PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SOFTWARE COMO SERVIÇO: UM *FRAMEWORK* PARA FORNECER FERRAMENTAS DE SIMULAÇÃO ANALÍTICA

RAFAEL TWEEDIE CAMPOS

Dissertação de Mestrado apresentada como requisito para obtenção do título de Mestre em Ciência da Computação pelo Programa de Pós-graduação da Faculdade de Informática. Área de concentração: Ciência da Computação.

Orientador: Paulo Henrique Lemelle Fernandes

Porto Alegre
2012

C198s Campos, Rafael Tweedie
Software como serviço : um framework para fornecer ferramentas de simulação analítica / Rafael Tweedie Campos. – Porto Alegre, 2012.
95 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Paulo Henrique Lemelle Fernandes.

1. Informática. 2. Redes De Autômatos Estocásticos. 3. World Wide Web – Ferramentas. 4. Framework. I. Fernandes, Paulo Henrique Lemelle. II. Título.

CDD 004.67

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Software como Serviço: Um Framework para Fornecer Ferramentas de Simulação Analítica", apresentada por Rafael Tweedie Campos como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 09/08/2012 pela Comissão Examinadora:

Prof. Dr. Paulo Henrique Lemelle Fernandes - PPGCC/PUCRS
Orientador

Prof. Dr. César Augusto Fonticielha De Rose - PPGCC/PUCRS

Prof. Dr. Afonso Henrique Corrêa de Sales - FACIN/PUCRS

Homologada em 23/10/2012, conforme Ata No. 023 pela Comissão Coordenadora.

Prof. Dr. Paulo Henrique Lemelle Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

"UNIX is very simple, it just needs a genius to understand its simplicity."

(Dennis Ritchie)

AGRADECIMENTOS

Agradeço à minha esposa Grazziane pela compreensão e paciência. Foram difíceis os momentos de dedicação à este trabalho, sem a colaboração da minha família não seria possível o desenvolvimento deste trabalho. Ao meu orientador que, pela sua simplicidade, facilitou o meu trabalho e por tudo que eu aprendi. Obrigado por ter me aceitado como orientando. Agradeço aos pesquisadores do grupo de pesquisas PEG-PUCRS, Afonso Sales, Ricardo Czeskter e Thais Webber, por me apoiarem e me ajudarem durante o curso de mestrado e também durante o desenvolvimento deste trabalho.

Muito obrigado a HP pela bolsa fornecida que tanto me ajudou nesta jornada.

Aos demais colegas, professores e funcionários do programa de pós-graduação, obrigado pelo apoio, paciência e sugestões.

À minha esposa dedico esta dissertação.

SOFTWARE COMO SERVIÇO: UM FRAMEWORK PARA FORNECER FERRAMENTAS DE SIMULAÇÃO ANALÍTICA

RESUMO

Atualmente, existe uma crescente demanda por se disponibilizar, através da Internet, *software* na forma de serviços. Este novo modelo de negócio é denominado de "*Software* como Serviço", aonde aplicações não são mais vendidas como um produto, mas sim cedidas na forma de um serviço e utilizados por clientes, que pagam apenas de acordo com a utilização. Entretanto, soluções que não foram desenvolvidas com o foco em integração, não conseguem entrar facilmente nesta nova tendência. PEPS (*Performance Evaluation of Parallel Systems*) e PLAT (*Production Line Analysis Tool*) são exemplos de ferramentas que se enquadram nesta situação. Apesar de serem soluções eficientes e confiáveis para a solução de, respectivamente, modelos de Redes de Autômatos Estocásticos e modelos teóricos de linhas de produção seriais, nenhuma das duas possui mecanismos de integração. Neste sentido, este trabalho propõe: Primeiro, a especificação de um *framework* genérico e reutilizável para a disponibilização de ferramentas na forma de serviços; Segundo, as implementações deste *framework* para a disponibilização dos *software* PEPS e PLAT na forma de serviços.

Palavras-chave: Software como Serviço; Arquitetura Orientada a Serviços; Computação na Nuvem; Soluções numéricas; Redes de Autômatos Estocásticos.

SOFTWARE AS A SERVICE: A FRAMEWORK TO PROVIDE ANALYTICAL SIMULATION TOOLS

ABSTRACT

Currently, there is a growing trend for exposing software, through the Internet, as services. This new business model is called "Software as a Service", where applications are no longer sold as a product, but instead are "rented" as a service and used by customers, who pay only according to use. However, solutions that were not developed with a focus on integration, can not easily enter into this new trend. PEPS (Performance Evaluation of Parallel Systems) and PLAT (Production Line Analysis Tool) are examples of tools into this situation. Although they are efficient and reliable solutions to solve, respectively, Stochastic Automata Networks models and theoretical performance models of serial production lines, neither have mechanisms for integration. Thus, this work proposes: Firstly, the specification of a generic and reusable framework for the provision of legacy codes as services; and secondly, the implementation of this framework for the provision of PEPS and PLAT as services.

Keywords: Software as a Service; Service-Oriented Architecture; Cloud computing; Numerical solutions; Stochastic Automata Networks.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

API	*Application Programming Interface*
ATM	*Asynchronous Transfer Mode*
B2B	*Business-to-Business*
BEEP	*Block Extensible Exchange Protocol*
CEO	*Chief executive officer*
CORBA	*Common Object Request Broker Architecture*
DCOM	*Distributed Component Object Model*
EPS	*Encapsulated PostScript*
ERP	*Enterprise resource planning*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IaaS	*Infrastructure as a Service*
IDE	*Integrated Development Environment*
IoC	*Inversion of Control*
J2EE	*Java2 Platform Enterprise Edition*
JDK	*Java Development Kit*
JPA	*Java Persistence API*
MDD	*Multi-valued Decision Diagrams*
MVC	*Model-View-Controller*
OASIS	*Organization for the Advancement of Structured Information Standards*
PaaS	*Platform as a Service*
PEPS	*Performance Evaluation of Parallel Systems*
PLAT	*Production Lines Analysis Tool*
PNG	*Portable Network Graphics*
PSS	*Product State Space*
REST	*Representational State Transfer*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
RSS	*Reachable State Space*
SaaS	*Software as a Service*
SAN	*Stochastic Automata Networks*

SGBD	*Sistema Gerenciador de Banco de Dados*
SGML	*Standard Generalized Markup Language*
SHA	*Secure Hash Algorithm*
SMTP	*Simple Mail Transfer Protocol*
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
TCP	*Transmission Control Protocol*
TI	*Tecnologia da Informação*
UDDI	*Universal Description Discovery and Integration*
URL	*Unified Resource Locator*
VDP	*Vector-Descriptor Product*
XML	*Extensible Markup Language*
XSLT	*eXtensible Stylesheet Language for Transformation*
W3C	*World Wide Web Consortium*
WSDL	*Web Services Description Language*

SUMÁRIO

1. INTRODUÇÃO

Nos dias de hoje, os sistemas operacionais e os aplicativos que usamos estão cada vez mais se fundindo com a Internet. Muitas vezes temos a impressão de estar utilizando um *software* instalado localmente, mas, na verdade, estamos acessando um serviço que foi disponibilizado na *Web*. Serviços *on-line* estão cada vez mais inseridos em nosso cotidiano, trazendo consigo um novo modelo de negócio. Este novo negócio é o modelo de distribuição *Software* como Serviço, aonde aplicações não são mais vendidas como um produto, mas sim cedidas na forma de um serviço por um fornecedor e utilizadas por um cliente consumidor. Neste modelo, o fornecedor do serviço se responsabiliza por toda a infraestrutura necessária para a disponibilização do sistema e o cliente utiliza o *software* via Internet, pagando um determinado valor pela utilização.

Entretanto, aplicativos antigos, legados ou que não tenham sido planejados para integração, não conseguem ser inseridos neste modelo facilmente. Neste contexto, este trabalho propõe um *framework* para facilitar a disponibilização como serviço desses aplicativos.

O PEPS [PLA88] é um *software* criado para solucionar numericamente grandes modelos Markovianos, expressos na forma de Redes de Autômatos Estocásticos (*Stochastic Automata Networks* - SAN) [PLA85]. E apesar de ser uma ferramenta eficiente e em constante evolução, o PEPS não possui funcionalidades que facilitem a sua integração com outras ferramentas ou sistemas.

Carente do mesmo tipo de funcionalidade, o PLAT [FER11b] é uma ferramenta criada para calcular soluções exatas para linhas de produção confiáveis.

Neste sentido, este trabalho visa apresentar uma solução para estes problemas, oferecendo uma solução para tornar simples e eficiente a criação de mecanismos de integração para ambos PEPS e PLAT, e assim, tornar possível a disponibilização destes *software* como um serviço. Apesar do foco deste trabalho estar centrado nas ferramentas PEPS e PLAT, o *framework* proposto pode ser facilmente estendido e utilizado na disponibilização como serviço de qualquer ferramenta.

1.1 Objetivo

Os objetivos deste trabalho são propor um *framework* para disponibilização de ferramentas não integráveis na forma de serviços. E apresentar as implementações desta solução para a exposição das ferramentas PEPS e PLAT através do modelo de distribuição *software* como serviço.

1.2 Contribuição

Ao alcançar os objetos, é esperada uma solução genérica que possa ser reutilizada para a disponibilização de diversas ferramentas que atualmente não possuem funcionalidades de integração, além de fornecer à comunidade científica as ferramentas PEPS e PLAT na forma de serviços.

1.3 Organização

Esta dissertação está organizada da seguinte forma. O Capítulo 2 contem uma introdução aos conceitos de Computação na Nuvem, *Software* como Serviço, *Web Services* e Arquitetura Orientada a Serviços e ao final apresenta uma descrição dos trabalhos relacionados a este projeto. A seguir, o Capítulo 3 fornece uma visão geral sobre o formalismo de Redes de Autômatos Estocásticos, juntamente com a apresentação das ferramentas de simulação que são alvo do estudo deste trabalho. O Capítulo 4 contem a contribuição principal deste trabalho, apresentando o *framework* proposto para solucionar os problemas identificados. Ao final, o Capítulo 5 é reservado para as conclusões.

2. SOFTWARE COMO SERVIÇO

Este capítulo apresenta os principais conceitos de *Software* como Serviço (*Software as a Service - SaaS*). A Seção 2.1 apresenta a Arquitetura Orientada a Serviços e a Seção 2.2 mostra os conceitos e as especificações da tecnologia *Web Services*. Ambas as Seções são pré-requisitos para a Seção 2.3, aonde é introduzido o conceito de computação na nuvem, juntamente com o conceito de *Software* como Serviço.

2.1 Arquitetura Orientada a Serviço

SOA (*Service-Oriented Architecture*) [ERL05] representa uma recente abordagem para a utilização dos recursos de TI (Tecnologia da Informação) em apoio ao negócio de uma organização. É um tipo de arquitetura de *software* que utiliza processos de negócio e distribui funções em forma de serviços. Apesar de ainda carecer de certa maturidade, SOA apresenta a renovação da forma como a TI deve ser usada para apoiar a estratégia de uma organização [MAR09].

Utilizar SOA significa modularizar uma aplicação na forma de diversos serviços interoperáveis. Em seu nível mais abstrato, uma arquitetura orientada a serviços é composta por três elementos que representam papéis distintos: o prestador de serviço, o consumidor do serviço e o registro do serviço. Estes três elementos compõem o modelo operacional triangular. A Figura 2.1 representa a interação entre os elementos desse modelo.

Figura 2.1: Modelo operacional triangular de SOA

Cada um dos elementos do modelo triangular se comporta da seguinte forma:

- **Provimento do serviço (Provedor):** Determina o comportamento daquele que está disponibilizando o serviço, ou seja, é considerado o dono do serviço; é o responsável por fornecer toda a infraestrutura de acesso, tipicamente via rede, e é capaz de responder as requisições.
- **Consumo do serviço (Consumidor):** Determina o comportamento daquele que representa o cliente da organização provedora do serviço. Em SOA, um consumidor pode ser representado por uma pessoa, uma organização, uma máquina ou um componente de *software*. A identificação do cliente é irrelevante no sentido de que um consumidor representa aquele que localiza um serviço, entende seu protocolo de operação e se utiliza desse protocolo para executá-lo.

- **Registro do serviço (Registro):** Determina o comportamento que a organização deve ter para divulgar seu serviço e o do cliente que deve proceder para localizar o serviço desejado. É responsável por gerenciar os repositórios que armazenam informações sobre os serviços e entidades que os fornecem. Normalmente, esses registros contêm:

  - Informações de identificação do serviço, tais como nome, descrição e contato. São as denominadas páginas brancas;
  - Informações técnicas como linguagens, tecnologias utilizadas, infraestrutura de acesso. São as denominadas páginas verdes;
  - Informações sobre o serviço, tais como as operações existentes e versões disponíveis. São as denominadas páginas amarelas;

É da responsabilidade do registro oferecer mecanismos de publicação e busca para que esse possa ser usado, garantindo facilidade de localização e utilização do serviço.

O modelo operacional triangular determina que os provedores de serviços registrem suas informações em um repositório central [ERL05]. Por sua vez, o registro deve ser capaz de aceitar a publicação, mediante um protocolo específico de comunicação e autenticação, organizar semanticamente as informações que irão representar o serviço dentro do registro e oferecer mecanismos de busca para que esse serviço possa ser descoberto por eventuais clientes/consumidores. Normalmente, um registro mantém seu repositório associado a duas interfaces de comunicação: a de publicação e a de busca [MAR09].

O conceito de SOA está fortemente ligado com o de *Web Services* [BOO04a], pois o seu modelo é um ótimo exemplo de SOA, uma vez que possui as características desta arquitetura e é considerada, atualmente, a tecnologia padrão para implementação de uma arquitetura orientada a serviços. Entretanto, a implementação de um sistema SOA pode ser realizada utilizando qualquer outra tecnologia padronizada para *Web*, como por exemplo CORBA (*Common Object Request Broker Architecture*) [BOO04b], RMI (*Remote Method Invocation*) [WOL96], DCOM (*Distributed Component Object Model*) [GRI97] e REST (*Representational State Transfer*) [RIC07]. Uma boa comparação para esta situação é mostrar SOA como se fosse um algoritmo. Ambos não pressupõem o uso de nenhuma tecnologia ou linguagem de programação específica, permitindo a escolha de implementação por parte do desenvolvedor.

A Seção a seguir irá detalhar a tecnologia *Web Services*, pois além de ser a implementação mais comum para SOA, também foi a tecnologia utilizada na implementação do *framework* que será apresentado no Capítulo 4.

2.2 Web Services

O W3C (*World Wide Web Consortium*) [W3C11] define *Web Services* como um sistema de *software* projetado para suportar a interoperabilidade entre máquinas sobre uma rede. Deve possuir uma

interface descrita em um formato compreensível pelas máquinas, chamado de WSDL (*Web Services Description Language*). As chamadas para *Web Services* são feitas utilizando-se o descritor WSDL e através de mensagens SOAP (*Simple Object Access Protocol*), tipicamente transportadas usando HTTP (*Hypertext Transfer Protocol*) com a serialização XML (*Extensible Markup Language*) em conjunto com outros padrões para *Web* [BOO04a]. O W3C juntamente com o OASIS (*Organization for the Advancement of Structured Information Standards*) [OAS11] são os órgãos responsáveis pela especificação de todas estas tecnologias.

Web Service é uma solução utilizada na integração de sistemas e na comunicação entre diferentes aplicações. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Existe uma grande motivação sobre a tecnologia *Web Service*, pois ela possibilita que diferentes aplicações comuniquem entre si e utilizem diferentes recursos. Por exemplo, um *software* escrito na linguagem C e rodando em uma plataforma UNIX pode fazer chamadas para funções de um outro *software* construído em Java e rodando em uma plataforma *Windows*. Basicamente, *Web Services* são pequenas funcionalidades (funções, procedimentos ou métodos) de uma aplicação que pode ser chamado através de um protocolo de rede, ambos os parâmetros de entrada quanto os valores de retorno são especificados em uma estrutura XML.

Os *Web Services* surgiram da necessidade de padronizar a comunicação entre diferentes plataformas e diferentes linguagens de programação. Já haviam sido feitas algumas tentativas na criação deste padrão, porém nenhuma obteve êxito considerável. CORBA [BOO04b], RPC (*Remote Procedure Call*) [THU09] e RMI são exemplos destas tentativas não sucedidas. O fator crucial para o fracasso destas tecnologias em se tornar o padrão de intercomunicação, foi o protocolo de comunicação utilizado. Enquanto *Web Services* trafega através de HTTP, as outras tecnologias procuraram criar seus próprios protocolos de comunicação, a fim de otimizar a transferência de dados. Esta otimização trouxe como consequência a necessidade de utilização de portas de comunicação específicas. Esta obrigatoriedade de disponibilização de portas específicas praticamente inviabilizou o uso destas tecnologias em um ambiente como a Internet, pois a abertura de portas de comunicação representa uma ameaça à segurança das redes das corporações. A impossibilidade de troca de mensagens via Internet também impossibilitou a integração de serviços entre distintas organizações. *Web Services* foi planejado para contornar este problema, o uso de HTTP como protocolo de comunicação não necessita que regras de *firewall* sejam modificadas para permitir a comunicação, assim sendo, não representa uma ameaça à segurança da rede.

Proposto em 1999 através de um consórcio entre empresas, foi desenvolvido o modelo *Web Services*. Após uma adoção significativa deste modelo, surgiu a iniciativa UDDI (*Universal Description Discovery and Integration*) [CLE04]. O UDDI é um diretório universal de descrição de serviços. Como o próprio nome sugere, tal iniciativa se propõe a listar todos os *Web Services* disponíveis na Internet. Nos anos seguintes, muitas linguagens de programação criaram APIs (*Application Programming Interface*) para o desenvolvimento de *Web Service*. Surgiram, então, padrões de desenvolvimento, de qualidade e de segurança, características responsáveis por tornar *Web Services*

uma tecnologia cada vez mais sólida [NEW02].

Os padrões para *Web Services* definem os formatos para as mensagens de entrada e saída, e também especificam como a mensagem será enviada. Estes padrões também descrevem as convenções de mapeamento de conteúdo das mensagens dentro e fora dos programas que implementam o serviço, também definem mecanismos para publicar e descobrir interfaces de *Web Services*. *Web Services* podem ser executados tanto em computadores *desktops* como em dispositivos portáteis. Podem ainda ser utilizados para integração *business-to-business* (B2B), conectando aplicações que estão rodando em várias organizações em um mesmo aplicativo.

As Seções a seguir apresentam os principais conceitos relacionados à *Web Services*.

2.2.1 XML - eXtensible Markup Language

XML [BRA03] representa uma família de especificações relacionadas que são publicadas e mantidas pelo W3C. Pode-se considerar o XML como a base sobre a qual os *Web Services* estão construídos, pois provê o armazenamento de descrições e determina o formato de transmissão para a troca de todos os dados via *Web Services* [SOU10].

Baseada na SGML (*Standard Generalized Markup Language*), em 1996, oitenta peritos uniram forças ao W3C para definir uma linguagem de marcação com o poder da SGML, mas com maior facilidade de implementação. O surgimento da linguagem XML transformou o modo como dados são representados e alcançou o patamar de padrão para descrição e transmissão de dados na Internet. O uso desse modelo se popularizou devido a sua flexibilidade, refino e capacidade de representação estruturada de qualquer informação [MAR09].

