

**UNIVERSIDADE FEDERAL DE SANTA CATARINA**

**Ambiente para Execução Automática de Composições de Web  
Services**

**Augusto César Ferreira**

Florianópolis – SC

2013/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA  
CURSO DE SISTEMAS DE INFORMAÇÃO

Ambiente para Execução Automática de Composições de Web Services

Augusto César Ferreira

Trabalho de conclusão de curso  
apresentado como parte dos requisitos  
para obtenção do grau de Bacharel em  
Sistemas de Informação.

Florianópolis – SC

2013/2

Augusto César Ferreira

Ambiente para Execução Automática de Composições de Web Services

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Banca Examinadora

---

Prof. Frank Augusto Siqueira  
Orientador

---

Prof. Renato Fileto  
Membro

---

Prof. Jim Lau  
Membro

## Sumário

1. Introdução.....	13
1.1. Objetivos .....	14
1.2. Justificativa.....	16
1.3. Delimitação do Escopo.....	16
1.4. Metodologia.....	17
1.5. Estrutura do Trabalho.....	17
2. Serviços Web.....	19
2.1. Arquitetura Orientada a Serviços.....	19
2.2. Web Services .....	22
2.2.1. XML.....	23
2.2.2. SOAP .....	25
2.2.3. WSDL.....	26
2.2.4. UDDI .....	28
2.2.5. Orquestração e Coreografia de Web Services.....	29
2.2.6. WS-BPEL.....	31
2.2.7. Web Services Semânticos .....	36
2.3. Composição e Execução de Serviços .....	38
3. Plataforma para Descoberta e Execução de Composições de Web Services Semânticos.....	42
3.1. Composição de Web Services.....	44
3.2. Execução das Composições de Web Services .....	45

3.3.	Arquitetura .....	49
3.3.1.	Implantação de uma Composição .....	51
3.3.2.	Execução de uma Composição.....	54
3.4.	Composition Repository .....	54
4.	Implementação e Avaliação do Protótipo.....	56
4.1.	Linguagem de Programação e Framework de Desenvolvimento .....	57
4.2.	Componentes .....	58
4.3.	Modelos.....	60
4.4.	Apache ODE.....	61
4.5.	Artefatos de Implantação.....	62
4.6.	Avaliação do Protótipo.....	63
5.	Conclusão.....	69
5.1.	Trabalhos Futuros .....	71
6.	Referências Bibliográficas .....	73
	APÊNDICE I - Artigo .....	78
	APÊNDICE II – Código-fonte.....	90

## Lista de Figuras

Figura 1 O Modelo SOA (GISOLFI, 2001 apud BIH, 2006).....	20
Figura 2 Arquitetura orientada a serviços baseada em Web Services (AYALA et al. 2002) .....	21
Figura 3 Interação tradicional da Web e Interação de Web Services (HAAS, 2003) .....	22
Figura 4 Registro, localização e invocação de um Web Service .....	23
Figura 5 Estrutura do documento WSDL (VIEGAS, 2008) .....	27
Figura 6 Orquestração de Web Services (JURIC; MATHEW; SARANG, 2006).....	29
Figura 7 Coreografia de Web Services(JURIC; MATHEW; SARANG, 2006)...	30
Figura 8 Arquitetura da Plataforma de Composição e Execução de <i>Web Services</i> (HOBOLD; SIQUEIRA, 2012) .....	42
Figura 9 Diagrama de Atividades para geração de processo WS-BPEL.....	48
Figura 10 Composição de <i>Web Services</i> transformada em um processo executável WS-BPEL conforme abordagem proposta .....	49
Figura 11 Arquitetura proposta para execução de composições de <i>Web Services</i> .....	50
Figura 12 Diagrama de sequência do processo de implantação de uma composição .....	53
Figura 13 Diagrama de Sequência para Execução de uma Composição .....	54
Figura 14 Modelo entidade-relacionamento do banco de dados de composições adequado .....	55
Figura 15 Diagrama de Implantação do protótipo .....	56
Figura 16 Diagrama de classes para os componentes da arquitetura .....	59
Figura 17 Diagrama de classes para os Modelos .....	61

Figura 18 Diagrama de classes representando elementos do documento WS-BPEL .....	63
Figura 19 <i>Workflow</i> da composição exemplo .....	64
Figura 20 Visualização gráfica do documento WS-BPEL gerado pelo protótipo para a composição exemplo.....	65
Figura 21 Tempo de execução para cada etapa da geração do pacote de implantação em cada iteração.....	68

## Lista de Quadros

Quadro 1 Um documento XML.....	24
Quadro 2 Processo abstrato em WS-BPEL .....	32
Quadro 3 Processo executável em WS-BPEL .....	32
Quadro 4 Escopo em WS-BPEL .....	32
Quadro 5 Atividade receive em WS-BPEL .....	33
Quadro 6 Atividade reply em WS-BPEL.....	33
Quadro 7 Atividade invoke em WS-BPEL .....	34
Quadro 8 Atividades de controle condicional e de repetição em WS-BPEL.....	35
Quadro 9 Atividade sequence em WS-BPEL.....	35
Quadro 10 Atividade flow em WS-BPEL .....	36
Quadro 11 Manipulador de falhas para um processo em WS-BPEL.....	36
Quadro 12 Exemplo de referência entre elementos em arquivos XML .....	62
Quadro 13 Declaração do elemento partnerLinkType referenciado pelo elemento partnerLink.....	63
Quadro 14 Requisição da execução da composição exemplo .....	66
Quadro 15 Resposta da execução da composição exemplo.....	66
Quadro 16 Utilização da ferramenta AB.....	67



## Lista de Tabelas

Tabela 1 Classificação das abordagens de Composição e Execução de Web Services. (AGARWAL et al. 2008).....	41
Tabela 2 Comparação de <i>engines</i> BPEL resumida (WIKIPEDIA, 2013).....	62

## Lista de Reduções

DBMS	Data Base Management System
BPEL4WS	Business Process Execution Language for Web Services
CORBA	Common Object Request Broker Architecture
CSS	Cascade Style Sheets
DCOM	Distributed Component Object Model
FTP	File Transfer Protocol
HTML	HyperText Markup Language
IDE	Integrated Development Environment
HTTP	Hyper Text Transfer Protocol
MathML	Mathematical Markup Language
MIME	Multipurpose Internet Mail Extensions
MVC	Model-View-Controller
OASIS	Organization for the Advancement of Structured Information Standards
ODE	Orchestration Director Engine
ORB	Object Request Broker
OWL-S	Ontology Web Language for Services
PHP	PHP: Hypertext Preprocessor
RDF	Resource Description Framework
RMI	Remote Method Invocation
SAWSDL	Semantic Annotations for WSDL and XML Schema
SGML	Standard Generalized Markup Language
SMTP	Simple Mail Transfer Protocol

SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SVG	Scalable Vector Graphics
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WS-BPEL	Web Services Business Process Execution Language
WS-CDL	Web Services Coreography Description Language
WSDL	Web Services Description Language
WSML	Web Services Modeling Language
WSMO	Web Service Modeling Ontology
WSMX	Web Services Modeling Execution Environment
XHTML	eXtensible Hyper Text Markup Language
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language for Transformation

## Resumo

O presente trabalho visa abordar as tarefas relacionadas a permitir, de forma automática, que uma composição de *Web Services* previamente descoberta seja executada como um serviço único, de forma transparente ao solicitante, seguindo a arquitetura proposta por Hobold e Siqueira (2012).

Conceitos relacionados à Arquitetura Orientada a Serviços, *Web Services*, Orquestração e Coreografia de *Web Services* além de *Web Semântica* e *Web Services Semânticos*, abordando diversas tecnologias que possibilitam a prática destes conceitos, são revistos a fim de obter o fundamento teórico que possibilitaram o desenvolvimento do presente trabalho.

Este trabalho apresenta uma abordagem para representação de uma composição de *Web Services* seguindo o padrão WS-BPEL, geração dos artefatos de implantação necessários para implantação e execução da composição em um *engine* de execução, ambos de forma automática. Um protótipo foi desenvolvido a fim de verificar o funcionamento da abordagem proposta.

**Palavras-chave:** Arquitetura Orientada a Serviços, *Web Services Semânticos*, Orquestração de *Web Services*, Composição de *Web Services*, Execução de Composições de *Web Services*, WS-BPEL.

## 1. Introdução

As principais tendências da indústria do software, em geral, surgem após diversos visionários e pioneiros conceberem independentemente o mesmo conceito, e com a Arquitetura Orientada a Serviços (SOA – *Service Oriented Architecture*) não foi diferente. O termo SOA foi cunhado pelo *Gartner Group* em 1996, que publicou os primeiros relatos sobre a tecnologia, porém a abordagem de aplicação distribuída empregada por SOA já existia há pelo menos uma década (TOWNSEND, 2008). SOA surge como uma evolução dos paradigmas de programação, que foi do modelo procedural, passando pelo estruturado, pela orientação a objetos, pelo modelo de componentes e, elevando ainda mais o nível de abstração, chegando aos serviços.

Na Arquitetura Orientada a Serviços, os serviços, funções bem definidas, autônomas e independentes do contexto ou estado de outros serviços, são disponibilizados como componentes de softwares através de uma rede de computadores. Com a adoção de Web Services, que utilizam linguagens e protocolos abertos, é possível implementar a Arquitetura Orientada a Serviços através da criação de uma camada adicional entre os serviços, facilitando a integração destes através de uma rede de computadores, independentemente da plataforma ou linguagem de programação em que foram desenvolvidos.

Web Services podem ser compostos a fim de criar novos serviços que atendam a requisições que não poderiam ser atendidas por um Web Service individualmente, porém a tarefa de compô-los manualmente é complexa em razão da grande quantidade de Web Services disponíveis e da frequência com

que estes são atualizados. Com a utilização de linguagens semânticas, e não somente sintáticas, para descrição de serviços, possibilita-se a composição automática de Web Services.

Hobold e Siqueira (2012) descrevem uma plataforma para descoberta e execução de composições de *web services* semânticos. Esta plataforma compreende um mecanismo para dinamicamente identificar composições baseada na descrição semântica e uma *engine* para execução das composições, que devem ser invocadas transparentemente como um único serviço. Em Hobold (2012), dissertação de mestrado intitulada “Uma Abordagem para Descoberta Automática de Composições de Serviços Web Semânticos”, é abordada a fase de descoberta das composições prevista na plataforma e tem como um de seus resultados o *workflow* para execução de uma composição, descrevendo a ordem de invocação das operações e troca de mensagens entre os *web services* participantes.

## 1.1. Objetivos

O presente trabalho tem como objetivo principal, a partir dos resultados obtidos por Hobold (2012), permitir de maneira automática a execução das composições descobertas, desonerando desenvolvedores de *software* na realização de tarefas para obter a execução das composições de forma manual. Para alcançar este objetivo foram traçados os seguintes objetivos específicos:

1. Realizar a transformação dos dados que descrevem a composição descoberta para o padrão WS-BPEL (*Web Services Business Process Execution Language*);

2. Estabelecer a seleção de uma *engine* de execução de processos compatível com o padrão WS-BPEL;
3. Implantar de maneira automatizada as composições na *engine* de execução de processos selecionada;
4. Permitir que as composições implantadas sejam executadas de forma transparente, como um único serviço.

Para alcançar os objetivos estabelecidos serão desenvolvidas as seguintes atividades:

- Estudar o padrão WS-BPEL e definir como uma composição de *web services* pode ser descrita utilizando esse padrão;
- Buscar *engines* de execução de processos que suportem o padrão WS-BPEL e atendam os requisitos do projeto;
- Analisar os requisitos para implantação de uma composição descrita no padrão WS-BPEL na *engines* de execução de processos encontradas;
- Desenvolver um componente de *software* capaz de efetuar a conversão de uma composição de *web services* descoberta pela implementação de Hobold (2012) para o padrão WS-BPEL;
- Desenvolver um componente de *software* que faça a implantação automatizada das composições na *engine* de execução de processos selecionada;
- Testar o funcionamento e analisar o desempenho da solução desenvolvida.

## 1.2. Justificativa

A justificativa para elaboração deste trabalho se deve à grande complexidade na tarefa executar composições de *Web Services* manualmente. Essa atividade demanda grande esforço e tempo de desenvolvimento, desviando o foco dos desenvolvedores da lógica de negócio da aplicação em questão. Diversas ferramentas como IDEs (*Integrated Development Environment*), juntamente com editores gráficos, estão disponíveis para auxiliar desenvolvedores na definição de um processo para execução de uma composição de serviços, porém trata-se ainda de um processo manual que necessita de intervenção humana e demanda tempo para ser realizada. A automatização do processo de descrição da composição de serviços em um padrão existente e a sua implantação em uma *engine* de execução evitam que desenvolvedores necessitem de fluência em padrões de descrição de composições, ter conhecimento de detalhes técnicos para implantação e invocação das composições na *engine* em uso ou mesmo ter que programar de maneira *ad-hoc* uma composição, manipulando os dados de entrada e saída e invocando diretamente os serviços participantes.

## 1.3. Delimitação do Escopo

Este trabalho visa à execução das composições de *web services* descobertas pelo protótipo desenvolvido por Hobold (2012). Não estão incluídos no escopo deste trabalho o tratamento e recuperação de falhas que possam ocorrer durante a execução de uma composição. Não será abordado o controle transacional na execução das composições. Métricas de qualidade de serviço (QoS) não serão coletadas. Não serão empregadas estruturas de



controle e paralelismo, pois conforme Hobold (2012), na fase de descoberta estas já não são suportadas.

#### **1.4. Metodologia**

Primeiramente será realizada a revisão bibliográfica a fim de se obter a fundamentação teórica sobre a descoberta e execução de composições de *web services* semânticos, com ênfase nos assuntos e tecnologias relacionados à execução das composições. Em seguida será feita a análise do resultado obtido por Hobold (2012) na fase de descoberta das composições e com isso determinar a abordagem que será adotada para realizar a implantação automática possibilitando a execução das composições de *web services* descobertas. Finalmente será desenvolvido um protótipo para avaliação da abordagem adotada e verificação dos resultados obtidos.

#### **1.5. Estrutura do Trabalho**

Este trabalho está organizado em seis capítulos, incluindo o capítulo introdutório, estruturados da seguinte forma:

- O capítulo 2 apresenta a base teórica para entendimento deste trabalho, abordando *Web Services* em seu conceito fundamental, as tecnologias empregadas na sua utilização prática e conceitos da web semântica. Apresenta também conceitos sobre o uso de *Web Services* como uma arquitetura de desenvolvimento de aplicações.
- O capítulo 3 apresenta a Plataforma para Descoberta e Execução de Composições de *Web Services Semânticos*, proposta por

Hobold e Siqueira (2012), descrevendo as fases de descoberta e execução, apresenta a abordagem proposta para execução das composições descobertas e propõe uma arquitetura de *software* para a fase de execução;

- O capítulo 4 descreve o desenvolvimento do protótipo para a fase de execução das composições a fim de validar a abordagem proposta, detalhando as tecnologias utilizadas e o ambiente de execução. Por fim apresenta os resultados alcançados através da execução de testes sobre o protótipo desenvolvido,
- O capítulo 5 finaliza o trabalho apresentando as conclusões obtidas e proposições para trabalhos futuros relacionados ao tema abordado.

## 2. Serviços Web

### 2.1. Arquitetura Orientada a Serviços

A Arquitetura Orientada a Serviços apresenta o conceito onde os serviços fornecidos por softwares são disponibilizados através de uma rede de computadores, na internet ou intranet, de forma a reduzir o acoplamento e aumentar o reuso. Estes serviços se comunicam através de simples troca de mensagens ou pela interação de dois ou mais serviços para execução de determinada atividade. Neste contexto, serviços são vistos como uma função bem definida, autônoma e que não depende do contexto ou estado de outros serviços.

