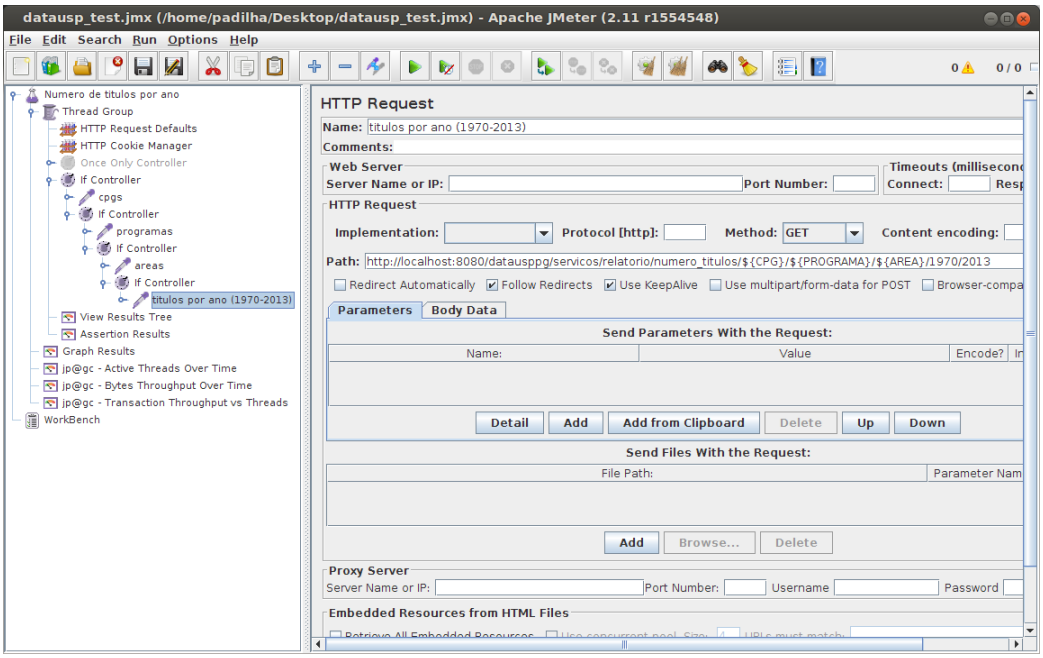
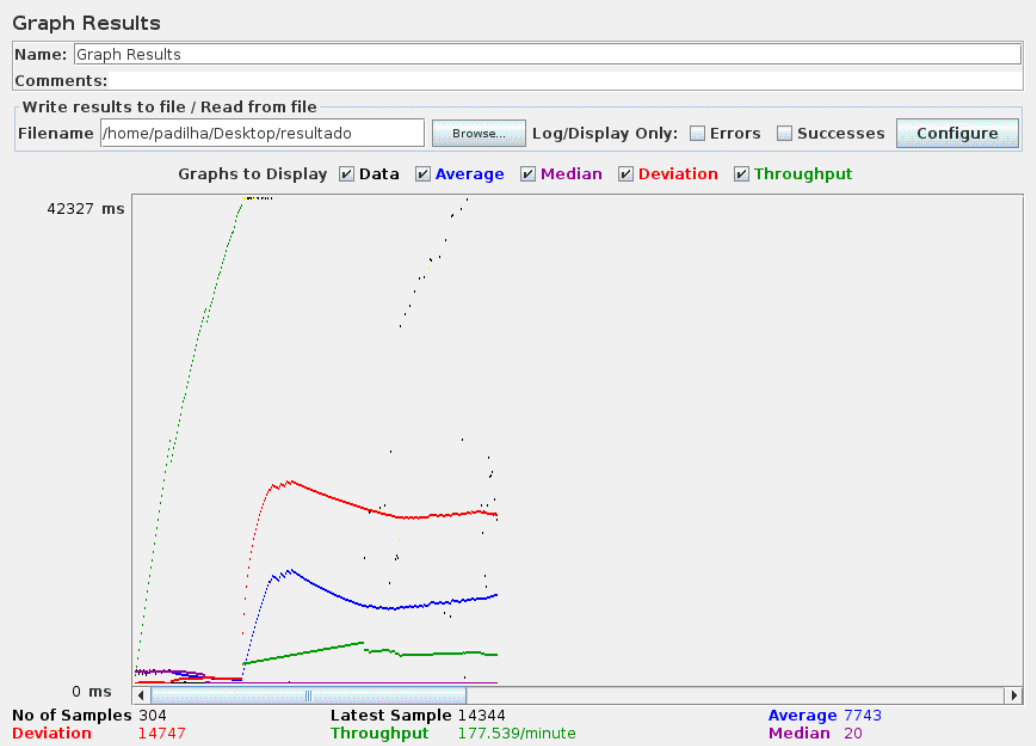