Esta grande flexibilidade implica em alguns problemas, pois permite a definição de estruturas diferentes para representar a mesma informação. Em uma integração entre sistemas, é difícil assegurar que ambas as partes usarão a mesma estrutura para representar os dados. A Figura 2.2 representa duas estruturas em XML que representam a mesma informação de um endereço.

```
<elemento-raiz>
    <endereco>
        <pais>Brasil</pais>
        <estado>Rio Grande do Sul</estado>
        <cidade>Porto Alegre</cidade>
        <bairro>Partenon</bairro>
        <rua>Av. Ipiranga</rua>
        <numero>6681</numero>
    </endereco>
</elemento-raiz>
```

```
<elemento-raiz>
    <local>
        <pais>Brasil</pais>
        <estado>Rio Grande do Sul</estado>
        <cidade>Porto Alegre</cidade>
        <bairro>Partenon</bairro>
    </local>
    <logradouro>
        <rua>Av. Ipiranga</rua>
        <numero>6681</numero>
    </logradouro>
</elemento-raiz>
```

Figura 2.2: Duas estruturas em XML representando a mesma informação

XSLT (*eXtensible Stylesheet Language for Transformation*) [CLA99b], *XQuery* [BOA03] e *XPath* [CLA99a] são algumas das tecnologias utilizadas para fazer a tradução entre diferentes esquemas durante uma integração. Com estas tecnologias é possível fazer um mapeamento entre os elementos de cada esquema, obtendo assim o dado em uma estrutura homogênea.

Documentos XML conseguem difundir um conhecimento específico de forma inteligente, estruturada e eficaz. Sua aceitação deve-se ao conjunto de ferramentas de suporte existentes atualmente, como banco de dados XML, ferramentas de desenvolvimento, *browsers* e diversos outros, que suportam e permitem manipular documentos XML de maneira simples e rápida.

As estruturas em XML são compostas por elementos denominados nós (*tags*). Elementos podem ser vazios, simples ou complexos (quando o elemento contem outro elemento aninhado). Sua estrutura pode ser vista como uma hierarquia em formato de árvore e todo documento deve ser iniciado por um único elemento raiz.

Atualmente, o XML é uma tecnologia amplamente adotada, principalmente em grandes sistemas como os ERPs (*Enterprise resource planning*), onde é necessária a customização da solução para se adaptar ao modelo de negócio de cada empresa. Podemos considerar, então, que o XML pode ser utilizado em uma grande variedade de aplicações, como na formatação, serialização e transformação de dados. *Web Services* se comunicam trocando instâncias formatadas em documentos XML. Assim pode-se afirmar que o principal benefício proporcionado pelo XML aos *Web Services* é a independência das estruturas e dos tipos de dados, deixando livre a intercomunicação entre aplicativos [BRA03].

2.2.2 WSDL - Web Service Definition Language

A linguagem de descrição de *Web Services* pode ser vista como uma gramática em XML que tem o objetivo de descrever um *Web Service* como uma coleção de pontos de acesso. Ponto de acesso é o termo usado para definir uma URL (*Unified Resource Locator*) para a qual requisições de serviço são efetuadas. Deve também definir as interfaces públicas que um *Web Service* pode expor, incluindo suas funcionalidades e a forma de como invocá-las, além de declarar os parâmetros das operações e a forma de tratamento das exceções.

WSDL foi originalmente criada pela IBM, Microsoft e Ariba, através da união de 3 propostas anteriores: *Microsoft's SOAP Contract Language*, *Service Description Language* e o *Network Accessible Service Specification Language*. Atualmente a WSDL está na versão 2.0 [CHI07].

A WSDL estabelece um formato comum para descrever e publicar informações de serviços na *Web*. Utiliza elementos que contêm uma descrição de dados, que tipicamente utiliza um ou mais esquemas XML, de modo a tornar as informações compreensíveis de ambos os lados da comunicação.

A especificação WSDL é frequentemente caracterizada por uma parte abstrata e uma parte concreta. A parte abstrata é composta por uma coleção de operações lógicas. Cada operação define uma troca de mensagem simples. Uma mensagem é uma unidade de comunicação, representando os dados trocados em uma transmissão. A parte concreta define protocolos de conexão e os endereços dos serviços [ALO03].

2.2.3 UDDI - Universal Description Discovery and Integration

O UDDI é um padrão para publicação e localização de *Web Services* pelo uso de consultas (*queries*) baseadas em documentos XML. É uma iniciativa da indústria no sentido de criar uma arquitetura livre de plataforma capaz de propagar serviços.

A analogia normalmente feita é com uma lista telefônica, onde as páginas brancas possuem uma lista de organizações, informações de contato e de serviços que estas organizações proporcionam. Nas páginas amarelas, há uma classificação dos serviços e das empresas de acordo com o tipo de serviço que oferecem. Já as páginas verdes descrevem como os serviços devem ser invocados. Esta organização normalmente é muito flexível e permite a fácil consulta de *Web Services*, bem como possibilita um fácil comparativo entre serviços similares oferecidos por uma mesma empresa [ALO03].

O objetivo inicial do UDDI era a criação de um diretório mundial de descrição de serviços gratuitos, onde qualquer pessoa ou empresa poderiam publicar os seus serviços ou pesquisar por serviços já implementados. Porém este objetivo não obteve sucesso, e com o tempo uma nova abordagem foi introduzida. Hoje em dia UDDI está mais direcionado para implementação em ambientes privados, tipicamente empresariais, aonde uma corporação publica os serviços disponíveis para serem consumidos por demais aplicações.

UDDI pode ser visto como um motor de busca para serviços, é possível navegar pelos registros em busca de um serviço que melhor atenda às necessidades do cliente consumidor.

2.2.4 HTTP - Hypertext Transfer Protocol

HTTP é um protocolo de transferência amplamente utilizado na Internet. Basicamente, ele especifica as mensagens que os clientes podem enviar aos servidores e quais as respostas que eles receberão. Por usar TCP (*Transmission Control Protocol*), o controle das mensagens (confirmações de entrega, reenvio de mensagens) é garantido, deixando o HTTP livre para tratar melhor de outros assuntos. Ainda podemos acrescentar que a partir da versão 1.1, foi adicionada a função de conexões persistentes, que permite que mensagens adicionais sejam trocadas através de uma mesma conexão a TCP. Este tipo de conexão diminui o *overhead* das conexões [TAN03].

Apesar de ter sido projetado para a utilização na Internet, o HTTP foi criado visando futuras aplicações orientadas a objetos. Por este motivo ele aceita operações através de métodos. Foi por permitir este tipo de funcionalidade que o protocolo de *Web Services* (SOAP) pôde surgir. Na Tabela 2.1 são descritos os métodos de solicitações HTTP.

Em *Web Services*, HTTP é o protocolo mais utilizado. Ele é o grande facilitador que permite que os *Web Services* se comuniquem de forma transparente através de *firewalls*. É importante deixar claro que o uso do HTTP não é obrigatório quando falamos de *Web Services*, pode-se ainda utilizar SMTP (*Simple Mail Transfer Protocol*) [POS82], BEEP (*Block Extensible Exchange Protocol*) [ROS01] entre outros.

Tabela 2.1: Métodos de Solicitações HTTP

Método	**Descrição**
GET	Solicita algum recurso
HEAD	Solicita a leitura de um cabeçalho
PUT	Envia certo recurso
POST	Envia dados para serem processados
DELETE	Remove determinado recurso
TRACE	Ecoa o pedido enviado
CONNECT	Serve para uso com um proxy que possa se tornar um túnel
OPTIONS	Consulta certas opções

2.2.5 SOAP - Simple Object Access Protocol

O protocolo SOAP é uma especificação do W3C para troca de informação estruturada em ambientes descentralizados e distribuídos. Usa o XML como linguagem de criação das mensagens e o protocolo HTTP como infraestrutura de transmissão. Além disso, o protocolo SOAP é visto como o padrão internacional de interoperabilidade entre aplicações.

Em uma mensagem SOAP, o elemento Envelope é obrigatório, pois define a raiz da estrutura XML da mensagem. A Figura 2.3 apresenta a estrutura de uma mensagem SOAP.

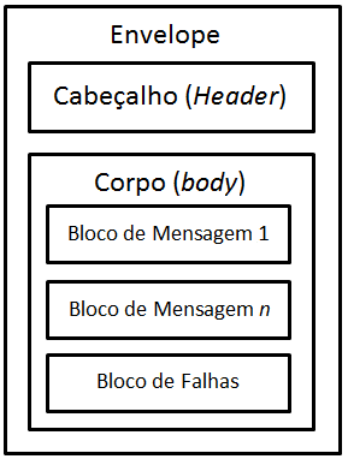

Figura 2.3: Estrutura de uma mensagem SOAP

Elemento *Header*

O elemento *Header* é opcional e quando existe, deve conter informações específicas sobre a mensagem SOAP. Normalmente, é nessa seção que são enviadas informações sobre o contexto da execução de um serviço. Por exemplo, se a execução do serviço necessita de pagamento, no *header* podem ser enviados os dados bancários do cliente para autorizar o débito. Outro cenário comum é enviar informações de segurança, como *tokens* de autenticação.

Elemento *Body*

O elemento *Body* é obrigatório e é ele que contem a informação que está sendo enviada ao *Web Service*. O *Body* pode conter um ou mais blocos de mensagens, também denominados de Partes (*Parts*). Dentro das Partes é que estão efetivamente os valores que são enviados ao serviço.

Elemento *Fault*

Erros e exceções de negócio podem ocorrer durante a execução do serviço, em *Web Services* os erros são retornados ao cliente através de Falhas SOAP (SOAP Faults). O elemento opcional *Fault* serve para indicar códigos e mensagens de erro provenientes da execução do serviço.

2.3 Computação na Nuvem

A computação na nuvem (*Cloud Computing*) surgiu recentemente como um novo paradigma para hospedagem e prestação de serviços através da Internet. A computação em nuvem é atraente para os donos de negócios pois elimina a necessidade de planejar antecipadamente a infraestrutura necessária para suportar as soluções, e permite que as empresas iniciem com uma operação pequena e que possam crescer de acordo com o aumento na demanda de serviços. No entanto, apesar do fato de computação em nuvem oferecer grandes oportunidades para a indústria de TI, o desenvolvimento da tecnologia está recem iniciando, com muitas questões e desafios em aberto para serem abordados [ZHA10].

A ideia principal por trás da computação na nuvem não é nova. John McCarthy na década de 1960 já previa que as instalações de computação seriam fornecidas ao público em geral como um utilitário (*computing as a utility*) [PAR66]. O termo "nuvem" também não é novo, nos anos de 1990 foi utilizado para descrever grandes redes ATM (*Asynchronous Transfer Mode*). No entanto, foi depois que o CEO do Google, Eric Schmidt, usou a palavra para descrever o modelo de negócio de prestação de serviços através da Internet em 2006, que o termo realmente começou a ganhar popularidade [ZHA10]. Desde então, o termo "computação na nuvem" tem sido usado principalmente como um termo de *marketing* em uma variedade de contextos para representar muitas ideias diferentes.

A falta de uma definição formal juntamente com um grande apelo publicitário, gerou confusão e desconfiança com relação à usabilidade e aplicabilidade de computação na nuvem. Visando obter um consenso, em [VAQ08] foram comparados mais de 20 definições diferentes a partir de uma variedade de fontes para chegar em uma definição padrão. Neste trabalho, adotamos a definição fornecida pelo NIST (*National Institute of Standards and Technology*) [MEL09].

> **Definição NIST:** "Computação na nuvem é um modelo conveniente para permitir o acesso sob demanda à uma rede de recursos compartilhados (por exemplo redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente fornecidos e

> liberados com o esforço mínimo de gerenciamento ou pouca interação com o prestador de serviços."

Ou seja, computação na nuvem é um termo genérico para qualquer funcionalidade que envolva a entrega de serviços hospedados na Internet.

Alguns benefícios da utilização de computação na nuvem são:

- É pago de forma incremental, economizando dinheiro às organizações;
- As organizações podem armazenar mais dados do que no ambiente computacional privado;
- Equipe de TI não precisa consumir tempo em manter os sistemas atualizados;
- Oferece muito mais flexibilidade do que métodos tradicionais;
- A informação fica disponível a partir de qualquer lugar;
- Separação do serviço de negócio da infraestrutura necessária para executá-lo;
- Flexibilidade para escolher vários fornecedores que prestam serviços confiáveis e escaláveis, ambientes de desenvolvimento e infraestrutura, com o custo medido conforme a utilização;
- Natureza elástica da infraestrutura para rapidamente alocar e de-alocar recursos, tornando a solução altamente escalável com base na demanda;
- Redução de custos devido à eficiência operacional e implantação mais rápida de novos serviços.

Computação na nuvem é dividida em três categorias: Infraestrutura como Serviço (IaaS - *Infrastructure as a Service*), Plataforma como Serviço (PaaS - *Platform as a Service*) e *Software* como Serviço (SaaS - *Software as a Service*).

2.3.1 Infraestrutura como Serviço

O termo "Infraestrutura como Serviço" refere-se a uma combinação de hospedagem, *hardware* e fornecimento de serviços básicos necessários para executar um serviço na nuvem [IaaS11]. Este modelo ganhou popularidade nos últimos anos devido à flexibilidade, escalabilidade e confiabilidade que ela oferece às empresas e instituições que não tem os meios ou o interesse de gerenciar suas próprias infraestruturas de rede.

IaaS é a categoria mais simples em computação na nuvem. Provedores de IaaS entregam o acesso a computadores (máquinas físicas ou, mais frequentemente, máquinas virtuais), sistemas de arquivos, *firewalls*, balanceadores de carga e redes. Prestadores de IaaS fornecem estes recursos sob demanda, alocando-os dinamicamente de seus grandes em centros de dados. Neste modelo, o cliente é responsável pela instalação, correção e manutenção dos sistemas operacionais e *softwares* aplicativos. Prestadores de serviços de nuvem tipicamente cobram com base no número de utilização

dos recursos, isto é, o custo irá refletir a quantidade de recursos alocados e consumidos. São exemplos de ofertas de IaaS o *Amazon Web Services Elastic Compute Cloud* (EC2) [CLO11] e o *Secure Storage Service* (S3) [PAL08].

2.3.2 Plataforma como Serviço

Plataforma como Serviço é o conceito de que alguém pode fornecer o infraestrutura (como em IaaS) adicionando-se uma certa quantidade de *software* no produto entregue, como por exemplo o sistema operacional, sistemas gerenciadores de banco de dados, suporte a linguagens de programação entre outros. Em PaaS, além do *hardware*, são fornecidas todas as bases necessárias para a construção e implantação de aplicações que irão atender a necessidade do negócio. Isto facilita o desenvolvimento e implantação de aplicações sem o custo e a complexidade de compra e gestão da infraestrutura básica, proporcionando todas as facilidades necessárias para apoiar o ciclo de vida completo de construção e entrega de aplicações web e serviços, totalmente disponível a partir da Internet.

Normalmente, a infraestrutura utilizada para fornecer PaaS consiste de um ambiente virtualizado e clusterizado de grades (*grids*) computacionais. Algumas ofertas de PaaS possuem uma API ou linguagem de programação específica. Por exemplo, o Google AppEngine é uma solução de PaaS aonde desenvolvedores podem construir aplicações em Python ou em Java.

2.3.3 *Software* como Serviço

Software como serviço, também conhecido como *software* sob demanda (*on-demand software*), é uma forma de distribuição e comercialização de *software*. No modelo SaaS o fornecedor do serviço se responsabiliza por toda a infraestrutura necessária para a disponibilização do sistema e o cliente utiliza o *software* via Internet, pagando um determinado valor pela utilização. A forma de cobrança pode variar entre o pagamento recorrente de um favor fixo, ou de acordo com a intensidade do uso. Este modelo se caracteriza por retirar o processamento da custódia do cliente e disponibilizá-lo através da Internet (ou outra infraestrutura de rede) na forma de um serviço de fácil acesso e que possa ser consumido por diversos clientes. Esta transferência só foi possível devido ao grande avanço da Internet e da larga adoção de SOA.

Comparando com o modelo tradicional de distribuição de *software* (no qual o cliente adquire a licença de uso e se responsabiliza pela instalação e manutenção em produção) podemos destacar as seguintes vantagens do modelo SaaS para os clientes:

- Não exige que o cliente crie uma estrutura e capacite os profissionais para manter o sistema funcionando, permitindo que ela se foque no seu negócio;
- Permite uma abordagem gradual de implantação, podendo começar com poucas licenças e expandir conforme tiver um retorno positivo do seu investimentos, reduzindo os riscos e o tempo para o retorno do investimento;

- Permite aumentar ou reduzir as licenças ao longo do tempo, de acordo com as necessidades do negócio;
- Redução de custos do quadro funcional das equipes de TI;
- Redução dos custos de *Hardware* e gastos com energia elétrica, pois o provedor do serviço é responsável pela manutenção dos servidores.

Uma pesquisa realizada pelo instituto *AMR Research*, indicou que o mercado de SaaS cresce 20% a cada ano, enquanto que o modelo tradicional cresce menos do que 10% [LAC06]. Um questionário realizado pela consultoria *ThinkStrategies* mostrou que um terço dos 118 entrevistados já estava usando SaaS, e outro terço já estava avaliando a possibilidade e utilização para os próximos 12 meses [KAP05].

2.4 Trabalhos Relacionados

Com o objetivo de entender e levantar o estado da arte no quesito *frameworks* para exposição de ferramentas como serviço, foram analisadas diversas iniciativas de pesquisa, direta e indiretamente relacionadas com este trabalho.

Existem diversas pesquisas na área de *Software* como serviço na exposição de ferramentas, algumas focadas na interação entre o usuário e o sistema, outras focadas no ponto de vista do fornecedor de SaaS e outras focadas no modelo de implementação. Entretanto, nenhuma possui o objetivo deste trabalho, criar um *framework* reutilizável para a exposição de ferramentas de simulação analítica.

- Em [JER05] os autores propõem um *framework* em SOA para modelar serviços colaborativos. Primeiramente é definido um modelo genérico, aonde a arquitetura de um serviço de colaboração pode ser dividida em quatro componentes: A lógica do serviço, dados do serviço, conteúdo do serviço e o perfil do serviço. Este modelo é então mapeado para uma arquitetura orientada a serviços. Ao final, o *framework* é elaborado baseado nos resultados obtidos a partir do mapeamento. Como contribuição do trabalho, o *framework* resultante oferece aos desenvolvedores de serviços colaborativos uma camada de colaboração, aonde são fornecidos serviços para acesso simultâneo de recursos (*locking*), controle de apresentação, gestão de presença do usuário, gerenciamento organizacional e controle de comunicação.
- Em [CAL11] é proposto o *framework* CloudSim que visa oferecer os recursos necessários para a simulação de ambientes computacionais em nuvem. É extensível, facilmente adaptável e permite a criação de simulações em grande escala com alto grau de customização. Através do CloudSim pesquisadores e desenvolvedores podem testar o desempenho dos serviços da aplicação em desenvolvimento em um ambiente controlado e de fácil configuração. Com base nos resultados da avaliação do CloudSim, é possível modificar e configurar os serviços para

otimizar o desempenho. As principais vantagens da utilização CloudSim para teste de desempenho inicial incluem: (i) eficácia tempo: exige pouco esforço e tempo para implementação de um ambiente de testes em nuvem para as aplicações e (ii) flexibilidade e aplicabilidade: os desenvolvedores podem modelar e testar o desempenho de seus serviços de aplicações em ambientes heterogêneos de Cloud com pouco de programação e esforço de implantação.

- Em [CAN06] os autores apresentam um modelo de interação baseado em um autômato finito para migrar sistemas legados em *Web Services*. É utilizada uma técnica de encapsulamento (*wrapping*) de caixa preta, ou seja, o sistema legado não é modificado e uma nova camada entre o usuário e o sistema é introduzida para mapear os valores de entrada e saída. O *wrapper* tem a responsabilidade de (i) interagir com o sistema legado durante a execução de cada possível de interação, (ii) efetuar a troca de dados e comandos entre o sistema e o usuário até que a resposta final seja obtida. O autômato finito é definido como: AF = (S, T, A, S_{in}, S_{fin}), aonde S é o conjunto de estados de interação, T é o conjunto de transições entre estados, A o conjunto de ações do usuário e S_{in} e S_{fin} são os estados inicias e finais da interação. Esta abordagem de encapsulamento com caixa preta é a mesma adotada na solução proposta por este trabalho que será detalhada no Capítulo 4.

- Em [FRE10] é proposto o modelo CloudMIG para migrar sistemas legados para ambientes na nuvem. É apresentada uma abordagem semiautomática para auxiliar na reengenharia de sistemas. O processo é composto por seis atividades. (i) Extração: Aonde é feito um levantamento manual da atual arquitetura do sistema a ser migrado. Também são obtidos dados estatísticos do comportamento da solução. (ii) Seleção: Com base nos resultados obtidos na atividade de extração, é necessário selecionar manualmente as características do ambiente na nuvem de destino, como por exemplo, a quantidade de instâncias de máquinas virtuais, a quantidade de *threads* de trabalho por instância, etc. (iii) Geração: Nesta atividade são produzidos três artefatos, a arquitetura destino, o modelo de mapeamento e uma listagem das características do sistema legado que não estão conforme a com arquitetura de destino na nuvem. Esta listagem contem as partes da aplicação que deverão ser reconstruídas manualmente pela reengenharia. (iv) Adaptação: A quarta atividade é utilizada para ajustar manualmente requisitos específicos que não puderam ser cobertos pela fase de geração. (v) Avaliação: Com o objetivo de avaliar a arquitetura produzida, a atividade de avaliação verifica os resultados das fases Geração e Adaptação. A avaliação envolve análises estáticas e dinâmicas da arquitetura resultante. (vi) Transformação: Esta atividade compreende a transformação real do sistema legado a partir da arquitetura gerada e melhorada para o ambiente de nuvem selecionado.

- Em [ZHA04] é apresentado um processo de reengenharia para decompor sistemas legados em sub-rotinas que possam representar um serviço a ser exposto. Um algoritmo baseado em análise aglomerativa (*Agglomerative Clustering Analysis*) [BEE00] é utilizado para transformar o código procedural em um modelo orientado a objetos. Funções, procedimentos e classes são

definidas como entidades de aglomeração e funcionalidades são utilizadas para medir a similaridade entre as entidades. As funcionalidades são definidas conforme a ocorrência de nomes identificadores no código legado. Um dendrograma é obtido como resultado da execução do algoritmo e este dendrograma é manualmente analisado para identificar as funcionalidades chaves do aplicativo legado. Sub-árvores do dendrograma são manualmente selecionadas e a qualidade desta analise manual influencia diretamente na granularidade dos serviços, impactando diretamente a coesão e o acoplamento dos mesmos. As sub-rotinas selecionadas passam então por um processo manual de refatoração e os serviços são gerados utilizando-se as interfaces de entrada e saída.

3. FERRAMENTAS DE SIMULAÇÃO ANALÍTICA

O objetivo deste trabalho é propor um *framework* para disponibilizar ferramentas de simulação analítica na forma de *Web Services*, utilizando o modelo de distribuição de *software* como serviço. Como forma de validar o trabalho proposto, foram feitas duas implementações do *framework* para disponibilizar duas ferramentas de simulação já existentes. Os simuladores PEPS [BRE07] e PLAT [FER11b] foram os escolhidos para a prova de conceito do *framework*. Ambos simuladores são implementações que visam solucionar redes de autômatos estocásticos. Este capítulo introduz aos conceitos de redes de autômatos estocásticos e em seguida apresenta uma visão geral sobre as ferramentas PEPS e PLAT.

3.1 Redes de Autômatos Estocásticos

O formalismo de Redes de Autômatos Estocásticos foi inicialmente proposto por Plateau em 1984 [PLA85]. No início da década de 90 as primeiras soluções foram formalizadas para modelos em escala de tempo contínua e discreta [PLA91]. Também chamada de SAN (*Stochastic Automata Networks*), essa técnica permite que modelos *Markovianos* possam ser descritos de forma compacta e eficiente.

A ideia básica das Redes de Autômatos Estocásticos é que esta descreva um modelo global de um sistema em diversos subsistemas (submodelos) quase independentes entre si, onde cada dois ou mais submodelos interagem somente em alguns casos. Estes subsistemas, definidos como autômatos estocásticos, são caracterizados por três aspectos: estados, transições e eventos [BAL07]. A denominação de estocásticos atribuída aos autômatos deve-se ao fato de que o tempo é tratado como uma variável aleatória com distribuição exponencial [BRE01].

Autômato estocástico é um modelo matemático de um sistema que possui entradas e saídas discretas. O sistema pode se encontrar em qualquer um dentre o número finito dos estados do sistema ou das configurações internas. O estado interno em que o sistema se encontra sumariza as informações sobre as entradas anteriores e indica ainda o que é necessário para determinar o comportamento do sistema para as entradas seguintes [HOP06].

Baseado nessa definição, pode-se descrever um autômato estocástico como um conjunto finito de estados e um conjunto finito de transições entre esses estados. O estado local do sistema modelado em SAN é o estado individual de cada autômato do modelo. Por sua vez, o estado global do mesmo é definido pela combinação dos estados locais de todos os autômatos que compõem o modelo. A mudança do estado global do sistema dá-se pela mudança do estado local de qualquer um dos autômatos do modelo.

3.2 Estados Locais e Globais

O estado de um autômato resume toda a informação referente a seu passado, pois a entrada passada é necessária para determinar qual será o próximo comportamento adotado pelo autômato mediante novas entradas. Ou seja, para um determinado conjunto de estados, um sistema poderá assumir somente um estado a cada momento, e este estado irá variar de acordo com a nova entrada recebida [BRE01, PLA91].

O estado individual de cada autômato é chamado de estado local. O estado global de uma SAN é definido como a combinação de todos os estados locais de cada autômato componente da SAN.

A mudança de um determinado estado local para outro é feita através de transições. As transições são construções que indicam a possibilidade de mudança entre um estado e outro. No entanto, cada transição necessita ter ao menos um evento associado a ela para que essa possa ser disparada [CHA05].

3.3 Eventos

Evento é a entidade do modelo responsável pela ocorrência de uma transição, a qual muda o estado global do modelo. Um ou mais eventos podem estar associados a uma transição e esta é disparada através da ocorrência de qualquer um dos eventos a ela associada.

No formalismo SAN existem dois tipos eventos: locais e sincronizantes.

Eventos locais são os eventos que alteram o estado local de um único autômato do modelo. São utilizados para demonstrar o comportamento do processamento individual de cada autômato.

Eventos sincronizantes são os eventos que alteram o estado de dois ou mais autômatos do modelo simultaneamente. Estes eventos são utilizados para descrever a interação entre dois ou mais autômatos do modelo.

3.4 Taxas e Probabilidades Funcionais

Um outro tipo de denominação para os eventos locais e sincronizantes é que ambos podem ser chamados de transições funcionais. Isto ocorrerá quando suas taxas não forem constantes, ou seja, tem-se uma função do estado local de outros autômatos da SAN, avaliada conforme os estados atuais do modelo. Logo, as taxas funcionais podem ser colocadas tanto em transições locais como em transições sincronizadas, e estas podem ser definidas por funções que refletem a avaliação dos estados atuais da rede de autômatos estocásticos em questão.

A Figura 3.1 apresenta um modelo em SAN com 2 autômatos. O autômato A possui três estados e o autômato B possui dois estados. Esta SAN utiliza cinco eventos, aonde os eventos e1, e2, e3 e e5 são eventos locais e o evento e4 é sincronizante, visto que este está presente em ambos autômatos. Este evento e4 possui ainda uma probabilidade associada a diferentes transições no autômato A. Além disso, o evento e5 possui ainda uma função $f1$ associada à sua taxa de ocorrência. A tabela com as taxas de ocorrência dos eventos para este modelo pode ser vista em 3.1.

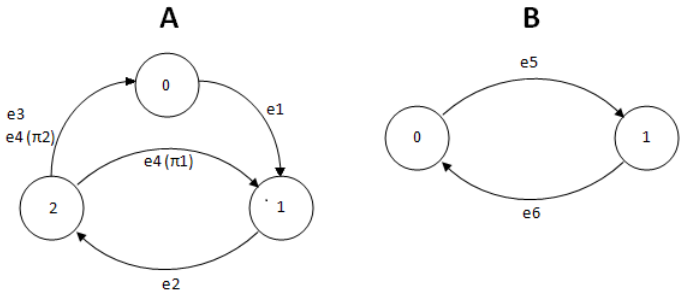

Figura 3.1: Exemplo de uma rede de autômatos estocásticos

Evento	Taxa de Ocorrência	Tipo
e1	$t1$	local
e2	$t2$	local
e3	$t3$	local
e4	$t5$	sincronizante
e5	$f1$	local

Tabela 3.1: Taxa de ocorrência dos eventos do modelo SAN da Figura 3.1

$$f1 = \begin{cases} \lambda_1 \text{ se autômato A está no estado 0;} \\ 0 \text{ se autômato A está no estado 1;} \\ \lambda_2 \text{ se autômato A está no estado 2.} \end{cases}$$

Como pode-se observar na definição da função $f1$, a taxa de ocorrência da transição do estado 0 para o estado 1 do autômato B é igual a λ_1 quando autômato A estiver está estado 0, igual a zero quando o autômato A estiver no estado 1 (neste caso, a transição não ocorrerá) e igual a λ_2 quando o autômato A estiver no estado 2.

Outra possibilidade dentro do formalismo é que para cada modelo pode-se definir uma função de atingibilidade. É uma função booleana que determina os estados atingíveis do modelo dentro do espaço total de estados. Esta função definirá os estados atingíveis do modelo SAN. Quando esta for igual a 1, significa que todos os estados do modelo são atingíveis. Geralmente isto não representa a realidade, em muitas situações existem condições que previnem alguns estados serem atingidos.

3.5 A Ferramenta PEPS

O projeto PEPS (*Performance Evaluation of Parallel Systems*) iniciou no final dos anos 80, com o objetivo de criar um *software* para modelar e calcular soluções numéricas para redes de autômatos estocásticos. Ele usa um descritor Kronecker para representar as transições do modelo em uma forma eficiente em termos de consumo de memória [FER11b]. O modelo é solucionado usando este descritor e aplicando-se métodos VDP (*Vector-Descriptor Product*), como *Shuffle* [FER98] ou *Split* [CZE07].

A primeira versão do PEPS foi apresentada em [PLA88] e foi implementada como uma multiplicação vetor-matriz simples, aonde as colunas da matriz são geradas, coluna por coluna, a cada iteração.

Em 2000 uma nova versão foi lançada, esta versão implementa um conjunto de novos algoritmos [FER98]. A maior contribuição desta versão é o método de multiplicação do vetor-descritor, o algoritmo *Shuffle*. Com este método, cada termo do tensor produto multiplica uma parte do vetor, evitando assim a geração de partes da matriz completa. Após todas as multiplicações do tensor, obtemos uma multiplicação vetor-matriz completa.

A versão 2003 do PEPS trouxe uma nova interface de compilação, manipulação de vetores esparsos e avaliações de funções em tempo de execução (*just-in-time*). A nova interface de compilação é mais intuitiva e compacta. Também foram introduzidos os operadores de replicação de autômatos. A implementação do formato de vetores esparsos, permitiu o cálculo exclusivo dentro do espaço de estados alcançáveis. Este tipo de abordagem reduz consideravelmente o espaço de armazenamento para alguns determinados modelos. Outra característica é a avaliação de funções *just-in-time*, este método gera, para cada função descrita no modelo SAN um código de linguagem de programação. Os códigos gerados são compilados e ligados com o método de solução do PEPS e são chamados toda vez que uma função precisa ser avaliada [BEN03b].

O último lançamento oficial do PEPS é a versão 2007. Nesta versão a ferramenta foi dividida em módulos independentes, com o intuito de facilitar a manutenção e o desenvolvimento de novas funcionalidades. Nesta versão as funcionalidades de replicação de autômatos foram aperfeiçoadas, possibilitando a replicação de autômatos com diferentes transições, eventos ou taxas [BRE07].

O PEPS é uma ferramenta em desenvolvimento, e apesar de novas versões não terem sido lançadas, algumas melhorias foram implementadas, com por exemplo a execução paralela do algoritmo *Split* [FRA08].

A entrada da ferramenta é um arquivo texto descrevendo um modelo SAN em uma determinada gramatica. O PEPS carrega este modelo e constrói um sistema de equações usando os eventos e as taxas informadas. Este sistema de equações é resolvido utilizando-se soluções iterativas e ao final um arquivo texto é gerado, contendo a solução do sistema de equações.

Todo o código fonte da ferramenta é escrito na linguagem de programação C++, e apesar de ser um código aberto, até hoje não foram feitos esforços na direção de facilitar a integração da ferramenta com outras aplicações ou outros sistemas.

3.5.1 Modelagem de SAN na Ferramenta PEPS

A Interface Textual

O módulo de interface textual mantém a característica chave do formalismo SAN: a modularidade. O PEPS incorpora uma abordagem baseada em grafos que é próximo aos modelos semânticos. Nesta abordagem cada autômato é representado por um grafo, em que os nodos são os estados e os arcos representam transições pela ocorrência de eventos. Esta descrição textual foi mantida simples,

extensível e flexível [BEN03a].

- É bastante simples, porque há poucas palavras reservadas, o suficiente para delimitar os diferentes níveis de modularidade;
- Extensível porque a definição do modelo SAN é executado hierarquicamente;
- Flexível porque a inclusão de estruturas de replicação permite a reutilização de autômatos idênticos, e a construção de autômatos repetindo blocos de estado com o mesmo comportamento.

A descrição da SAN é composta por 5 blocos, conforme a Figura 3.2, que são facilmente localizados com os seus delimitadores (em negrito). As outras palavras reservadas são indicadas com a fonte itálica. Os símbolos "<" e ">" indicam informação mandatória e os símbolos "{" e "}" indicam informação opcional.

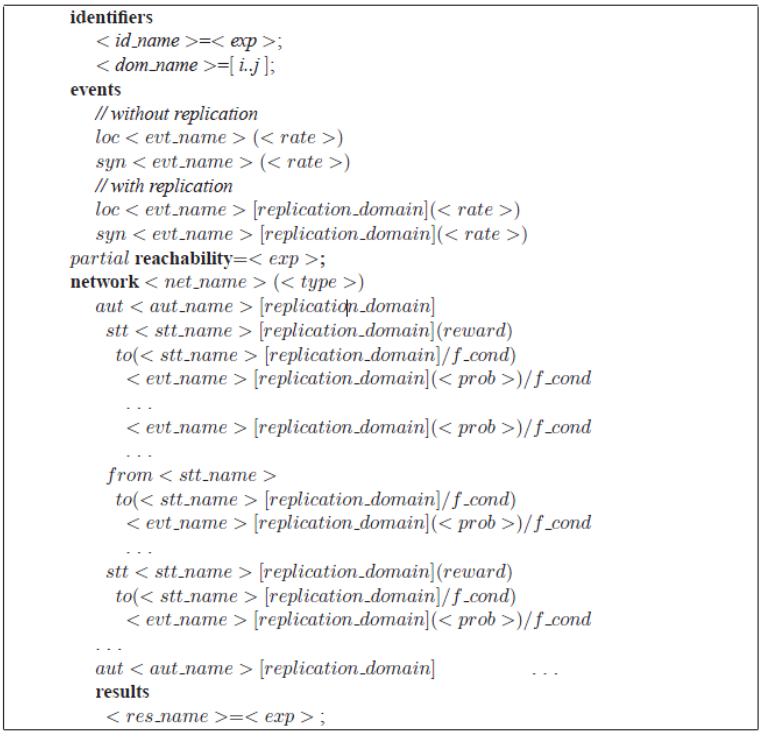

Figura 3.2: Estrutura Modular do formato textual da SAN

Identificadores e Domínios

O primeiro bloco, identificadores (*identifiers*), contêm todas as declarações dos parâmetros: valores numéricos, funções, ou conjuntos de índices que serão utilizados na definição do modelo. Um identificador (< id name >) pode ser qualquer *string* de caracteres alfanuméricos. Os valores numéricos e funções são definidos de acordo com a sintaxe da linguagem C. Em geral, as expressões são similares as expressões matemáticas com lógicas e operadores aritméticos. Os argumentos

destas expressões podem ser números constantes, identificadores de autômatos ou identificadores de estados. Neste último caso, as expressões são funções definidas no espaço de estados do modelo SAN. Por exemplo, "o número de autômatos no estado n0" (que dado um resultado inteiro) pode ser expresso como "nb n0". Uma função que retorna o valor 4 se dois autômatos (A1 e A2) estão em diferentes estados, e o valor 0, caso contrário, é expresso como "(stA1! $=$ stA2) * 4". Operadores de comparação retornam o valor "1" para o resultado verdadeiro ou o valor "0" para o resultado falso. É possível declarar quantos identificadores forem necessários e a sua definição é a seguinte:

- "$<$ *id_name* $>$" É um identificador de expressão que começa com uma letra e é seguido por uma sequência de letra ou números. O comprimento máximo de uma expressão é 128 caracteres;
- "$<$ *dom_name* $>$" É um identificador de domínio. É um conjunto de índices. Um domínio pode ser por um intervalo "[1..3]", por uma lista "[1,2,3]" ou por lista de intervalos "[1..3,5,7..9]". Identificadores podem ser usados para definir um intervalo "[1..ID1,5,7..ID2]", onde ID1 e ID2 são identificadores com valores constantes. Em todos os casos, o domínio deve respeitar uma ordem crescente de valores;
- "$<$ *exp* $>$" É um número real ou uma expressão matemática. Um número real tem um dos seguintes formatos: Um número inteiro, tal como "12345"; um número real com ponto flutuante, tal como "12345,6789"; um número real com mantissa e exponente, tal como "12345.6789e+100".

Conjuntos de índices são usados para definir números de eventos, autômatos ou estados que podem ser descritos como replicações. Um grupo de autômatos replicados de A com o conjunto em índices [0..2; 5; 8..10] define o conjunto contendo os autômatos A[0]; A[1]; A[2]; A[5]; A[8]; A[9]; e A[10].

Eventos

O bloco de eventos define cada evento do modelo dado:

- seu tipo (local ou sincronizante);
- seu nome (um identificador);
- seu índice (uma constante ou função previamente definida no bloco de identificadores).

Adicionalmente, eventos podem ser replicados usando os conjuntos de índices (domínios). Esta facilidade pode ser usada quando eventos com os mesmos índices aparecem em um conjunto de autômatos.

- "*loc*" define o tipo de evento como um evento local;

- "*syn*" define o tipo do evento como evento sincronizado;

- "< *evt_name* >" o identificador do evento inicia com uma letra e é seguida por uma sequência de letras ou números. O tamanho máximo de um identificador é de 128 caracteres.

- "*[replication_domain]*" é um conjunto de índices. O *replication domain* deve ter um identificador definido no bloco de identificadores. Um evento pode ser replicado em até três níveis. Cada nível é definido um por *replication domain*. Por exemplo, um evento replicado em dois níveis é definido como <*evt_name*> *[replication domain][replication domain]*;

- "< *rate* >" define a taxa dos eventos. Deve ser um identificador de expressão declarado no bloco de identificadores.

Rede de autômatos

O bloco rede (*network*) é o principal componente do descritor SAN e é composto de uma estrutura hierárquica. É formado de um conjunto de autômatos, aonde cada autômato é composto por um conjunto de estados. Cada estado é conectado a outro estado através da identificação de um evento.

- "< *net_name* >" define o nome do modelo. É uma *string* de caracteres alfanuméricos iniciados por uma letra;

- "< *type* >" define o tipo do modelo. A definição prevê dois tipos, contínuo (*continuous*) e discrito (*discrete*), porém atualmente a versão atual suporta somente modelos contínuos;

- "< *aut_name* >" define o nome de um autômato. É um identificador alfanumérico e pode ser usado na definição de funções;

- "< *number_of_replications* >" é um conjunto de índices. Pode ser definido com um intervalo "[0..4]" ou por um identificador de domínio declarado;

- "< *stt_name* >" É o identificador de um estado, o qual pode ser usado para avaliação de funções;

- "< *(evt_name)* >" É o identificado do evento que dispara a transição de estado. O evento deve ser declaro no bloco *events*.

- "< *(prob)* >" É a taxa de probabilidade da ocorrência do evento. Deve ser utilizando quando um evento possui mais do que um estado de destino. Pode ser um número real ou um identificador declarado.

Função de Atingibilidade

O bloco *reachability* é uma função definindo o espaço de estados atingíveis do modelo SAN. Usualmente, é uma função Booleana, que retorna um valor diferente de zero para os estados atingíveis do conjunto completo de estados do modelo SAN. Um modelo onde todos os estados são atingíveis tem a função de atingibilidade definida como qualquer constante diferente de zero, e.g., o valor 1. Opcionalmente, a função de atingibilidade parcial pode ser definida pela adição da palavra reservada "*partial*".

Resultados

Neste bloco são definidas as funções usadas para computar os índices de desempenho do modelo. Os resultados dado pelo PEPS são os valores integrais dessas funções com a distribuição estacionária do modelo.

3.6 A Ferramenta PLAT

A ferramenta PLAT (*Production Lines Analysis Tool*) foi criada para calcular soluções exatas para linhas de produção confiáveis [FER11b]. Como resultado, o PLAT retorna o rendimento (*throughput*), a utilização dos *buffers* (*buffer occupation*), a utilização dos servidores (*server utilization*) e o tempo *sojourn* (*sojourn time*) de uma linha de produção, considerando-se que a quantidade de estações confiáveis, a taxa de serviço e o tamanho dos *buffers* são enviados como parâmetros para a ferramenta.

O diferencial desta ferramenta é calcular a solução exata para a linha de produção com grandes espaços de estados, enquanto que outras soluções não proveem valores precisos o suficiente. A partir dos valores de entrada, a ferramenta gera automaticamente o modelo Markoviano correspondente a linha de produção, na forma de uma rede de autômatos estocásticos. Esta SAN gerada é então analisada e resolvida utilizando-se ferramentas auxiliares específicas.

O PLAT é dividido em três módulos. O primeiro gera o modelo SAN equivalente a linha de produção especificada pelo usuário. O segundo módulo determina a solução numérica exata do modelo. E o terceiro módulo calcula as métricas de performance da linha de produção.

3.6.1 Módulo 1 - Geração do modelo

Neste primeiro modulo, o usuário informa o número de estações da linha de produção serial, assim como a capacidade do *buffer* de cada estação e também a taxa de serviço para cada servidor. Este módulo gera automaticamente um modelo SAN equivalente a linha de produção informada, ou seja, é feita transformação da linha de produção em um modelo em redes de autômatos estocásticos. O algoritmo que realiza esta transformação pode ser visto em [FER11a]. O produto cartesiano de todos os estados locais do modelo definem o PSS (*Product State Space*), no entanto, considerando-se um estado inicial, apenas um subconjunto do PSS é atingível de fato. Este novo subconjunto é

denominado de RSS (Reachable State Space) [FER11a]. Ambos PSS e RSS são calculados neste primeiro módulo do PLAT, e com base nos valores obtidos o métodos de solução mais apropriado é recomendado para a execução no próximo módulo.

3.6.2 Módulo 2 - Solução do modelo

O segundo módulo é o responsável pela solução do modelo SAN previamente gerado. O PLAT fornece distribuições probabilísticas estacionárias ou distribuições probabilísticas transientes. A solução exata do modelo é computada através de ferramentas pré-existentes para resolução de redes de autômatos estocásticos, como o PEPS2007 [BRE07] ou o GTAEXPRESS [CZE09]. O PEPS2007 é uma versão específica da ferramenta já apresentada na Seção 3.5 deste mesmo Capítulo. O GTAEXPRESS é outra ferramenta para solução de modelos SAN, se caracteriza por armazenar as transições do modelo como uma matriz esparsa *Harwell-Boeing* [STE94] que é calculada a partir dos estados atingíveis do modelo, o qual é armazenado em uma estrutura MDD (Multi-valued Decision Diagrams) [SAL09].

De acordo com as características da linha de produção e dos valores RSS e PSS do modelo SAN gerado, o PLAT sugere a ferramenta de solução mais adequada. Cabe ao usuário aceitar a sugestão ou selecionar outra forma de solução. Assim que definida a forma de solução, as ferramentas auxiliares para solução do modelo são executadas e as probabilidades de distribuição dos estados do modelo são computadas.

3.6.3 Módulo 3 - Métricas de performance

Dada as probabilidades de distribuições dos estados do modelo SAN, o PLAT calcula as medidas de desempenho de cada estação, e seu respectivo *buffer*, da linha de produção modelada. Mais especificamente, ele calcula par a par (estação e *buffer*) a taxa média de transferência (*throughput*), taxa média de ocupação do *buffer* (*buffer occupation*), taxa de ocupação do servidor da estação (*station server utilization*) e o tempo sojourn (*sojourn time*).

4. FRAMEWORK PARA DISPONIBILIZAR FERRAMENTAS DE SIMULAÇÃO ANALÍTICA

Este capítulo apresenta, na Seção 4.1, o problema a ser solucionado. Logo após, a Seção 4.2 descreve a solução proposta e em seguida, as Seções 4.3 e 4.4 apresentam as duas implementações realizadas com o objetivo de verificar se a proposta deste trabalho atende ao problema identificado. Ao final, na Seção 4.5 são apresentados os resultados das análises de *overhead* introduzidos na adoção do *framework*.

4.1 Problema

Atualmente, existe uma crescente tendência de se disponibilizar soluções, aplicações e ferramentas na forma de serviços que possam ser utilizados por usuários ou integrados a outras aplicações. Neste contexto, o Capítulo 3 apresentou duas ferramentas de simulação analítica, denominadas PEPS e PLAT, que não possuem mecanismos de integração. Ambas as ferramentas possuem dependências com relação ao sistema operacional suportado e bibliotecas de compilação, além de possuírem apenas uma interface de usuário em modo texto. Estas restrições exigem um conhecimento prévio por parte dos usuários que não está diretamente relacionado ao objetivo dessas ferramentas. Assim sendo, dois problemas são identificados:

- Impossibilidade de integração com outras ferramentas.
- Grande complexidade na utilização das ferramentas por parte dos usuários.

4.2 Framework

Com o objetivo de solucionar os problema identificados na Seção 4.1, este trabalho propõe um *framework* reutilizável, para expor ferramentas de simulação analíticas na forma de *Web Services*, utilizando como forma de distribuição o conceito de *Software* como Serviço. Apesar do foco deste trabalho estar em simulação analítica, este *framework* pode ser facilmente estendido para suportar praticamente qualquer ferramenta na qual se deseje disponibilizar através de SaaS.

O objetivo central é reduzir a complexidade na utilização, fornecer acessibilidade a qualquer usuário e possibilitar a integração das ferramentas. Para atender aos problemas apontados, dois módulos distintos são propostos. O primeiro visa atender ao requisito de integração e o segundo o requisito de usabilidade.

4.2.1 Módulo *Web Service*

O primeiro módulo é uma interface *Web Services* para receber as requisições dos usuários, transformar o pedido em uma execução da ferramenta e retornar ao usuário o resultado obtido.

Seguindo os conceitos de *Software* como Serviço, o usuário não precisa se preocupar como ou aonde a execução da ferramenta será feita, o usuário preocupa-se apenas em realizar o pedido e receber o resultado. A Figura 4.1 representa a visão do ponto de vista do cliente consumidor do serviço, aonde a única interação é realizada através da interface. Pedidos de execução são recebidos e retornados ao cliente através de *Web Services* trafegando sobre o protocolo de transporte HTTP.

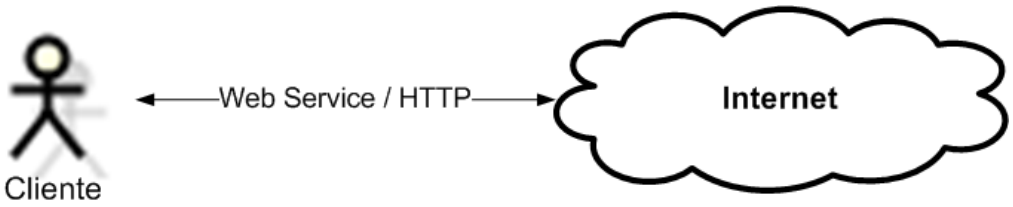

Figura 4.1: Visão do ponto de vista do usuário

O *framework* foi projetado para suportar chamadas *Web Service* síncronas e assíncronas. Independentemente do modo de execução, ao receber os pedidos de clientes, a interface analisa os valores enviados e verifica se a entrada é válida. Uma mensagem de erro é retornada sempre sincronamente caso existam inconsistências.

- **Execução Síncrona:** Durante uma execução síncrona, o cliente envia o pedido e fica aguardado, em um estado bloqueado, o término por completo da execução da ferramenta. A Figura 4.2 representa este cenário. No contexto deste trabalho, as ferramentas disponibilizadas através do *framework* possuem a tendência de consumir grandes quantidades de tempo durante sua execução, pois se tratam de ferramentas de simulação iterativa que buscam um algo grau de precisão. Em situações como estas, idealmente buscamos minimizar o impacto do tempo de execução para o usuário, e isto é feito utilizando-se chamadas assíncronas.

- **Execução Assíncrona:** No caso de execuções assíncronas, os padrões de *Web Services* possibilitam duas alternativas, *One Way* e *Request/Callback*.

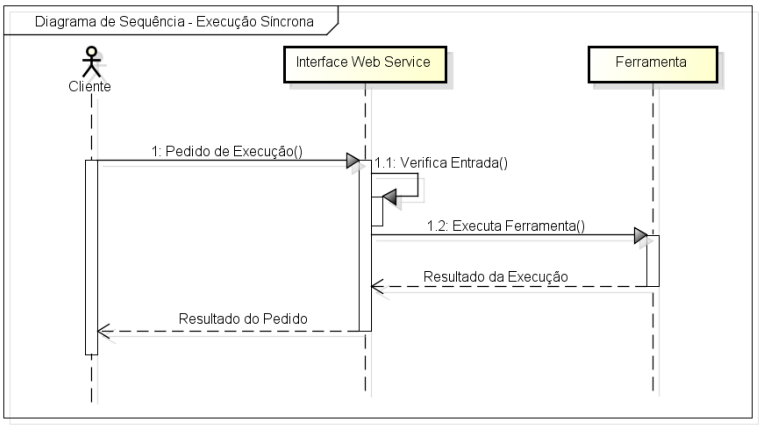

Figura 4.2: Diagrama de Sequência - Execução Síncrona

– ***One Way*:** Em chamadas assíncronas *One Way*, não existe nenhuma resposta ao consumidor do serviço. Ao efetuar a invocação do serviço, o cliente envia um identificador único de execução e não fica em estado bloqueado nem recebe qualquer informação de resposta. Neste momento o cliente apenas tem ciência de que o pedido de execução foi recebido, ou seja, não houve erros na comunicação de dados durante a chamada. Para se obter o resultado de uma execução *One Way* é necessário que seja disponibilizado, por parte do produtor de serviços, um serviço auxiliar. Este novo serviço deve receber o mesmo identificador único e verificar o estado da execução em questão. Caso a execução já tenha terminado, os resultados são retornados ao cliente. Fica sob responsabilidade do consumidor do serviço chamar o serviço auxiliar periodicamente para verificar o resultado da execução. O diagrama de sequência da Figura 4.3 apresenta o cenário de uma execução assíncrona *One Way*.

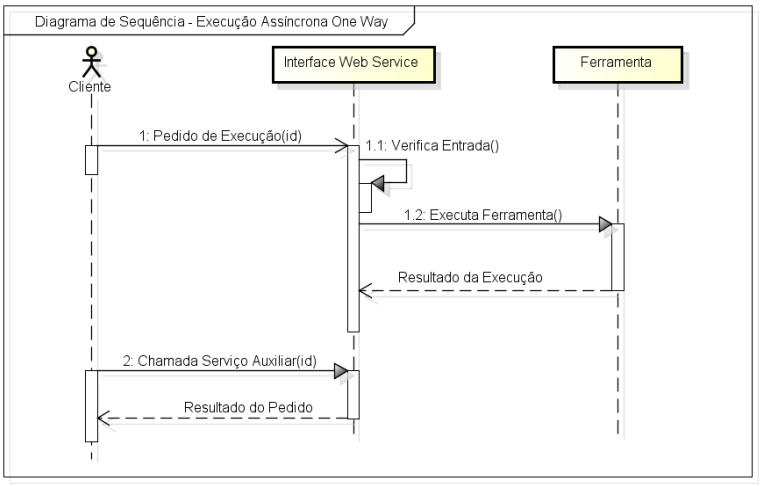

Figura 4.3: Diagrama de Sequência - Execução Assíncrona *One Way*

– ***Request/Callback*:** Em chamadas assíncronas *Request/Callback*, o consumidor é responsável por disponibilizar um serviço que será chamado pelo produtor, quando do final da execução. Este serviço que fica exposto pelo cliente é denominado de *Callback*. Durante a chamada assíncrona, o consumidor envia um identificador único juntamente com a assinatura do serviço *Callback* e não aguarda pelo retorno da chamada. Neste momento o produtor irá executar e ao final irá invocar o serviço de *Callback*, enviado o identificar único juntamente com o resultado da execução. O diagrama de sequência da Figura 4.4 apresenta este cenário.

O *framework* disponibiliza nativamente tanto o modo síncrono quanto os modos assíncronos. Entretanto, algumas customizações são necessárias durante a criação de uma solução. As validações dos valores de entrada e a transformação dos valores de entrada em parâmetros para a ferramenta sendo exposta, devem ser codificados manualmente durante a fase de implementação, respeitando-se os nomes de métodos e parâmetros provenientes do *framework*.

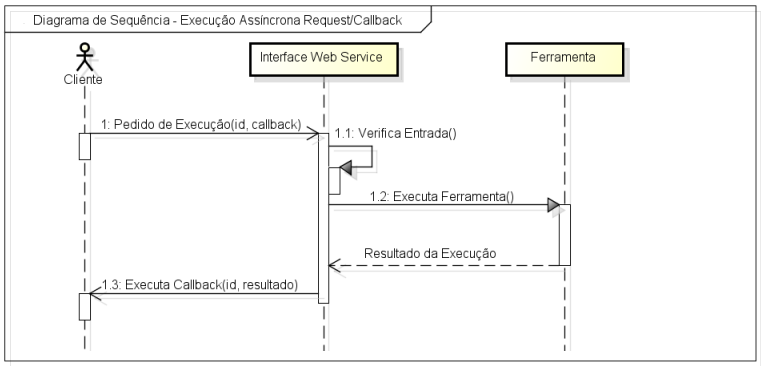

Figura 4.4: Diagrama de Sequência - Execução Assíncrona *Request/Callback*

Após a transformação dos valores de entrada vem a fase de execução. Com exceção de casos específicos, nesta fase não são necessárias customizações, basta apenas que seja informado nos arquivos de configuração os caminhos no sistema de arquivos onde estão armazenados os binários executáveis da ferramenta. De posse dos parâmetros transformados e dos caminhos para os executáveis, o *framework* realiza o controle de execução de processos. Uma chamada ao sistema operacional é efetuada para a inicialização de um novo processo, paralelo ao processo aonde o *framework* esta sendo executado, para a execução da ferramenta. Ao longo da execução o *framework* monitora tanto os arquivos de log quanto os sinais do sistema operacional para identificar interrupções por erro ou o término com sucesso da execução.

Terminada a fase de execução, os resultados precisam ser traduzidos para o formato esperado pelo consumidor do serviço. Neste ponto, novamente se faz necessária uma customização de código, aonde os resultados obtidos pela execução da ferramenta devem ser transformados no formato que o cliente espera receber.

O *framework* provê ainda um controle de enfileiramento de execuções. Um semáforo de P posições assegura que somente P execuções paralelas ocorram ao mesmo tempo. Este controle se faz necessário para evitar uma sobrecarga sobre o sistema operacional encarregado dos processos. O valor de P deve ser especificado nos arquivos de configuração do *framework*.

Para viabilizar as execuções assíncronas *One Way*, se faz necessário o armazenamento de resultados. Um banco de dados relacional é utilizado para este fim e o diagrama entidade-relacionamento da Figura 4.5 demonstra as entidades utilizadas para este armazenamento. Este banco de dados também é utilizado para os serviços auxiliares que efetuam por exemplo: autenticação de usuários, reexecução de pedidos e listagem de execuções anteriores. A partir desta estrutura relacional simples, também é possível a extração de métricas de utilização e desempenho da ferramenta.

A entidade Execução, representa a tabela principal do sistema. E é nela que ficam armazenadas as execuções e seus respectivos resultados. Associadas a tabela principal estão as entidades Usuário, Grupo_Execução, Parâmetros_Entrada e Arquivo_Entrada. A tabela Usuário armazena dados pessoais dos usuários e é utilizada para realizar a autenticação do sistema. Este mecanismo de segurança se faz necessário para evitar que execuções prévias e seus resultados, cujo conteúdo possivelmente contenha dados sigilosos, sejam visualizadas indiscriminadamente por outras pessoas.

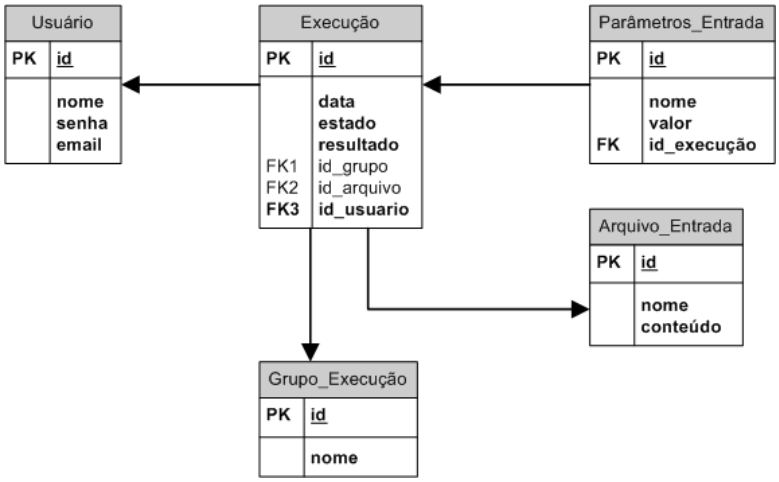

Figura 4.5: Diagrama ER do banco de dados

O *framework* possui também uma opção de execução em lote, aonde pode-se definir um intervalo de variação e um valor de incremento para popular automaticamente parâmetros da execução. De posse do intervalo e do incremento, o *framework* gera uma combinação de execuções que são enfileiradas na fila de processamento. A tabela Grupo_Execução serve para agrupar execuções em lote. A partir do momento em que todas as execuções do lote estão finalizadas, a interface habilita um serviço que gera gráficos com os resultados das variações da execução em lote. Esta funcionalidade é particularmente interessante em simulações analíticas, aonde rotineiramente diversas simulações são executadas, variando-se valores de entrada, com o objetivo de identificar quais variações impactam mais no resultado final. A Figura 4.6 apresenta um exemplo desse gráfico. A tabela Parâmetros_Entrada salva os valores dos parâmetros utilizados na execução e a tabela Arquivo_Entrada é utilizada quando os parâmetros são enviados na forma de um arquivo texto.

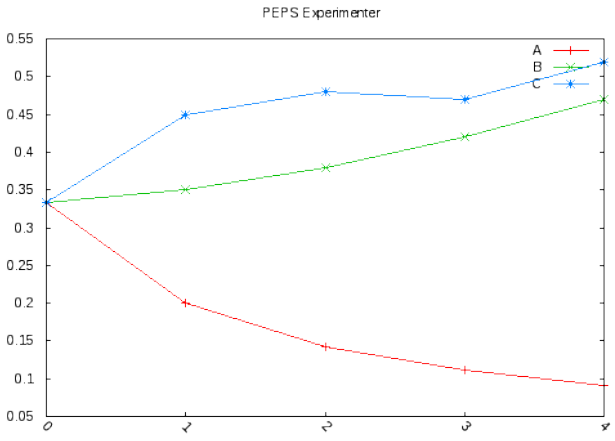

Figura 4.6: Exemplo de gráfico gerado na execução em lote

Os serviços expostos pelo módulo *Web Service* são descritos a seguir e o arquivo WSDL correspondente pode ser encontrado no Apêndice B.

- **Autentica:** Serviço que recebe como parâmetro um usuário e uma senha. Utilizando cripto-

grafia SHA-1 [EAS01], verifica se as informações enviadas conferem com os dados armazenados no banco de dados.

- **BuscaUsuario:** Serviço que recebe uma *String* com o nome de usuário (*login*) e retorna os dados armazenados no banco de dados para este usuário.

- **SalvaUsuario:** Serviço que recebe os dados pessoais de um usuário e armazena no banco de dados um nove registro.

- **SolicitaNovaSenha:** Serviço que recebe o endereço de e-mail do usuário e gera uma nova senha. Esta nova senha é gerada utilizando criptografia SHA-1 e é enviada através de e-mail ao usuário.

- **BuscaExecução:** Serviço que retorna os parâmetros e resultados de uma execução, selecionada através do identificador único.

- **ExecutaFerramenta:** Serviço que executa a ferramenta disponibilizada na forma de serviço. Este serviço recebe os parâmetros para a execução e faz a comunicação com o sistema operacional que irá executar a ferramenta. Primeiramente é feita a transformação dos dados recebidos para o formato de entrada esperado pela ferramenta. Em seguida ocorre a verificação do semáforo para entrada na seção crítica, assim que a entrada for permitida ocorre uma chamada ao sistema operacional para criação do processo que irá executar a ferramenta. Ao final da execução, os resultados são armazenados no banco de dados. O Algoritmo 4.1 apresenta a lógica deste serviço.

Algoritmo 4.1: Lógica do serviço *ExecutaFerramenta*.

```
1: parametros[] ← transformaParametros() {função customizada}
2: if semaforo.adquire() then
3:    processo ← iniciaProcesso(parametros)
4:    while processo.terminou() = false do
5:       leLog()
6:    end while
7:    if processo.status() = sucesso then
8:       armazenaResultado(processo)
9:    end if
10:   semaforo.libera()
11:   return processo.status()
12: end if
```

O módulo *Web Services* foi implementado utilizando a linguagem de programação *Java* na versão 1.6 e em um ambiente Linux/Ubuntu versão 10.10, codinome *the Maverick Meerkat*. Roda sob um servidor de aplicação *Apache Tomcat* na versão 7.0.11. Por se tratar de uma solução que segue os padrões J2EE (*Java2 Platform Enterprise Edition*), pode ser instalado em qualquer plataforma

ou em qualquer servidor de aplicação compatível com J2EE. Além disso, este módulo e utiliza uma série de tecnologias e *frameworks* de código aberto. Dentre elas podemos citar:

- ***Apache Axis* 2:** Axis 2 [PER06] é um projeto mantido pela *Apache Software Foundation* e é uma implementação em Java para o protocolo SOAP. Proporciona grande agilidade no desenvolvimento, pois possui geradores de código capazes de criar toda a infraestrutura necessária para a exposição de classes Java via Web Services;

- ***Apache Derby*:** É uma implementação puramente em *Java* de um sistema gerenciador de banco de dados (SGBD). Este banco de dados foi selecionado por ser de código aberto e por ser um *software* considerado leve. É utilizado para armazenar as solicitações de execução e seus respectivos resultados.

- ***Hibernate* 3:** É uma *framework* de mapeamento objeto-relacional. Sua função é transformar tabelas e registros do banco de dados em objetos que possam ser manipulados em uma programação orientada a objetos. Está atualmente na terceira versão e é a implementação mais popular para a especificação JPA (Java Persistence API) [BAU06];

- ***Spring Framework*:** É um *framework* para facilitar a implementação de padrões de projeto (*Design Patterns*) [VLI95]. Neste caso são utilizados os padrões Inversão de Controle (*Inversion of Control* ou IoC) e Injeção de Dependência (*Dependency Injection*) com o objetivo de ter um baixo acoplamento entre as classes da aplicação. [JOH03].

4.2.2 Módulo *Web*

O segundo módulo é responsável por atender ao requisito de usabilidade, reduzindo a complexidade na manipulação das ferramentas. Este módulo é uma aplicação *Web* que atua como um intermediário entre o usuário final e os *Web Services* expostos pelo primeiro módulo. Esta aplicação recebe os valores de entrada do usuário, faz a comunicação com os serviços e retorna ao usuário o resultado, formatado amigavelmente, como páginas HTML (*HyperText Markup Language*) [RAG99].

Neste cenário, o consumidor do serviço não é o usuário final, mas sim a aplicação *Web* intermediária. É nesta aplicação que serão criadas as mensagens SOAP e são feitas as chamadas aos serviços. Do ponto de vista do usuário, a ferramenta se comporta como uma página comum da Internet, aonde entradas são enviadas e os resultados são exibidos através do navegador.

O Diagrama de Casos de Uso da Figura 4.7 apresenta as operações da aplicação que são disponibilizadas ao usuário.

Os Casos de Uso são descritos abaixo:

- **Faz *Login*:** O usuário fornece o seu *login* e senha. A aplicação *Web* envia ambos os valores para o serviço de autenticação do módulo *Web Service* que por sua vez confronta os dados com a informação armazenada no banco de dados e retorna ao módulo *Web* a confirmação.

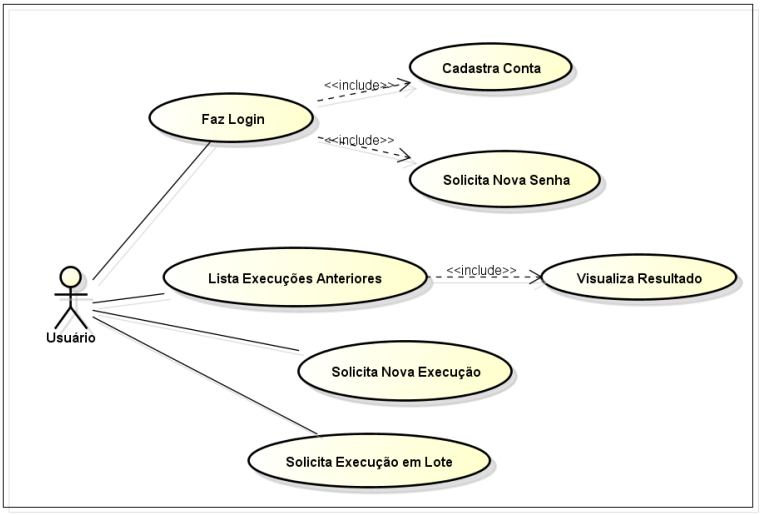

Figura 4.7: Diagrama de Casos de Uso

Tabela 4.1: Caso de Uso Faz *Login*

Característica	Descrição
Ator Primário	Usuário
Objetivo	Permitir ao usuário acesso ao sistema.
Fluxo Principal	1. O usuário acessa o endereço da aplicação através do navegador. 2. Usuário insere *login* e senha. 3. Autenticação é verificada pelo módulo *Web Service*. 4. Sistema encaminha o usuário ao Caso de Uso "Lista Execuções Anteriores".
Fluxo Alternativo 1	3a. Login e senha não são válidos. 3b. O Sistema exibe a mensagem "Login ou senha Inválidos". 3c. Retorna ao passo 2.
Fluxo Alternativo 2	2a. Usuário não possui conta de acesso. 2b. Usuário acessa o *link* "Criar Conta". 2c. Sistema encaminha o usuário ao Caso de Uso "Cadastra Conta".
Fluxo Alternativo 3	2a. Usuário não lembra da senha. 2b. Usuário acessa o *link* "Esqueci minha senha". 2c. Sistema encaminha o usuário ao Caso de Uso "Solicita Nova Senha".

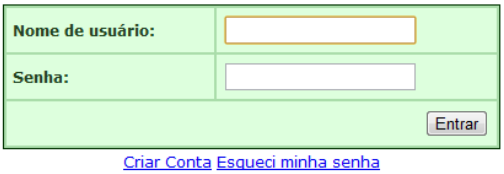

Figura 4.8: Interface do Usuário - Fazer Login

Caso retorno verdadeiro, o acesso as outras funcionalidades da aplicação é garantido (Tabela 4.1 e Figura 4.8).

- **Cadastra Conta:** Antes de poder acessar a aplicação, o usuário necessita efetuar o cadastramento de uma conta de acesso. Neste Caso de Uso, o usuário informa seus dados pessoais

como: nome de usuário (*login*), senha, nome completo, e-mail e linguagem de preferência. Utilizando o serviço *BuscaUsuario* do módulo *Web Service*, o sistema verifica se o usuário já esta cadastrado. Caso ainda não esteja, o serviço *SalvaUsuario* armazena o registro de uma nova conta. Todos os campos da tela são obrigatórios (Tabela 4.2 e Figura 4.9)..

Tabela 4.2: Caso de Uso Cadastra Conta

Característica	Descrição
Ator Primário	Usuário
Objetivo	Cadastrar uma conta de usuário.
Fluxo Principal	1. O usuário preenche os campos da tela. 2. Sistema verifica se todos os campos foram preenchidos. 3. Sistema verifica se o campo e-mail foi corretamente inserido. 4. Sistema verifica se o usuário já existe através do serviço *BuscaUsuario*. 5. Sistema salva a conta do usuário através do serviço *SalvaUsuario*.
Fluxo Alternativo 1	2a. Usuário não preenche todos os campos. 2b. O Sistema indica os campos em branco. 2c. Retorna ao passo 1.
Fluxo Alternativo 2	3a. Usuário não preenche corretamente o campo e-mail. 3b. O Sistema exibe a mensagem "e-mail inválido". 3c. Retorna ao passo 1.
Fluxo Alternativo 3	4a. Usuário informa *login* já existente. 4b. O Sistema exibe a mensagem "Nome de usuário já existente". 4c. Retorna ao passo 1.

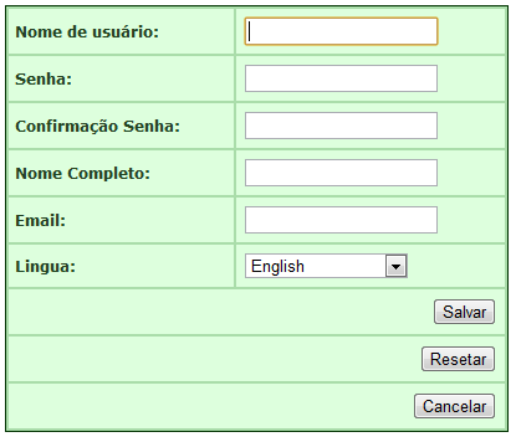

Figura 4.9: Interface do Usuário - Cadastra Conta

- **Solicita Nova Senha:** O usuário fornece o e-mail informado durante a criação da conta. A aplicação *Web* valida a entrada e encaminha o endereço de e-mail do usuário ao serviço *SolicitaNovaSenha* do módulo *Web Service*, que por sua vez irá gerar uma nova senha aleatoriamente e enviará via SMTP ao usuário (Tabela 4.3 e Figura 4.10).

- **Lista Execuções Anteriores:** O sistema retorna a listagem de todas as execuções efetuadas pelo usuário. A forma de apresentação dos itens deve ser customizada em cada implementa-

Tabela 4.3: Caso de Uso Solicita Nova Senha

Característica	Descrição
Ator Primário	Usuário
Objetivo	Gerar uma nova senha e enviar ao usuário via e-mail.
Fluxo Principal	1. O usuário informa o e-mail utilizado durante a criação da conta de acesso. 2. Sistema verifica se o valor informado é valido. 3. Sistema gera uma nova senha aleatória, criptografa utilizando SHA-1 e armazena o novo valor no registro de conta do usuário. 4. Sistema envia a nova senha via SMTP ao usuário
Fluxo Alternativo 1	2a. Usuário não informa um endereço de e-mail válido. 2b. O Sistema exibe a mensagem "e-mail inválido". 2c. Retorna ao passo 1.

Figura 4.10: Interface do Usuário - Solicita Nova Senha

ção do *framework*. Para cada registro apresentado, são possíveis as operações de exclusão, visualização dos parâmetros e a visualização dos resultados (Tabela 4.4).

Tabela 4.4: Caso de Uso Lista Execuções Anteriores

Característica	Descrição
Ator Primário	Usuário
Objetivo	Exibir as execuções previamente solicitadas e visualizar os resultados
Fluxo Principal	1. Sistema apresenta a listagem de execuções solicitadas. 2. Usuário acessa a opção de visualizar os resultados. 3. Sistema exibe os resultados da execução.
Fluxo Alternativo 1	2a. Usuário acessa a opção de exclusão. 2b. Sistema efetua a exclusão do registro de execução e atualiza a listagem do usuário. 2c. Retorna ao passo 1.
Fluxo Alternativo 1	2a. Usuário acessa a opção de visualização de parâmetros. 2b. Sistema informa os parâmetros que foram utilizados na execução selecionada. 2c. Retorna ao passo 1.

- **Visualiza Resultado:** O usuário seleciona a execução na qual deseja visualizar o resultado, a partir do Caso de Uso "Lista Execuções Anteriores". A aplicação *Web* executa o serviço *BuscaExecução* do módulo *Web Service*, passando como parâmetro o identificador único do registro. O serviço retorna todos os dados associados a execução solicitada e a aplicação *Web* deve customizar a formatação dos resultados ao usuário (Tabela 4.5).

- **Solicita Nova Execução:** Através de aplicação *Web*, o usuário informa os parâmetros desejados para efetuar uma nova execução. Os parâmetros são validados e então enviados ao

Tabela 4.5: Caso de Uso Visualiza Resultado

Característica	Descrição
Ator Primário	Usuário
Objetivo	Visualizar o resultado de uma execução
Fluxo Principal	1. Usuário seleciona a opção de visualizar resultados, a partir do Caso de Uso "Lista Execuções Anteriores". 2. Sistema executa o serviço *BuscaExecução* do módulo *Web Service*, passando como parâmetro o *id* da execução. 3. Sistema formata e retorna ao usuário os resultados associados à execução retornada pela serviço.
Fluxo Alternativo 1	3a. A execução selecionada ainda não finalizou ou terminou com erro. O serviço *BuscaExecução* não retorna dados de resultado 3b. Sistema informa ao usuário que não existem dados para visualização.

serviço *ExecutaFerramenta* do módulo *Web Service*. Neste serviço ocorre a transformação dos parâmetros em entradas para a ferramenta que será executada. Após a transformação é feito o teste de entrada da seção critica através do semáforo de P posições. No momento em que o semáforo permitir a entrada, é feita uma chamada ao sistema operacional para criação de um novo processo, paralelo ao processo do serviço, para a execução da ferramenta. Durante a execução, o *framework* monitora os arquivos de *log* e os sinais do sistema operacional para identificar o término ou a morte do processo. Ao final da execução, os resultados são inseridos no banco de dados para posterior visualização (Tabela 4.6).

Tabela 4.6: Caso de Uso Solicita Nova Execução

Característica	Descrição
Ator Primário	Usuário
Objetivo	Gerar uma execução da ferramenta.
Fluxo Principal	1. Usuário informa os parâmetros da execução. 2. Sistema transforma os parâmetros em entradas para a ferramenta e cria uma nova execu 3. Sistema armazena o resultado da execução no banco de dados. 4. Usuário é encaminhado ao Caso de Uso "Lista Execuções Anteriores".

- **Solicita Execução em Lote:** O usuário, além de informar os parâmetros para a execução, também indica intervalos de variação e incrementos para os valores dos parâmetros. A quantidade máxima de intervalos é fixada em três para evitar um número muito grande de execuções, visto que, é feita uma combinação de todas as possibilidades. Cada combinação gerada é executada através do serviço *ExecutaFerramenta*. Por exemplo, em uma solicitação aonde os parâmetros $P1$ e $P2$ possuem respectivamente os intervalos $[1..5]$ e $[1..3]$, ambos com incremento de 1. Serão geradas um universo de 15 execuções, sendo elas $\{\{P1$=1, $P2$=1$\}$, $\{P1$=1, $P2$=2$\}$, $\{P1$=1, $P2$=3$\}$, $\{P1$=2, $P2$=1$\}$, $\{P1$=2, $P2$=2$\}$, $\{P1$=2, $P2$=3$\}$, $\{P1$=3, $P2$=1$\}$, $\{P1$=3, $P2$=2$\}$, $\{P1$=3, $P2$=3$\}$, $\{P1$=4, $P2$=1$\}$, $\{P1$=4, $P2$=2$\}$, $\{P1$=4, $P2$=3$\}$, $\{P1$=5, $P2$=1$\}$, $\{P1$=5, $P2$=2$\}$, $\{P1$=5, $P2$=3$\}\}$ (Tabela 4.7 e

Figura 4.11).

Tabela 4.7: Caso de Uso Solicita Execução em Lote

Característica	Descrição
Ator Primário	Usuário
Objetivo	Gerar uma combinação de execuções.
Fluxo Principal	1. Usuário informa os parâmetros da execução. 2. Sistema identifica os parâmetros passíveis de variação e fornece a interface da Figura 4.11. 3. Usuário informa de um a três parâmetros para variação, com seus respectivos intervalos. 4. Sistema gera a combinação e executa cada uma através do serviço *ExecutaFerramenta*.

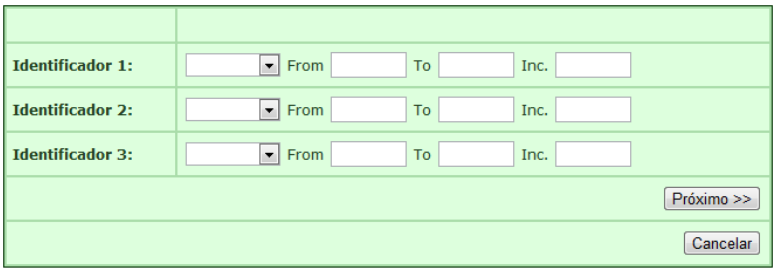

Figura 4.11: Interface do Usuário - Solicita Execução em Lote

Assim como o módulo *Web Service*, o módulo *Web* também foi implementado na linguagem de programação Java em sua versão 1.6 e em um ambiente Linux. Faz uso do mesmo servidor de aplicação *Apache Tomcat*, porém o conjunto de tecnologias utilizadas na implementação são distintos. Neste módulo foram utilizados os seguintes *frameworks*:

- ***Apache Struts* 2:** É um *framework* MVC (*Model-View-Controller*) para separação de camadas da aplicação [CAV04]. A divisão de camadas da proporciona um baixo acoplamento e uma maior coesão, melhorando a qualidade e consequentemente facilitando a manutenção do código. A camada de apresentação (*View*) é responsável por ditar como os dados devem ser organizados, independente de seus valores. *Model* é a representação lógica dos dados e *Controller* é a camada responsável por manipular os dados e fazer a ligação entre as outras duas camadas.

- ***Apache Axis* 2:** É uma biblioteca para criação de *Web Services* em *Java*. No caso do módulo *Web*, o *Apache Axis* é utilizado para a geração do código cliente, ou seja o código que irá acessar os serviços expostos no módulo *Web Service*.

Como forma de validar esta solução e provar a sua funcionalidade, duas implementações do *framework* foram realizadas. Foram disponibilizadas como um serviço as ferramentas PEPS e PLAT, apresentadas no Capítulo 3.

4.3 PEPS como Serviço

O objetivo principal em disponibilizar o PEPS como um serviço é possibilitar sua integração com outras ferramentas e tornar o acesso e a utilização por parte dos usuários mais simples. Além disso, no contexto deste trabalho, também serviu como forma de validar a funcionalidade do *framework*. O PEPS é uma ferramenta para solucionar Redes de Autômatos Estocásticos e possui como entrada um arquivo texto representando um modelo SAN. Este arquivo deve necessariamente seguir as regras da Interface Textual apresentada na Subseção 3.5.1.

Nesta implementação foram criadas três opções de execução, cada uma delas focada num perfil de usuário diferente.

- ***Simple Run* (Execução Simples):** Este é o modo de execução mais simples e visa atender aos usuários menos familiarizados com o PEPS. Nesta opção o usuário precisa apenas informar o arquivo contendo o modelo SAN e aguardar o término da execução. Todos os outros parâmetros necessários para a execução recebem um valor padrão. A Figura 4.12 apresenta a interface do usuário aonde se informa o arquivo a ser executado.

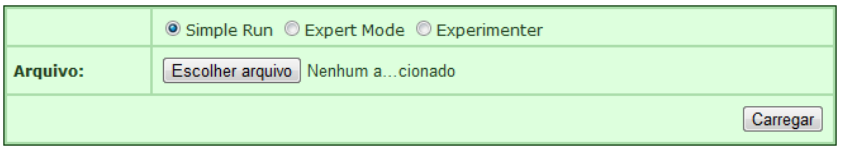

Figura 4.12: Interface do Usuário - PEPS - Execução Simples

- ***Expert Mode* (Modo Avançado):** Este modo é voltado para usuários com experiência na utilização do PEPS. Neste caso, além de informar o arquivo com o modelo SAN, o usuário também pode informar todas as outras opções de parâmetros suportadas pelo PEPS. A Figura 4.13 mostra a interface do usuário com todas as opções.

- ***Experimenter* (Experimentador):** Este modo é voltado para experimentos aonde diversas execuções serão feitas com o objetivo de identificar tendências nos modelos. Este modo faz uso da funcionalidade de execução em lote.

Independentemente do modo de execução escolhido, os resultados são visualizados através da listagem de execuções anteriores. A Figura 4.14 apresenta um exemplo de listagem. Através da listagem de execuções são disponibilizadas diversas operações, são elas:

- **Visualizar arquivo SAN:** Através do *link* em (A), é apresentado o arquivo texto do modelo SAN que foi utilizado na execução;

- **Agrupador de execução em lote:** O agrupador em (B) mostra como as execuções em lote são agrupadas;

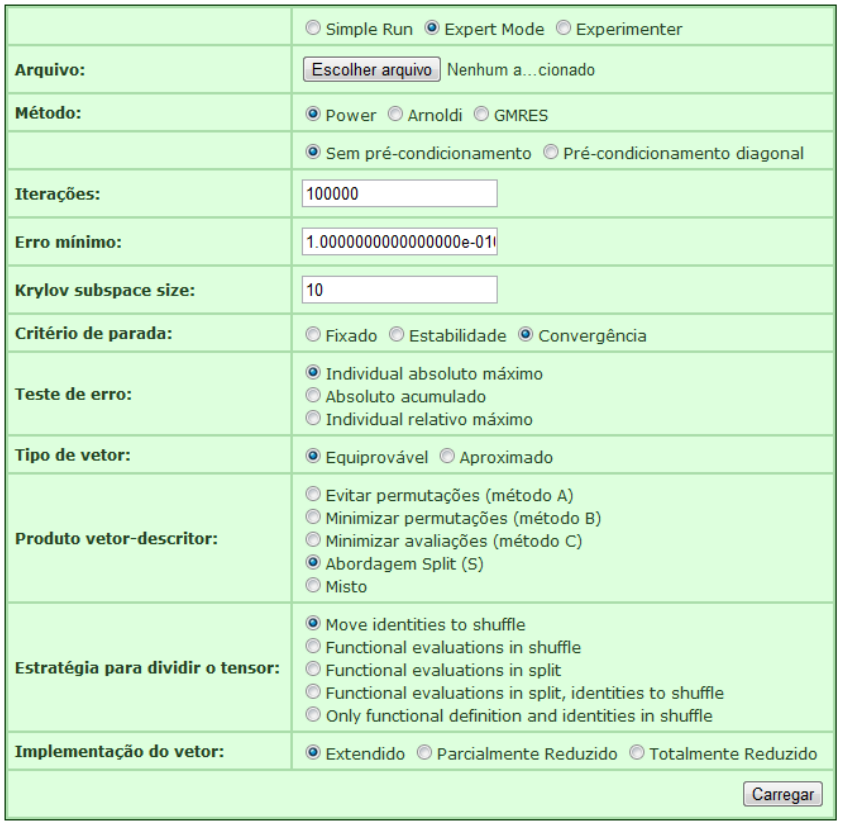

Figura 4.13: Interface do Usuário - PEPS - Execução Avançada

- **Visualizar o *log* de execução:** O ícone em (C) é utilizado para abrir o arquivo de *log* de uma execução; No caso de uma execução em andamento, a visualização é atualizada a cada três segundos;

- **Excluir execução:** O ícone em (D) exclui do bando de dados todas as informações relativas a execução. No caso de uma execução em andamento, além de efetuar a exclusão, o processo no sistema operacional também é destruído;

- **Visualizar resultado da execução em lote:** O ícone (E) é utilizado para visualizar os resultados da execução em lote na forma de uma planilha Excel;

- **Visualizar gráfico:** Os ícones em (F) e (G) são utilizados para exportar as informações da execução em lote na forma de gráficos. Ambos geram o mesmo resultado, porém o primeiro gera o gráfico em formato EPS (*Encapsulated PostScript*) e o segundo em formato PNG (*Portable Network Graphics*);

- **Visualizar matriz de resultados:** O ícone em (H) serve para apresentar a matriz de resultados da execução em lote. Esta matriz é a mesma utilizada para a formação dos gráficos;

- **Visualizar arquivo *TIM*:** A operação em (I) serve para visualizar o arquivo TIM gerado durante a execução do PEPS;

- **Visualizar resultado:** O ícone (J) é utilizado para visualizar o resultado de uma única execução. O resultado é formatado como uma planilha Excel;
- **Visualizar arquivo *VCT*:** A operação em (K) serve para visualizar o arquivo VCT gerado durante a execução do PEPS;

Figura 4.14: Interface do Usuário - PEPS - Listagem de Execuções Anteriores

Esta implementação está disponível na Internet através do endereço `http://paleoprospec.pucrs.br:16000/peps2011` desde o dia 22 de maio de 2011. O sistema possui 28 usuários cadastrados e até a presente data 1.043 execuções foram solicitadas. O tempo médio de execuções é de 1 minutos e 49 segundos.

4.4 PLAT como Serviço

O PLAT é uma ferramenta para obtenção de índices (*throughput*, *buffer occupation*, *server utilization* e *sojourn time*) de linhas de produção seriais. Assim como o PEPS, o objetivo em disponibilizar como serviço é possibilitar a integração com outras ferramentas e tornar a usabilidade mais amigável com o usuário final. No PLAT todos os parâmetros são informados pela interface do usuário. O usuário inicia informando a quantidade de estações (K) da linha de produção. Aonde K pode variar de 3 a 18. O sistema automaticamente atualiza a interface de acordo com a quantidade de estações. A Figura 4.15 apresenta, respectivamente, a formatação da tela para $K = 3$ e $K = 5$. O passo seguinte é informar o tamanho do *buffer* de cada estação (com exceção da primeira estação, que possui *buffer* infinito) e o valor da taxa de atendimento de serviço para cada estação.

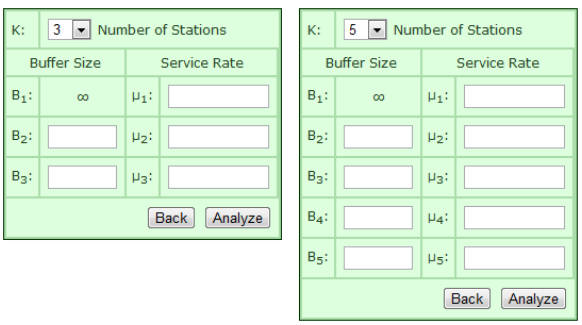

Figura 4.15: Interface do Usuário - PLAT - Dados da Linha de Produção

O PLAT analisa os parâmetros de entrada e calcula ambos PSS e RSS do modelo SAN equivalente à linha de produção. O método de solução é sugerido com base nestes valores. A Figura 4.16 demonstra a interface aonde o usuário seleciona o método e o modo de execução, aonde os modos disponíveis são estacionário e transiente. Em execuções transientes, o usuário precisa ainda informar a unidade de tempo e o estado inicial.

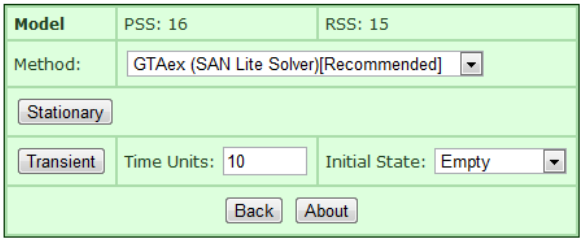

Figura 4.16: Interface do Usuário - PLAT - Seleção do Método de Execução

Após a seleção do método e modo de execução, o usuário é direcionado para a listagem de execuções anteriores. Esta listagem, além de permitir o acompanhamento das execuções em andamento, também permite ao usuário efetuar as seguintes operações:

Figura 4.17: Interface do Usuário - PLAT - Listagem de Execuções Anteriores

- **Excluir execução:** O ícone (A) exclui do bando de dados todas as informações relativas a execução;
- **Visualizar arquivo SAN:** O ícone (B) permite visualizar o arquivo SAN equivalente a linha de produção;
- **Visualizar resultado:** O ícone (C) apresenta o resultado da execução. Por exemplo a Figura 4.18;
- **Editar parâmetros de execução:** O ícone (D) permite ao usuário criar uma nova execução tomando como exemplo os parâmetros informados anteriormente;
- **Cancelar execução**: O ícone (E) permite ao usuário cancelar uma execução em andamento.

O PLAT como serviço está disponível, no endereço `http://paleoprospec.pucrs.br:16000/plat`, deste o dia 26 de setembro de 2011, possui 4 usuários e já obteve índices de 19 linhas de produção.

Performance Indexes							
	Buffer Size	Service Rate		Throughput	Buffer Occupation	Server Utilization	Sojourn Time
1	∞	1.0		-	-	-	-
2	1	1.0		0,6705	0,6228	0,8436	2,1872
3	1	1.0		0,6705	0,3949	0,6705	1,5889

Figura 4.18: Interface do Usuário - PLAT - Resultado

4.5 *Overhead* Introduzido

Os resultados apresentados a seguir são baseados em observações feitas a partir da execução das ferramentas PEPS e PLAT, utilizando-se modelos exatamente iguais, nas versões originais e nas versões implementadas através do *framework*. Estes resultados tem como objetivo analisar o *overhead* gerado para inclusão do *framework*. O objeto dessa análise é comparar a utilização de memória e o tempo de resposta de cada uma das soluções.

Para obter um resultado mais confiável, foram realizadas 30 execuções para cada teste. O consumo de memória foi calculado considerando-se a média de utilização durante toda a execução das ferramentas e o tempo de resposta foi obtido através da subtração do instante marcado pelo relógio interno do sistema operacional quando do término da execução e do instante inicial da execução. Para ambos, consumo de memória e tempo de resposta, foi realizada uma média aritmética simples sobre os valores obtidos sobre a quantidade de execuções. Nesta análise foram ignorados os tempos gastos em função de transmissões de dados sobre redes. Os testes foram efetuados em um único servidor, uma máquina virtual instalada com ambiente Linux/Ubuntu versão 10.10, com o CPU Intel(R) Core(TM)2 Duo T6600 de 2.20GHz e 2GBytes de memória RAM.

Nos testes de comparação de desempenho do PEPS foram utilizados dois modelos. O primeiro modelando o famoso problema do Jantar dos Filósofos [DIJ71], aonde foi utilizada uma mesa com 12 filósofos. E o segundo representando uma Rede de Filas Fechadas [CHA05] com um conjunto de 15 filas. Os modelos utilizados podem ser encontrados em [AFO12]. Já para as execuções da ferramenta PLAT, foram utilizadas duas configurações de linhas de produção. A primeira com 10 estações confiáveis e a segunda com 12 estações confiáveis.

4.5.1 Consumo de Memória

Antes de analisar o consumo de memória gasto pelas ferramentas, cabe aqui ressaltar que as versões implementadas através do *framework* fazem uso tanto do servidor de aplicação *Apache Tomcat* quanto do banco de dados *Apache Derby*. Análise feita no mesmo servidor de teste, constatou-se que o banco de dados consome cerca de 38 *Megabytes* de memória e o servidor de aplicação cerca de 119 *Megabytes*, totalizando em torno de 157 *Megabytes* de memória que são utilizados independentemente da execução das ferramentas.

A Tabela 4.8 apresenta o comparativo de consumo de memória obtidos nos testes do PEPS. Observa-se um incremento de 17% e de 20% sobre a utilização de memória, o que podemos consi-

derar um impacto significante.

Tabela 4.8: *Overhead* - Consumo de Memória - PEPS

Modelo	**Sem *Framework***	**Com *Framework***	**%**
Jantar dos Filósofos	897 Megabytes	1057 Megabytes	+17,8%
Rede de Filas Fechadas	780 Megabytes	940 Megabytes	+20,5%

Assim como nos testes anteriores, as execuções da ferramenta PLAT também apresentaram um *overhead* significante em termos de consumo de memória. Foram observados incrementos de 25% e 13% na versão exposta como serviço. A Tabela 4.9 mostra os resultados obtidos na comparação.

Tabela 4.9: *Overhead* - Consumo de Memória - PLAT

Modelo	**Sem *Framework***	**Com *Framework***	**%**
Linha de Produção - 10 Estações	639 Megabytes	801 Megabytes	+25,3%
Linha de Produção - 12 Estações	1.198 Megabytes	1.361 Megabytes	+13,6%

O gráfico da Figura 4.19 apresenta a comparação de consumo de memória entre todas as execuções e é possível identificar um incremento linear com relação ao *overhead* introduzido pela adoção do *framework*, ou seja, independentemente do tamanho do modelo a ser solucionado pelas ferramentas, a diferença de utilização de memória permanece a mesma. Pelos testes analisados, as diferenças obtidas estiveram em torno de 160 *Megabytes*, que se deve em grande parte pela adoção do servidor de aplicação e do banco de dados. Entretanto, este *overhear*, mesmo sendo significativo, nos dias de hoje é facilmente absorvível pelas infraestruturas de *hardware*.

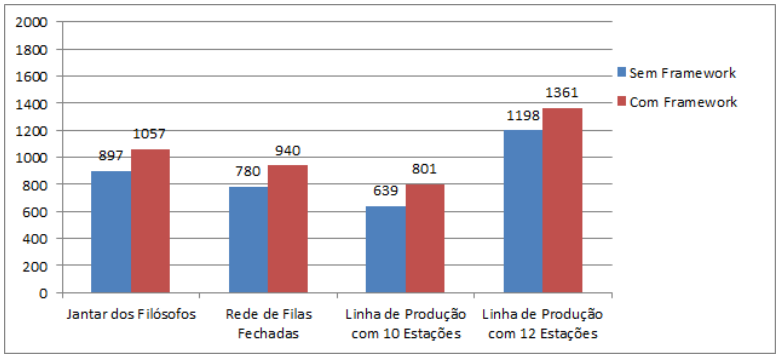

Figura 4.19: *Overhead* - Consumo de memória

4.5.2 Tempo de Resposta

Para a análise de *overhead* sobre o tempo de resposta foram desconsiderados os tempos de interação humana, necessários na utilização das ferramentas, assim como os tempos gastos em transmissões de dados. Restando assim, apenas o tempo efetivo de processamento para obtenção da solução numérica dos modelos testados. O tempo de resposta foi obtido subtraindo-se o instante de término de processamento do instante de início de processamento. Os instantes inicial e final

são obtidos através do arquivo de *log* que registra o processamento de uma execução. Este arquivo registra, utilizando o relógio interno do sistema operacional, as etapas do ciclo de execução das ferramentas. É considerado o instante inicial o momento registrado na primeira operação de escrita realizada no arquivo de *log* e instante final o momento da última operação de escrita.

A Tabela 4.10 apresenta os tempos de resposta, medidos em segundos, obtidos nos testes do PEPS. Verifica-se um incremento de, em média, apenas 5 segundos no tempo total de execução, representando 2,45% e 0,24% a mais no tempo de espera. Ao contrário do que foi verificado na análise de consumo de memória, neste caso o *overhead* gerado é praticamente desprezível.

Tabela 4.10: *Overhead* - Tempo de Resposta - PEPS

Modelo	**Sem *Framework***	**Com *Framework***	**%**
Jantar dos Filósofos	204 segundos	209 segundos	+2,45%
Rede de Filas Fechadas	1.639 segundos	1.643 segundos	+0,24%

Seguindo a mesma tendência, os testes realizados com a ferramenta PLAT também mostraram um acréscimo em torno de 5 segundos no tempo de processamento. Neste caso, o incremento representou um adicional de apenas 0,55% e 0,02% do tempo total. Na Tabela 4.11 são apresentados os tempos médios, em segundos, obtidos nos testes.

Tabela 4.11: *Overhead* - Tempo de Resposta - PLAT

Modelo	**Sem *Framework***	**Com *Framework***	**%**
Linha de Produção - 10 Estações	904 segundos	909 segundos	+0,55%
Linha de Produção - 12 Estações	16.315 segundos	16.319 segundos	+0,02%

Comparando-se todos os resultados dos testes de tempo de resposta, nota-se um *overhead* constante, sempre em torno de 5 segundos. Tendo em mente que a principal usabilidade das ferramentas PEPS e PLAT é solucionar problemas que exigem grande quantidade de processamento, e em geral, problemas que exigem a execução da ferramenta por diversos minutos ou horas, o incremento de 5 segundos no tempo total de resposta pode ser considerado desprezível tendo em vista os benefícios que o *framework* agrega. O gráfico da Figura 4.20 mostra a comparação dos resultados obtidos nos testes de tempo de resposta para ambas as ferramentas.

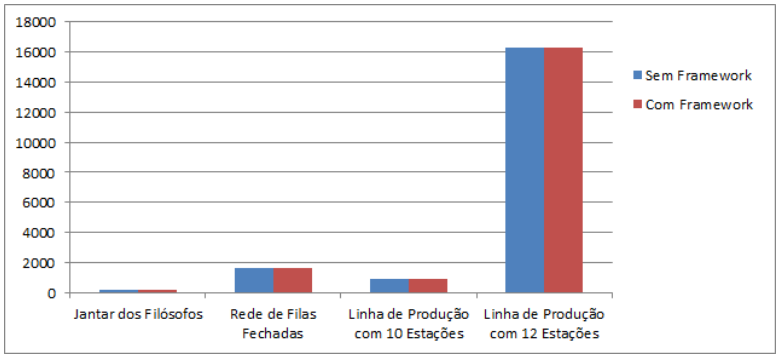

Figura 4.20: *Overhead* - Consumo de memória

A análise geral do *overhead* introduzido indica um aumento considerável com relação ao consumo de memória e um aumento aceitável no tempo de resposta. Este incremento considerável não é considerado um problema ou impedimento na implementação de soluções utilizando o *framework*, exceto para situações aonde o consumo de memória seja um fator crítico. Para estes cenários, a sugestão para tentar mitigar o excesso de memória consumida é efetuar a troca do servidor de aplicação e do banco de dados por soluções que possuam um *footprint* menor, visto que foi identificado que o aumento no consumo está relacionando à infraestrutura utilizada pelo *framework*. Por exemplo, o *Apache Tomcat* poderia ser substituído pelo *Jetty* e o banco de dados *Apache Derby* pelos bancos *SQLite* ou pelo *HyperSQL DB*. Por seguir fielmente a especificação J2EE, não seria necessária nenhuma alteração no código do *framework* para utilizar servidores de aplicação e banco de dados diferentes.

5. CONCLUSÃO

Este trabalho apresentou, no Capítulo 4, a especificação e implementação de um *framework* para a disponibilização de ferramentas de simulação analítica na forma de *software* como serviço. Apesar do foco deste trabalho estar em simulação analítica, este *framework* poderia ser facilmente adaptado para disponibilizar qualquer solução como um serviço. Foram definidos métodos, assinaturas e comportamentos dos serviços criados pela solução, assim como identificado os pontos aonde se faz necessário a customização de código para transformar a execução do *software* em uma chamada *Web Service*.

A abordagem proposta utiliza uma técnica de encapsulamento de caixa preta, ou seja, a aplicação original não é modificada e introduz-se uma nova camada entre o usuário cliente e o *software*. As vantagens em utilizar esta abordagem são possibilitar o reuso da camada de interface para diferentes aplicações e também diminuir o tempo de desenvolvimento, pois um processo de reengenharia dos *software* legados demandaria uma quantidade de tempo muito maior.

Além da demonstração das funcionalidades da solução, também está presente neste trabalho as implementações para expor como serviço as ferramentas PEPS e PLAT. Ambos requisitos de usabilidade e integralidade foram alcançados, atingindo assim os objetivos propostos. As ferramentas estão atualmente em produção e disponíveis ao publico em geral. Podem ser utilizadas por usuários através da interface *Web* (módulo *Web*) e também podem ser parte de uma orquestração de serviços, utilizando-se os serviços do módulo *Web Service*.

Algumas dificuldades foram encontradas ao longo do desenvolvimento deste trabalho. O principal e mais crítico problema foi o consumo exagerado de memória do servidor quando diversas solicitações de execução eram requisitadas ao mesmo tempo. Dependendo do tamanho dos modelos e do número de estados atingíveis o servidor chegava a abortar os processos no sistema operacional por falta de recurso de memória. Para solucionar este problema, foi inserido um esquema de enfileiramento de execuções, aonde um semáforo é responsável por permitir somente P execuções simultâneas. Além deste problema, também foram encontradas dificuldades no gerenciamento dos processos do sistema operacional. Diversas horas de depuração e pesquisa foram utilizadas para tentar construir um mecanismo dinâmico de alocação de recursos. Este mecanismo tinha como objetivo analisar a quantidade de memória disponível e a quantidade de memória necessária para a execução dos processos, e com base nestas informações, determinar em tempo real, a quantidade de processos paralelos suportados pelo *framework*. Apesar do tempo investido nessa solução, não foi possível obter resultados satisfatórios e tomou-se a decisão de implementar o controle de execuções paralelas estático, aonde apenas P execuções paralelas são permitidas com o uso de um semáforo.

5.1 Contribuição

Este trabalho contribui para a comunidade acadêmica no sentido de fornecer uma solução reutilizável para ser usada na disponibilização, através de *Web Services*, de aplicativos que não foram projetados com foco na integração. Além de demonstrar a solução genérica, este trabalho também apresentou duas implementações concretas, disponibilizando como serviço as ferramentas PEPS e PLAT. Ambos trazem como benefício a possibilidade de integração através de SOA e uma grande melhoria na interface com o usuário, tornando suas usabilidades mais simples. Além de facilitar o processo de instalação do *software*, pois basta ao usuário simplesmente acessar o endereço aonde as aplicações estiverem disponíveis, evitando assim todo o processo complexo de compilação da ferramenta. A utilização destas aplicações pelos acadêmicos proporcionará ao leigos uma melhor aprendizagem do formalismo SAN. E para os mais experientes, será útil no sentido de facilitar a utilização e visualização dos resultados.

5.2 Trabalhos Futuros

Como trabalho futuro é sugerido uma extensão do *framework* para que os serviços disponibilizados possam ser utilizados na forma de computação na nuvem (*Cloud Computing*). A disponibilização das ferramentas na forma de *Web Services* já é um primeiro passo na direção à nuvem, porém adaptações seriam necessárias para fazer uso dos conceitos de computação elástica (*Elastic Computing*) [WER10], com o objetivo de tornar o *framework* capaz de alocar e desalocar outras unidades de processamento de acordo com a demanda exigida e também, possuindo a habilidade de particionar paralelamente o processamento entre as diversas instâncias alocadas. Neste caso, o ideal seria substituir a abordagem de encapsulamento de caixa preta e adotar uma estratégia de reengenharia, como proposto em [FRE10] e [ZHA04], e reescrever novamente a ferramenta sendo exposta, tornando-o capaz de solucionar problemas em paralelo.

Também é sugerido a disponibilização de outras ferramentas além das já apresentadas neste trabalho. Novas implementações serviriam para validar ainda mais a solução proposta e também trariam o benefício da integração e melhor usabilidade para os usuários.

REFERÊNCIAS BIBLIOGRÁFICAS

[AFO12] A. Sales "Utilities - Afonso Sales.". Capturado em: `https://sites.google.com/site/afonsosales/utilities`, maio 2012.

[ALO03] G. Alonso, F. Casati, H. Kuno e V. Machiraju. "Web Services - Concepts, Architectures and Applications". Springer, 2003, 1ª edição, 354p.

[BAL07] L. J. Baldo, P. Fernandes e L. G. Leão. "Predição de Desempenho de aplicações Paralelas para Máquinas Agregadas Utilizando Modelos Estocásticos". Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2007, 80p

[BAU06] C. Bauer e G. King. "Java Persistance with Hibernate". Manning, 2007, 904p.

[BEN03a] A. Benoit, L. Brenner, P. Fernandes, B. Plateau e W. J. Stewart. "Peps 2003 user manual". *Computer Performance Evaluation. Modelling Techniques and Tools*, vol. 2794, 2003, pp. 98-115.

[BEN03b] A. Benoit, L. Brenner, P. Fernandes, B. Plateau e W. J. Stewart. "The peps software tool". In: Computer Performance Evaluations, Modelling Techniques and Tools. 13th International Conference., 2003, pp. 98-115.

[BOA03] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, J. Siméon e M. Stefanescu. "Xquery 1.0: An xml query language". *W3C working draft*, vol. 12, 2003, pp. 1-191.

[BOO04a] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris e D. Orchard "Web Service Architecture". Technical Report, W3C, Working Group Note, 2004.

[BOO04b] OMG Object Management Group. "Documents Associated with Corba, 3.1". Capturado em: `http://www.omg.org/spec/CORBA/3.1/`, julho 2011.

[BRA03] T. Bray, J. Paoli, C. M. Sperberg-Mcqueen, E. Maler e F. Yergeau. "Extensible Markup Language (XML) 1.0". W3C Recommendation. 2003, 4ª edição.

[BRE01] L. Brenner, P. Fernandes e A. Sales "MQNA - Markovian Queueing Networks Analyser". In: International Symposium on Modeling, Analysis, and Simulation of Computer Systems, 2003, 194p.

[BRE07] L. Brenner, P. Fernandes, B. Plateau e I. Sbeity. "Peps2007 - stochastic automata networks software tool". In: Fourth International Conference on the Quantitative Evaluation of Systems - QEST, 2007, pp. 163-164.

[CAL11] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. f. De Rose e R. Buyya, R. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms". Software: Practice and Experience, vol. 41, 2011, pp. 23-50.

[CAN06] G. Canfora, A. R. Fasolino, G. Frattolillo e P. Tramontana. "Migrating interactive legacy systems to Web services". In: Proceedings of the 10th European Conference on Software Maintenance and Reengineering, 2006, pp. 10-36.

[CAV04] C. Cavaness. "Programming jakarta struts". O'Reilly Media, 2004, 550p.

[CHA05] T. Chanin, F. L. Dotti, P. Fernandes e A. Sales. "Avaliação quantitativa de sistemas" PUCRS, 2005, 76p.

[CHI07] R. Chinnici, J. Moreau, A. Ryman e S. Weerawarana. "Web services description language (WSDL) version 2.0 part 1: Core language". *W3C Recommendation*, vol. 26, 2007.

[CLA99a] J. Clark, S. DeRose e outros. "XML path language (XPath) version 1.0". *W3C recommendation*, vol. 16, 1999.

[CLA99b] J. Clark e outros. "XSL transformations (XSLT) version 1.0". *W3C recommendation*, vol. 16, 1999.

[CLE04] L. Clement, A. Hately, C. von Riegen e T. Rogers. "UDDI Technical Specification version 3.0.2". *Published specification, Oasis*, vol. 5, 2004, pp. 16-18.

[CLO11] A. E. C. Cloud. "Amazon Web Services". *Retrieved November*, vol. 9, 2011.

[CZE07] R. M. Czekster, P. Fernandes, J. M. Vincent e T. Webber. Split: a flexible and efficient algorithm to vector-descriptor product. In: Proceedings of the 2nd international conference on Performance evaluation methodologies and tools, 2007, 83p.

[WER10] J. R. Wernsing e G. Stitt. Elastic computing: a framework for transparent, portable, and adaptive multi-core heterogeneous computing. In: ACM SIGPLAN Notices, 2010, pp 115-124.

[CZE09] R. M. Czekster, P. Fernandes e T. Webber. "Gtaexpress: a software package to handle kronecker descriptors". In: Sixth International Conference on the Quantitative Evaluation of Systems, 2009, pp. 281-282.

[DIJ71] E. W. Dijkstra. "Hierarchical ordering of sequential processes". *Acta Informatica, vol. 1-2, 1971, pp. 115-138.*

[EAS01] D. Eastlake e P. Jones. "Us secure hash algorithm 1 (sha1)". Technical Report, RFC Editor, 2001.

[ERL05] T. Erl. "Service-oriented architecture: concepts, technology, and design". Prentice Hall PTR, 2005, 792p.

[FER11a] P. Fernandes, M. E. J. O'Kelly, C. T. Papadopoulos e A. Sales. "Modeling exponential reliable production lines using kronecker descriptors". TR 66, FACIN, PUCRS, Porto Alegre, RS, Brazil, August, 2011.

[FER11b] P. Fernandes, M. E. J. O'Kelly, C. T. Papadopoulos e A. Sales. "Production Line Analysis Tool (PLAT)". In: 41st International Conference on Computers Industrial Engineering (CIE41), 248-253, Los Angeles, LA, USA, October, 2011.

[FER98] P. Fernandes, B. Plateau e W. J. Stewart. "Efficient descriptor-vector multiplication in stochastic automata networks". Journal of the ACM (JACM), *vol. 45, Maio 1998, pp. 381-41.*

[FRA08] M. F. Franciosi. "Uma Abordagem Paralela para o Algoritmo Split". Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2008, 67p

[FRE10] S. Frey e W. Hasselbring. "Model-Based Migration of Legacy Software Systems to Scalable and Resource-Efficient Cloud-Based Applications: The CloudMIG Approach". In: CLOUD COMPUTING 2010, The First International Conference on Cloud Computing, GRIDs, and Virtualization, 2010, pp. 155-158.

[GRI97] R. Grimes e D. R. Grimes. "Professional DCOM programming". Wrox Press Ltd., 1997, 565p.

[HOP06] J. E. Hopcroft, R. Motwani e J. D. Ullman. "Introduction to Automata Theory, Languages, and Computation". Boston:Addison-Wesley Longman Publishing Co., 2006, 3ª Edição, 750p.

[IaaS11] Educational Resources from the Cloud Computing Data Center "IaaS Definition - A reference resource about the infrastructure service standards within cloud computing.". Capturado em: `http://www.iaasdefinition.com/`*, junho 2012.*

[JER05] I. Jerstad, S. Dustdar e D. V. Thanh. "A service oriented architecture framework for collaborative services". In: Enabling Technologies: Infrastructure for Collaborative Enterprise, 2005. 14th IEEE International Workshops on., 2005, pp. 121-125.

[JOH03] R. Johnson e outros. "Expert one-on-one J2EE design and development". Wrox, 2003, 768p.

[KAP05] J. Kaplan. "Sorting through software as a service". Network World, *2005.*

[LAC06] S. Lacy. "The on-demand software scrum". Business Week, *2006.*

[MAR09] F. P. Marzullo. "SOA na prática - Inovando seu negócio por meio de soluções orientadas a serviço". Novatec, 1ª edição, 2009, 390p.

[MEL09] P. Mell e T. Grance. "The NIST definition of cloud computing". National Institute of Standards and Technology, *vol. 53, 2009, pp. 50.*

[NEW02] E. Newcomer. "Understanding Web Services: XML, Wsdl, Soap, and UDDI". Addison-Wesley Professional, 2002, 368p.

[OAS11] Oasis. "Oasis - advancing open standards for the information society". Capturado em: `http://www.oasis-open.org/`*, julho 2011.*

[PAL08] M. R. Palankar, A. Iamnitchi, M. Ripeanu e S. Garfinkel. "Amazon S3 for science grids: a viable solution?". Proceedings of the 2008 international workshop on Data-aware distributed computing*, 2008, pp. 55-64.*

[PAR66] D. F. Parkhill "The challenge of the computer utility". In: Addison-Wesley Reading, MA.

[PER06] S. Perera, C. Herath, J. Ekanayake, E. Chinthaka, A. Ranabahu, D. Jayasinghe, S. Weerawarana e G. Daniels. "Axis2, middleware for next generation web services". In: International Conference on Web Services, 2006, pp. 833-840.

[PLA85] B. Plateau. "On the stochastic structure of parallelism and synchronization models for distributed algorithms". In: ACM SIGMETRICS Performance Evaluation Review, 1985, pp. 147-154.

[PLA91] B. Plateau e K. Atif. "Stochastic automata network for modeling parallel systems". IEEE Trans. Software Eng.*, vol. 17, 1991, pp. 1093-1108.*

[PLA88] B. Plateau, J. M. Fourneau e K. H. Lee. "PEPS: A package for solving complex Markov model of parallel systems". Université de Paris-Sud, Centre d'Orsay, Laboratoire de Recherche en Informatique, 1988, 20p.

[POS82] J. Postel. "Simple mail transfer protocol". Information Sciences*, 1982.*

[RAG99] D. Raggett, A. Le Hors, I. Jacobs e outros. "Html 4.01 specification". W3C recommendation*, vol. 24, 1999.*

[RIC07] L. Richardson e S. Ruby. "RESTful web services". O'Reilly Media, 2007, 448p.

[ROS01] M. Rose. "The blocks extensible exchange protocol core". Technical Report, Invisible Worlds, Inc., 2001, 58p.

[SAL09] A. Sales e B. Plateau. "Reachable state space generation for structured models which use functional transitions". In: Sixth International Conference on the Quantitative Evaluation of Systems, 2009, pp. 269-278.

[SOU10] S. C. Souza. "Segurança para Web Services com criptografia heterogênea baseada em proxy". Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2010, 87p

[STE94] W. J. Stewart. "Introduction to the numerical solution of Markov chains". Princeton University Press NJ, 1994, volume 41, 539p.

[TAN03] A. S. Tanenbaum. "Computer networks". Prentice-Hall, 2003, 4ª Edição, 960p.

[THU09] R. Thurlow. "RPC: Remote Procedure Call Protocol Specification Version 2". Technical Report, Sun Microsystems, 2009, 18p.

[VAQ08] L. M. Vaquero, L. Rodero-Merino, J. Caceres e M. Lindner. "A break in the clouds: towards a cloud definition". ACM SIGCOMM Computer Communication Review, *vol. 39, 2008, pp. 50-55.*

[VLI95] J. Vlissides, R. Helm, R. Johnson e E. Gamma. "Design patterns: Elements of reusable object-oriented software". Addison-Wesley Professional, 1995, 416p.

[W3C11] W3C "W3C - world wide web consortium.". Capturado em: `http://www.w3.org/`, *julho 2011.*

[WOL96] A. Wollrath, R. Riggs e J. Waldo. "A distributed object model for the Java system". Computing Systems, *vol. 9, 1996, pp. 265-290.*

[ZHA04] Z. Zhang e H. Yang. "Incubating services in legacy systems for architectural migration". In: Software Engineering Conference, 2004, pp. 196-203.

[BEE00] D. Beeferman e A. Berger. "Agglomerative clustering of a search engine query log". In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, 2000, pp. 407-416.

[ZHA10] Q. Zhang, L. Cheng e R. Boutaba. "Cloud computing: state-of-the-art and research challenges". Journal of Internet Services and Applications, *vol. 1, 2010, pp. 7-18.*

A. Manual Implementação do *Framework*

As Seções a seguir irão demonstrar como utilizar o *framework* proposto neste trabalho para a disponibilização de outras soluções através de *Web Services*. Primeiro é apresentado um passo-a-passo de como configurar o ambiente de desenvolvimento e logo após são apontados os arquivos que necessitam ser customizados.

A.1 Configurando o Ambiente de Desenvolvimento

Este passo-a-passo irá cobrir as atividades necessárias para configurar o ambiente de trabalho para a execução do código do *framework*.

A.1.1 Pré-requisitos

Os pré-requisitos necessários estão listados abaixo:

1. **Java 7**. É necessário possuir a máquina virtual e o kit de desenvolvimento Java (JDK - *Java Development Kit*). Os arquivos de instalação podem ser encontrados no endereço `http://www.oracle.com/technetwork/java/javase/downloads/index.html`. No momento da criação deste manual, a versão mais atual é a 7u6.

2. **Apache Tomcat**. É necessário possuir o servidor de aplicação Apache Tomcat na versão 7. O pacote com os arquivos do servidor podem ser encontrados no endereço `http://tomcat.apache.org/download-70.cgi`, não é necessário efetuar instalação, basta apenas descompactar os arquivos em algum diretório do sistema de arquivos. No momento da criação deste manual, a versão atual é a 7.0.29.

3. **Apache Derby**. É necessário possuir o banco de dados Apache Derby. O pacote com os arquivos do servidor podem ser encontrados no endereço `http://db.apache.org/derby/derby_downloads.html`, assim como no Tomcat, não é necessário efetuar instalação, basta apenas descompactar os arquivos em algum diretório do sistema de arquivos. No momento da criação deste manual, a versão atual é a 10.8.2.2.

4. **Eclipse**. É necessário possuir alguma IDE (*Integrated Development Environment*) de desenvolvimento. Neste manual estaremos cobrindo a utilização da IDE Eclipse, entretando diversas outras poderiam ser utilizadas, como por exemplo o Netbeans. O pacote com os arquivos pode ser encontrado no endereço `http://www.eclipse.org/downloads/` e recomenda-se a versão *Classic*. No momento da criação deste manual, a versão corrente é a 4.2, codinome *Juno*.

A.1.2 Configuração da IDE

Os passos necessários para configurar a IDE Eclipse são os seguintes:

1. Executar o arquivo *eclipse.exe* no diretório aonde foi extraído o Eclipse.
2. Selecionar o diretório de trabalho (*Workspace*). Pode ser qualquer diretório do sistema de arquivos.
3. Acessar o menu *Window* e logo após o item *Preferences*.
4. Abrir o grupo Java e selecionar o item *Installed JREs*. Garantir que a JRE selecionada é a máquina virtual Java descrita nos pré-requisitos.
5. Acessar o menu *Help* e selecionar o item *Install New Software...*. No campo *Work With* selecionar a opção "Juno - http://download.eclipse.org/releases/juno". Logo após selecionar os seguintes itens:
    - Collaboration
        - Eclipse EGit
    - General Purpose Tools
        - m2e - Maven Integration for Eclipse
    - Web, XML, Java EE and OSGi Enterprise Development
        - Axis2 Tools
        - Eclipse Java EE Developer Tools
        - Eclipse Java Web Developer Tools
        - Eclipse Web Developer Tools
        - Eclipse XML Editors and Tools
        - Javascript Development Tools
        - JST Server Adapters
        - JST Server Adapters Extensions
        - JST Server UI
        - Web Page Editor
        - WST Server Adapters
6. Pressionar o botão *Next*, ler e aceitar os termos de licença e aguardar pelo *download* dos arquivos. Ao final da instalação será necessário reiniciar o Eclipse.
7. Após a reinicialização, acessar o menu *Window*, selecionar o item *Show View*, selecionar o item *Others...*, expandir o item *Server* e selecionar o item *Servers*. A visão de servidores se abrirá na área inferior da IDE.