Segundo HURWITZ et al. (2007)

[...] arquitetura orientada a serviços é uma arquitetura para construção de aplicações de negócios como um conjunto de componentes “caixa preta” orquestrados, fracamente acoplados para entregar um nível de serviço bem definido pela ligação dos processos de negócio.

Conforme BIH (2006), a arquitetura orientada a serviços é dividida em três principais componentes: o provedor de serviços (*service provider*), o solicitante de serviços (*service requester*) e o agente de serviços (*service broker*).

O provedor de serviços fornece a interface de serviço para um software que suporta um conjunto específico de tarefas. O provedor de serviços pode representar a entidade responsável por executar as tarefas ou apenas representar a interface para um subsistema que as executa.

O solicitante de serviços faz a descoberta e a invocação de serviços a fim de fornecer uma solução para o negócio. São componentes de aplicações que executam chamadas remotas aos provedores de serviços. O acesso aos provedores de serviços pode ser feito localmente ou através da Internet.

O agente de serviços atua como um repositório para as interfaces de serviços que são publicadas pelos provedores de serviço e é onde o solicitante de serviços efetua a busca por serviços.

A Figura 1 ilustra a Arquitetura Orientada a Serviços com seus componentes principais e as ações que estes executam. O provedor de serviços publica (*publish*) informações sobre o serviço prestado no agente de serviços. O solicitante de serviços efetua a busca (*find*) por serviços no agente de serviços e, quando encontra o serviço desejado, faz a ligação (*bind*) ao serviço no provedor de serviços.

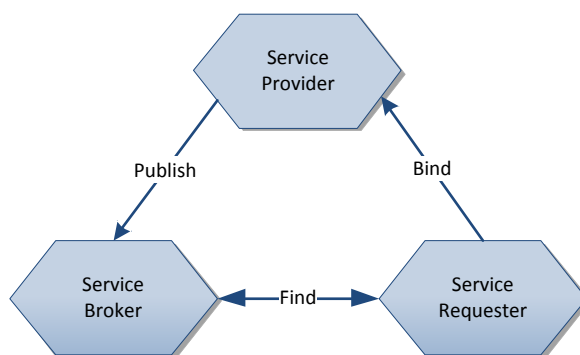


Figura 1 O Modelo SOA (GISOLFI, 2001 apud BIH, 2006)

As primeiras iniciativas com o objetivo de fornecer uma arquitetura orientada a serviços utilizavam tecnologias proprietárias que restringiam a utilização destes componentes a certas plataformas, como por exemplo, o RMI (*Remote Method Invocation*), mecanismo presente na plataforma Java para execução de chamadas de métodos remotos, ou o DCOM (*Distributed*

*Component Object Model*) similar ao anterior, porém desenvolvido pela Microsoft.

Conforme HURWITZ et al. (2007), por outro lado, a arquitetura orientada a serviços baseada em Web Services, descrita na Figura 2, utiliza diversos padrões abertos baseados em XML, entre eles o SOAP (*Simple Object Access Protocol*), WSDL (*Web Services Description Language*) e UDDI (*Universal Description, Discovery and Integration*). Com a adoção destes padrões é criada uma camada adicional entre os serviços, favorecendo a integração entre estes, independentemente da plataforma ou linguagem de programação em que são desenvolvidos.

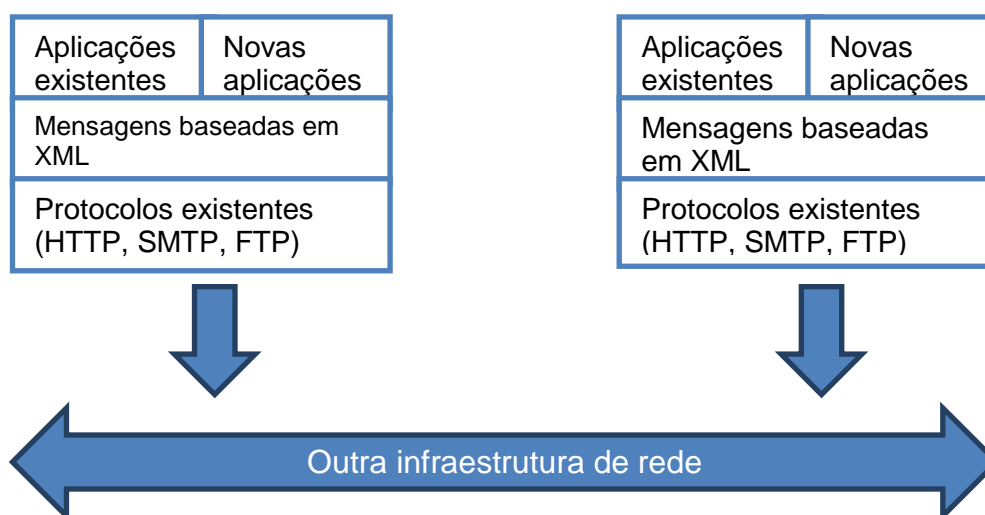


Figura 2 Arquitetura orientada a serviços baseada em Web Services (AYALA et al. 2002)

Conforme Juric, Mathew e Sarang (2006), a arquitetura orientada a serviços baseada em *Web Services* apresenta diversas e importantes mudanças comparadas às arquiteturas distribuídas anteriores:

- Suporte ao baixo acoplamento em operações de troca de dados;
- Suporte a interações síncronas e assíncronas;

- Uso de protocolos de internet padrão, como HTTP (*Hyper Text Transfer Protocol*), SMTP (*Simple Mail Transfer Protocol*), FTP (*File Transfer Protocol*), and MIME (*Multipurpose Internet Mail Extensions*), facilitando a comunicação através de conexões padrão de internet, mesmo através de *firewalls*.

## 2.2. Web Services

Os *Web Services* fornecem uma forma alternativa à interação tradicional da Web, onde um ser humano por meio de um navegador de internet acessa documentos através de *links* em uma página escrita em HTML (*Hyper Text Markup Language*), disponibilizadas por um servidor remoto. A interação através de *Web Services* permite que aplicações façam o intercâmbio de informações estruturadas, através do uso da linguagem XML, como ilustrado pela Figura 3.

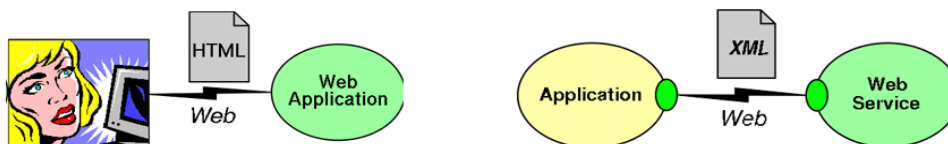


Figura 3 Interação tradicional da Web e Interação de Web Services (HAAS, 2003)

Em definição, *Web Services* são aplicativos independentes, modulares que podem ser descritos, publicados, localizados e chamados por uma rede (IBM, 2013). Para Gartner (apud AYALA et al. 2002), *Web Services* são componentes de software fracamente acoplados que interagem um com o outro dinamicamente através de tecnologias padrão para Internet.

Diversas tecnologias, como UDDI, WSDL e SOAP, todas baseadas em XML, suportam o desenvolvimento de Web Services, permitindo que estes sejam descobertos, descritos e invocados através da Internet. A adoção destes padrões permite a interoperabilidade entre sistemas heterogêneos.

Web Services são registrados em um repositório que permite que clientes localizem (*find*) o serviço esperado. Assim que localizado, o cliente obtém uma referência ao serviço e pode invocar os métodos que são disponibilizados pelo serviço através da interface pública deste, conforme a Figura 4.

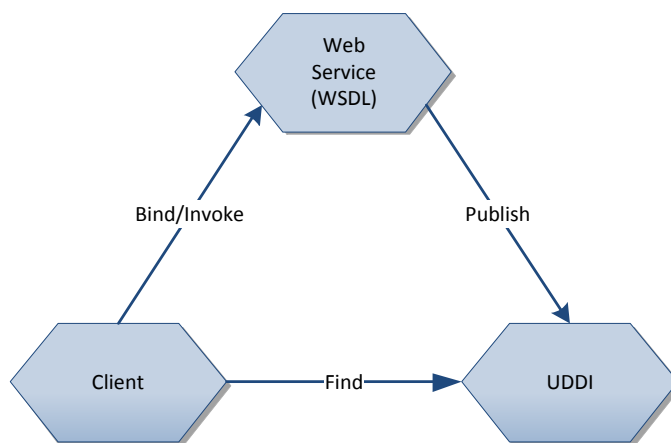


Figura 4 Registro, localização e invocação de um Web Service

### 2.2.1. XML

Segundo HURWITZ et al. (2007), XML (*eXtensible Markup Language*) é uma meta-linguagem de marcação padrão do W3C (*World Wide Web Consortium*) derivada da SGML (*Standard Generalized Markup Language*), utilizada para descrever estruturas de dados. Por ser baseada em texto e ter alta expressividade, é muito utilizada na troca de informações entre aplicações através da internet.

Os elementos em XML são definidos hierarquicamente a partir de um elemento raiz. Um elemento é a marca de abertura e fechamento, seus atributos e texto incluídos, ou uma marca vazia e seus atributos (TITTEL, 2002). Estas marcas, também conhecidas como *tags*, delimitam as estruturas de dados nos documentos XML. Diferentemente do HTML, todas as *tags* em XML devem ser obrigatoriamente fechadas, seja por auto-fechamento `<marca atributo="valor" />`, no caso das *tags* vazias, ou por uma marca de fechamento correspondente `<marca atributo="valor">conteúdo</marca>`. O conteúdo pode ser texto, outro(s) elemento(s) ou vazio. O Quadro 1 apresenta um exemplo de documento XML.

```
01 <Employee name=John>
02 <departament>systems</departament>
03 <age>25</age>
04 </Employee>
```

Quadro 1 Um documento XML

Por ser tratar de uma metalinguagem, XML não define um conjunto de *tags* específicas, como ocorre em HTML, por exemplo. A definição de quais *tags* e como estas serão utilizadas ficam a cargo de quem define o documento XML, podendo ser definido um *XML Schema*, que é o documento que descreve um conjunto de regras que definem quais elementos e como estes devem aparecer no documento, os atributos que podem ser definidos para um elemento, os tipos de dados e valores válidos para elementos e atributos.

Diversas linguagens, com as mais variadas finalidades, são baseadas em XML, por exemplo, MathML (*Mathematical Markup Language*) para representação de símbolos e fórmulas matemáticas (CARLISLE; ION; MINER,



2010), XHTML (*eXtensible Hypertext Markup Language*) para formatação de páginas Web (PEMBERTON et al., 2002) e SVG (*Scalable Vector Graphics*) para definição de gráficos vetoriais (DAILEY, 2010).

### 2.2.2. SOAP

SOAP é um protocolo baseado em XML que descreve um formato de mensagens para comunicação entre aplicações. SOAP fornece o mecanismo de comunicação para interação com Web Services (NEWCOMER, 2002).

Segundo Newcomer (2002), uma mensagem SOAP é composta pelos seguintes elementos, sendo que somente o envelope e corpo são obrigatórios:

- **Envelope** (*Envelope*): define o início e fim da mensagem;
- **Cabeçalho** (*Header*): contém qualquer atributo opcional utilizado para processamento da mensagem;
- **Corpo** (*Body*): contém os dados XML que compõem a mensagem que está sendo enviada;
- **Anexos** (*Attachment*): consiste em um ou mais documentos anexados na mensagem principal (Somente para SOAP com anexos);
- **Interação RPC** (*RPC interaction*): define como modelar interações estilo RPC com SOAP;
- **Codificação** (*Encoding*): define como representar informações simples e complexas que estão sendo transmitidas juntamente com a mensagem.

### 2.2.3. WSDL

Conforme Newcomer (2002), *Web Service Description Language* (Linguagem de Descrição de Web Services) é uma linguagem padrão, baseada em XML, para descrever e publicar os formatos e protocolos de um *Web Service* de uma maneira padrão. Foi definida em um esforço conjunto entre Microsoft, IBM e Ariba.

Os elementos em um documento WSDL contém a descrição dos dados trocados pelo *Web Service* e a descrição das operações executadas sobre estes dados. WSDL descreve sintaticamente como um *Web Service* deve ser acessado por um cliente, agindo como um contrato entre estes e desta maneira permite a comunicação efetiva entre ambos.

Viegas (2008) divide o documento WSDL em dois blocos, o abstrato e o concreto conforme Figura 5. O bloco abstrato apresenta a interface do serviço, ou seja, define as operações com seus parâmetros de entrada e saída descrevendo como os clientes devem invocar o serviço. O bloco concreto por sua vez, fornece informações sobre a localização do serviço e qual o protocolo utilizado na troca de mensagens entre serviço e clientes. A separação entre endereço e protocolo permite que um serviço seja disponibilizado através de endereços e protocolos distintos.

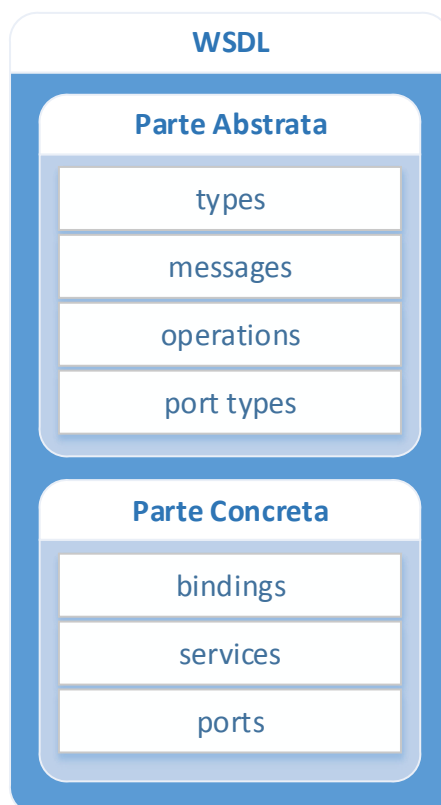


Figura 5 Estrutura do documento WSDL (VIEGAS, 2008)

No bloco abstrato são definidos os elementos *types*, *messages*, *operations* e *port types*. O elemento *types* permite especificar os tipos de dados que serão empregados nas mensagens trocadas pelo serviço. É possível utilizar *XML Schema* para definir as estruturas de dados. As mensagens que serão trocadas entre serviço e seu cliente são definidas pelo elemento *message*. Uma mensagem pode possuir diversas partes, cada uma possuindo nome e um tipo de dado associado. As operações, definidas através do elemento *operations*, descrevem as ações que o serviço pode realizar. Um conjunto de operações correlatas pode ser agrupado através de um elemento *port type*, semelhante à ideia de classes e métodos na programação orientada a objetos.

O bloco concreto é composto pelos elementos *bindings*, *ports* e *services*. O elemento *binding* prove detalhes sobre a transmissão das mensagens entre serviço e cliente informando o protocolo adotado, sendo este usualmente o SOAP, porém como o WSDL não fixa o uso deste protocolo outros podem ser utilizados, como por exemplo, HTTP ou SMTP. O element *port* define um *endpoint* individual pela especificação de um endereço único para um *binding*. O element *service* define quais são os *ports* suportados pelo serviço, sendo este um conjunto de elementos *port*.

#### **2.2.4. UDDI**

UDDI (*Universal Description, Discovery, and Integration*) segundo AYALA et al. (2002), é um registro baseado em XML para entidades globais se registrarem na Internet, que tem como objetivo principal permitir que empresas encontrem umas as outras e tornem seus sistemas interoperáveis para comércio eletrônico. É comparado ao serviço fornecido pelas listas telefônicas, permitindo a descoberta de serviços com base em seu nome (páginas brancas), seu ramo de negócios (páginas amarelas) ou suas características técnicas (páginas verdes).