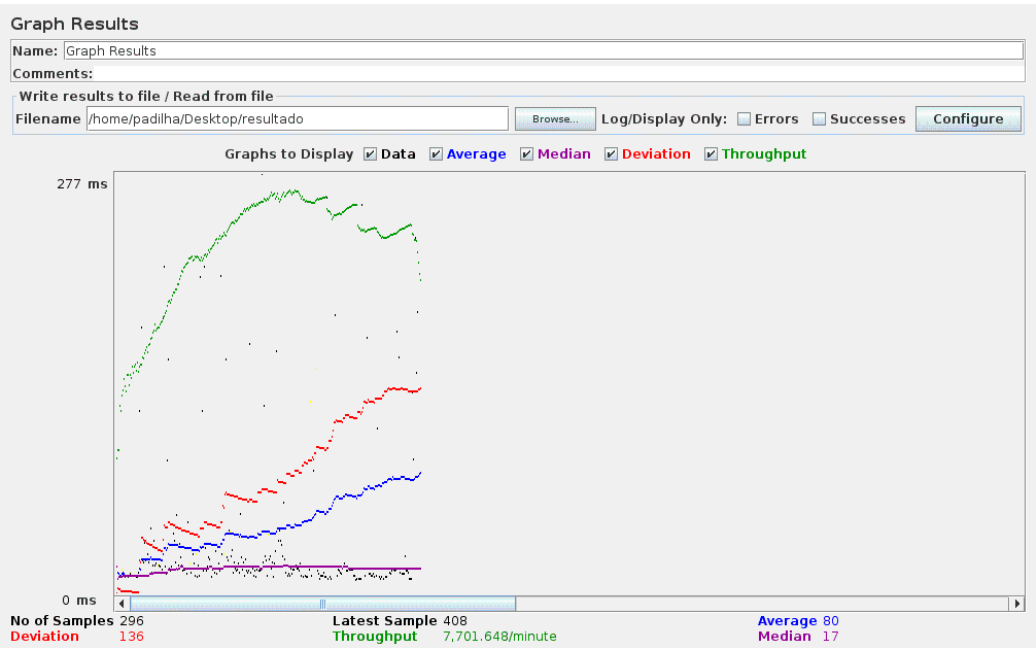
Figura 4.1: Interface gráfica do JMeter

4.2.2 Resultado dos testes

Utilizando o banco de dados Sybase ASE foram feitos testes simulando 30, 90 e 360 usuários simultâneos. Os resultados são apresentados nas figuras a seguir.

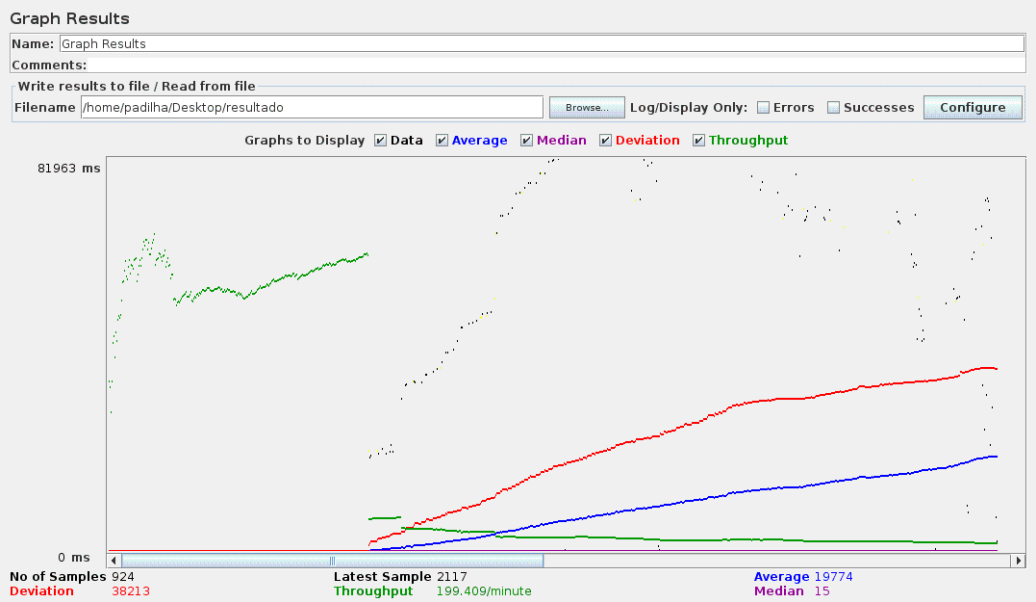


(a) Sybase ASE

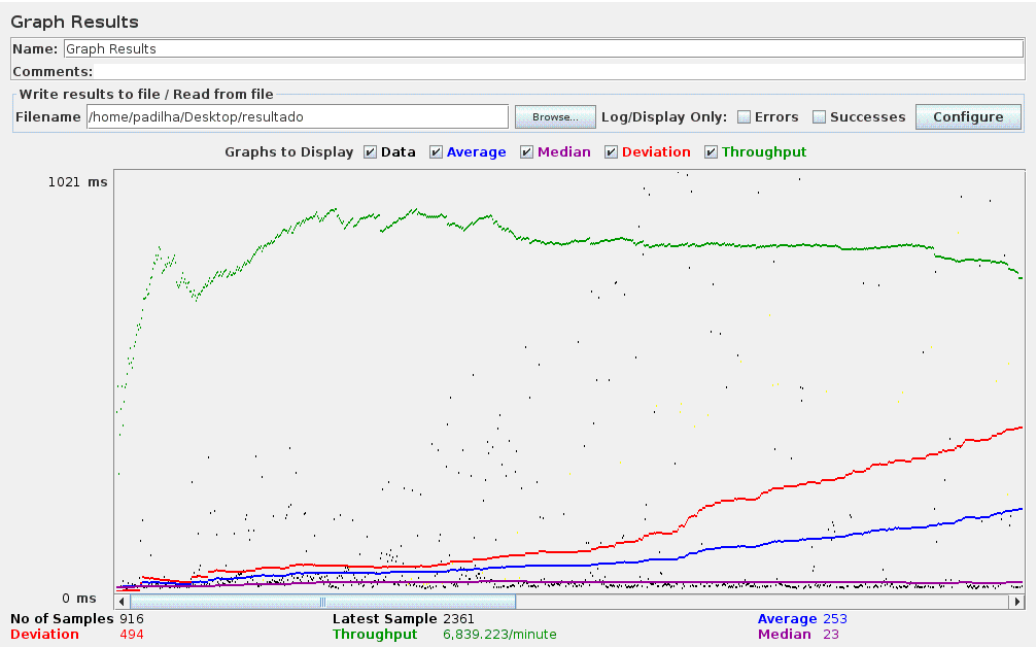


(b) MS SQL Server

Figura 4.2: Gráfico gerado pelo JMeter: 30 usuários

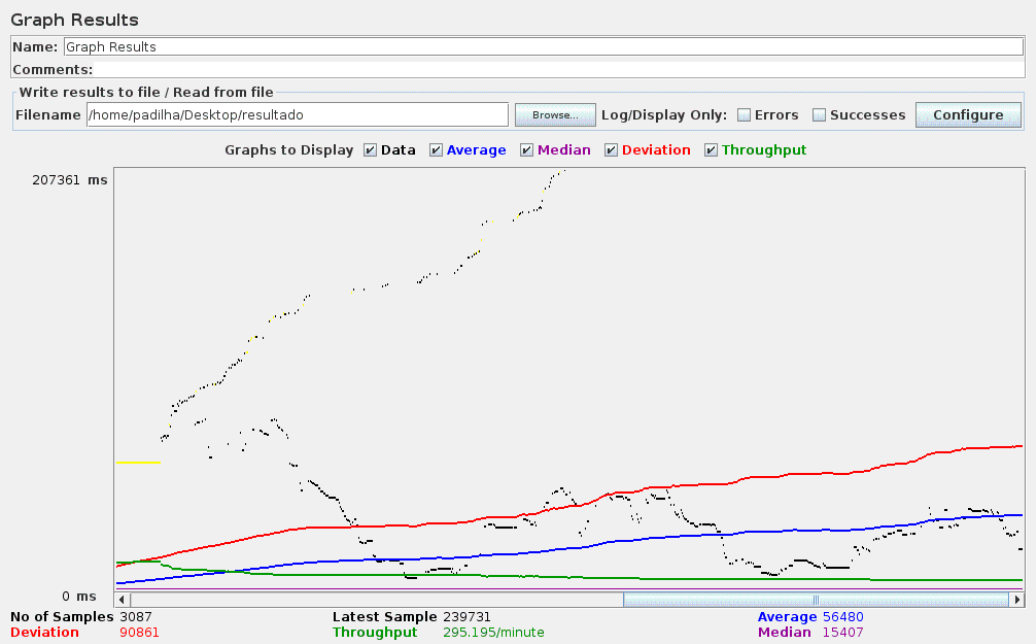


(a) Sybase ASE

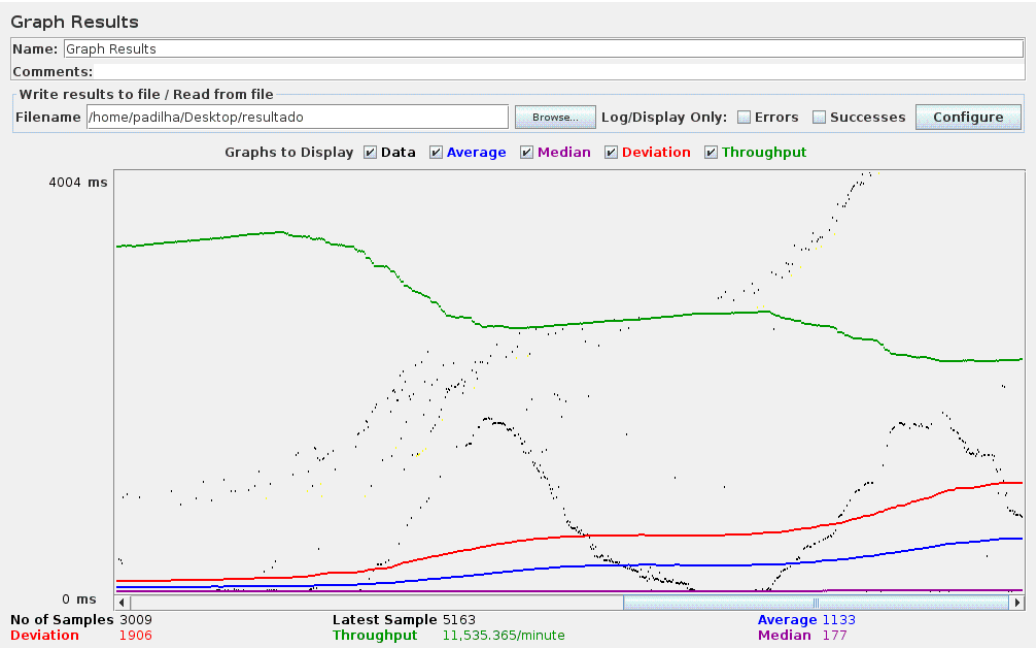


(b) MS SQL Server

Figura 4.3: Gráfico gerado pelo JMeter: 90 usuários



(a) Sybase ASE



(b) MS SQL Server

Figura 4.4: Gráfico gerado pelo JMeter: 360 usuários

A linha verde nos gráficos indica a vazão de dados do servidor em transa-

ções por minuto, enquanto que as linhas vermelha, azul e lilás correspondem respectivamente ao desvio padrão, média e mediana do tempo de resposta das requisições.

Analisando o tempo médio aproximado de resposta das requisições, tem-se que:

- 30 usuários 4.2: 7,7s em Sybase e 80ms em MS SQL Server;
- 90 usuários 4.3: 19,7s em Sybase e 253ms em MS SQL Server;
- 30 usuários 4.4: 56,5s em Sybase e 1100ms em MS SQL Server;

4.2.3 Conclusão dos testes

Ao utilizar o banco de dados Sybase ASE no DataUSP-PósGrad não foi possível avaliar a escalabilidade dos Serviços Web REStful, uma vez que o Servidor de Recursos passou a maior parte do tempo ocioso, esperando dados do banco. Porém, ao utilizar-se o banco de dados Microsoft SQL Server que, apesar de na USP ser um banco de testes executado em uma máquina virtual, foi centenas de vezes mais rápido do que o Sybase ASE, fazendo com que o Servidor de Recursos esgotasse o processamento e também a memória ram da máquina onde foi executado nos testes.

Não apenas o tempo de resposta das requisições foi consideravelmente menor no Microsoft SQL Server, como sua vazão de dados foi milhares de vezes maior, aumentando conforme o número de usuários simultâneos. O Sybase ASE, do contrário, manteve sua vazão ao longo de todos os testes, saturando sua capacidade com apenas 30 usuários simultâneos.

Apenas a título de curiosidade, foi feito um teste com dois mil usuários simultâneos (um pouco mais do que todos os usuários com acesso ao sistema). Com o Microsoft SQL Server o tempo médio de resposta do Servidor de Recursos foi de 5,2s. Ao utilizar o Sybase ASE não foi possível completar o teste pois, após alguns minutos de espera, o banco de dados repentinamente encerrava a conexão.

O uso do banco de dados Sybase ASE no DataUSP-PósGrad acarreta sérios problemas de desempenho em um cenário de alta demanda por seus serviços. Mesmo assim é o banco padrão de praticamente todos os outros sistemas administrativos da USP.

Capítulo 5

Conclusões

Para se compreender a situação atual de uma instituição do tamanho da USP, é fundamental fazer uma análise abrangente e precisa dos dados acumulados ao longo dos anos. É nesse contexto que surgiu o DataUSP-PósGrad como sistema analítico da pós-graduação da Universidade de São Paulo.

Os dois principais objetivos desse trabalho foram a criação de um Data Warehouse e um sistema baseado em Serviços Web para gerar relatórios sob demanda referentes aos dados da pós-graduação da USP.

O Data Warehouse foi construído com dados de fontes internas, no caso dos dados dos programas, áreas e pessoas, e também de fontes externas, para se obter dados da produção acadêmica dos docentes e seu impacto na comunidade científica. As fontes internas são os dados coletados pelos demais sistemas administrativos da USP, por exemplo o sistema Janus, enquanto que as principais fontes externas são as páginas web do Google Scholar e também do Scopus.

Seguindo os conceitos da arquitetura REST, o sistema foi dividido em duas componentes: o Servidor de Recursos e a Interface Web. O Servidor de Recursos, que é propriamente o Serviço Web, foi implementado em Java por meio da biblioteca Jersey. Já a Interface Web, responsável por consumir os dados do Servidor de Recursos e construir os relatórios, foi implementada em JavaScript, por meio das bibliotecas JQuery e Fusion Charts, e HTML5. Além disso um sistema auxiliar escrito em Python, o Web Crawler, foi construído para obter informações da produção intelectual dos docentes da USP nas fontes de dados externas, além de integrá-las ao Data Warehouse.

Embora o DataUSP-PósGrad utilize o banco de dados Sybase ASE, como

praticamente todos os sistemas da USP, utilizar uma arquitetura orientada a serviços provou-se bastante adequada para o ambiente USP. Desde que não haja um gargalo no banco de dados, esse modelo arquitetural garante alta escalabilidade de recursos computacionais e na implementação de novas funcionalidades. Facilita a comunicação e a integração com outros futuros Serviços Web e, ainda, definine um novo paradigma para se construir sistemas administrativos dentro da Universidade.

Os dados fornecidos pelo DataUSP-PósGrad são essenciais para se conhecer a evolução dos programas da pós-graduação, assim como para detectar tendências e manter o padrão de excelência dos cursos. Conseguir esses dados em tempo real é indispensável para agilizar a tomada de decisões e corrigir eventuais problemas, ou ainda entender alguma anomalia.

Com o sucesso do DataUSP-PósGrad a tendência é que apareçam novos sistemas na USP orientados a serviços, e até mesmo a modernização de sistemas já existentes, aumentando o foco nos dados e de como eles representam o estado dos processos administrativos.

Referências Bibliográficas

[1] O K Takai, I C Italiano e J E Ferreira,
Introdução a Banco de Dados, cp.10 - Data Warehouse, Uma visão geral
http://www.ime.usp.br/~jef/apostila.pdf

[2] C Adamson, M Venerable. Data Warehouse Design Solutions, John Wiley & Sons, 1998.

[3] M Golfarelli, D Maio, S Rizzi. "Conceptual Design of Data Warehouses from E/R Schemes", Proceedings of the Hawaii International Conference on System Sciences, Kona, Hawaii, USA, 1998

[4] Vincent Rainardi, Building a Data Warehouse: With Examples in SQL Server, *Chapter 8: Populating the Data Warehouse*,
http://books.google.com.br/books?id=KEW_AUD6GegC

[5] E. R. Elmasri and S. Navathe and. Fundamentals of Database Systems. Editora Addison Wesley Pub, 2001.

[6] R. Ramakrishnan and J. Gehrke. Database Management Systems. 3a. Edição. McGraw-Hill, 2003.

[7] Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures, 2000
http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[8] Snehal Mumbaikar, Puja Padiya. Web Services Based On SOAP and REST Principles, International Journal of Scientific and Research Publications, Volume 3, Issue 5, May 2013, *ISSN 2250-3153* http://www.ijsrp.org/research-paper-0513/ijsrp-p17115.pdf

[9] Leonard Richardson e Sam Ruby, RESTful Web Services (O'Relly), 2008
http://books.google.com.br/books?id=XUaErakHsoAC

[10] Bill Burke, RESTful Java with JAX-RS (O'Relly), 2010
http://books.google.com.br/books?id=_jQtCL5_vAcC

[11] Shriram Krishnamurthi,
Programming Languages: Application and Interpretation (Second Edition), 2012, cp. 15.2.7 - Type Soundness

[12] Shriram Krishnamurthi,
Programming Languages: Application and Interpretation (Second Edition), 2012, cp.10 - Objects

[13] Jesse James Garrett, Ajax: A New Approach to Web Applications, 2005
https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php

[14] Vladislav Shkapenyuk and Torsten Suel, Design and implementation of a high-performance distributed web crawler, Polytechnic University Brooklyn, NY 2002 http://cis.poly.edu/~suel/papers/crawl.pdf

[15] J.E. Hirsch, An index to quantify an individual's scientific research output, 2005
http://arxiv.org/abs/physics/0508025

[16] J. P. Mena-Chalco e R. M. Cesar-Jr. scriptLattes: An open-source knowledge extraction system from the Lattes platform. Journal of the Brazilian Computer Society, vol. 15, n. 4, páginas 31–39, 2009.
http://scriptlattes.sourceforge.net/

Apêndice A

O trabalho e o curso

Nesse capítulo irei descrever minha experiência pessoal com o projeto DataUSP-PósGrad, descrevendo as dificuldades encontradas , a relevância do conteúdo adquirido no curso para a realização do trabalho e os trabalhos futuros.

A.1 Desafios

O projeto DataUSP-PósGrad iniciou-se em janeiro de 2013 quando o Prof. Dr. João Eduardo Ferreira (também conhecido por Jef) convidou-me para fazer um estágio na Pró-Reitoria de pós-graduação da USP, onde ele é o assistente de informática.

O grande desfio inicial do projeto foi construir o *Data Warehouse* 2.1,entender os dados e seus relacionamentos. Foi a primeira dificuldade encontrada.

Tradicionalmente os softwares desenvolvidos na USP são escritos em Java, que embora seja uma linguagem bastante difundida e utilizada, eu não a conhecia profundamente. Além disso tive que aprender em detalhes o funcionamento de um *Serviço Web* e a sua implementação em Java (essa etapa custou o mês de fevereiro praticamente inteiro).

Com o esqueleto do projeto pronto e funcionando com algum relatórios, chegou o momento de integrá-lo ao sistema de login único da USP, onde uma única senha é utilizada para acessar todos os sistemas a que um usuário tenha acesso. Devido à arquitetura cliente servidor do DataUSP-PósGrad, apenas a interface web teve que ser integrada, o que já deu bastante trabalho mesmo

com o suporte do Departamento de Informática da USP. Para fazer autenticação no servidor tive que estudar a fundo o funcionamento do Jersey 2.3 e o funcionamento de suas classes de filtro.

Para colocar o sistema em produção foi preciso configurar o ambiente juntamente com a equipe do Departamento de Informática da USP, pois a distribuição de Linux usada por eles (no caso o Red Hat) não atendia a todas as dependências do projeto, a maioria por causa dos scripts python (Web Crawler) dos relatórios de citações.

Depois que a primeira versão do DataUSP-PósGrad foi ao ar no dia 5 de junho de 2013, o Prof. Jef teve a ideia de fazer um banco de dados de citações 3.5 da produção intelectual dos pesquisadores da USP. Para isso seria preciso obter os dados do Google Scholar e também do Scopus. Isso seria feito com a técnica de *Web Crawling*.

Obter os dados do Google parecia impossível a princípio, já que aparecia um *captha*, imagem com caracteres a serem digitados, em um intervalo curto de requisições (precisava obter os dados de mais de 5000 pesquisadores), e não pretendíamos violar esse sistema de verificação. Isso se resolveu quando o Jef entrou em contato com o Google, que muito atenciosamente disponibilizou uma pagina com todos os pesquisadores da USP cadastrados, e assim não precisaríamos mais buscar os nomes dos docentes individualmente, embora o problema de associar os nomes encontrados aos nomes no banco de dados persistisse.

A navegação no site do Scopus foi relativamente mais complicada. O site gera identificadores temporários para cada requisição e, para navegar de forma autônoma, foi preciso fazer a engenharia reversa do processo (foi o mês de julho e parte de agosto). Depois de entender o mecanismo das verificações foi feita a primeira tentativa de se obter os dados para todos os mais de 5000 pesquisadores da USP. Como era necessário dar um intervalo aleatório de tempo (de 1 a 3 segundos) entre cada requisição (para se obter os dados de cada pesquisador são 7 requisições), o processo levou cerca de 15 horas para completar. Já pensando em como manter essa base de dados atualizada, o Web Crawler foi paralelizado e executado novamente, mas agora com 100 threads. Resultado: obtive os mesmos dados em menos de uma hora! Mas como toda ação gera uma reação, o Scopus bloqueou o acesso para o endereço IP da Pró-Reitoria (e também um dos endereços IP do IME) e a Capes enviou um ofício pedindo esclarecimentos (felizmente o caso foi resolvido sem prejuízo ao DataUSP-PósGrad)

A.2 disciplinas relevantes

As algumas disciplinas do bacharelado em ciência da computação foram fundamentais para que eu pudesse compreender o construir o DataUSP-PósGrad (mesmo algumas que não foram citadas contribuíram de forma indireta, por exemplo as matérias de base que ajudam a desenvolver o raciocínio lógico e introduzem os princípios da computação). São elas:

A.2.1 Introdução a Bancos de dados

Aqui eu aprendi o funcionamento de um banco de dados relacional e como usar a linguagem SQL para interagir com o mesmo. Apesar de ser uma matéria introdutória achei interessante o foco voltado a lógica e não a linguagem de consulta em si. Não esperava menos do curso.

A.2.2 Engenharia de Software

Essa disciplina me pareceu um tanto superficial a principio mas foi aqui que tive o primeiro contato com a linguagem de programação Java. Durante o meu estágio na Pró-Reitoria pude perceber o quão importante é saber trabalhar em equipe e aplicar os conceitos e as boas práticas de programação adquiridos na disciplina.

A.2.3 Estrutura de Dados

No projeto inúmeras vezes precisei associar dados oriundos de bases distintas antes sumarizá-los. O uso de árvores rubro-negras (TreeMap em Java) para inserir e buscar os dados da maneira mais eficiente o possível contribuiu para a eficiência de alguns métodos. Outras estruturas como os vetores associativos (tabelas de Hash) também foram muito utilizadas no projeto.

A.2.4 Análise de Algoritmos

Em Java muitas estruturas de dados e algoritmos clássicos já estão implementadas em forma de bibliotecas. Saber compará-los assintoticamente foi decisivo para buscar o melhor desempenho possível do sistema.

A.3 Trabalhos futuros

Pretendo continuar atuando na área de Web Services estudando mais a fundo não só a tecnologia mas também os processos de modelagem de negócio, como as redes de Petri, álgebras de processos e também a abordagem do Prof. Jef chamada *WED-flow*[1].

A.4 Agradecimentos

Gostaria de agradecer primeiramente ao Prof. Dr. João Eduardo Ferreira pela oportunidade de participar do projeto DataUSP-PósGrad, e também por sua paciência e por acreditar em mim. Agradeço também a toda a equipe de informática da Pró-reitoria de Pós-graduação, sob orientação do Marino Hilário Catarino e com a ajuda do Felipe Augusto Araujo Dias, Eduardo Dias Filho e do Rafael G. Rossi na implementação do sistema.

Contamos também com amplo suporte do Departamento de Informática da USP (agora conhecido por DTI) sob a coordenação do Prof. Dr. Luiz Natal Rossi e Luis Carlos Moreira Gomes, e gerência de Silvio Fernandes de Paula.

Finalmente agradeço ao Pró-reitor de Pós-graduação Prof. Dr. Vahan Agopyan pela oportunidade e apoio ao desenvolvimento do DataUSP-PósGrad.

[1] WED-flow - `http://data.ime.usp.br/wedflow`