8. Acionar o botão direito do *mouse* sobre a visão de servidores e selecionar a opção *New* e a opção *Server*. Conforme a Figura A.1.

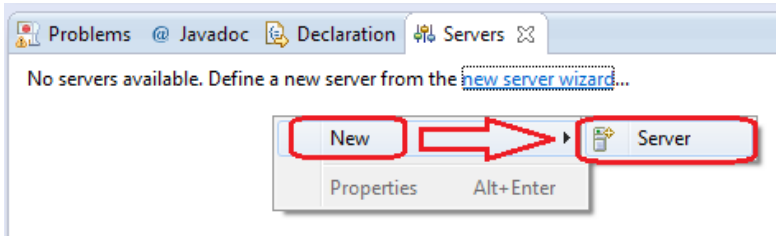

Figura A.1: Criação de servidor no Eclipse

9. Expandir o grupo *Apache* e selecionar o item *Tomcat v7.0 Server*. Pressionar o botão *Next*.

10. No campo *Tomcat instalation directory* selecionar o diretório do sistema de arquivos aonde foi extraído o Apache Tomcat. Pressionar o botão *Finish*.

11. Acessar o menu *Window*, selecionar o item *Open Perspective*, selecionar o item *Other...* e selecionar o item *Git Repository Exploring*.

12. Selecionar o item *Clone a Git repository*, conforme a Figura reffig:manual06.

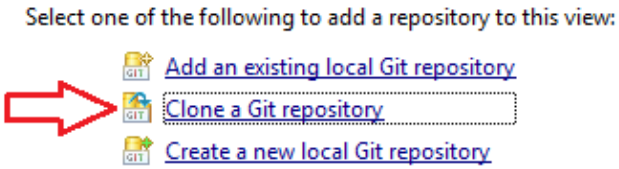

Figura A.2: Clonando um repositório Git

13. Informar o campo URI o endereço `git://git.code.sf.net/p/saasframework/code`. Pressionar o botão *Next*. Garantir que o *branch Master* está selecionado e pressionar o botão *Next*. Pressionar o botão *Finish*. Aguardar o download do *código*.

14. Acessar o menu *Window*, acessar o submenu *Open Perspective* e abrir a perspectiva Java (*default*).

15. Acessar o menu *File* e selecionar o item *Import...*.

16. Expandir o grupo *General* e selecionar a opção *Existing Projects into Workspace*. No campo *Select root directory* selecionar o local no sistema de arquivos aonde foi feito o *download* do repositório Git. Pressionar o botão *Finish*.

17. Clicar com o botão direito do *mouse* sobre o servidor Apache Tomcat na visão de servidores e acessar a opção *Add or Remove*. Adicionar o projeto ao servidor, pressionando o botão *Add*.

18. Inicializar o banco de dados Derby, executando o arquivo "startNetworkServer". Inicializar o Tomcat utilizando o botão *Start* na visão de servidores e acessar o endereço `http://localhost:8080/saasframework/` e a tela de login do *framework* deverá ser apresentada.

A.1.3 Customizando o *framework*

Para implementar o *framework* são necessários efetuar as seguintes ações:

1. Editar o arquivo "config.properties" informando corretos diretórios e o correto comando para execução da ferramenta que se deseja expor como SaaS.

2. Tomando como exemplo a classe *EchoExecution*, implementar uma classe que extenda a classe abstrata *SaasExecutionBase* e faça as interações necessárias com a ferramenta que se está disponibilizando.

3. Modificar o método *solveModel* da classe *Util* para utilizar a nova classe implementada no passo anterior. A alteração deve estar contida no bloco delimitado pelos comentários "START OF CUSTOMIZATION BLOCK" e "END OF CUSTOMIZATION BLOCK".

4. Customizar o arquivo *Input.jsp* e a classe *Execute* com os parâmetros necessários para a execução da ferramenta sendo exposta.

B. Arquivo WSDL

A listagem abaixo apresenta o arquivo WSDL dos serviços expostos pelo *framework*.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:ns1="http://org.apache.axis2/xsd" xmlns:ns="http://service.
    peg.pucrs.com.br" xmlns:wsaw="http://www.w3.org/2006/05/
    addressing/wsdl" xmlns:ax25="http://model.peg.pucrs.com.br/xsd"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs="http
    ://www.w3.org/2001/XMLSchema" xmlns:mime="http://schemas.xmlsoap.
    org/wsdl/mime/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap
    /" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    targetNamespace="http://service.peg.pucrs.com.br">
     <wsdl:documentation>
                 Please Type your service description here
         </wsdl:documentation>
     <wsdl:types>
         <xs:schema xmlns:ax26="http://model.peg.pucrs.com.br/xsd"
             attributeFormDefault="qualified" elementFormDefault="
             qualified" targetNamespace="http://service.peg.pucrs.com.
             br">
              <xs:import namespace="http://model.peg.pucrs.com.br/xsd
                 "/>
              <xs:element name="solicitaNovaSenha">
                  <xs:complexType>
                      <xs:sequence>
                          <xs:element minOccurs="0" name="email"
                              nillable="true" type="xs:string"/>
                      </xs:sequence>
                  </xs:complexType>
              </xs:element>
              <xs:element name="salvaUsuario">
                  <xs:complexType>
                      <xs:sequence>
                          <xs:element minOccurs="0" name="user"
                              nillable="true" type="ax26:User"/>
                      </xs:sequence>
```

```
21                  </xs:complexType>
                </xs:element>
                <xs:element name="salvaUsuarioResponse">
                    <xs:complexType>
                        <xs:sequence>
26                          <xs:element minOccurs="0" name="return"
                                nillable="true" type="ax26:User"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="buscaUsuario">
31                  <xs:complexType>
                        <xs:sequence>
                            <xs:element minOccurs="0" name="login"
                                nillable="true" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
36              </xs:element>
                <xs:element name="buscaUsuarioResponse">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element minOccurs="0" name="return"
                                nillable="true" type="ax26:User"/>
41                      </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="buscaExecução">
                    <xs:complexType>
46                      <xs:sequence>
                            <xs:element minOccurs="0" name="id" type="xs
                                :int"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
51              <xs:element name="buscaExecuçãoResponse">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element minOccurs="0" name="return"
                                nillable="true" type="ax26:PepsExecution
```

```
                        "/>
                    </xs:sequence>
56              </xs:complexType>
            </xs:element>
            <xs:element name="autentica">
                <xs:complexType>
                    <xs:sequence>
61                      <xs:element minOccurs="0" name="login"
                            nillable="true" type="xs:string"/>
                        <xs:element minOccurs="0" name="senha"
                            nillable="true" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="autenticaResponse">
66              <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" type
                            ="xs:boolean"/>
                    </xs:sequence>
71              </xs:complexType>
            </xs:element>
            <xs:element name="ExecutaFerramenta">
                <xs:complexType>
                    <xs:sequence>
76                      <xs:element maxOccurs="unbounded" minOccurs
                            ="0" name="parametros" nillable="true"
                            type="ax26:ExpertParameters"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
81      <xs:schema attributeFormDefault="qualified"
            elementFormDefault="qualified" targetNamespace="http://
            model.peg.pucrs.com.br/xsd">
            <xs:complexType name="User">
                <xs:sequence>
                    <xs:element minOccurs="0" name="email" nillable
                        ="true" type="xs:string"/>
```

```
                <xs:element minOccurs="0" name="fullName"
                    nillable="true" type="xs:string"/>
86              <xs:element minOccurs="0" name="id" nillable="
                    true" type="xs:int"/>
                <xs:element minOccurs="0" name="language"
                    nillable="true" type="xs:string"/>
                <xs:element minOccurs="0" name="password"
                    nillable="true" type="xs:string"/>
                <xs:element minOccurs="0" name="username"
                    nillable="true" type="xs:string"/>
            </xs:sequence>
91      </xs:complexType>
        <xs:complexType name="PepsExecution">
            <xs:sequence>
                <xs:element minOccurs="0" name="group" nillable
                    ="true" type="ax25:PepsExecGroup"/>
                <xs:element minOccurs="0" name="id" nillable="
                    true" type="xs:int"/>
96              <xs:element minOccurs="0" name="params" nillable
                    ="true" type="xs:string"/>
                <xs:element minOccurs="0" name="pepsFile"
                    nillable="true" type="ax25:PepsFile"/>
                <xs:element minOccurs="0" name="status" nillable
                    ="true" type="xs:string"/>
                <xs:element minOccurs="0" name="tim" nillable="
                    true" type="xs:string"/>
                <xs:element minOccurs="0" name="timestamp"
                    nillable="true" type="xs:dateTime"/>
101             <xs:element minOccurs="0" name="vector" nillable
                    ="true" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="PepsExecGroup">
            <xs:sequence>
106             <xs:element minOccurs="0" name="id" nillable="
                    true" type="xs:int"/>
                <xs:element minOccurs="0" name="name" nillable="
                    true" type="xs:string"/>
            </xs:sequence>
```

```
        </xs:complexType>
        <xs:complexType name="PepsFile">
111         <xs:sequence>
                <xs:element minOccurs="0" name="absolutePath"
                    nillable="true" type="xs:string"/>
                <xs:element minOccurs="0" name="fileContent"
                    nillable="true" type="xs:string"/>
                <xs:element minOccurs="0" name="filename"
                    nillable="true" type="xs:string"/>
                <xs:element minOccurs="0" name="id" nillable="
                    true" type="xs:int"/>
116             <xs:element minOccurs="0" name="timestamp"
                    nillable="true" type="xs:dateTime"/>
                <xs:element minOccurs="0" name="user" nillable="
                    true" type="ax25:User"/>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="ExpertParameters">
121         <xs:sequence>
                <xs:element minOccurs="0" name="
                    diagonalPreconditioning" nillable="true" type
                    ="xs:string"/>
                <xs:element minOccurs="0" name="errorTest"
                    nillable="true" type="xs:string"/>
                <xs:element minOccurs="0" name="from_identifier1
                    " nillable="true" type="xs:string"/>
                <xs:element minOccurs="0" name="from_identifier2
                    " nillable="true" type="xs:string"/>
126             <xs:element minOccurs="0" name="from_identifier3
                    " nillable="true" type="xs:string"/>
                <xs:element minOccurs="0" name="identifier1"
                    nillable="true" type="xs:string"/>
                <xs:element minOccurs="0" name="identifier2"
                    nillable="true" type="xs:string"/>
                <xs:element minOccurs="0" name="identifier3"
                    nillable="true" type="xs:string"/>
                <xs:element minOccurs="0" name="inc_identifier1"
                     nillable="true" type="xs:string"/>
```

```
                    <xs:element minOccurs="0" name="inc_identifier2"
                        nillable="true" type="xs:string"/>
                    <xs:element minOccurs="0" name="inc_identifier3"
                        nillable="true" type="xs:string"/>
                    <xs:element minOccurs="0" name="iteractions"
                       nillable="true" type="xs:string"/>
                    <xs:element minOccurs="0" name="
                       krylovSubspaceSize" nillable="true" type="xs:
                       string"/>
                    <xs:element minOccurs="0" name="minimalError"
                       nillable="true" type="xs:string"/>
                    <xs:element minOccurs="0" name="solutionMethod"
                       nillable="true" type="xs:string"/>
                    <xs:element minOccurs="0" name="splitType"
                       nillable="true" type="xs:string"/>
                    <xs:element minOccurs="0" name="stopCriterion"
                       nillable="true" type="xs:string"/>
                    <xs:element minOccurs="0" name="to_identifier1"
                       nillable="true" type="xs:string"/>
                    <xs:element minOccurs="0" name="to_identifier2"
                       nillable="true" type="xs:string"/>
                    <xs:element minOccurs="0" name="to_identifier3"
                       nillable="true" type="xs:string"/>
                    <xs:element minOccurs="0" name="
                       vectorDescriptorProduct" nillable="true" type
                       ="xs:string"/>
                    <xs:element minOccurs="0" name="
                       vectorImplementation" nillable="true" type="
                       xs:string"/>
                    <xs:element minOccurs="0" name="vectorType"
                       nillable="true" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
    </wsdl:types>
    <wsdl:message name="buscaExecuçãoRequest">
        <wsdl:part name="parameters" element="ns:buscaExecução"/>
    </wsdl:message>
    <wsdl:message name="buscaExecuçãoResponse">
```

```
        <wsdl:part name="parameters" element="ns:
            buscaExecuçãoResponse"/>
    </wsdl:message>
    <wsdl:message name="salvaUsuarioRequest">
        <wsdl:part name="parameters" element="ns:salvaUsuario"/>
    </wsdl:message>
    <wsdl:message name="salvaUsuarioResponse">
        <wsdl:part name="parameters" element="ns:
            salvaUsuarioResponse"/>
    </wsdl:message>
    <wsdl:message name="ExecutaFerramentaRequest">
        <wsdl:part name="parameters" element="ns:ExecutaFerramenta
            "/>
    </wsdl:message>
    <wsdl:message name="buscaUsuarioRequest">
        <wsdl:part name="parameters" element="ns:buscaUsuario"/>
    </wsdl:message>
    <wsdl:message name="buscaUsuarioResponse">
        <wsdl:part name="parameters" element="ns:
            buscaUsuarioResponse"/>
    </wsdl:message>
    <wsdl:message name="autenticaRequest">
        <wsdl:part name="parameters" element="ns:autentica"/>
    </wsdl:message>
    <wsdl:message name="autenticaResponse">
        <wsdl:part name="parameters" element="ns:autenticaResponse
            "/>
    </wsdl:message>
    <wsdl:message name="solicitaNovaSenhaRequest">
        <wsdl:part name="parameters" element="ns:solicitaNovaSenha
            "/>
    </wsdl:message>
    <wsdl:portType name="ServicosPortType">
        <wsdl:operation name="buscaExecução">
            <wsdl:input message="ns:buscaExecuçãoRequest" wsaw:
                Action="urn:buscaExecução"/>
            <wsdl:output message="ns:buscaExecuçãoResponse" wsaw:
                Action="urn:buscaExecuçãoResponse"/>
        </wsdl:operation>
```

```
        <wsdl:operation name="salvaUsuario">
            <wsdl:input message="ns:salvaUsuarioRequest" wsaw:Action
                ="urn:salvaUsuario"/>
            <wsdl:output message="ns:salvaUsuarioResponse" wsaw:
                Action="urn:salvaUsuarioResponse"/>
        </wsdl:operation>
        <wsdl:operation name="ExecutaFerramenta">
            <wsdl:input message="ns:ExecutaFerramentaRequest" wsaw:
                Action="urn:ExecutaFerramenta"/>
        </wsdl:operation>
        <wsdl:operation name="buscaUsuario">
            <wsdl:input message="ns:buscaUsuarioRequest" wsaw:Action
                ="urn:buscaUsuario"/>
            <wsdl:output message="ns:buscaUsuarioResponse" wsaw:
                Action="urn:buscaUsuarioResponse"/>
        </wsdl:operation>
        <wsdl:operation name="autentica">
            <wsdl:input message="ns:autenticaRequest" wsaw:Action="
                urn:autentica"/>
            <wsdl:output message="ns:autenticaResponse" wsaw:Action
                ="urn:autenticaResponse"/>
        </wsdl:operation>
        <wsdl:operation name="solicitaNovaSenha">
            <wsdl:input message="ns:solicitaNovaSenhaRequest" wsaw:
                Action="urn:solicitaNovaSenha"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="ServicosSoap11Binding" type="ns:
        ServicosPortType">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/
            http" style="document"/>
        <wsdl:operation name="buscaExecução">
            <soap:operation soapAction="urn:buscaExecução" style="
                document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
```

```
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="salvaUsuario">
            <soap:operation soapAction="urn:salvaUsuario" style="
               document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="ExecutaFerramenta">
            <soap:operation soapAction="urn:ExecutaFerramenta" style
               ="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
        </wsdl:operation>
        <wsdl:operation name="buscaUsuario">
            <soap:operation soapAction="urn:buscaUsuario" style="
               document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="autentica">
            <soap:operation soapAction="urn:autentica" style="
               document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
```

```
        <wsdl:operation name="solicitaNovaSenha">
            <soap:operation soapAction="urn:solicitaNovaSenha" style
                ="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="ServicosSoap12Binding" type="ns:
        ServicosPortType">
        <soap12:binding transport="http://schemas.xmlsoap.org/soap/
            http" style="document"/>
        <wsdl:operation name="buscaExecução">
            <soap12:operation soapAction="urn:buscaExecução" style="
                document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="salvaUsuario">
            <soap12:operation soapAction="urn:salvaUsuario" style="
                document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="ExecutaFerramenta">
            <soap12:operation soapAction="urn:ExecutaFerramenta"
                style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
        </wsdl:operation>
```

```
        <wsdl:operation name="buscaUsuario">
281         <soap12:operation soapAction="urn:buscaUsuario" style="
                document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
286             <soap12:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="autentica">
            <soap12:operation soapAction="urn:autentica" style="
                document"/>
291         <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
296         </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="solicitaNovaSenha">
            <soap12:operation soapAction="urn:solicitaNovaSenha"
                style="document"/>
            <wsdl:input>
301             <soap12:body use="literal"/>
            </wsdl:input>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="ServicosHttpBinding" type="ns:
        ServicosPortType">
306     <http:binding verb="POST"/>
        <wsdl:operation name="buscaExecução">
            <http:operation location="Servicos/buscaExecução"/>
            <wsdl:input>
                <mime:content type="text/xml" part="buscaExecução"/>
311         </wsdl:input>
            <wsdl:output>
                <mime:content type="text/xml" part="buscaExecução"/>
            </wsdl:output>
```

```
        </wsdl:operation>
316     <wsdl:operation name="salvaUsuario">
            <http:operation location="Servicos/salvaUsuario"/>
            <wsdl:input>
                <mime:content type="text/xml" part="salvaUsuario"/>
            </wsdl:input>
321         <wsdl:output>
                <mime:content type="text/xml" part="salvaUsuario"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="ExecutaFerramenta">
326         <http:operation location="Servicos/ExecutaFerramenta"/>
            <wsdl:input>
                <mime:content type="text/xml" part="
                    ExecutaFerramenta"/>
            </wsdl:input>
        </wsdl:operation>
331     <wsdl:operation name="buscaUsuario">
            <http:operation location="Servicos/buscaUsuario"/>
            <wsdl:input>
                <mime:content type="text/xml" part="buscaUsuario"/>
            </wsdl:input>
336         <wsdl:output>
                <mime:content type="text/xml" part="buscaUsuario"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="autentica">
341         <http:operation location="Servicos/autentica"/>
            <wsdl:input>
                <mime:content type="text/xml" part="autentica"/>
            </wsdl:input>
            <wsdl:output>
346             <mime:content type="text/xml" part="autentica"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="solicitaNovaSenha">
            <http:operation location="Servicos/solicitaNovaSenha"/>
351         <wsdl:input>
```

```
                    <mime:content type="text/xml" part="
                        solicitaNovaSenha"/>
                </wsdl:input>
            </wsdl:operation>
        </wsdl:binding>
356     <wsdl:service name="Servicos">
            <wsdl:port name="ServicosHttpSoap11Endpoint" binding="ns:
                ServicosSoap11Binding">
                <soap:address location="http://localhost:8080/peps2011/
                    services/Servicos.ServicosHttpSoap11Endpoint/"/>
            </wsdl:port>
            <wsdl:port name="ServicosHttpSoap12Endpoint" binding="ns:
                ServicosSoap12Binding">
361             <soap12:address location="http://localhost:8080/peps2011
                    /services/Servicos.ServicosHttpSoap12Endpoint/"/>
            </wsdl:port>
            <wsdl:port name="ServicosHttpEndpoint" binding="ns:
                ServicosHttpBinding">
                <http:address location="http://localhost:8080/peps2011/
                    services/Servicos.ServicosHttpEndpoint/"/>
            </wsdl:port>
366     </wsdl:service>
    </wsdl:definitions>
```