O registro armazena informações sobre as atividades da organização e relacionadas aos serviços publicados de maneira categorizada, de acordo com diferentes critérios. Com isso, permite a outras empresas e parceiros procurar por um serviço e efetuar transações de modo rápido, fácil e dinâmico.

## 2.2.5. Orquestração e Coreografia de Web Services

Conforme Juric, Mathew e Sarang (2006), dependendo dos requisitos, composições de *Web Services* podem tratar de processos privados ou públicos, para isso são empregados dois termos: Orquestração e Coreografia, respectivamente.

A Orquestração de *Web Services* é caracterizada pela existência de um coordenador central, que é responsável por controlar a execução de um processo de negócio pela invocação de operações dos *Web Services* participantes deste processo. Os *Web Services* participantes não tem ciência, e nem precisam ter, de que fazem parte deste processo de negócio, apenas o coordenador central conhece as definições do processo, os *Web Services* participantes, as operações e a ordem de invocação. O fluxo da troca de mensagens definida pela orquestração é frequentemente utilizada em processos de negócios privados e é apresentada na Figura 6.

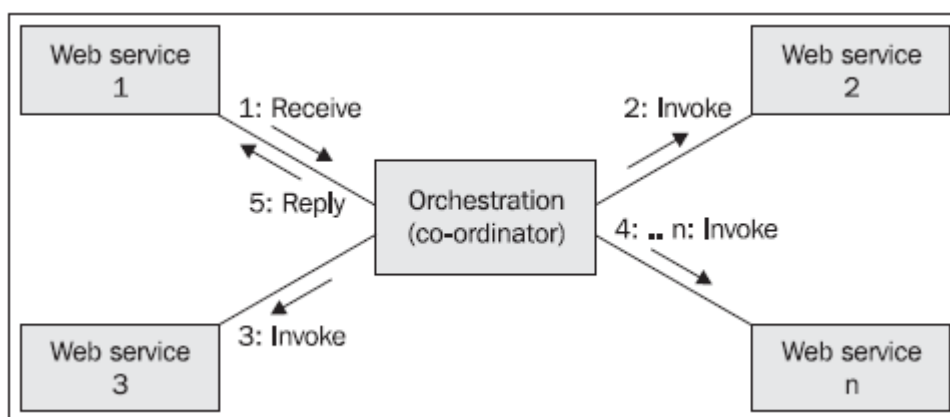


Figura 6 Orquestração de Web Services (JURIC; MATHEW; SARANG, 2006)

Coreografia por outro lado, não possui um coordenador central, cada *Web Service* participante envolvido na coreografia sabe exatamente quando atuar e com quem interagir. A coreografia é um esforço colaborativo com foco

na troca de mensagens em processos de negócios públicos. Todos os participantes devem estar cientes do processo de negócio, as operações a executar, trocas de mensagens e momento para troca de mensagens. A Figura 7 apresenta a coreografia, destacando os participantes, a troca de mensagens diretamente entre eles.

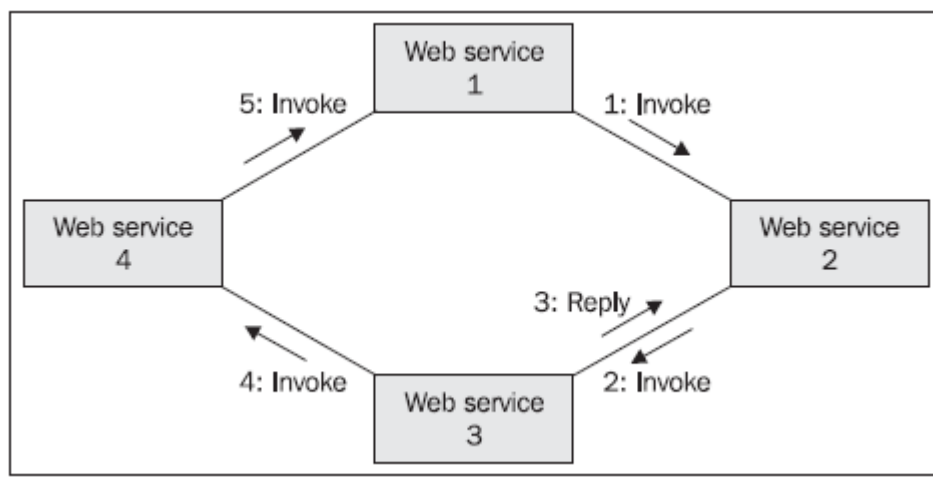


Figura 7 Coreografia de Web Services(JURIC; MATHEW; SARANG, 2006)

Ainda conforme Juric, Mathew e Sarang (2006), do ponto de vista da composição de Web Services para execução de processos de negócio a orquestração é um paradigma mais flexível e apresenta algumas vantagens em relação à coreografia. Na orquestração fica claro quem é o responsável pela execução do processo como um todo. A incorporação de novos participantes (*Web Services*) é facilitada, pois estes não precisam estar cientes de que fazem parte de um processo de negócio. Além disso, é possível definir cenários alternativos em caso de falhas.

Padrões existentes permitem a orquestração e coreografia de Web Services, por exemplo, o WS-BPEL para orquestração (ALVES, 2007) e o WS-CDL (*Web Services Coreography Description Language*) para coreografia (KAVANTZAS, 2005).

### 2.2.6. WS-BPEL

A especificação WS-BPEL é uma extensão do modelo de interação de Web Services para suporte a transações de negócios. A linguagem definida por WS-BPEL é baseada em XML e fornece meios para descrever formalmente processos de negócio e interação de protocolos de negócios.

A primeira versão da especificação, chamada de BPEL4WS (*Business Process Execution Language for Web Services*), foi lançada em julho de 2002, em esforço conjunto pela IBM, Microsoft e BEA. Em maio de 2003, com a adesão de empresas como SAP e Siebel Systems, foi lançada a versão 1.1 da especificação. Pouco antes deste lançamento, a especificação BPEL4WS foi submetida ao comitê técnico da OASIS (*Organization for the Advancement of Structured Information Standards*) para que esta pudesse ser desenvolvida como um padrão oficial e aberto.

De acordo com a especificação, um processo BPEL é um contêiner onde é possível declarar relacionamentos com parceiros externos, declarações para informações de processo, manipuladores para fins diversos e, principalmente, atividades a serem executadas.

Por meio de BPEL, é possível descrever tanto processos abstratos quanto processos executáveis. Processos abstratos descrevem processos sem detalhamento da execução destes, sendo usados para descrever modelos de processos ou processos de parceiros externos (Web Services), sem expor a lógica de negócio interna. A definição de um processo abstrato é feita conforme o exemplo apresentado no Quadro 2.

```
01 <process name="nome-processo-abstrato" targetNamespace="URI"  
02 xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/abstract">  
03 </process>
```

Quadro 2 Processo abstrato em WS-BPEL

Por outro lado, os processos executáveis definem o comportamento completo do negócio, tanto a parte visível externamente quando o processamento interno. Um processo executável é definido conforme Quadro 3.

```
01 <process name="nome-processo-executavel" targetNamespace="URI"  
02 xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable">  
03 </process>
```

Quadro 3 Processo executável em WS-BPEL

Sendo o processo o contêiner mais externo, quaisquer parceiros, informações de processo ou manipuladores declarados no contêiner do processo são considerados globais. Entretanto, também há o conceito de escopo, o que permite que estas declarações sejam feitas localmente. Para definição de um escopo é utilizado o elemento *scope*, conforme visto no Quadro 4.

```
01 <scope name="NomeEscopo" />
```

Quadro 4 Escopo em WS-BPEL

Com a utilização de escopos, pode-se dividir o processo de negócio, onde cada parte contém uma fração da lógica de negócio.

BPEL fornece a possibilidade de agregar Web Services e definir a lógica de negócio entre cada uma das iterações de serviços, ou seja, BPEL faz a orquestração destas interações de serviços.



A construção de processos em BPEL se dá pela definição de blocos chamados atividades. Estas atividades são classificadas em básicas, quando possuem propósitos específicos, e estruturadas, quando contêm outras atividades e definem lógica de negócio entre estas.

Para a troca de mensagens com parceiros externos (Web Services), BPEL fornece um conjunto de atividades. A atividade *receive* define o recebimento de mensagens de um parceiro externo, contendo o *link* para o parceiro e uma operação disponibilizada por este, além de poder definir variáveis para armazenamento da resposta e também uma associação a uma atividade *reply*, seu uso é exemplificado no Quadro 5.

```
01 <receive name="ReceiveRequestFromPartner" createInstance="yes"  
02 partnerLink="ClientStartUpPLT" operation="StartProcess" ... />
```

Quadro 5 Atividade receive em WS-BPEL

A atividade *reply*, com uso exemplificado no Quadro 6, é normalmente utilizada em associação a uma atividade *receive* na implementação de uma operação de requisição e resposta. Contém a resposta de uma requisição ou, no caso de falha, uma exceção.

```
01 <reply name="ReplyResponseToPartner" partnerLink="ClientStartUpPLT"  
02 operation="StartProcess" ... />
```

Quadro 6 Atividade reply em WS-BPEL

Para efetuar chamada a uma operação disponibilizada por um parceiro externo deve ser declarada a atividade *invoke*. Dentre as operações (primitivas de transmissão) definidas em WSDL em sua versão 1.1, são suportadas pela atividade *invoke* as operações do tipo *one-way* (parceiro externo recebe uma mensagem) e *request-response* (parceiro externo recebe uma mensagem e responde com uma mensagem correlata). Para tanto, deve-se definir uma

variável de entrada, tanto para operações *one-way* como *request-response*, além de uma variável de saída para operações do tipo *request-response*. O Quadro 7 apresenta a declaração de uma atividade *invoke* para uma operação *request-response*.

```
01 <invoke name="RequestResponseInvoke"  
02 partnerLink="BusinessPartnerServiceLink"  
03 operation="RequestResponseOperation" inputVariable="Input"  
04 outputVariable="Output" />
```

Quadro 7 Atividade invoke em WS-BPEL

Para a estruturação da lógica de negócio BPEL fornece atividades que se assemelham às estruturas de controle condicionais e de repetição das linguagens de programação tradicionais, além também de atividades para o controle do fluxo das chamadas aos parceiros externos.

Como estrutura de controle condicional BPEL fornece a atividade *if-else*. Para comandos de repetição são disponibilizadas as atividades *while*, *repeatUntil* e *forEach*. O uso destas estruturas de controle é apresentado no Quadro 8.

```

01 <if name="isOrderBiggerThan5000Dollars">
02   <condition>$order > 5000</condition>
03   <invoke name="calculateTenPercentDiscount" ... />
04   <else>
05     <reply name="sendNoDiscountInformation" ... />
06   </else>
07 </if>
08 <while>
09   <condition>$iterations > 3</condition>
10   <invoke name="increaseIterationCounter" ... />
11 </while>
12 <repeatUntil>
13   <invoke name="increaseIterationCounter" ... />
14   <condition>$iterations > 3</condition>
15 </repeatUntil>
16 <forEach parallel="no" counterName="N" ...>
17   <startCounterValue>1</startCounterValue>
18   <finalCounterValue>5</finalCounterValue>
19   <scope>
20     <documentation>
21       check availability of each item ordered
22     </documentation>
23     <invoke name="checkAvailability" ... />
24   </scope>
25 </forEach>

```

Quadro 8 Atividades de controle condicional e de repetição em WS-BPEL

Atividades que devem ser executadas em sequência devem ser contidas por uma atividade *sequence*, conforme exemplo no Quadro 9.

```

01 <sequence name="InvertMessageOrder">
02   <receive name="receiveOrder" ... />
03   <invoke name="checkPayment" ... />
04   <invoke name="shippingService" ... />
05   <reply name="sendConfirmation" ... />
06 </sequence>

```

Quadro 9 Atividade *sequence* em WS-BPEL

Por outro lado, as atividades que devem ser executadas paralelamente devem ser contidas por uma atividade *flow*, conforme Quadro 10.

```

01 <flow ...>
02   <links> ... </links>
03   <documentation>
04     check availability of a flight, hotel and rental car
05     concurrently
06   </documentation>
07   <invoke name="checkFlight" ... />
08   <invoke name="checkHotel" ... />
09   <invoke name="checkRentalCar" ... />
10 </flow>

```

Quadro 10 Atividade flow em WS-BPEL

Também é suportado por WS-BPEL o tratamento de exceções através do conceito *fault handlers* (manipuladores de falhas), que podem estar associadas a um escopo, processo ou diretamente a uma atividade do tipo *invoke*. Seu uso é apresentado no Quadro 11.

```

01 <faultHandlers>
02   <catch faultName="BookOutOfStockException"
03     faultVariable="BookOutOfStockVariable">
04     ...
05   </catch>
06   <catchAll>...</catchAll>
07 </faultHandlers>

```

Quadro 11 Manipulador de falhas para um processo em WS-BPEL

### 2.2.7. Web Services Semânticos

A web semântica estende a web atual descrevendo e gerenciando semanticamente as informações e serviços disponíveis, possibilitando que computadores compreendam e interpretem seu significado. Segundo BERNERS-LEE et al. (2001), na web semântica são fornecidas informações estruturadas e um conjunto de regras de inferência que computadores podem utilizar para executar raciocínio automatizado. Essas informações estruturadas e conjuntos de regras de inferências são descritas em linguagens de representação do conhecimento, como o RDF (*Resource Description*

*Framework*), que faz o uso de triplas formadas por sujeito, predicado e objeto para descrição de características ou propriedades de recursos, e ontologias, que são um modelo formal de informações que representa conhecimento sobre um domínio por meio da definição de conceitos e os relacionamentos entre estes.

Assim como a Web Semântica acrescenta dinamismo à Web, possibilitando que uma nova gama de aplicações seja desenvolvida, os Web Services Semânticos (SWS - *Semantic Web Services*) o fazem em relação aos Web Services, possibilitando que estes sejam descobertos, invocados, compostos e executados de forma automática. Os SWS são serviços, bem como suas mensagens de entrada e saída, descritos semanticamente através da adoção de ontologias. Com isto é possível inferir relacionamentos entre serviços e construir composições de serviços baseados nos serviços existentes.

Diversos padrões foram definidos com o objetivo de descrever Web Services semanticamente, entre eles OWL-S (*Ontology Web Language for Services*) (MARTIN et al., 2004), WSMO (*Web Service Modeling Ontology*) (BRUIJN et al., 2005) e SAWSDL (*Semantic Annotations for WSDL and XML Schema*) (FARRELL; LAUSEN, 2007).

OWL-S é uma ontologia OWL de alto nível que permite a descrição das propriedades e capacidades de um Web Service de forma que possa ser processado por computador. Consiste em três sub-ontologias relacionadas: *Profile*, *Process Model* e *Grounding*. Um serviço é descrito por uma instância da classe *Service*, e o perfil do serviço por uma instância da classe *ServiceProfile*. Além disso, a especificação contém também o *Grounding*, que é

o mapeamento da especificação abstrata do serviço para sua especificação concreta, que pode ser uma especificação WSDL.

Segundo Bruijn et al. (2005), a abordagem WSMO (*Web Service Modeling Ontology*) é composta por três elementos: a ontologia - modelo conceitual para Web Services Semânticos -, a linguagem WSML (*Web Services Modeling Language*) - provê sintaxe e semântica formal - e o ambiente WSMX (*Web Services Modeling Execution Environment*) - ambiente de execução.

Segundo Farrell e Lausen (2007), SAWSDL (*Semantic Annotations for WSDL and XML Schema*) define a maneira com a qual se adiciona anotações às diversas partes de um documento WSDL, como por exemplo, estruturas de mensagens de entrada e saída, interfaces e operações, com categorização da informação que pode ser usada para publicar um Web Service em um registro e posteriormente na descoberta e composição. As anotações semânticas referenciam um conceito em uma ontologia ou um documento de mapeamento. A especificação não requer nem restringe o uso de uma linguagem de expressão de ontologias ou de linguagem de mapeamento. Além disso, SAWSDL também define um mecanismo para especificação de mapeamento de tipos em um XML Schema a partir de e para uma ontologia.

### **2.3. Composição e Execução de Serviços**

Quando um único Web Service é incapaz de atender as necessidades de um consumidor de serviços, é possível combinar um conjunto de Web Services, desta forma criando um novo serviço a fim de satisfazer os requisitos de uma solicitação de um usuário. A composição é o processo de seleção,

combinação e execução de Web Services para atingir o objetivo de um usuário (MARTIN et al., 2007, p. 36).

Combinações de Web Services podem ser definidas manualmente, com ou sem o auxílio de ferramentas gráficas, onde o usuário define o fluxo de informações através dos diversos Web Services selecionados para, em conjunto, executar determinada operação ou ainda através de métodos automatizados, que são baseados, principalmente, em técnicas de *workflow* e Inteligência Artificial (RAO; SU, 2004).

Definir composições de Web Services é uma tarefa complexa, pois existe uma grande quantidade de Web Services disponíveis na Web para seleção, sendo que mudanças em serviços existentes e a criação de novos serviços podem ocorrer em tempo real. Adicionalmente, como os Web Services e suas operações são descritos apenas sintaticamente pelo WSDL, seu significado semântico só pode ser compreendido por humanos. Para que seja possível a integração de sistemas sem intervenção humana é necessária a descrição semântica dos Web Services.

Conforme AGARWAL et al. (2008), as abordagens de composição e execução de Web Services podem ser separadas em quatro grupos: intercalada, monolítica, estagiada e baseada em modelo.

Na **abordagem intercalada** não é possível separar composição e execução em diferentes fases, pois o funcionamento é do tipo caixa-preta, e o desenvolvedor não pode intervir no processo. Dinamicamente os serviços implantados são pesquisados com base em nos requisitos funcionais e não funcionais do serviço composto.

A **abordagem monolítica** impede a intervenção na fase de composição, porém o *workflow* pode ser inspecionado e possivelmente alterado antes do início da fase de execução. O processo de composição considera apenas as descrições dos serviços, ignorando se este está implantado e em execução. Por ser baseado apenas na descrição do serviço, um conjunto ou todos os serviços são selecionados e a rotina de planejamento encontra uma ou mais composições que podem ser passadas para a fase de execução. Isto possibilita que o processo de execução selecione o *workflow* mais apropriado baseado em algum critério de otimização.

Na **abordagem estagiada** as descrições dos serviços são separadas em tipos de serviços e instâncias. Primeiramente é gerado um *workflow* abstrato, baseado nos tipos de serviços, também chamado de composição lógica, que é posteriormente concretizado em um *workflow* executável através da seleção das instâncias de Web Services apropriadas, o que é chamado de composição física.

A **abordagem baseada em modelo** parte de linhas de workflows predefinidos (ou modelos) e os instancia para obter um *workflow* executável. A seleção e instanciação podem ser baseadas em políticas que descrevem o comportamento de acordo com objetivos, eventos ou situações em tempo de execução.

Além de agrupar as abordagens de composição e execução, AGARWAL et al. (2008) também as classificam segundo os critérios abaixo. A sumarização da classificação das abordagens elencadas em termos dos critérios propostos está na Tabela 1.

- As fases de composição e execução são separáveis (sim ou não);



- Quando ocorre a composição (em tempo de desenvolvimento ou em tempo de execução);
- Como ocorre a composição (baseada em busca, baseada em modelo);
- Qual informação é utilizada para a composição (tipos de serviços, instâncias de serviços publicadas, serviços implantados, modelos/políticas);
- Como eventos externos são tratados em tempo de execução ou adaptação (sob demanda, quando os eventos são tratados assim que ocorrem, ou gradual, quando eventos são tratados após um intervalo de tempo).

<b>Critério</b>	<b>Intercalada</b>	<b>Monolítico</b>	<b>Estagiado</b>	<b>Baseado em modelo</b>
<b>Separável</b>	Não	Sim	Sim	Sim
<b>Quando</b>	Execução	Desenvolvimento	Desenvolvimento	Desenvolvimento/Execução
<b>Como</b>	Busca	Busca	Busca	Modelo
<b>Qual informação</b>	Instâncias implantadas	Instâncias publicadas	Tipos e instâncias publicados	Modelos/Políticas, Tipos e instâncias publicados
<b>Eventos externos</b>	Sob demanda	Gradual	Gradual	Sob demanda/Gradual

Tabela 1 Classificação das abordagens de Composição e Execução de Web Services. (AGARWAL et al. 2008)

### 3. Plataforma para Descoberta e Execução de Composições de Web Services Semânticos

Hobold e Siqueira (2012) propõem uma plataforma que permite a descoberta automática e a execução de composições de *web services* semânticos. Nesta plataforma, anotações SAWSDL são extraídas de documentos WSDL e empregadas na construção de grafos de composição, ou seja, as composições são descobertas com base na semântica dos serviços. Após a descoberta das composições estas são convertidas para o padrão WS-BPEL e podem ser invocadas como um serviço único. A fase de descoberta das composições foi desenvolvida por Hobold (2012) e a fase de execução das composições será abordada no presente trabalho.

A plataforma proposta tem como característica ser independente de linguagem de programação e plataforma de execução e é composta por nove componentes, apresentados na Figura 8.

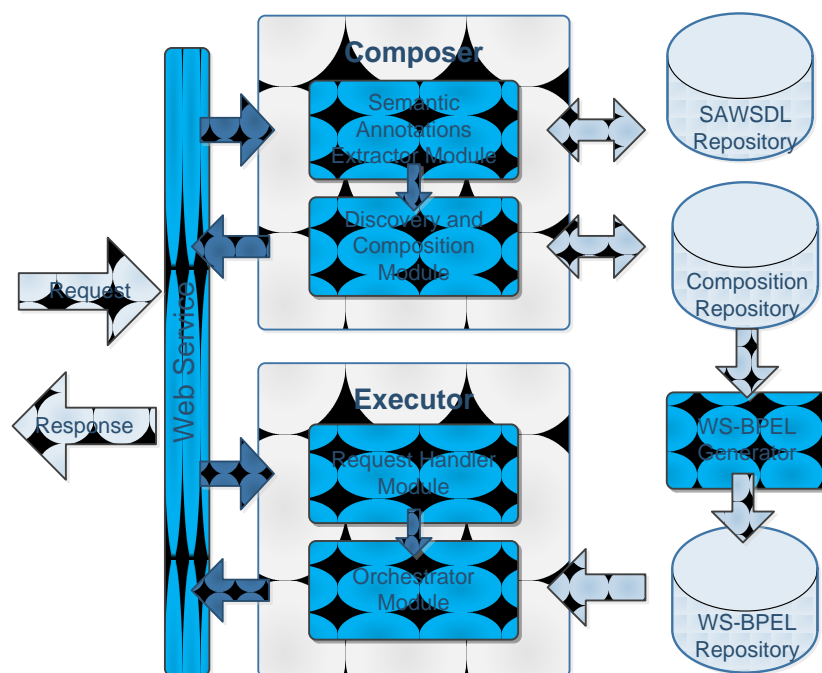


Figura 8 Arquitetura da Plataforma de Composição e Execução de *Web Services* (HOBOLD; SIQUEIRA, 2012)

O componente *Web Service* expõe três operações que são invocadas por clientes para requisição de um serviço específico, publicação de um novo serviço e execução de uma composição de serviços e fornecem como resultado, respectivamente, a composição descoberta, armazena o serviço no repositório e retorna a resposta produzida pela execução da composição.

O *Semantic Annotations Extractor Module* é responsável por extrair as anotações semânticas das descrições WSDL durante o processo de publicação e as armazenar em um banco de dados relacional representado pelo *SAWSDL Repository*. Este módulo também é responsável pela recuperação das anotações do *SAWSDL Repository* durante o processo de descoberta.

O *Discovery and Composition Module* é responsável por descobrir os Web Services compatíveis com os requisitos da solicitação, com base nas anotações extraídas anteriormente, efetuando a verificação de similaridade através da correspondência semântica. Este módulo constrói o grafo de composição baseado nas relações entre os conceitos associados às entradas, saídas e operações. Ao final do processo as composições descobertas são armazenadas no *Compositions Repository* e retornadas ao cliente pelo *Web Service*.

Assim que descobertas, as composições são processadas pelo *WS-BPEL Generator*. Este componente obtém a composição do *Composition Repository*, gera a representação da composição como um processo executável descrito no padrão WS-BPEL e a armazena no *WS-BPEL Repository*.

Clientes podem invocar a execução de uma composição através de uma requisição direta ao componente *Web Service*. Esta requisição será tratada

pelo *Request Handler Module*, que gerencia as requisições de entrada. Este módulo faz a ativação do *Orchestrator Module*, que executa os processos WS-BPEL que representam as composições, coordenando a invocação dos serviços envolvidos. A saída produzida por esta execução é então retornada ao cliente.

Seguindo os critérios de avaliação do processo de composição e execução de *web services* proposto por AGARWAL et al. (2008) pode-se afirmar que para a plataforma apresentada, as fases de composição e execução são separáveis, a composição ocorre em tempo de desenvolvimento e é realizada pela busca entre as instâncias publicadas. Conforme estas características a abordagem adotada pela plataforma pode ser considerada, dentre as classificações indicadas por AGARWAL et al. (2008), como uma abordagem monolítica, porém sem possibilidade de intervenção no *workflow* gerado.

### **3.1. Composição de Web Services**

O trabalho de Hobold (2012), que tem como objetivo analisar e desenvolver uma plataforma para a composição automática de Web Services Semânticos, segue a arquitetura proposta por Hobold e Siqueira (2012), cobrindo a descoberta das composições, porém não abrangendo a execução das composições encontradas.

Conforme Hobold (2012), para a descoberta de um serviço ou composição de serviços que satisfaça os requisitos de uma solicitação, um usuário (desenvolvedor) da plataforma deve descrever estes requisitos através dos seguintes parâmetros:

- Lista de entradas disponíveis;
- Lista de saídas desejadas;
- Lista de operações desejadas;
- Profundidade máxima da composição;
- Tempo máximo de espera pela resposta (*timeout*);
- Permissão para reconstruir composições (*allowRebuild*).

Esta requisição deve ser codificada pelo desenvolvedor, sendo as listas de entradas disponíveis, saídas desejadas e operações classificadas por conceitos definidos em ontologias de domínio. Como resposta a esta requisição o usuário obtém um serviço ou composição de serviços que atenda aos requisitos especificados, podendo este serviço ou composição de serviços atender total ou parcialmente as saídas desejadas.

### **3.2. Execução das Composições de Web Services**

Para a execução das composições, de acordo com a arquitetura proposta por Hobold e Siqueira (2012), os componentes envolvidos são o *WS-BPEL Generator*, o *WS-BPEL Repository*, o *Request Handler Component* e o *Orchestrator Module*. Estes componentes e as tarefas que estes realizam fazem parte do escopo deste trabalho.

Implicitamente, em razão da existência do *WS-BPEL Repository*, observa-se a opção pela estratégia de execução baseada na orquestração de serviços em vez da coreografia, justificado pelo motivo de que os serviços participantes não necessariamente estão cientes de que fazem parte de um processo de negócio.

Durante a fase de descoberta das composições pelo protótipo desenvolvido por Hobold (2012), para cada uma das saídas desejadas em uma composição é gerado um caminho. Este caminho é composto por uma sequência de operações associadas aos seus parâmetros de entrada e saída.

A abordagem para transformação de uma composição de Web Services em um processo executável descrito no padrão WS-BPEL é realizada com base nos caminhos obtidos na descoberta da composição, que são mapeados diretamente para uma sequência de invocações de serviços. Cada caminho envolve a execução de vários serviços, que são encadeados por meio de seus parâmetros de entrada e saída. Os caminhos que formam uma composição são independentes entre si, permitindo a execução paralela destes. Essa abordagem é apresentada detalhadamente a seguir.

Primeiramente, deve-se definir uma atividade *receive* para recebimento dos parâmetros de entrada (as entradas disponíveis na fase de descoberta da composição). Na sequência, uma atividade *assign* deve ser utilizada na definição e inicialização das variáveis que serão utilizadas durante a execução do processo.

Em seguida, para cada um dos caminhos deve ser gerada uma atividade *sequence*, para garantir a execução sequencial das atividades contidas. Esta deve possuir atividades *assign* e *invoke* declaradas sequencialmente, a fim de definir, respectivamente, os parâmetros de entrada para invocação de um serviço e a invocação do serviço, para cada uma das operações que compõem o caminho atual. Após, uma última atividade *assign*, responsável por associar o resultado da execução do caminho à saída do processo, é definida.

Como cada uma das atividades *sequence* definidas no passo anterior são independentes entre si, a execução destas pode ser realizada paralelamente a fim de melhorar o desempenho da execução da composição. Para que isto ocorra deve-se incluir este conjunto de atividades *sequence* em uma atividade *flow*.

Para enviar o resultado da execução da composição ao cliente que a requisitou, uma atividade *reply* deve ser utilizada.

Como as atividades de recebimento das entradas, de execução dos caminhos paralelamente e de fornecimento das saídas devem ser executadas de forma sequencial, estas devem ser definidas dentro de uma atividade *sequence*.

A Figura 9 apresenta o diagrama de atividades para a abordagem de geração do processo WS-BPEL a partir de uma composição de serviços descoberta.

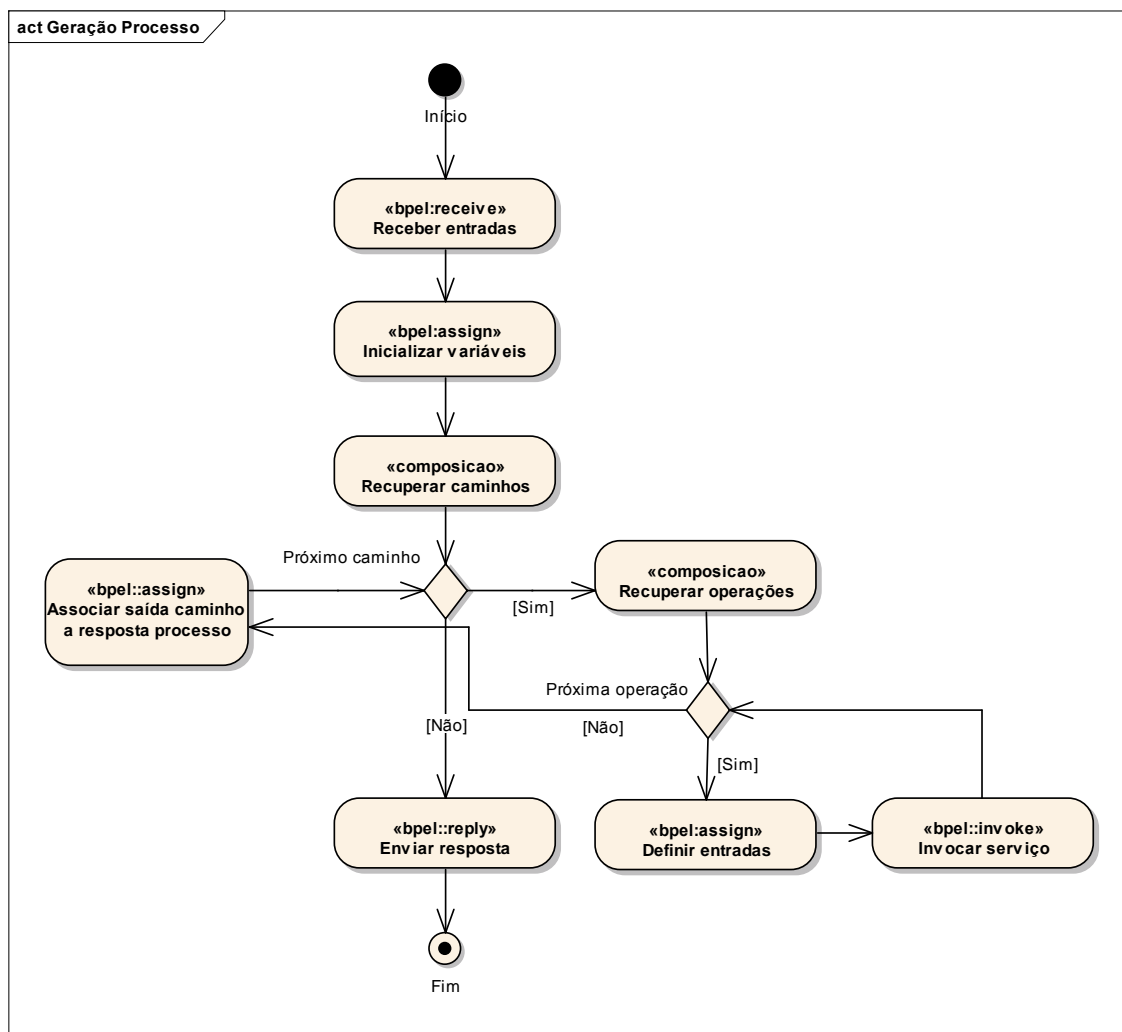


Figura 9 Diagrama de Atividades para geração de processo WS-BPEL

A Figura 10 apresenta uma composição transformada em um processo executável WS-BPEL, criado através do BPEL Designer<sup>1</sup>, conforme a abordagem proposta. Primeiramente, uma atividade *sequence*, chamada "Composicao" é definida. Em seguida, uma atividade *receive* realiza o recebimento das entradas da composição. Após, cada um dos caminhos é representado por uma atividade *sequence*, todos incluídos em uma atividade *flow*. No "Caminho 1", pode-se observar uma atividade *assign* inicial, responsável por atribuir as entradas da atividade *invoke* subsequente, a

<sup>1</sup><http://www.eclipse.org/bpel/>



atividade *invoke* que efetua a invocação de um Web Service externo e em seguida outra atividade *assign*, essa por sua vez responsável por associar a saída produzido pelo caminho ao resultado do processo. O “Caminho 2” é definido de maneira similar. Finalmente a atividade *reply*, que retorna o resultado ao solicitante é declarada.

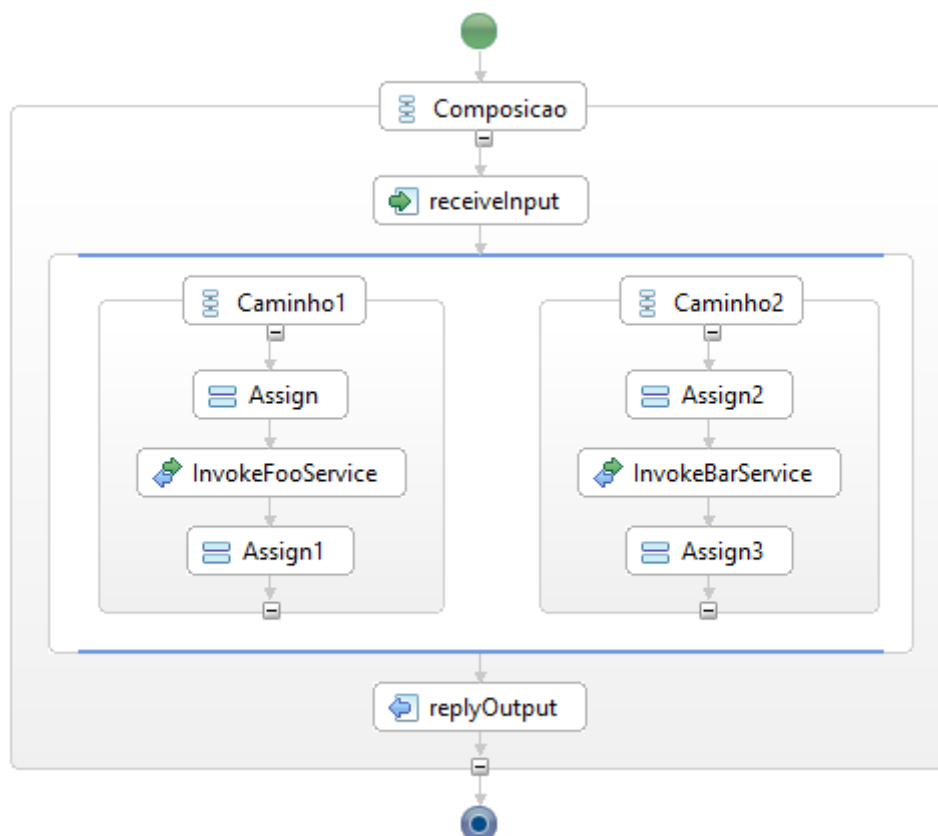


Figura 10 Composição de *Web Services* transformada em um processo executável WS-BPEL conforme abordagem proposta

### 3.3. Arquitetura

As operações que devem ser realizadas na fase de execução, conforme a arquitetura da Plataforma para Descoberta e Execução de Composições de Web Services Semânticos (Hobold e Siqueira, 2012), são a implantação e a execução de uma composição de serviços.

Para implantação de um processo descrito no padrão WS-BPEL em uma *engine* de orquestração, além da própria descrição do processo, os *engines* pesquisados exigem também outros artefatos para implantação. Estes artefatos em sua grande maioria são: um arquivo WSDL que descreve o acesso ao processo, cópias dos arquivos WSDL dos serviços participantes e, em alguns casos, arquivos descritores de implantação com padrões próprios de cada *engine*. Por esta razão, a arquitetura proposta por Hobold e Siqueira (2012) foi estendida, sendo o *WS-BPEL Repository* substituído pelo *Artifacts Repository* e o *WS-BPEL Generator* desmembrado em cinco componentes (*Deployment Module*, *WS-BPEL Generator*, *WSDL Generator*, *Deploy Generator* e *WSDL Reader*), conforme apresentado na Figura 11 e descrito a seguir.

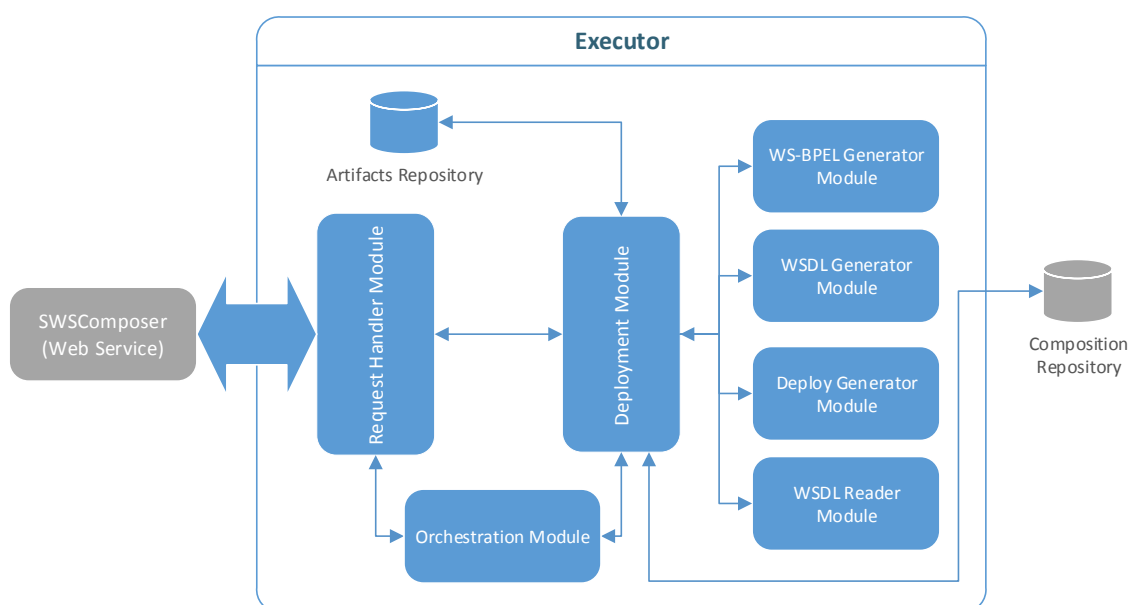


Figura 11 Arquitetura proposta para execução de composições de Web Services

O *Request Handler Module*, além de receber as requisições de execução de uma composição como previsto inicialmente, também será responsável pela comunicação entre as camadas de descoberta e execução da

plataforma, fornecendo as operações para implantação e reimplantação de uma composição.

No *Deployment Module* as composições são obtidas a partir do *Composition Repository* e encaminhadas ao *WS-BPEL Generator*, *WSDL Generator* e *Deploy Generator*, para geração dos respectivos artefatos de implantação. A seguir, estes artefatos são empacotados e encaminhados ao *Orchestration Module* para implantação da composição.

O *Artifacts Repository* é responsável pelo armazenamento dos artefatos de implantação gerados para cada composição e das cópias dos documentos WSDL de cada um dos serviços participantes de uma composição.

O *WS-BPEL Generator*, *WSDL Generator* e *Deploy Generator* são os módulos geradores dos artefatos de implantação, os arquivos WS-BPEL, WSDL e descritor de implantação respectivamente. Estes módulos interpretam a composição e geram os arquivos de acordo com a especificação de cada padrão.

O *WSDL Reader Module* é responsável por adquirir uma cópia do arquivo WSDL de cada serviço pertencente à composição e enviá-la ao *Artifacts Repository* para armazenamento. Além disso, efetua a leitura e análise destes arquivos a fim de obter referências para as operações e estruturas de dados utilizadas na composição.

### **3.3.1. Implantação de uma Composição**

Quando o *Request Handler Module* recebe uma solicitação para implantação de uma composição, o *Deployment Module* é acionado e primeiramente este recupera a composição informada pelo solicitante a partir

do *Composition Repository*. A partir deste momento, cópias dos arquivos WSDL são obtidas pelo *WSDL Reader Module*. Estas cópias também são analisadas a fim de obter as referências às operações e estruturas de dados utilizadas pela composição. Em seguida, os geradores dos artefatos de implantação são invocados e cada um deles, após a geração do respectivo artefato, os armazena no *Artifacts Repository*. Com os artefatos gerados e cópias dos arquivos WSDL dos serviços participantes obtidas, o *Deployment Module* deve gerar o pacote de implantação a partir dos artefatos e encaminhá-lo ao *Orchestration Module*, finalizando com isto a operação. O diagrama de sequência apresentado na Figura 12 mostra a interação entre os componentes da arquitetura, no processo de implantação de uma composição.

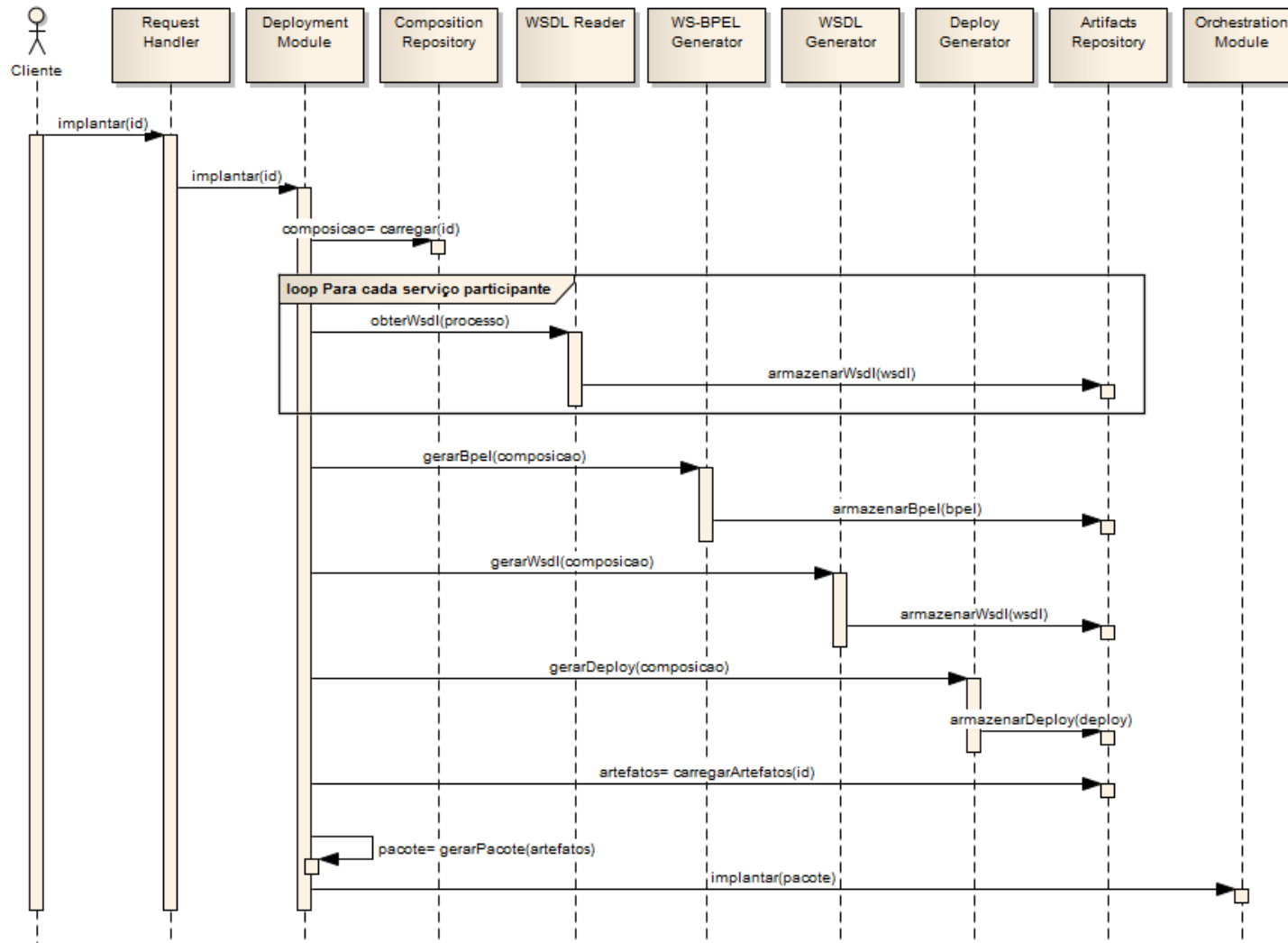


Figura 12 Diagrama de sequência do processo de implantação de uma composição

### 3.3.2. Execução de uma Composição

Para a execução de uma composição, assim que o *Request Handler Module* recebe uma requisição para tal, este primeiramente deve obter a composição a partir do *Composition Repository* e após, verificar junto ao *Orchestration Module* se a composição encontra-se implantada e apta para execução. Estando a composição implantada, o *Request Handler Module* deve solicitar a execução ao *Orchestration Module*. Este deverá efetuar a execução da composição, invocando os *Web Services* participantes, de acordo com o processo definido no respectivo documento WS-BPEL, gerado durante a operação de implantação. Por fim, o resultado da execução é retornado ao solicitante da execução. O diagrama de sequência referente à operação de execução de uma composição é apresentado na Figura 13.

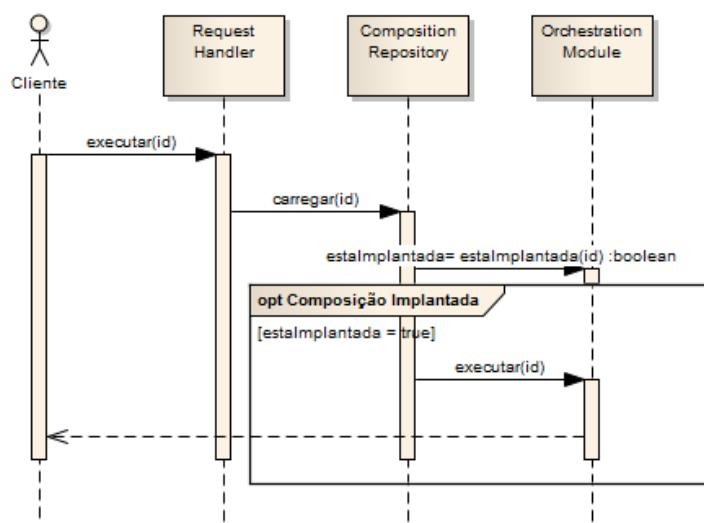


Figura 13 Diagrama de Sequência para Execução de uma Composição

### 3.4. Composition Repository

O modelo entidade-relacionamento do banco de dados para armazenamento das composições definido por Hobold (2012), por ter sido

planejado para atender a fase de descoberta das composições cobrindo o aspecto semântico, necessitou alterações a fim de permitir a execução destas. Para abranger os aspectos referentes à execução das composições foram incluídas colunas para definição dos tipos de dados das entradas e saídas e também definidas relações para indicação da origem dos parâmetros de entrada das operações e origem da saída dos caminhos. Este modelo é apresentado na Figura 14.

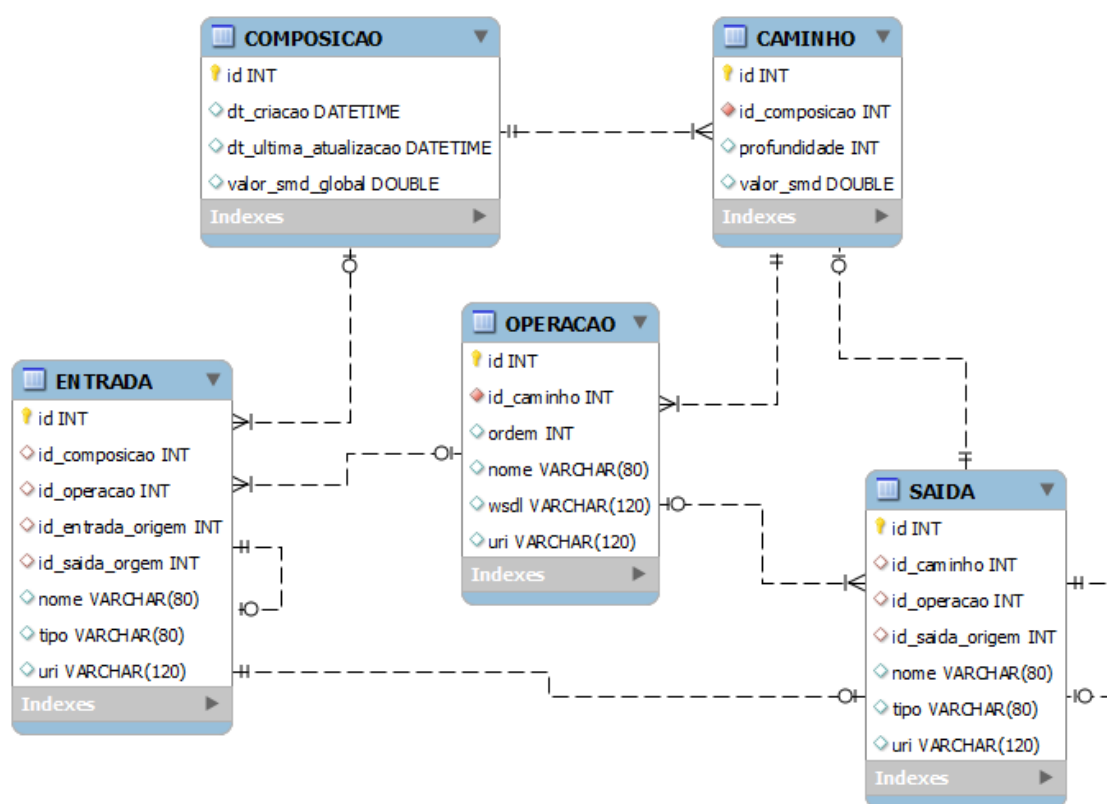


Figura 14 Modelo entidade-relacionamento do banco de dados de composições adequado

## 4. Implementação e Avaliação do Protótipo

O protótipo foi implementado tendo os componentes previstos na arquitetura escritos em linguagem PHP (*PHP: Hypertext Preprocessor*), baseados no *framework* CakePHP<sup>2</sup> e executando no Apache HTTP Server, exceto pelo *Orchestration Module*, que será representado pelo Apache ODE executando sobre o Apache Tomcat. Os servidores Apache HTTP Server e Apache Tomcat são desenvolvidos e distribuídos pela licença Apache<sup>3</sup> pela Apache Software Foundation.

As composições serão carregadas diretamente do *Composition Repository*, implementado como um banco de dados no MySQL, através de consultas SQL (*Structured Query Language*). O MySQL é um DBMS (*Data Base Management System*) *open source* desenvolvido e distribuído pela Oracle Corporation.

Os componentes de *software* do ambiente de execução, descritos anteriormente, são apresentados através de diagrama de implantação na Figura 15.

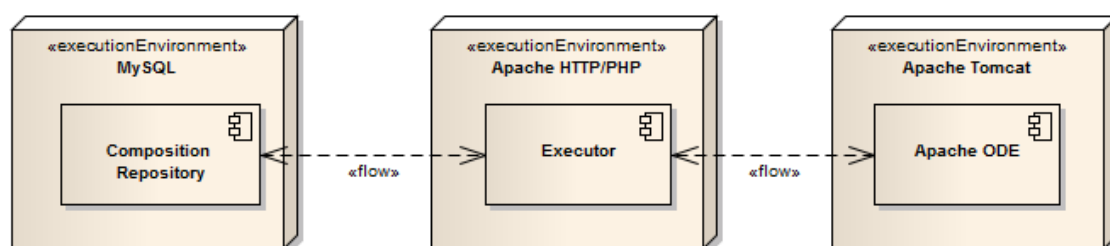


Figura 15 Diagrama de Implantação do protótipo

<sup>2</sup> <http://www.cakephp.org>

<sup>3</sup> <http://www.apache.org/licenses/>



#### 4.1. Linguagem de Programação e Framework de Desenvolvimento

Segundo The PHP Group (2013), PHP é uma linguagem de script *open-source* de uso geral, amplamente utilizada no desenvolvimento de aplicações Web. Mesmo tendo como foco *scripts* para execução do lado do servidor, é possível utilizá-lo para execução de *scripts* de linha de comando e até mesmo aplicações *desktop*.

Cake Software Foundation (2013) define o CakePHP como um *framework* de desenvolvimento rápido de aplicações *web*. O CakePHP fornece ferramentas para geração de código em tempo de desenvolvimento e execução além de bibliotecas embutidas para acesso a banco de dados, *caching*, validação, autenticação e segurança. Seu objetivo primário é permitir o trabalho de maneira estruturada e rápida, sem perda de flexibilidade. O desenvolvimento no *framework* CakePHP segue princípio da convenção sobre a configuração, auxiliando no desenvolvimento mais rápido e com menos código fonte, dispensando o uso de arquivos externos de configuração. É construído de acordo com a arquitetura MVC (*Model-View-Controller*) e implementa padrões de projeto como o *Active Record*. Sua distribuição é feita sob a licença MIT<sup>4</sup>.

A adoção da linguagem de programação PHP e do *framework* CakePHP para implementação do protótipo se deve à experiência do autor.

---

<sup>4</sup><http://opensource.org/licenses/MIT>

## 4.2. Componentes

Os módulos *Deployment Module*, *WS-BPEL Generator Module*, *WSDL Generator Module*, *Deploy Generator Module*, *WSDL Reader Module* e o *Artifacts Repository* foram implementados como *Components* do CakePHP, respectivamente nomeados de *DeploymentComponent*, *WsBpelGeneratorComponent*, *WsdGeneratorComponent*, *DeployGeneratorComponent*, *WsdIReaderComponent* e *ArtifactsRepositoryComponent*, por razão da convenção de nomes<sup>5</sup> adotada pelo CakePHP. As requisições HTTP, necessárias para a obtenção das cópias dos arquivos WSDL dos serviços participantes pelo *WsdIReaderComponent*, foram encapsuladas na classe *HttpClientComponent*, também implementada em forma de *Component* e utilizando a biblioteca cURL<sup>6</sup> do PHP.

Conforme consta na documentação do CakePHP (*Cake Software Foundation*, 2013), *Components* são pacotes de lógica que são compartilhados pelos *Controllers*. Além da convenção de nome adotada pelo *framework*, estas classes também devem estender a classe *Component*, presente no *core* do CakePHP.

Para os componentes *WsBpelGeneratorComponent*, *WsdGeneratorComponent* e *DeployGeneratorComponent*, responsáveis por gerar os arquivos WS-BPEL, WSDL e descritor de implantação respectivamente, todos estes arquivos XML, foi definida uma superclasse abstrata, chamada de *AbstractXmlGeneratorComponent*, que define a estrutura geral para um gerador de arquivo XML.

---

<sup>5</sup><http://book.cakephp.org/2.0/en/getting-started/cakephp-conventions.html>

<sup>6</sup><http://php.net/manual/en/book.curl.php>

O modelo de classes apresentado na Figura 16 apresenta estas classes, com a definição de seus atributos, métodos e relacionamentos estabelecidos entre as estas.

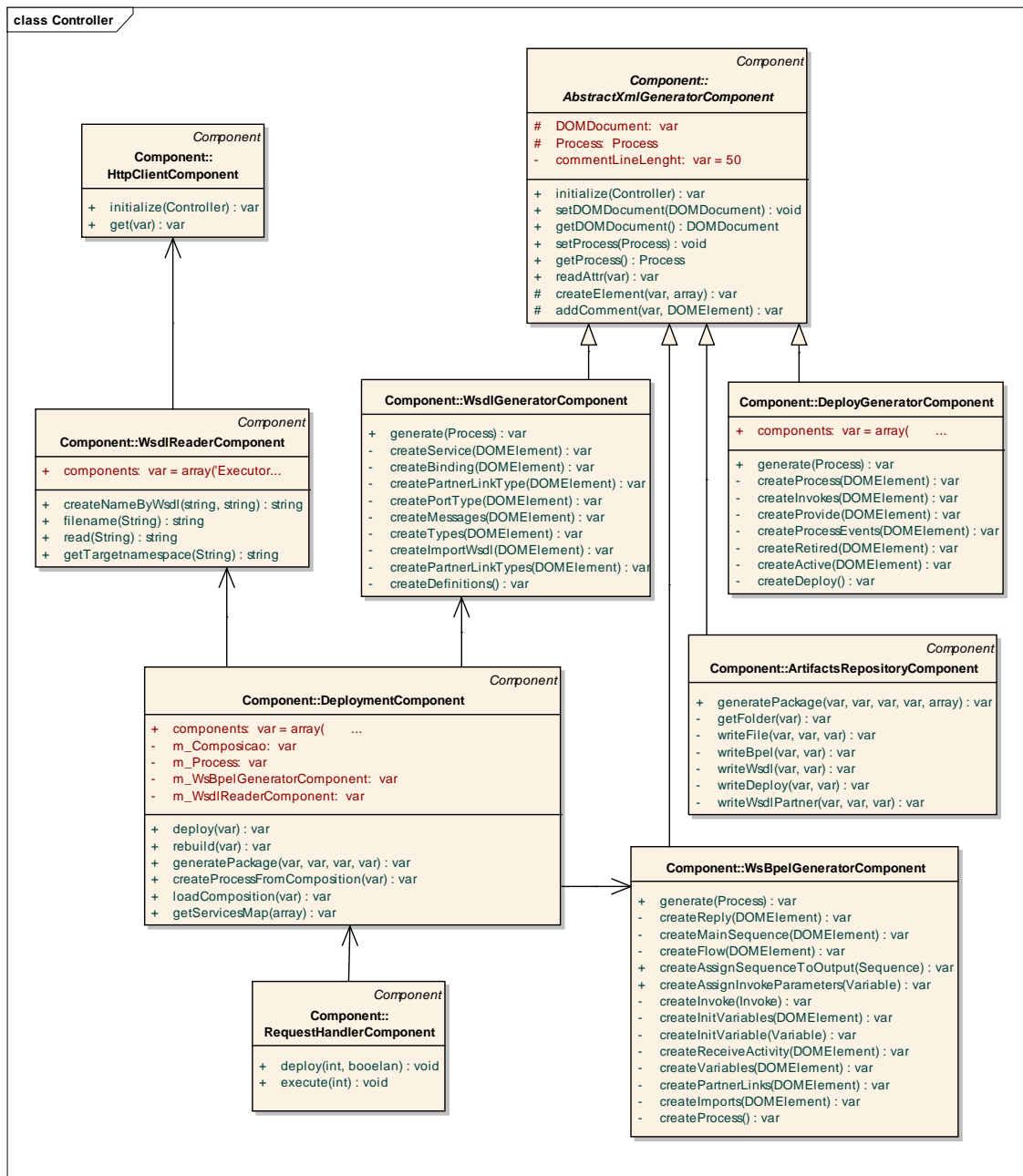


Figura 16 Diagrama de classes para os componentes da arquitetura

### 4.3. Modelos

Conforme os padrões de desenvolvimento definidos pelo CakePHP foram definidas classes modelo (estendendo a classe *Model* do *core* do CakePHP) para cada uma das tabelas presentes no banco de dados de composições (*Composition Repository*). Desta maneira, cada uma destas classes fornece métodos para recuperação e manipulação dos dados presentes nas tabelas referenciadas, seguindo o padrão de projeto *Active Record*. Conforme Flower et al. (2002), o padrão *Active Record* define que um objeto representa uma linha de uma tabela ou *view* de um banco de dados, encapsulando o acesso ao banco de dados e incluindo lógica de acesso aos dados.

No CakePHP os relacionamentos<sup>7</sup> entre as classes de modelos, de acordo com o modelo entidade-relacionamento, devem ser descritos através dos atributos de classe *hasOne* (um para um), *hasMany* (um para muitos), *belongsTo* (muitos para um) e *hasAndBelongsToMany* (muitos para muitos), estabelecendo a classe referenciada e nome da chave estrangeira no banco de dados, sendo que a relação de fato entre as classes é estabelecida em tempo de execução, pelo *framework*, a partir destas declarações.

O diagrama de classes destes modelos é apresentado na Figura 17, sendo que os atributos e métodos foram propositalmente ocultados, pois se referem a particularidades do *framework* irrelevantes neste contexto.

---

<sup>7</sup><http://book.cakephp.org/2.0/en/models/associations-linking-models-together.html>

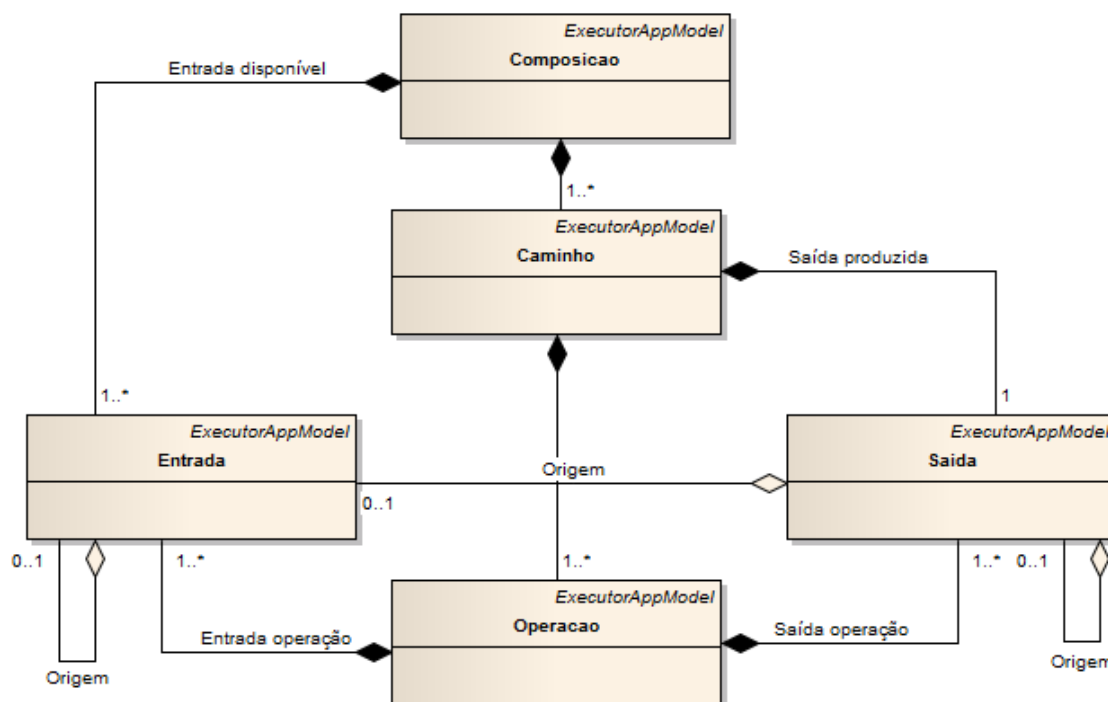


Figura 17 Diagrama de classes para os Modelos

#### 4.4. Apache ODE

Para o *Orchestrator Module* neste protótipo foi escolhido o Apache ODE (*Orchestration Director Engine*), *engine* de orquestração capaz de executar processos de negócios descritos no padrão WS-BPEL, porém outra *engine* compatível com o este padrão poderia ter sido empregada. As razões para opção pelo Apache ODE foram: a compatibilidade com os padrões BPEL4WS e WS-BPEL 2.0, o grau de maturidade do projeto em relação às outras *engines* pesquisadas e licença de distribuição *open-source*<sup>8</sup>.

A Tabela 2 apresenta de maneira resumida as *engines* de orquestração pesquisadas. Vale ressaltar que somente as *engines* com licença *open-source* foram consideradas.

Product	Release Date	Compatibility	License
Apache ODE	2006	BPEL4WS 1.1, WS-BPEL 2.0	Apache

<sup>8</sup><http://www.apache.org/licenses/LICENSE-2.0>

Open ESB	2013	WS-BPEL 2.0	CDDL
OW2 Orchestra	2012	WS-BPEL 2.0	LGPL
Petals BPEL Engine	2009	WS-BPEL 2.0, WSDL 1.1 and 2.0	LGPL

Tabela 2 Comparação de *engines* BPEL resumida (WIKIPEDIA, 2013)

#### 4.5. Artefatos de Implantação

Para implantação de uma composição no Apache ODE são necessários alguns artefatos que compõem o pacote de implantação. São eles: a composição descrita no padrão WS-BPEL, o arquivo WSDL que descreve como a composição poderá ser invocada, o arquivo de *deploy* do Apache ODE e cópias dos arquivos WSDL de cada um dos serviços participantes da composição.

Para a geração dos artefatos de implantação, em razão das diversas relações entre os elementos de cada artefato, além de relações entre elementos de artefatos diferentes, foi definida uma estrutura de classes intermediária na qual cada composição é convertida, antes de ser encaminhada aos geradores respectivos. Como todos os artefatos gerados são arquivos XML, para cada elemento destes documentos foram criadas classes que os representam, compostas pelos atributos do próprio elemento e pelas relações de composição com subelementos e atributos cujo valor faz referência a outros elementos. Por exemplo, no Quadro 12 o trecho XML mostra o elemento *partnerLink*, que é declarado no documento WS-BPEL, que faz referência ao elemento *partnerLinkType*, definido no *namespace* com prefixo “tns”, este declarado no documento WSDL.

```
01 <bpel:partnerLink name="GlobalWeatherPL"
02 partnerLinkType="tns:GlobalWeatherPLT"
03 partnerRole="GlobalWeatherPLRole"></bpel:partnerLink>
```

Quadro 12 Exemplo de referência entre elementos em arquivos XML

No quadro 13 é apresentada a declaração do elemento *partnerLinkType*, referenciado pelo elemento *partnerLink*, como descrito acima.

```

01 <plnk:partnerLinkType name="GlobalWeatherPLT">
02   <plnk:role name="GlobalWeatherPLRole"
03     portType="wsdl:GlobalWeatherSoap" />
04 </plnk:partnerLinkType>

```

Quadro 13 Declaração do elemento *partnerLinkType* referenciado pelo elemento *partnerLink*

A Figura 18 apresenta a estrutura de classes para os elementos exemplificados acima, com a definição dos atributos e relacionamentos definidos de acordo com a abordagem adotada na definição das classes para representar cada um dos elementos que compõem os documentos XML necessários.

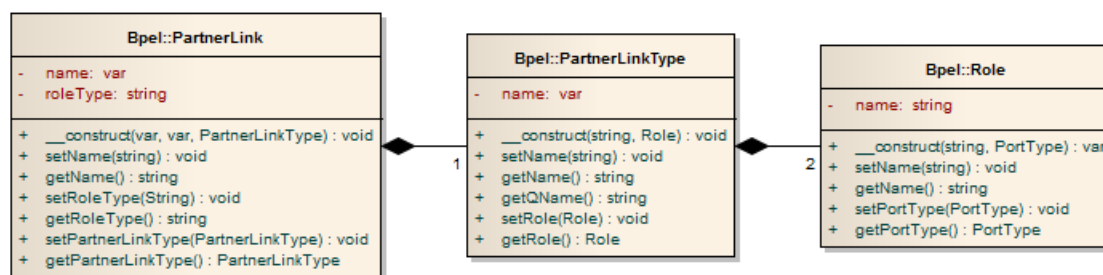


Figura 18 Diagrama de classes representando elementos do documento WS-BPEL

## 4.6. Avaliação do Protótipo

Para avaliar o funcionamento do protótipo foi definida uma composição fictícia contendo diversas situações que podem ocorrer em uma composição a fim de verificar se o protótipo é capaz de gerar corretamente os artefatos de implantação de acordo com as definições da composição. Foram utilizados dois *web services* executando em máquina local, o primeiro chamado *echoService* recebe uma *string* como entrada e retorna a *string* “echo” concatenado com o parâmetro de entrada como saída e o segundo chamado *concatService* recebe

duas *strings* como parâmetros de entrada e fornece como saída estas duas *strings* concatenadas. A ferramenta SoapUI<sup>9</sup> foi utilizada para realizar a invocação da composição.

O exemplo consiste em uma composição contendo dois caminhos, dois parâmetros de entrada (“a” e “b”) e dois parâmetros de saída (“c” e “d”). Para este exemplo todos os parâmetros de entrada e saídas são tipo *string*. No primeiro caminho, responsável por gerar a saída “c” da composição, são invocadas as operações *echo* e *concat*, nesta ordem. A primeira operação recebe como entrada o parâmetro “a” da composição e a segunda operação tem como entradas a saída da operação anterior e o parâmetro “b” da composição. O segundo caminho é composto por três operações, sendo as duas primeiras *echo* e a terceira *concat*, e é responsável por gerar a saída “d” da composição. Os parâmetros de entrada da composição “a” e “b” são passados como entrada das duas operações *echo* e a saída gerada por estas são passadas como entrada da operação *concat*. O *workflow* desta composição é apresentado na Figura 19.

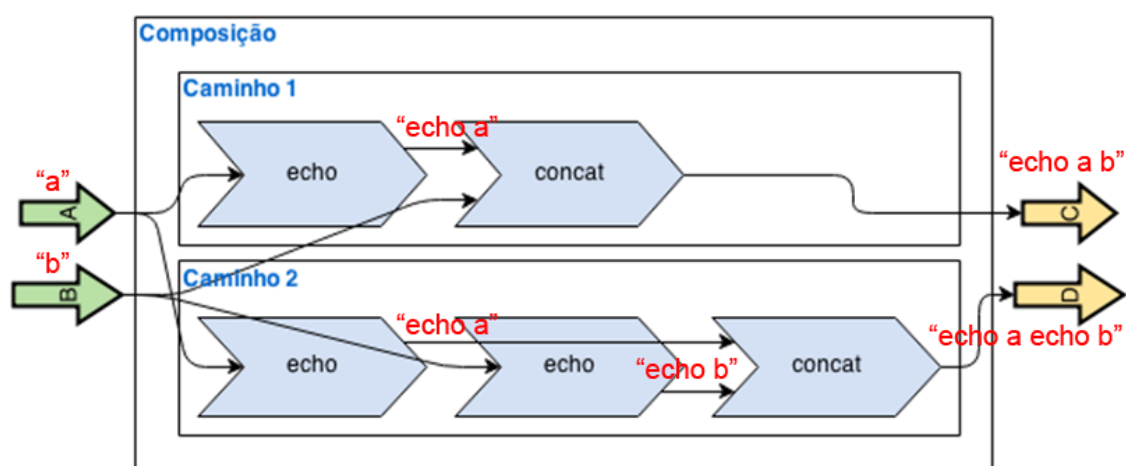


Figura 19 *Workflow* da composição exemplo

<sup>9</sup><http://www.soapui.org/>



O processo escrito no padrão WS-BPEL gerado pelo protótipo e visualizado no editor gráfico BPEL Designer pode ser visto na Figura 20. É possível observar, além das atividades de recebimento das entradas, inicialização de variáveis e resposta para o solicitante, os dois caminhos gerados – Caminho0 e Caminho1 – cada qual com as operações *assign* e *invoke*, conforme definido pela abordagem proposta.

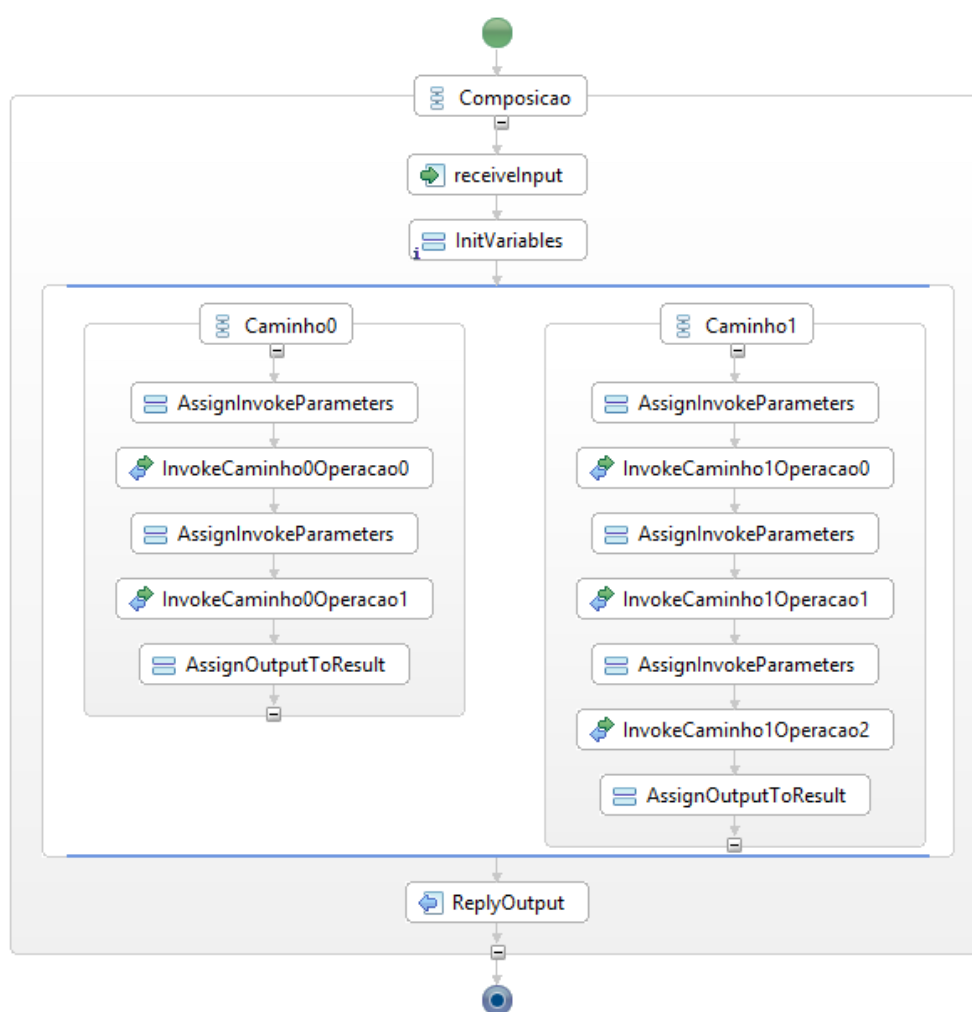


Figura 20 Visualização gráfica do documento WS-BPEL gerado pelo protótipo para a composição exemplo

Para testar a execução da composição foram definidos os parâmetros de entrada desta com os valores “a” e “b”, e após a execução da composição, espera-se como saída o valor “echo a b” para a saída “c” e “echo a echo b”

para a saída “d”. A mensagem de requisição, gerada pelo SoapUI, pode ser vista no Quadro.

```

01 <soapenv:Envelope
02   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
03   xmlns:com="http://executor.guto.me/Composicao00001">
04   <soapenv:Header/>
05   <soapenv:Body>
06     <com:Composicao00001Request>
07       <com:a>a</com:a>
08       <com:b>b</com:b>
09     </com:Composicao00001Request>
10   </soapenv:Body>
11 </soapenv:Envelope>

```

Quadro 14 Requisição da execução da composição exemplo

A resposta obtida pela execução da composição é apresentada no Quadro 15. Pode-se observar que o resultado obtido está de acordo com o resultado esperado.

```

01 <soapenv:Envelope
02   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
03   <soapenv:Body>
04     <Composicao00001Response
05       xmlns="http://executor.guto.me/Composicao00001">
06       <tns:c
07         xmlns:tns="http://executor.guto.me/Composicao00001">
08         echo a b
09       </tns:c>
10       <tns:d
11         xmlns:tns="http://executor.guto.me/Composicao00001">
12         echo a echo b
13       </tns:d>
14     </Composicao00001Response>
15   </soapenv:Body>
16 </soapenv:Envelope>

```

Quadro 15 Resposta da execução da composição exemplo

Para avaliar o desempenho do protótipo na geração dos artefatos de implantação foi utilizada a composição definida como exemplo. Foi realizada a execução do protótipo sobre esta composição, repetidas vezes, a fim de verificar o tempo de execução de cada uma das etapas envolvidas na geração dos artefatos de implantação.

Para realização do experimento foi utilizado o Apache HTTP *server benchmarking tool*<sup>10</sup>, ferramenta presente no Apache HTTP Server capaz de aferir o desempenho do servidor durante o atendimento a requisições. A definição do comando executado é apresentado no Quadro 16. O parâmetro `-n 200` define que serão executadas duzentas requisições e `-c 1` limita o número máximo de acessos concorrentes a um.

```
ab -n 200 -c 1 http://localhost/executor/executor/executor/deploy/1
```

Quadro 16 Utilização da ferramenta AB

Para cada etapa da geração dos artefatos foram capturados os tempos de execução e estas informações foram gravadas em arquivos de *log*. O experimento foi realizado em um PC com processador Intel Core i3 2,53GHz, 4GB RAM executando o sistema operacional Windows 8.1 64 bits e todos os recursos necessários disponíveis localmente, sem necessidade de acesso via internet.

A Figura 21 apresenta o tempo de execução, para cada uma das iterações de cada uma das etapas de geração do pacote de implantação: recuperação e conversão da composição, geração dos artefatos (WS-BPEL, WSDL e *deploy*), recuperação dos documentos WSDL dos serviços invocados e do pacote de implantação. Pode-se observar que a etapa com maior custo é a recuperação e conversão da composição, com tempo médio de 1,13s (tempos mínimo e máximo de 1,11s e 1,54s, respectivamente), justamente por ser a etapa responsável por recuperar a composição do banco de dados (*Composition Repository*) e efetuar a conversão desta na estrutura de classes intermediária, preparando a geração dos artefatos de implantação. As etapas

---

<sup>10</sup><http://httpd.apache.org/docs/2.2/programs/ab.html>

restantes apresentaram tempos de execução bastante semelhantes, com valores variando entre 1,61ms e 64,04ms. A apresentação dos resultados é feita em escala logarítmica para facilitar a visualização.

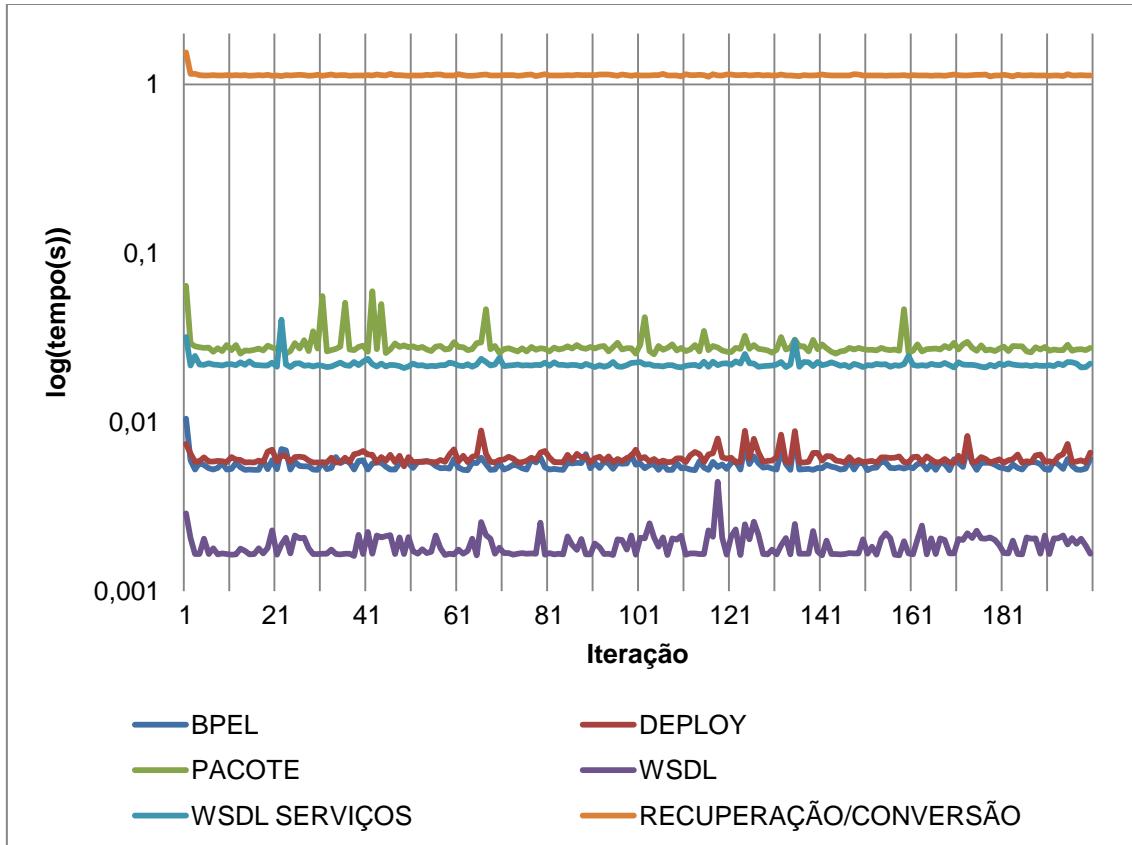


Figura 21 Tempo de execução para cada etapa da geração do pacote de implantação em cada iteração

## 5. Conclusão

A necessidade de compor *web services* surge no momento em que nenhum *web service* disponível é capaz de individualmente atender determinada solicitação. A tarefa de descobrir os serviços adequados, organizá-los de modo que os resultados esperados sejam gerados a partir das entradas disponíveis e executá-los é uma tarefa complexa e que demanda tempo por diversos fatores. Na etapa de composição pode-se citar a crescente quantidade de *web services* disponíveis e a frequência com que os *web services* sofrem alterações. Já na etapa de execução, tem-se a necessidade de obter fluência em uma linguagem que permita a definição de um processo de execução dos serviços participantes ou a programação manual da troca de mensagens entre estes serviços. Com a adoção da semântica sobre a descrição de *web services* é possível compô-los de modo automático. A partir da definição destas composições, utilizando linguagens padronizadas para descrição de um processo executável que represente a composição, é possível também automatizar o processo de execução das composições. Com isto, a tarefa de compor *web services* é reduzida a descrever as entradas disponíveis, operações e saídas desejadas semanticamente e obter como resultado a referência para um serviço composto que pode ser invocado diretamente.

Este trabalho abordou a etapa de execução das composições, prevista na Plataforma para Descoberta e Execução de Composições de Web Services Semânticos proposta por Hobold e Siqueira (2012). A fase de descoberta das composições, abordada na dissertação de mestrado de Hobold (2012), foi definida seguindo princípio de que a partir das entradas disponíveis são gerados caminhos em que os serviços participantes são invocados

sequencialmente, sendo que cada um desses caminhos é responsável por gerar uma das saídas desejadas para a composição. Com base nisto foi definida a abordagem para geração de um processo descrito no padrão WS-BPEL para possibilitar a execução de uma composição (objetivo específico 1). Cada caminho foi mapeado para uma atividade *sequence* que contém atividades *assign* e *invoke*, estas responsáveis por fazer a definição dos parâmetros de entrada de cada serviço e a invocação do serviço, respectivamente. Como estes caminhos são independente entre si, esse conjunto de atividades *sequence* foi incluído em uma atividade *flow* a fim de permitir a execução paralela dos caminhos. Importante salientar que a análise semântica dos serviços se restringe apenas à fase de descoberta e composição.

Após a definição da abordagem, foi implementado um protótipo com o intuito de validá-la. Além da geração do processo descrito no padrão WS-BPEL a partir da especificação da composição conforme a abordagem adotada, também foi necessário: gerar o documento WSDL para descrever como a composição pode ser acessada; obter cópias dos documentos WSDL dos serviços participantes; e gerar o descritor de implantação para a *engine* de execução de processos. Com todos estes arquivos gerados foi possível efetuar a implantação automática (objetivo específico 3) e possibilitar a execução de composições no Apache ODE, *engine* de orquestração de serviços utilizada neste trabalho (objetivo específico 2). Os testes realizados com o protótipo demonstraram que ele é capaz de converter corretamente uma composição descoberta em um processo executável descrito no padrão WS-BPEL e torná-

lo disponível de forma totalmente automática (objetivo específico 4), além de apresentar bom desempenho.

Neste contexto, é possível afirmar que o trabalho realizado alcançou o objetivo principal e os objetivos específicos propostos em sua totalidade. A partir do protótipo desenvolvido foi realizada de forma automática a conversão das composições no padrão WS-BPEL e a implantação destas em uma *engine* de execução de processos, com isso permitindo a execução das composições transparentemente como um único serviço, evitando que desenvolvedores tenham que programar as composições de maneira *ad-hoc*.

## 5.1. Trabalhos Futuros

Com realização do presente trabalho, mesmo com os objetivos propostos alcançados, observou-se diversas possibilidades para trabalhos futuros visando o aprimoramento da solução obtida. Estas propostas são elencadas a seguir:

- Identificar quais sequencias de operações, dentro de cada caminho ou mesmo entre caminhos, podem ser executadas paralelamente a fim de melhorar a desempenho da execução da composição;
- Identificar invocações de serviços ou sequencias que se repetem, dentro de cada caminho ou entre caminhos, para que sejam executadas uma única vez;
- Identificar métricas de qualidade de serviço relevantes (disponibilidade, custo, tempo de resposta, etc.) e monitorá-las, permitindo o fornecimento de *feedback* para a fase de

composição e conseqüentemente a descoberta de composições mais eficientes em relação às métricas definidas;

- Implementação de tratamento e manipulação de falhas e suporte ao controle de transações;
- Definição de um Schema XML para representação das composições, baseado no modelo apresentado em Hobold (2012), desta maneira tornando possível a conversão para o padrão WS-BPEL através da utilização de XSLT (eXtensible Stylesheet Language for Transformation), em vez de programaticamente como foi feito neste trabalho.



## 6. Referências Bibliográficas

ALVES, Alexandre (Ed.). **Web Services Business Process Execution Language Version 2.0 OASIS Standard**. 2007. Disponível em: <<http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>>. Acesso em: 19 nov. 2011.

AGARWAL, Vikas; CHAFLE, Girish; MITTAL, Sumit; SRIVASTAVA, Biplav. **Understanding Approaches for Web Service Composition and Execution**. IBM India Research Laboratory. 2008.

APACHE SOFTWARE FOUNDATION. **Apache ODE**: Welcome to Apache ODE. Disponível em: <<http://ode.apache.org/>>. Acesso em: 29 jun. 2012.

AYALA, Dietrich et al. **Professional Open Source Web Services**. Uk: Wrox Press, 2002.

BARRY, Douglas K.. **Service-oriented architecture (SOA) definition**. Disponível em: <[http://www.service-architecture.com/web-services/articles/service-oriented\\_architecture\\_soa\\_definition.html](http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html)>. Acesso em: 25 de jan. 2012.

BERNERS-LEE, Tim; HENDLER, James; LASSILA, Ora. **The Semantic Web**. Scientific American, Maio 2001.

BIH, Joseph. **Service oriented architecture (SOA) a new paradigm to implement dynamic e-business solutions**. Ubiquity, New York, 2006.

BRUIJN, Jos de et al. [Http://www.w3.org/Submission/WSMO/](http://www.w3.org/Submission/WSMO/). 2005. Disponível em: <<http://www.w3.org/Submission/WSMO/>>. Acesso em: 5 dez. 2013.

CAKE SOFTWARE FOUNDATION. **The CakePHP Cookbook**. 2013.  
Disponível em: < <http://book.cakephp.org/2.0/>>. Acesso em: 21 out. 2013.

CARLISLE, David; ION, Patrick; MINER, Robert (Ed.). **Mathematical Markup Language (MathML) Version 3.0**. 2010. Disponível em:  
<<http://www.w3.org/TR/MathML3/>>. Acesso em: 5 dez. 2013.

CHRISTENSEN, Erik et al. **Web Services Description Language (WSDL) 1.1**. Disponível em: <[http://www.w3.org/TR/wsdl#\\_porttypes](http://www.w3.org/TR/wsdl#_porttypes)>. Acesso em: 19 jun. 2012.

DAILEY, David (Ed.). **An SVG Primer for Today's Browsers**. 2010.  
Disponível em: <<http://www.w3.org/Graphics/SVG/IG/resources/svgprimer>>.  
Acesso em: 5 dez. 2013.

FARRELL, Joel; LAUSEN, Holger (Ed.). **Semantic Annotations for WSDL and XML Schema**. 2007. Disponível em:  
<<http://www.w3.org/TR/sawsdl/>>. Acesso em: 21 nov. 2013.

FLOWER, Martin et al. **Patterns of Enterprise Application Architecture**. Addison-Wesley, Boston, 2002. 560 p.

GARTNER. **IT Glossary – Service Oriented Architecture (SOA)**.  
Disponível em <<http://www.gartner.com/it-glossary/service-oriented-architecture-soa/>>. Acesso em 20 jun. 2012.

GUDGIN, Martin et al (Ed.). **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)**. 2007. Disponível em:  
<<http://www.w3.org/TR/soap12-part1/>>. Acesso em: 5 dez. 2013.

HASS, Hugo. **Designing the architecture for Web Services**. 2003.  
Disponível em <<http://www.w3.org/2003/Talks/0521-hh-wsa/>>. Acesso em 19 jun. 2012.

HOBOLD, Guilherme Coan. **Uma Abordagem para Descoberta Automática de Composições de Serviços Web Semânticos**. 2012. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2012.

HOBOLD, Guilherme Coan; SIQUEIRA, Frank. A Platform for Discovery and Execution of Semantic Web Services Compositions. In: INTERNATIONAL CONFERENCE ON SEMANTIC WEB AND WEB SERVICES, 2012., 2012, Las Vegas. **Proceedings of The 2012 International Conference on Semantic Web and Web Services**. Las Vegas: Csrea Press, 2012. p. 59 - 65. Disponível em: <<http://world-comp.org/p2012/SWW.html>>. Acesso em: 10 out. 2013.

HOLLEY, Kerrie; ARSANJANI, Dr. Ali. **100 SOA Questions: Asked and Answered**. New Jersey: Prentice Hall, 2011.

HURWITZ, Judith et al. **Service Oriented Architecture For Dummies**. Indianapolis: Wiley Publishing, Inc., 2007.

IBM. **Serviços da Web**: Introdução: Serviços da Web. Disponível em: <[http://www14.software.ibm.com/webapp/wsbroker/redirect?version=compass&product=was-nd-mp&topic=welc6tech\\_wbs\\_intro](http://www14.software.ibm.com/webapp/wsbroker/redirect?version=compass&product=was-nd-mp&topic=welc6tech_wbs_intro)>. Acesso em: 21 nov. 2013.

JURIC, Matjaz B.; MATHEW, Benny; SARANG, Poornachandra. **Business Process Execution Language for Web Services: An architect and developer's guide to orchestrating Web Services using BPEL4WS**. 2. ed. Olton: Packt Publishing, 2006. 372 p.

MARTIN, David et al. **OWL-S: Semantic Markup for Web Services**. 2004. Disponível em: <<http://www.w3.org/Submission/OWL-S/>>. Acesso em: 5 dez. 2013.

MARTIN, David et al. Bringing Semantics to Web Services with OWL-S. **World Wide Web**, Hingham, Ma, Usa, p. 243-277. 3 jul. 2007.

NEWCOMER, Eric. **Understanding Web Services: XML, WSDL, SOAP, and UDDI**. Boston, Ma: Addison-wesley, 2002. 332 p.

PEMBERTON, Steven et al (Ed.). **XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition)**. 2002. Disponível em: <<http://www.w3.org/TR/xhtml1/>>. Acesso em: 5 dez. 2013.

RAO, Jinghai; SU, Xiaomeng. **Survey of Automated Web Service Composition Methods**. International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004.

SILVA JUNIOR, Edson Nascimento. **Web Service – XML: Aula 08**, 2010. Notas de aula.

THE PHP GROUP (Org.). **Manual do PHP: O que é PHP?**. Disponível em: <[http://www.php.net/manual/pt\\_BR/intro-what-is.php](http://www.php.net/manual/pt_BR/intro-what-is.php)>. Acesso em: 27 out. 2013.

TITTEL, Ed. **Schaum's outline of theory and problems of XML: Schaum's outline series**. Blacklick, Oh, Usa: Mcgraw-hill, 2002. 214 p.

TOWNSEND, Erick. **The 25 Year History of Service Oriented Architecture**. 2008. Disponível em <[http://www.eriktownsend.com/white-papers-technology/doc\\_download/4-1-the-25-year-history-of-service-oriented-architecture.html](http://www.eriktownsend.com/white-papers-technology/doc_download/4-1-the-25-year-history-of-service-oriented-architecture.html)>. Acesso em 20 jun. 2012.

VIEGAS, Charles. **Anatomia WSDL**. 2008. Disponível em: <<http://www.aqueleblogdesoa.com.br/2008/08/anatomia-do-wsdl/>>. