UNIVERSIDADE ESTADUAL PAULISTA

"JÚLIO DE MESQUITA FILHO"

Faculdade de Ciências - Bauru

Bacharelado em Ciência da Computação

Thiago Gabriel Borges da Silva

Classificação de *Workflows* Baseados em *Web Services*

UNESP

2012

Thiago Gabriel Borges da Silva

Classificação de *Workflows* Baseados em *Web Services*

Orientador: Prof. Dr. Renê Pegoraro

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, UNESP, *campus* de Bauru, como parte do Trabalho de Conclusão de Curso.

UNESP

2012

Thiago Gabriel Borges da Silva

Classificação de *Workflows* Baseados em *Web Services*

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, UNESP, *campus* de Bauru, como parte do Trabalho de Conclusão de Curso.

BANCA EXAMINADORA

Prof. Dr. Renê Pegoraro
Professor Doutor
DCo – FC - UNESP – Bauru
Orientador

Profª. Drª. Simone das Graças Domingues Prado
Professora Doutora
DCo – FC - UNESP – Bauru

Prof. Dr. Wilson Massashiro Yonezawa
Professor Doutor
DCo – FC - UNESP – Bauru

Bauru, 29 de Outubro de 2012.

Dedico o trabalho à minha família, por me apoiar incondicionalmente e pela confiança creditada em todos os momentos.

AGRADECIMENTOS

Agradeço aos professores do curso de Bacharelado em Ciência da Computação da UNESP Bauru pelos ensinamentos e conselhos.

Aos meus colegas e companheiros de turma pela amizade e ajuda.

Ao Professor Renê Pegoraro pela orientação, pelas aulas e pelo apoio sempre que foi necessário.

Especial agradecimento a Camila Martins Marchetti, pela força e paciência nos momentos de dificuldade.

RESUMO

Nas últimas décadas ocorreram mudanças na comunicação, dentro e entre empresas, facilitadas por tecnologias como lojas virtuais, Internet, sistemas empresariais integrados e reuniões à distância e houve um rápido avanço na tecnologia de redes, o que mudou a maneira com que os negócios são feitos. Uma forma padronizada de oferecer serviços através da internet é utilizando *web services*. *Web services* são um tipo de chamada remota de procedimento e geralmente são usados para integrar sistemas, independente da linguagem, tanto do cliente quanto do servidor. É comum utilizar-se diversos *web services* executados em sequência para realizar um processo de negócio. A este tipo de processo, dá-se o nome de *workflow*. Dessa forma, os *web services* são os componentes primordiais dos *workflows*. Uma ferramenta que ofereça um modo de visualizar o comportamento de um *workflow* pode auxiliar o administrador e se faz necessária. O presente trabalho apresenta o desenvolvimento de uma ferramenta que permita ao administrador classificar de forma visual os serviços componentes e avaliar sua importância no desempenho final de um *workflow*. Como prova de conceito foram utilizados diversos servidores e computadores virtuais onde cada computador recebeu um conjunto de *web services*. Um *proxy* foi adicionado entre cada chamada dos *workflows* coletando informações relevantes e armazenando-as em um banco de dados para posterior análise. A analise foi baseada em parâmetros de Qualidade de Serviço.

PALAVRAS-CHAVE: *Web Service*, *Workflows*, Qualidade De Serviço, *Proxy*

ABSTRACT

Over the last decades changes have occurred in communication within and between enterprises, made easier by technologies such as E-commerce, Internet, ERP systems and remote meetings and there was a rapid progress in network technology, which has changed the way business is done. A standardized way to offer services over the internet is using web services. Web services are a kind of remote procedure call and are generally used to integrate systems, independent of language, both client and server. It is common to use several web services run in sequence to perform a business process. To this type of process, gives the name of workflow. Thus, Web services are the primary components of workflows. A tool that provides a way of visualizing the behavior of a workflow can assist the administrator and is required. The present work presents the development of a tool that allows the administrator to classify visually services components and evaluate their importance in the final performance of a workflow. As proof of concept we used several virtual servers and computers where each computer has received a set of web services. A proxy was added between each call of workflows collecting relevant information and storing them in a database for later analysis. The analysis was based on Quality of Service parameters.

KEYWORDS: *Web Service*, *Web Services*, Quality of Service, *Proxy*

LISTA DE FIGURAS

LISTA DE SIGLAS

BPEL	Business Process Execution Language
ERP	Enterprise Resource Planning
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IDE	Integrated Development Environment
J2EE	Java 2 Enterprise Edition
JVM	Java Virtual Machine
LTS	Long Term Support
ODE	Orchestration Director Engine
ORM	Mapeamento objeto-relacional
QoS	Quality of Service
SOA	Arquitetura Orientada a Serviços
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UUID	Universally Unique Identifier
W3C	World Wide Web Consortium
WSDL	*Web services* Description Language
XML	eXtensible Markup Language
XSD	XML Schema Definition

SUMÁRIO

1. INTRODUÇÃO

Houve um rápido avanço na tecnologia de redes nas últimas duas décadas, o que mudou a maneira com que os negócios são feitos através da Internet. Ocorreram mudanças na comunicação, dentro e entre empresas, facilitadas por tecnologias como lojas virtuais, Internet, sistemas empresariais integrados e reuniões à distância (ATKINSON et al., 1997). Junto a isso, cresce a quantidade de dispositivos móveis para acessar tais serviços. O acesso pode acontecer a partir da interface *web* comum ou de aplicativos nativos otimizados para cada plataforma. Isso traz inconveniências devido a falta de uniformização entre as várias plataformas envolvidas. Para uniformizar alguns aspectos os *web services* são utilizados.

Web services são softwares disponíveis na Internet que usa protocolos padrões definidos para executar funções ou processos de negócios (FENSEL & BUSSLER, 2002). *Web services* também podem ser vistos como uma forma de chamada remota de funções e geralmente são usados para integrar sistemas independente da linguagem do cliente e do servidor. *Web services* simplificam a interoperabilidade, e consequentemente, a integração de aplicativos. Eles fornecem um meio para envolver as aplicações existentes para que os desenvolvedores possam acessá-los através de linguagens e protocolos padrão.

A padronização simplifica a interoperabilidade. Em vez de interagir com sistemas heterogêneos, cada um com seu protocolo de transporte próprio, formato dos dados, protocolo de interação, e similares, as aplicações podem interagir com sistemas que são mais homogêneos. Uma abordagem padronizada ajuda a reduzir os custos de desenvolvimento e manutenção de sistemas integrados. Mais especificamente, padrões *web services* apoiam a adoção de baixo acoplamento e interações descentralizadas (NEZHAD et al., 2006).

Os *web services* são componentes importantes nos processos de negócio. Esses processos são definidos por sequências lógicas de atividades realizadas por indivíduos ou por *web services*. A automatização desses processos recebem o nome de *workflows*. Um *workflow* pode ser tratado como um simples *web service*, dependendo da perspectiva, visto que para ser um *web service* basta invocar um serviço e receber uma resposta. Então, um *workflow* também pode ser chamado por outros *workflows* gerando aplicações complexas. Uma linguagem utilizada para

modelar *workflows* de *web services* é Linguagem de Execução de Processo de Negócio (Business Process Execution Language - BPEL). Ela foi criada por empresas como Microsoft, IBM e BEA para definir e padronizar o modelo de negócios. Com ela, pode-se estruturar os processos de negócios nos *web services* e mostrar de forma visual todo o processo, semelhante à modelagem pela Linguagem Unificada de Modelagem (Unified Modeling Language - UML).

As melhorias nos workflows ocorrem com o desenho de novos processos, otimização e reengenharia de fluxo de trabalho. Um administrador, preocupado com otimização, não consegue avaliar com precisão os pontos em que ocorrem falhas ou atrasam a execução de seu *workflow*, logo, ferramentas de qualquer tipo que possam ajudá-lo em suas tarefas têm espaço e importância. Em um *workflow*, se uma chamada de serviço estiver muito lenta, então, diante desta situação, algumas decisões podem ser tomadas, tais como contatar o responsável pelo serviço e pedir melhorias ou então deixar de utilizar o serviço e encontrar outro que se adeque melhor às necessidades.

Um *workflow* organiza uma sequência de chamadas a *web services*. Estas operações são definidas estaticamente. Identificar seu comportamento dinâmico durante seu funcionamento de forma transparente é um desafio. Uma ferramenta que ofereça um modo de visualizar o comportamento de um *workflow* pode auxiliar o administrador e se faz necessária.

A troca de mensagens entre os *workflows* e *web services* ocorre através da rede. Uma forma para capturar o comportamento dinâmico é interceptar estas mensagens e medir os seus tempos. Esta intercepção pode ser criada através de um *proxy* modificado alojado no caminho dos *web services*.

A comunicação, orquestrada por um *workflow,* ocorre entre múltiplos serviços os quais também se comunicam com outros serviços. Para se saber quais serviços estão sendo chamados pelo *workflow* é importante conhecer qual a instância deste está sendo analisada. Apesar do *proxy* capturar algumas informações importantes do *workflow*, não existe informação disponível nos pacotes que indique a qual instância cada requisição/resposta pertence, assim, se faz necessária uma pequena modificação, que pode ser realizada automaticamente, nos *workflows* dos quais se deseja rastrear o comportamento dinâmico. No presente estudo, houve a necessidade

de incluir a chamada de um serviço que retorna um identificador único, que representa a instância, a cada vez que um novo serviço é chamado, este identificador é então capturado pelo *proxy* junto aos tempos de execução dos pacotes.

Este trabalho apresenta uma técnica de monitoramento dos tempos de execução do *workflow* e dos *web services* componentes. Uma forma de visualização é proposta e poderá ser utilizada para auxiliar um administrador do *workflow* na verificação do seu comportamento.

2. OBJETIVOS DO TRABALHO

2.1 Objetivo Geral

Desenvolver uma ferramenta que permita ao administrador classificar de forma visual os serviços componentes e avaliar sua importância no desempenho final de um *workflow*.

2.2 Objetivos Específicos

Desenvolver um ambiente para a validação da ferramenta contendo máquinas virtuais onde os *web services* e os *workflows* estejam alocados; aplicar um método de medição de tempos dos serviços; automatizar o processo de modificação dos arquivos que compõem a BPEL para manter as informações de instâncias dos *workflows*; desenvolver um aplicativo para exibir uma representação gráfica do *workflow* ao administrador.

3. METODOLOGIA

Para desenvolver este trabalho, um conjunto de computadores servidores foi necessário. Estes computadores servem para hospedar os *workflows* e os *web services* e devem estar conectados através de uma rede. A fim de simular um ambiente real, os serviços foram dispostos em servidores diferentes. A utilização de uma rede com computadores reais traz diversas dificuldades, entre elas: espaço físico, custos e manutenção. Para diminuir estas dificuldades, como se trata de um sistema onde várias máquinas estão envolvidas, o desenvolvimento ocorreu com servidores e computadores virtuais onde cada computador recebeu uma parte dos serviços ou *workflows*. Os *web services* e *workflows* foram hospedados em máquinas virtuais diferentes, para simular um ambiente real. O sistema operacional das máquinas virtuais é o Linux, que é utilizado por oferecer serviços gratuitos e se adaptar ao problema proposto. Estes servidores estão instalados em máquinas virtuais. A escolha por máquinas virtuais foi feita pois nos traz os seguintes benefícios apontados na documentação do *Virtual Box* (Oracle VM Virtualbox, 2012):

• A possibilidade de executar vários sistemas operacionais no mesmo hardware ao mesmo tempo.

• Fácil instalação de software, com isso pode-se ter soluções prontas como servidores pré-configurados, prontos para serem replicadas várias vezes.

• Possibilidade de teste e recuperação de falhas que é feita com snapshots, que salva o estado da maquina em diversos momentos e podem ser recuperados caso haja a necessidade.

• Máquinas virtuais também permitem a otimização da infraestrutura, que muitas vezes desperdiça recursos como energia e potencial executando apenas um sistema.

Outra vantagem é a flexibilidade de configuração da topologia de rede, que simula diversas possibilidades com poucas mudanças nas configurações. Caso essas mudanças fossem feitas em um ambiente físico, maiores alterações poderiam ser necessárias, como a realocação dos fios e necessidade de outros *hardwares*, como *switch*.

Todas as máquinas virtuais estão configuradas no modo de rede *Host Only*, o que quer dizer que todas as máquinas hospedeiras e servidores estão conectadas na mesma rede através de uma interface de rede previamente configurada no *Virtual Box*. Porém, a desvantagem de estar no modo *Host Only* é não ter acesso às redes externas, como a Internet, nesta interface. Segundo o documentação do Virtual Box (Oracle VM Virtualbox, 2012), *Host Only* são muito utilizadas em aplicações onde diversas máquinas virtuais são desenvolvidas para cooperar entre si, por exemplo, rodando um servidor *web* em uma e um banco de dados em outra.

Nos experimentos foram usadas seis máquinas virtuais, configuradas como servidores, sendo o sistema operacional de todos eles o Ubuntu Server 12.04 LTS. As máquinas virtuais que utilizam os nomes:

• SHA – Possui um servidor *proxy* desenvolvido em Java, usado para coletar as informações das chamadas SOAP, como utilizado na arquitetura de auto reparação (PEGORARO et al., 2008). Esses dados são armazenados em um servidor MySQL.

• *Arithmetics*, *Functions* e *Trigonometry* – Usados para hospedar os serviços básicos para criação de *workflows*. Esses *web services* são instalados sobre a plataforma Axis2.

• *Workflows1* e *Workflows2* – São usadas para hospedar os *workflows*, que também podem ser vistos como serviços. Tais *workflows* estão sendo servidos sobre a plataforma *Apache* ODE.

Alguns *workflows* foram criados para servir com prova na visualização da conexão entre os serviços. Foi utilizado um *proxy* entre cada chamada de cada serviço para coleta e posterior análise dos dados (PEGORARO et al., 2008). Esses dados são compostos por informações das chamadas e pelos tempos de execução de cada *web service*. Halima et al. (2008a) propõe o uso de quatro medidas, sendo duas no cliente e duas no servidor, para medir os tempos gastos na rede. Coletar quatro medidas implicaria na inserção de *proxies* juntos a cada serviço e aos seus clientes. Na arquitetura apresentada nesta monografia usou-se apenas duas medidas de tempo. O tempo de requisição e o tempo de resposta.

3.1 Representação Gráfica

Os dados coletados foram lidos e transformados em informações relevantes. Estas informações foram trabalhadas para exibir as chamadas em ordem de execução, com o tempo médio que elas levaram para concluir. Para contextualizar, esta representação gráfica é semelhante ao diagrama representado na Figura 1.

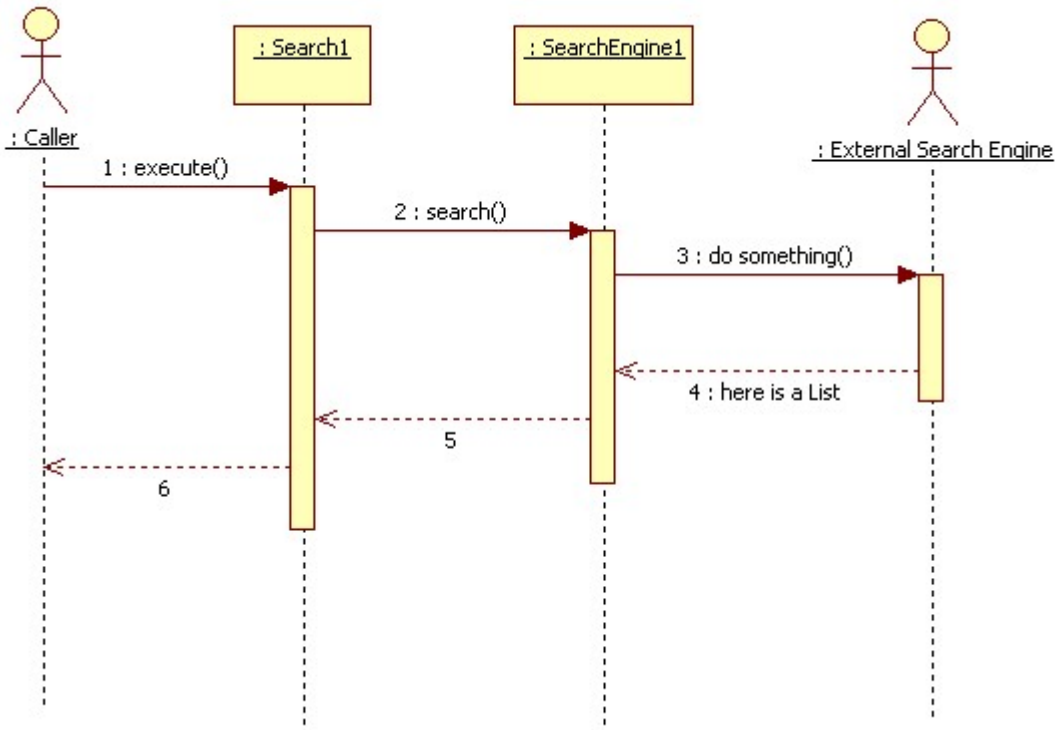

Figura 1: Exemplo de Diagrama de Sequência (LeGros B., 2012)

O diagrama da Figura 1 mostra uma representação semelhante à que será apresentada neste trabalho. O intuito desta figura é apenas traçar um paralelo entre um diagrama de sequência padrão UML e o diagrama desenvolvido como representação gráfica das chamadas dos *web services* participantes do *workflow.* Na Figura 1, as linhas tracejadas representam onde as operações de um serviço serão executadas. Sobre estas linhas tracejadas, são colocados retângulos verticais indicando o tempo que a operação ocupa durante sua execução. As setas horizontais representam as chamadas às funções. As setas são rotuladas com o nome do método e seus parâmetros, e o retorno das mensagens é representado por uma seta tracejada. Baseado nesta representação UML, desenvolveu-se uma representação usada. Nesta representação, as linhas verticais tracejadas

representam a execução de apenas uma operação, sendo que as execuções propriamente ditas continuam sendo representadas pelos retângulos verticais; os nomes das operações aparecem nos retângulos superiores junto ao nome do serviço e finalmente algumas representações de medidas estatísticas de tempo foram acrescentadas. Um melhor detalhamento da representação gráfica desenvolvida é apresentado na seção 5.4.

3.2 Instrumentalização e automatização

Para serem realizadas as capturas das informações de cada instância dos *workflows,* algumas alterações nos arquivos do *workflow* foram feitas, pois houve a necessidade de adicionar um identificador único para cada instância do *workflow*, caso contrário não seria possível rastrear qual a ordem de chamada dos serviços. Não existem informações de identificação das instâncias disponíveis nos pacotes de requisição e resposta que trafegam pela rede. Para gerar um identificador único, um *web service* extra*,* chamado de *instance*, foi criado. O identificador é obtido deste *web service* que retorna um Identificador Único Universal (*Universally Unique Identifier* - UUID). Este identificador é usado como dado em todas as requisições aos *web services* feitas pela instância do *workflow* e é registrado pelo *proxy*. As imagens da Figura 2 e Figura 3 mostram a diferença entre um *workflow* com *Instance* e sem *Instance*. Este *workflow* possui o serviço F3 chamando o F2 e foi criado para exemplificar o uso do *Instance*.

A Figura 2 apresenta dois *workflows*. O F2 calcula (x+y)/(x*y), e F3 calcula $\sqrt{\mathrm{F2}}$. As operações $\sqrt{\ }$ e aritméticas são *web services* que representam suas respectivas tarefas remotamente. Neste ambiente não existem informações das instancias executando, o que impossibilita a identificação da origem de uma chamada a um *web service*. A solução é a inserção do *web service* Instance (Figura 3) que gera um identificador único para cada instância de um *workflow*. Este identificador ficará disponível em todas as chamadas feitas pelo workflow. As alterações para a inclusão deste identificador no *workflow* pode ser feita manualmente, porém, por ser composta por diversos passos, torna-se impraticável. Para simplificar esta inclusão, propõe-se neste trabalho que elas sejam realizadas automaticamente.

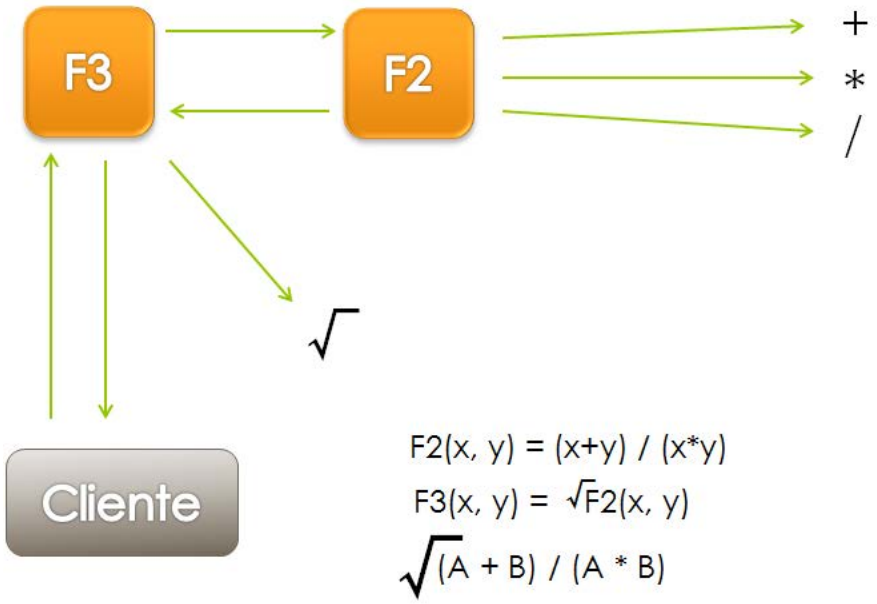

Figura 2: Workflow de web services sem a presença do Instance

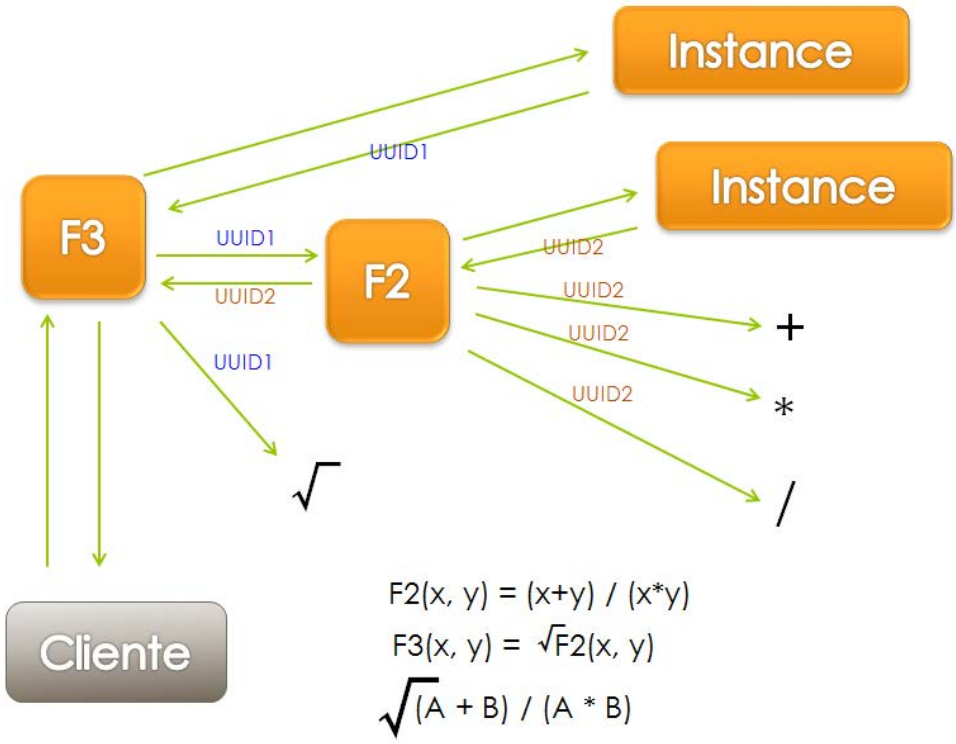

Figura 3: Workflow de web services com a presença do Instance

As modificações foram implementadas na linguagem Java, usando a biblioteca VTD-XML para percorrer as estruturas baseadas em XML dos arquivos WSDL e BPEL, identificar os pontos importantes, e inserir os campos necessários para incluir o *Instance* no *workflow*. A biblioteca VTD-XML foi usada como *parser*

pelo seu desempenho, eficiência, e principalmente pela simplicidade de ser usada por ter suporte a linguagem XPath. Com isso, algoritmos complexos são substituídos por uma sintaxe elegante e de fácil entendimento.

Os pacotes de requisições e de respostas dos *web services* e *workflows* trafegam pela rede formatados no protocolo SOAP*.* Duas abordagens de captura destes pacotes são propostas por Halima et al. (2008b), uma usando o *service container handler* do *Apache Axis2/Java* (2012) e outra usando um *proxy* modificado (PEGORARO et al., 2008). Uma vez que o *proxy* atua na camada HTTP, os *web services* não precisam sofrer alterações para que se possa realizar as medições de QoS. Por outro lado, o mecanismo *service container handler* exige a modificação do servidor Axis. Isso confere maior flexibilidade na abordagem usando proxy para realizar a medição de serviços que não estão no domínio do workflow. A inserção do *service container handler* depende da permissão do provedor do *web service*, o que dificulta a aplicação desta abordagem nestes outros domínios. Desta forma, restringe-se esse trabalho a aplicação do proxy como forma de medição da QoS.

O *proxy*, configurado no Apache ODE, capturou todas as informações relativas às chamadas, como por exemplo os tempos de chamada e de resposta, quem chamou, qual o serviço chamado e onde este serviço estava. Todas essas informações ficam guardadas no banco de dados na mesma máquina, e foram usadas para análise e criação dos gráficos.

4. FUNDAMENTAÇÃO TEÓRICA

Este projeto é constituído por *workflows* de *web services*. Para que um *workflow* possa invocar um *web service*, é necessário que o *workflow* importe os arquivos WSDL de cada *web service*. Estes arquivos descrevem aspectos do *web service*, como por exemplo as funções e parâmetros disponíveis. Para que ocorram as medidas de tempo citadas em 3.2, é necessário modificar os *workflows*. Os *workflows, web services* e arquivos WSDL são baseados na linguagem XML, então se optou por utilizar a biblioteca VTD-XML (VTD-XML, 2012) para processamento e modificação dos arquivos. Esta biblioteca possui suporte à XPath, linguagem utilizada para encontrar e percorrer pontos específicos das estruturas XML.

Após a modificação dos arquivos, será possível acompanhar a execução de cada instância de um workflow, e assim avaliar a qualidade de serviço utilizando um de seus requisitos que é o tempo resposta.

4.1 XML

A XML (Extensible Markup Language), assim como o HTML (Hyper Text Markup Language), é uma linguagem de marcação. A XML é utilizada para transporte e armazenamento de dados, além de ser flexível, podendo conter qualquer *tag*. Essa linguagem é uma recomendação da W3C desde 10/Fev/1998 (XML, 2012). Arquivos XML são estruturados usando elementos e atributos como exemplificado na Figura 4:

```
<elemento atributo=”valor”>
    <elemento1>valor 1</elemento1>
    <elemento2>valor 2</elemento2>
</elemento>
```

Figura 4: Modelo XML

4.1.1 XML Schema

O XML Schema descreve a estrutura de um documento XML, e também pode ser chamado de XSD (XML Schema Definition). Com ele, é possível definir quais elementos e atributos são possíveis, além de ser usado para validar o XML. XML Schema 1.0 foi aprovado como padrão pela W3C em Maio de 2001 e sua segunda edição (1.1) foi publicada em Outubro de 2004 (XML Schema, 2012).

No exemplo da Figura 5, o elemento “password” de um XML Schema deve seguir a expressão regular ”[a-zA-Z0-9]{8}” para ser validado por este XML Schema.

```
<xs:element name="password">
    <xs:simpleType>
      <xs:restriction base="xs:string">
          <xs:pattern value="[a-zA-Z0-9]{8}"/>
      </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Figura 5: Modelo XML Schema (W3Schools XML Schema, 2012)

4.1.2 XPath

A XPath (XML Path Language) é uma linguagem de consulta e navegação de estruturas XML. Essa linguagem, assim como XQuery, é orientada a caminho, ou seja, é necessário que o usuário tenha conhecimento prévio da estrutura que se deseja trabalhar para que ocorra o processamento das informações. Documentos XML possuem uma estrutura hierárquica, que começa do elemento raiz, passando por subelementos que por sua vez podem conter atributos e valores (Chan et al., 2002).

A linguagem XPath trata o elementos XML como árvores iteráveis e oferece uma maneira de especificar e selecionar partes da estrutura. Neste trabalho, apenas os operadores “/” e “//” foram usados. A barra “/”, indica um passo para dentro da estrutura XML e barra dupla “//”, indica um ou mais passos para dentro da árvore XML. Para exemplificar, considera-se o arquivo dados.xml na Figura 6 que é usado para apresentar o conceito e a utilização destes operadores da linguagem XPath.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>

  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Figura 6: Modelo XML para exemplo de XPath (W3Schools XPath, 2012)

Com base nesta estrutura, podemos selecionar:

• Utilizando-se o operador “/” e identificadores conhecidos, a sequência “/bookstore/book/title” representa as seções XML:

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
```

• Para obter a seção com o título do primeiro livro, utiliza-se “/bookstore/book[1]/title”:

```
<title lang="en">Everyday Italian</title>
```

• Com o código “/bookstore/book[price>35]/title”, obtém-se os títulos dos livros com o preço maior que 35. Como neste exemplo não existe nenhum livro com o preço (*price*) maior que 35, nada será retornado.

• Para se obter a lista de autores (*author*), independente da estrutura que se encontra anteriormente na árvore XML, o seletor “//author” é utilizado e retorna:

```
<author>Giada De Laurentiis</author>
<author>J K. Rowling</author>
```

4.2 *Web services*

Os *web services* foram criados para prover interoperabilidade entre uma aplicação web e qualquer tipo de cliente. É utilizado o protocolo SOAP (Simple Object Access Protocol), que é a combinação do HTTP com XML, e por usar um protocolo aberto, pode ser facilmente invocado. Os *web services* são a preferência de implementação da maioria dos desenvolvedores que pretendem integrar o sistema com outros, e isso ocorre devido ao alto nível de padronização requerido pelos protocolos (BECKER et al., 2008).

4.2.1 SOAP

O protocolo SOAP (Simple Object Access Protocol) foi desenvolvido para a troca de mensagens através do XML e HTTP, respectivamente usados para armazenar os dados e para transportar os dados pela Internet. É um protocolo flexível de comunicação e independente de plataforma. Originalmente desenvolvido pela Microsoft, ganhou apoio e suporte da IBM e outras grandes empresas. SOAP define o elemento *Envelope* como estrutura. Nela, ainda é obrigatória a presença do elemento *Body* e opcional a presença do *Header*.

```
<?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>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

Figura 7: Modelo de mensagem SOAP (W3Schools SOAP, 2012)

A estrutura do Header contém informações como expiração, roteamento, autenticação ou qualquer outra definida pelo desenvolvedor. Já o *Body* contém a mensagem em si.

O elemento *Fault*, também opcional, é usado para indicar mensagens de erro. Esse elemento deve aparecer como filho do elemento *Body*, e só pode aparecer uma única vez.

4.2.2 WSDL

A linguagem de descrição de *web services* (*Web Services* Description Language - WSDL) descreve o *web service* e como acessá-lo. Essa descrição é independente da implementação. Também escrito em XML, o WSDL contém os seguintes elementos:

• *types*: É o tipo de dado utilizado pelo *web service*. Pode ser importado um XML Scheme externo.

• *message*: Definição abstrata do que pode ser comunicado. Pode ser comparado ao parâmetro de uma chamada de função em uma linguagem de programação tradicional.

• *portType*: É o atributo mais importante do WSDL. Descreve as operações que podem ser realizadas e as mensagens que estão envolvidas.

• *binding*: Define o formato da mensagem e detalhes do protocolo para cada porta.

Essa estrutura pode ser melhor compreendida com o seguinte exemplo de WSDL:

```
    <message name="getTermRequest">
        <part name="term" type="xs:string"/>
    </message>

    <message name="getTermResponse">
        <part name="value" type="xs:string"/>
    </message>

    <portType name="glossaryTerms">
        <operation name="getTerm">
            <input message="getTermRequest"/>
            <output message="getTermResponse"/>
```

```
        </operation>
    </portType>
```

Figura 8: Parte da estrutura de um WSDL (W3Schools WSDL, 2012)

4.3 BPEL

Os *workflows* podem ser classificados de duas formas: orquestração ou coreografia. Na coreografia (Figura 9), cada serviço é programado para chamar outro serviço diretamente, sem nenhum intermediário.

Na orquestração (Figura 10), um motor coordena as chamadas e respostas dos serviços, ficando a cargo dele ter o conhecimento prévio destes serviços e da sequência lógica destes. A vantagem da orquestração é que os serviços não precisam saber que estão sendo utilizados como intermediários, tendo então apenas a responsabilidade de responder o que foi requisitado. Assim, o motor fica totalmente responsável pelo modelo de negócios.

Neste trabalho utiliza-se orquestração. A forma escolhida de orquestração para o desenvolvimento deste foi a BPEL, pois, segundo Nezhad et al. (2006), a BPEL é a linguagem líder para definição de *workflows*.

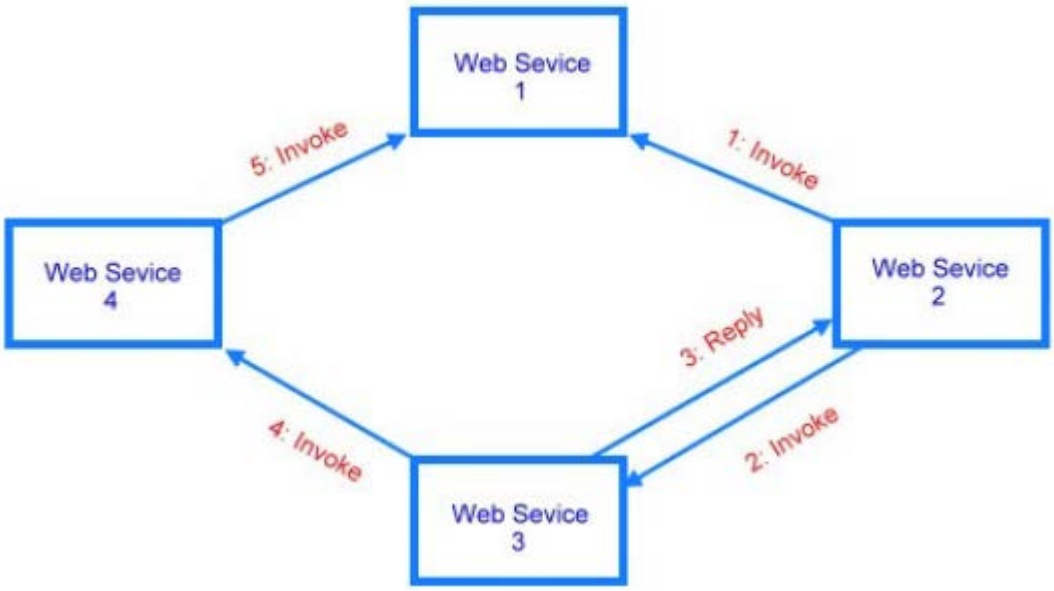

Figura 9: Workflow de web services com coreografia

Figura 10: Workflow de web services com orquestração

A BPEL é uma linguagem para descrever e executar processos de negócios usando *web services*. Essa linguagem possui um conjunto de componentes com ações (*Actions*) para invocar e receber mensagens, controles de fluxo (*Control*) permitindo a construção de estruturas condicionais e laços, além de elementos para tratamentos de falhas (*Faults*). Uma ferramenta utilizada para o desenvolvimento de um *workflow* em BPEL é o BPEL Designer, que pode ser instalado como um *plugin* na IDE (*Integrated Development Environment*) Eclipse.

A Figura 11 apresenta uma tela capturada da IDE Eclipse com o *plugin* BPEL Designer durante a edição de um *workflow.* Este *plugin* é utilizado na criação de *workflows* com a definição visual das sequências de execução dos *web services*. Os componentes disponíveis do *plugin* estão à esquerda e ao centro está o *workflow*. No restante da perspectiva deste *plugin* é possível a manipulação das propriedades e características embutidas no *workflow*.

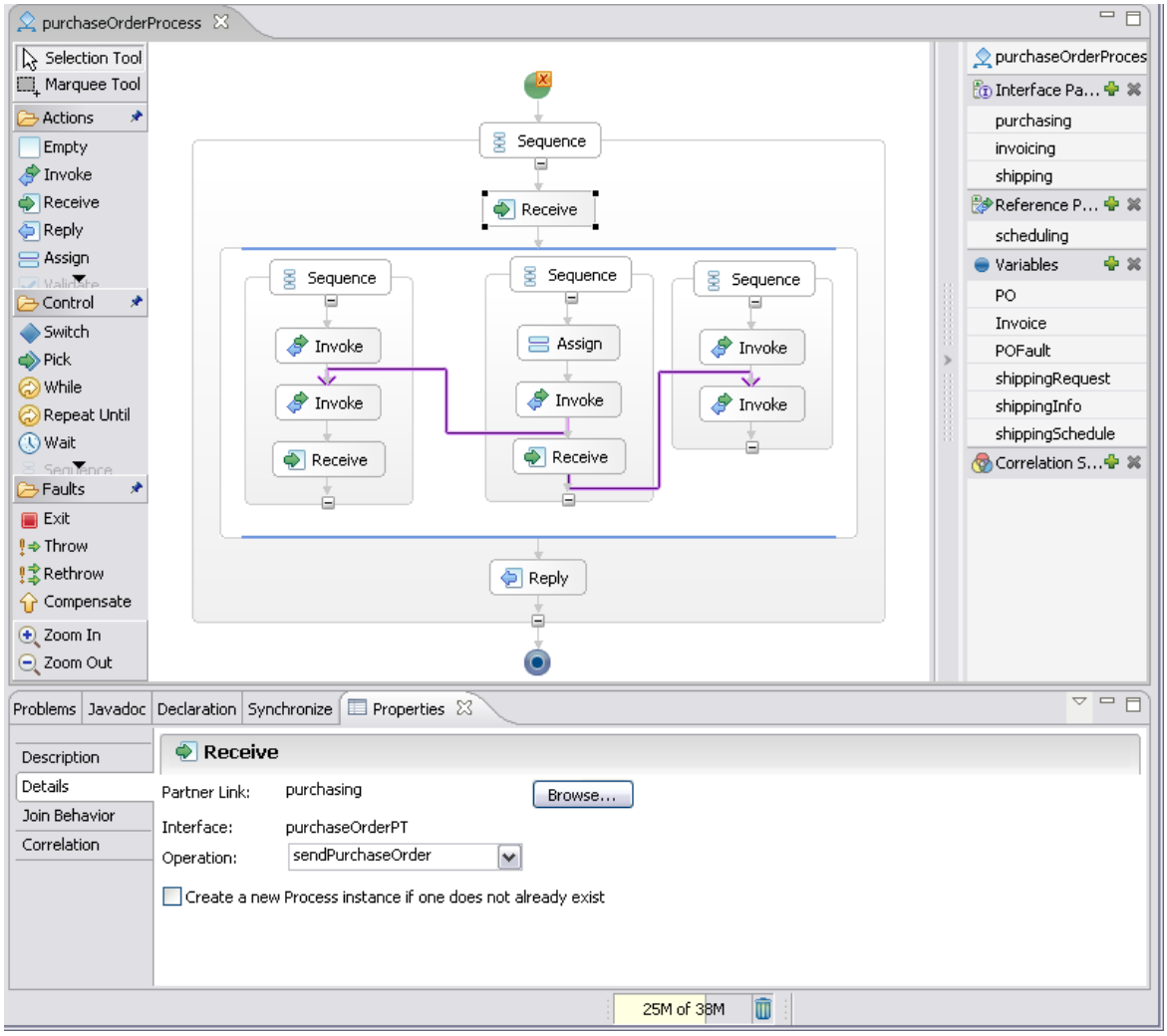

Figura 11: Processo BPEL modelado na IDE Eclipse (BPEL Designer, 2012)

Para executar uma BPEL, é necessário um motor que fica esperando uma requisição para ser executado. O motor escolhido neste trabalho foi o Apache ODE.

4.4 Apache ODE

O Apache ODE é um motor de orquestração que coordena e executa processos de negócios escritos pelo padrão WS-BPEL (Apache ODE, 2012). Ele conversa com os *web services*, enviando e recebendo mensagens, manipulando dados e lidando com os erros. Este motor funciona apenas com a especificação de *web services* SOAP.

4.5 QoS e medição dos tempos

Das diversas definições sobre QoS, uma das mais extensivas possuem os requisitos a seguir: performance, confiabilidade, escalabilidade, capacidade, robustez, tratamento de exceção, precisão, integridade, acessibilidade, disponibilidade, interoperabilidade, segurança e requisitos relacionados a redes. Em especial, o último requisito é muito interessante (REPP et al., 2006).

A análise da qualidade do serviço é um ponto importante nesse trabalho e pode ser realizada com monitoramento QoS (Quality of Service) e modelos de análise QoS. Com essa análise dos dados interceptados, pode-se conhecer qual a deficiência e identificar sua fonte. A medição é feita com base em funções estatísticas e valores relacionados ao tempo (HALIMA et al., 2008a). Alguns parâmetros podem ser utilizados, como: tempo de resposta, processamento ou disponibilidade (HALIMA et al., 2008a). Os parâmetros QoS considerados para medição são quatro tempos: t1: tempo em que a requisição foi realizada, t2: tempo em que a requisição é recebida pelo serviço provedor, t3: tempo em que o serviço provedor devolve a resposta, e t4: tempo em que a resposta é recebida pelo t4.

5. DESENVOLVIMENTO

Para acompanhar a execução e realizar a medição dos tempos nos *web services* de um *workflow*, diversas tecnologias foram utilizadas. O XML é usado como linguagem de suporte para a definição dos serviços com WSDL, SOAP e BPEL.

O desenvolvimento deste trabalho passou por várias fases. Inicialmente ocorreu a preparação das máquinas virtuais com a configuração dos servidores para que as medidas pudessem ser realizadas. Em seguida foi necessário a instalação do *proxy* coletor de informações da rede e a sua compreensão para a interpretação dos dados de QoS capturados e armazenados no banco de dados. Junto a isso, a instrumentalização do código BPEL foi desenvolvida e a sua automatização implementada. Finalmente a ferramenta de visualização foi construída. O desenvolvimento destas fases são detalhados nos itens a seguir.

5.1 Preparação das máquinas virtuais

Para criar os *web services* e *workflows*, foram necessários alguns *softwares* instalados nas máquinas virtuais com o sistema operacional *Ubuntu Server* 12.04 LTS. Utiliza-se o *Java Development Kit* (JDK) 1.6 como base para executar alguns *softwares*. Um desses softwares é o Apache Tomcat, um *container* de *servlets*, que por sua vez foi usado como base para hospedar o motor de *web services* Axis2 1.6.2 e também o motor de orquestração ODE 1.3.5.

5.2 Instalação do proxy coletor

Para que a coleta de dados fosse feita, um *proxy* modificado foi inserido entre as chamadas dentro de cada *workflow*. Para isso, foi necessário que o arquivo catalina.sh do Tomcat fosse alterado com a adição da linha *JAVA_OPTS="-Dhttp.proxyHost=192.168.2.10 –Dhttp.proxyPort=8080 –DproxySet=true"* sendo 192.168.2.10 e 8080 o IP fixo e a porta do servidor *proxy*, respectivamente. Com esta configuração no servidor Apache Tomcat do *workflow*, todas as chamadas passam pelo *proxy*, menos a primeira, realizada pelo cliente. Com isso, há a

necessidade do cliente também adaptar a primeira chamada do *workflow* para o uso do *proxy,* caso contrário, os dados dessa instância não serão utilizados para representação gráfica, pois seus dados não serão armazenados.

5.2.1 Banco de dados

Para armazenar os dados coletados pelo *proxy*, foi necessário um banco de dados. Optou-se por utilizar o servidor MySQL versão 5.0.27 e neste banco foi criada uma tabela denominada WS_LOG com os seguintes campos:

- ID_NUMBER: Identificador único de cada operação/transação.
- INSTANCE: Identificador único usando UUID do *workflow* que fez a chamada do *web service.*
- SOURCE_INSTANCE: Identificador único usando o UUID da instância retornada de um *workflow* para o *workflow* que chamou o primeiro
- SOURCE: Armazena o IP de quem realiza a chamada do *web service.*
- DESTINATION: Armazena o endereço URL do serviço chamado.
- ACTION: Armazena o nome da operação do serviço chamado.
- T1: Tempo em que a chamada foi realizada. Esse valor está no formato *epoch*, que é a representação numérica feita desde o dia 1 de Janeiro de 1970. O valor encontra-se em milissegundos.
- T4: É o tempo, no formato *epoch,* que o serviço que fez a chamada recebeu a resposta.

A Figura 12 exemplifica como os UUID`s são transferidos no *workflow*.

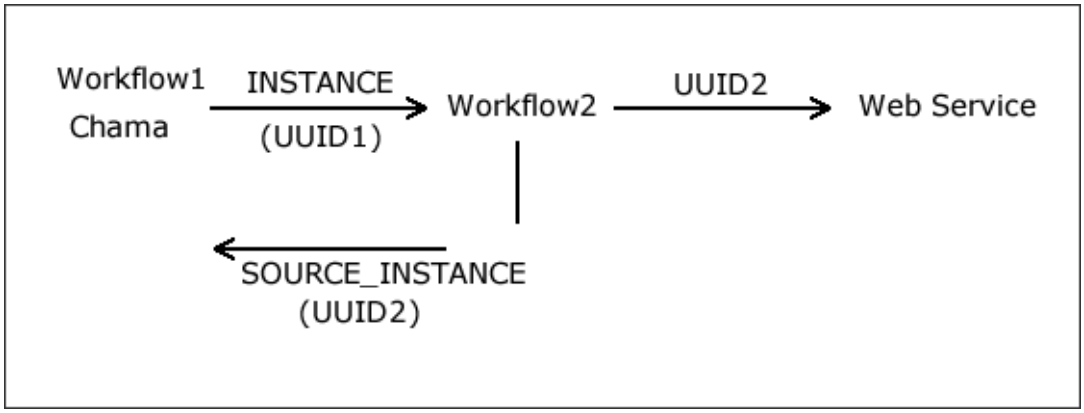

Figura 12: Esquema do banco de dados que armazena as informações

5.3 Instrumentalização do código BPEL

Os *workflows* em BPEL possuem arquivos no formato WSDL e BPEL, os quais são baseados em XML. A instrumentalização é o processo de inserir uma chamada ao *web service* Instance e adicionar o valor recebido a cada chamada que o atual serviço possa realizar. Este processo pode ser realizado através do *plugin* BPEL Designer, porém, neste trabalho, este processo foi automatizado. Para que a instrumentalização fosse automatizada, foi necessário que se tenha *scripts* preparados para ler, interpretar e modificar o conteúdo desses arquivos XML. Para isso, utilizou-se a linguagem Java com a biblioteca VTD-XML para auxiliar no processo de *parsing*, também conhecido como análise sintática, dos arquivos. Esse processo é facilitado usando a linguagem XPath (XPath, 2012), presente na biblioteca VTD-XML, que auxilia a navegação pela estrutura do XML. Existem 3 tipos de arquivos a serem modificados, com diferentes regras e estruturas. São eles:

- *Artifacts.wsdl - descritor do serviço gerado pelo *workflow*
- *.wsdl - arquivos descritores dos *web services* utilizados no *workflow*, incluindo *web service Instance*.wsdl.
- *.bpel - arquivo que possui a lógica das chamadas dos *web services*.

Os asteriscos acima indicam que o nome depende dos serviços utilizados ou descritos.

Sendo assim, foram desenvolvidos 3 *scripts* diferentes para realizar as modificações necessárias neste trabalho.

As modificações do arquivo *Artifacts.wsdl são:

No XPath `“//definitions”` é inserido `xmlns:nsInstance="http://Instance.wsdl"` como atributo. Após isso, é adicionado o *PartnerLink* do *instance* utilizando o XPath “`//definitions/plnk:partnerLinkType`”.

Com o seletor XPath `“//definitions/import`”, é adicionado o import do Instance.wsdl. E finalmente o elemento instance do tipo “string” foi adicionado a todos os filhos de `“//types/schema/element/complexType/sequence”`

Para modificar os arquivos *.bpel, os seguintes passos são realizados:

O atributo `xmlns:nsInstance="http://Instance.wsdl"` é adicionado no XPath `"//bpel:process"`. Em seguida o *namespace* referente ao Instance.wsdl é adicionado ao mesmo XPath anterior. Após isso, o partnerLink do Instance é adicionado no XPath `"//bpel:process/bpel:partnerLinks"`. Em seguida, são inseridas as variáveis `InstancePartnerResponse` e `InstancePartnerRequest` com os tipos `nsInstance:UUIDResponse` e `nsInstance:UUIDRequest` no XPath `"//bpel:process/bpel:variables"`. Ainda é necessário adicionar um "assign", que é onde são definidas as variáveis que serão enviadas na chamada do *web service*. O assign é adicionado na estrutura em que o XPath é `"//bpel:process/bpel:sequence/bpel:assign"`. Além deste assign, é necessário adicionar atributos referentes ao Instante nas estruturas `"//bpel:assign/bpel:copy/bpel:from/bpel:literal"` e `"//bpel:assign"`, as quais geralmente possuem mais de um item correspondente, desta forma, todos estes itens devem ter código adicionado.

Para modificar os arquivos *.wsdl, é chamado apenas um método, onde adiciona o elemento `<xs:element minOccurs="0" name="instance" type="xs:string" />` em todos os elementos que combinem com o XPath `"//wsdl:definitions/wsdl:types/xs:schema/xs:elemento"`.

5.4 Construção da ferramenta de visualização

O *proxy* insere os dados em um banco de dados a medida que captura as mensagens que trafegam entre *workflows* e *web services.* Estes dados são compostos por ID_NUMBER, INSTANCE, SOURCE_INSTANCE, SOURCE, DESTINATION, ACTION, T1, T4. Os campos INSTANCE e SOURCE_INSTANCE definem um grafo que é base para construção da representação gráfica. Além disso, os tempos do início (T1) e término (T4) da execução de cada serviço, que estão armazenados no banco de dados, servem para completar a representação gráfica.

No desenvolvimento do aplicativo para a representação gráfica foi necessário organizar os dados para utilizá-los como um grafo. O primeiro passo é encontrar a primeira chamada que o cliente faz ao *workflow*. Esta chamada pode ser encontrada

fazendo uma pesquisa do banco de dados por registros os quais possuem o valor de INSTANCE diferente de nulo e SOURCE_INSTANCE igual a nulo. Isso quer dizer que com um valor de INSTANCE, esta chamada faz parte do workflow. O SOURCE_INSTANCE nulo significa que o *web service* atual não foi invocado por nenhum outro *web service*, logo, é a primeira chamada. Tendo estes valores, chamados de índices, é possível encontrar a sequência de chamadas do *workflow.* Isto é feito procurando os campos onde o valor do INSTANCE do registro atual seja igual ao SOURCE_INSTANCE de um registro diferente, como pode ser visualizado na Figura 13. Feito isso, as informações dos novos registros, como o nome da operação e ID_NUMBER, são armazenadas na estrutura de dados semelhante a da Figura 14. Cada vez que uma chamada aponta para outra, um nível da profundidade é adicionado à lista. Com esta estrutura pronta, pode-se então desenhar o gráfico.

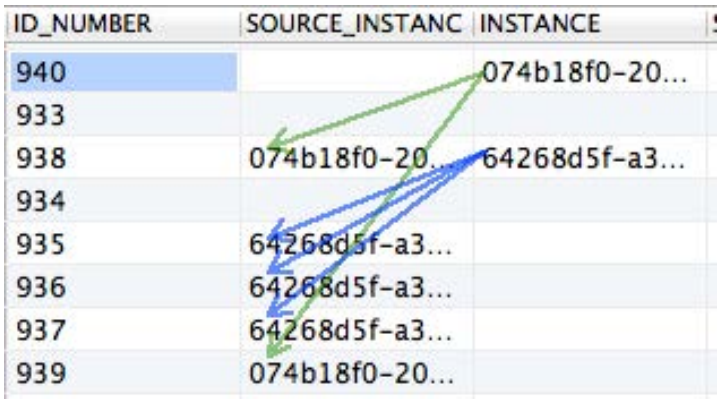

Figura 13: Parte da tabela correspondente a um workflow

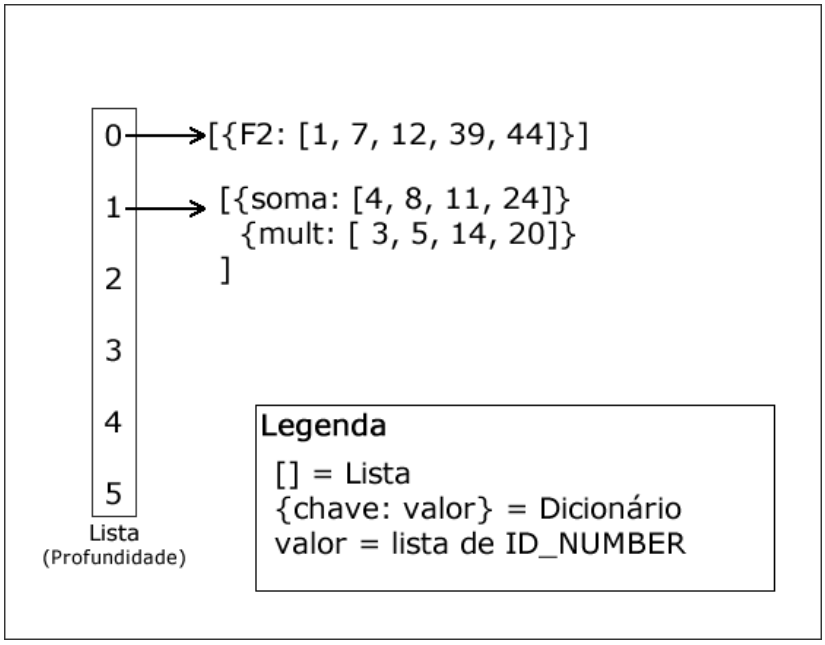

Figura 14: Estrutura de dados que contém informações do grafo

Foi criada uma biblioteca em Python utilizando a biblioteca gráfica Tkinter para realizar os desenhos dos componentes do *workflow* em forma semelhante ao diagrama de sequência. Foram criadas diversas classes. A classe Operation possui como principais variáveis o tempo inicial, final, titulo, subtítulo. A classe Arrow representa a chamada a outra operação de *web service*. Esta classe possui como parâmetro de construtor, instâncias de Operation que são referentes à origem e destino da chamada no *workflow.*

Com as classes Operation, Arrow e com a estrutura de dados criada é possível desenhar a representação proposta.

Para decidir a ordem das chamadas dos web services na hora de desenhar, a lista ordered_list armazena uma lista de ID_NUMBER's da última instância do workflow. Existe um método que retorna uma lista das operações referentes aos ID_NUMBER's. Em seguida, após os dados deste método serem gravados em uma variável workflow apenas com os objetos da última chamada do *workflow*, esta lista é iterada e recupera os ID_NUMBER's dos valores similares. Neste trabalho, valor similar quer dizer que o registro encontra-se na mesma profundidade do grafo e com operação de mesmo nome. Exemplificando com base na Figura 14, se o ID_NUMBER em questão for o 44, ele é referente à operação F2 e seus similares retornados são 1, 7, 12, 39 e 44. Iterando a lista de itens armazenados em workflow

e encontrados os registros similares, pode-se então procurar o tempo médio de início, que é a média de todos os tempos de início do *web service* atual subtraídos pelos tempos de início do primeiro *web service* do *workflow*, que é o primeiro *web service* a ser chamado pelo cliente. Como o tempo médio podia variar muito em cada workflow, foi preciso redimensionar os componentes para que fiquem com tamanho adequado à área gráfica onde o desenho será apresentado. Para isto, foi encontrado um valor que possa dimensionar o gráfico. Este valor, chamado de rate, é encontrado através da divisão do tamanho máximo da área gráfica, pelo maior tempo do workflow, que é o tempo da primeira chamada. Após encontrar este valor, ele é usado quando os tempos inicial e final são definidos, multiplicando estes valores e encontrando a posição relativa na área gráfica.

6. Experimentos e Resultados

Para que este trabalho fosse realizado, alguns *web services* e *workflows* foram desenvolvidos e alocados em máquinas virtuais. Foram usadas funções matemáticas, pois elas podem ser facilmente compreendidas e comparadas. Assim, *web services* como *Arithmetics*, *Functions* e *Trigonometry* forneceram suas operações para a criação de *workflows* com fórmulas mais complexas.

A Figura 15a é um *workflow* BPEL da função F1. Ele pode ser visto como $F1(x, y) = ((a + b) / (a * b))^2$. Para realizar todas as operações matemáticas, foram utilizados os *web services*.

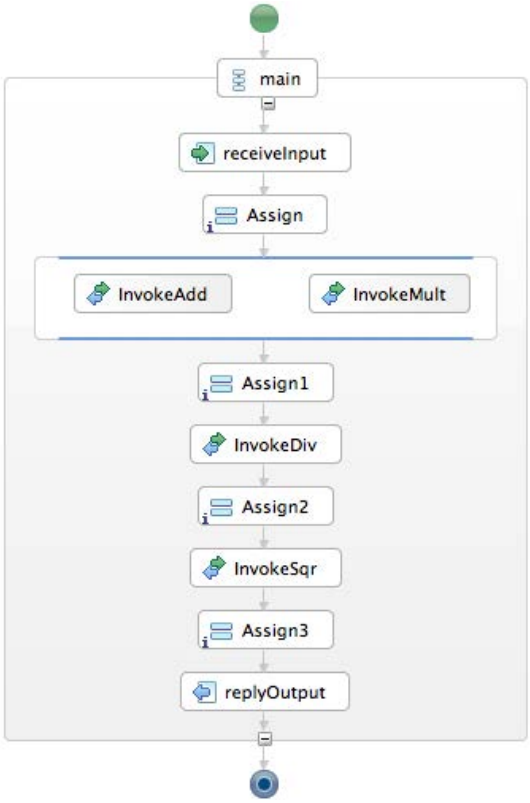

Figura 15a: BPEL do workflow F1 sem Instance

Na Figura 15b, a chamada ao *web service* Instance, no retângulo vermelho, foi adicionada no *workflow* para que os dados pudessem ser capturados corretamente pelo *proxy*. Esta nova chamada pode ser adicionada manualmente modificando o *workflow* através de um editor BPEL, ou de forma automatizada, como foi desenvolvido neste projeto.

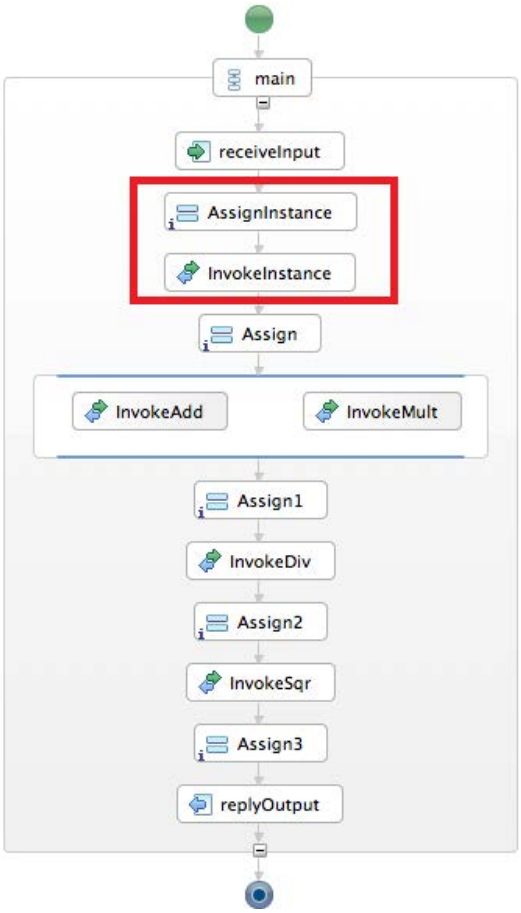

Figura 15b: BPEL do workflow F1 com *Instance*

A representação gráfica é criada na linguagem de programação *Python* usando a biblioteca gráfica *TKinter* para desenhar os componentes e *SQLAlchemy* para abstração do banco de dados com um mapeador objeto-relacional (ORM). Este ORM facilita a manipulação dos dados para que sejam trabalhados em forma de objeto.

O *workflow* BPEL apresentado na Figura 15a tem sua execução representada na Figura 16 pelo diagrama proposto. A representação da Figura 16 possui retângulos amarelos que exibem os nomes dos serviços na parte superior (F1) e a operação na parte inferior do retângulo (LAAS.com/process). As linhas tracejadas na vertical representam onde as operações de um serviço serão executadas. Sobre estas linhas tracejadas, são colocados retângulos verticais indicando o tempo que a operação ocupa durante sua execução. As setas horizontais representam as chamadas às funções, e o retorno dessa chamada é demonstrado pelas setas tracejadas.

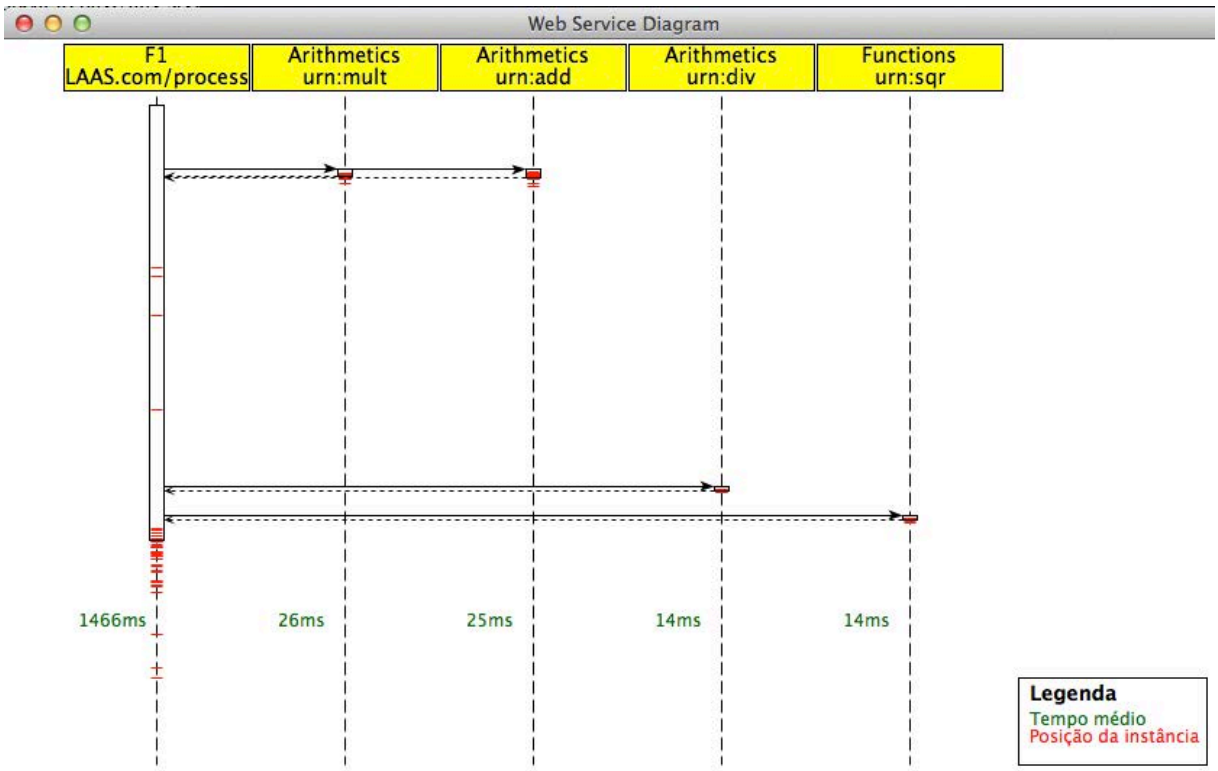

Figura 16: Representação gráfica dos tempos do workflow F1

Como pode ser visto na Figura 16, existem retângulos verticais de cor branca, seus topos representam o instante médio de início de execução da operação representada e suas bases os tempos médios de execução. Os tempos médios entre todas as chamadas de cada uma das operações foi calculado com as medidas realizadas em 39 execuções do workflow F1 e este valor está representado em verde. Em vermelho estão marcados os pontos em que cada uma das operações termina. Estes pontos são relativos ao início da própria instância da operação.

Um exemplo mais complexo pode ser visto na Figura 17, a qual representa a expressão $F3(x, y) = (F2(x, y))^2$. O diferencial deste *workflow* F3 é que ele chama outro *workflow,* F2, que também está modificado para ter suas instancias identificadas. A função F2 representa a expressão (a + b) / (a * b).

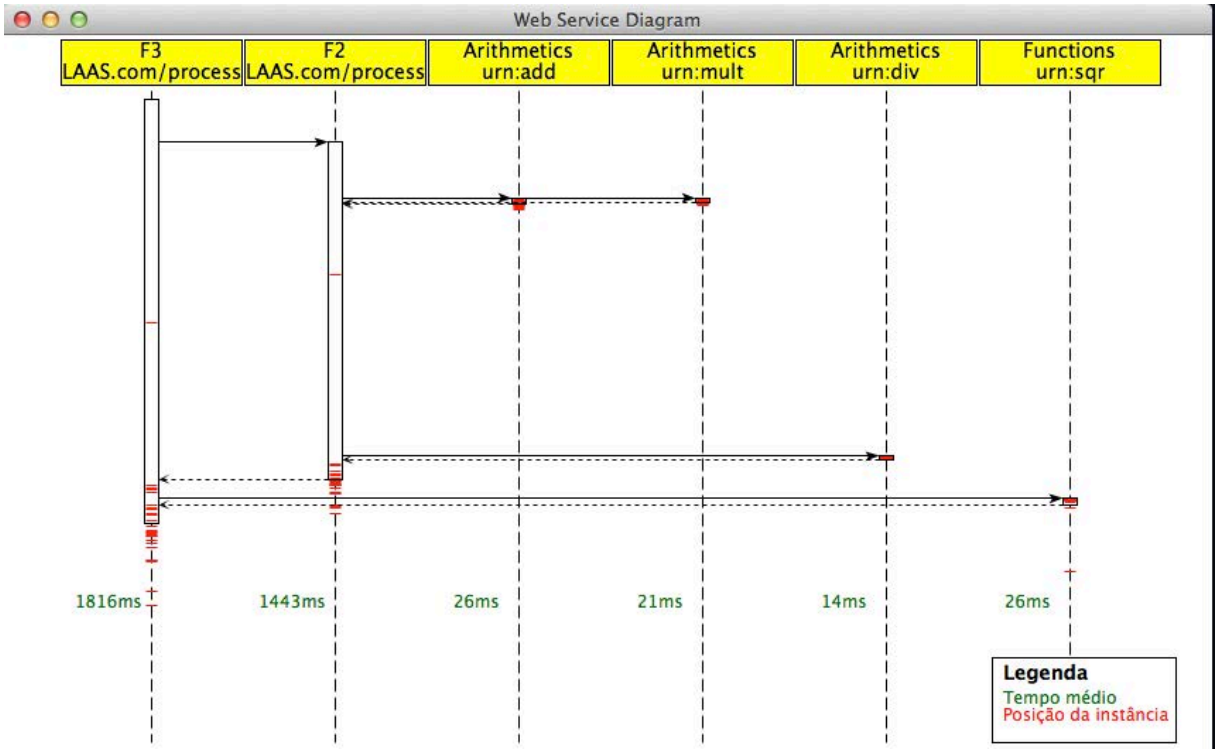

Figura 17: Representação do workflow F3

Um administrador observando diagramas das figuras 16 e 17, identifica que os *web services,* se otimizados, resultarão em pouco ganho no desempenho. Por outro lado, as execuções dos workflows indicam um tempo de aparente pausa na execução, uma vez que entre as chamadas dos *web services* existe um intervalo grande de tempo. Nos exemplos nada pode ser considerado sobre discrepâncias na disponibilidade dos serviços oferecidos, uma vez que as faixas vermelhas que indicam o término das operações aparecem concentradas próximas às medias dos tempos de execução de cada serviço.

Em um ambiente real, a primeira chamada pode ter uma variação muito grande, pois depende da conexão, latência e outros fatores que variam mais do que a disponibilidade dos serviços da rede.

As representações dos *workflows* mostram que existe atraso entre as chamadas, os quais não deveriam ser tão acentuados devido à estrutura ser totalmente local e simulada.

7. CONCLUSÃO

O projeto apresenta uma ferramenta de visualização do funcionamento de *workflows* capturado a partir das mensagens trocadas entre os *workflows* e os *web services*.

A ferramenta oferece recursos de visualização dos tempos médios das chamadas dos serviços e indicação dos tempos relativos observados em cada chamada na execução de cada serviço envolvido.

Para que a ferramenta de visualização pudesse funcionar, os dados foram capturados por um *proxy* que intercepta as mensagens e armazena informações de qualidade de serviço e dos serviços envolvidos em um banco de dados.

REFERÊNCIAS BIBLIOGRÁFICAS

Apache Axis2/Java. Disponível em: <http://axis.apache.org/axis2/java/core>. Acesso em 11 jun. 2012.

Apache ODE. Disponível em: <http://ode.apache.org>. Acesso em 3 mai. 2012.

ATKINSON, A.; BALAKRISHNAN, R.; BOOTH, P.; COTE, J. M.; GROOT, T.; MALMI, T.; ROBERTS, H.; ULIANA, E.; WU, A. New Directions in Management Accounting Research. **Journal of Management Accounting Research 9**. [S.l.: s.n.], 1997. 79-108 p.

BECKER, A., CLARO, D., SOBRAL, J. Web Services e XML, Um Novo Paradigma da Computação Distribuída, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, 2008.

BPEL Designer. Disponível em: <http://www.eclipse.org/bpel>. Acesso em 27 abr. 2012.

C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. 2002. Efficient filtering of XML documents with XPath expressions. The VLDB Journal 11, 4 (December 2002), 354-379.

FENSEL, D., & BUSSLER, C. The *Web service* Modeling Framework WSMF. **Electronic Commerce Research and Applications,** 2002.

HALIMA, R. B.; GUENNOUN, M. K.; DRIRA K.; JMAIEL M. Non-intrusive QoS monitoring and analysis for self-healing *web services*. 2008a. Disponível em: <http://hal.archives-ouvertes.fr/hal-00438851/en/>.

HALIMA, R. B.; GUENNOUN, M. K.; DRIRA K.; JMAIEL M. Providing Predictive Self-Healing for Web Services: A QoS Monitoring and Analysis-based Approach, Journal of Information Assurance and Security 3, 3 (2008b) 175-184.

LEGROS, B. Disponível em: http://www.brianlegros.com/blog/2007/08/09/architectual-styles-part-three-domains. Acesso em 19 jul. 2012.

NEZHAD, H. R. M.; BENATALLAH, B.; CASATI, F.; TOUMANI, F. "Web services interoperability specifications," **Computer**, vol.39, no.5, pp. 24-32, May 2006. Disponível em: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1631936&isnumber=34216 Acesso em 14 out. 2012.

Oracle VM VirtualBox. User Manual. Disponível em: <https://www.virtualbox.org/manual/UserManual.html>, Acesso em 27 ago. 2012.

PEGORARO, R.; HALIMA, R. B.; DRIRA K.; GUENNOUN, M. K.; ROSÁRIO, J. M. A framework for monitoring and runtime recovery of web service-based applications. In: **10th International Conference on Enterprise Information Systems**, Barcelona : Espagne (2008). Disponível em: <http://hal.archives-ouvertes.fr/hal-00438852>.

N. REPP, R. BERBNER, O. HECKMANN, AND R. STEINMETZ. A cross-layer approach to performance monitoring of web services. In Proceedings of the Workshop on Emerging Web Services Technology (in conjunction with IEEE ECOWS 2006). CEUR-WS, Dec 2006.

SOAP. Disponível em: <http://www.w3.org/TR/soap>. Acesso em 10 abr. 2012.

VTD-XML. Disponível em: <http://vtd-xml.sourceforge.net>. Acesso em 22 jun. 2012.

XML. Disponível em: <http://www.w3.org/XML/>. Acesso em 10 abr. 2012.

XML Schema. Disponível em: <http://www.w3.org/XML/Schema.html>. Acesso em 10 abr. 2012.

XPath, Disponível em: <http://www.w3.org/TR/xpath>. Acesso em 22 jun. 2012.

W3Schools SOAP. Disponível em: <http://www.w3schools.com/soap/soap_syntax.asp>. Acesso em 5 jul. 2012.

W3Schools XML Schema. Disponível em: <http://www.w3schools.com/xpath/xpath_examples.asp>. Acesso em 5 jul. 2012.

W3Schools XPath. Disponível em: <http://www.w3schools.com/xpath/xpath_examples.asp>. Acesso em 5 jul. 2012.

W3Schools WSDL. Disponível em: <http://www.w3schools.com/wsdl/wsdl_documents.asp>. Acesso em 9 mai. 2012.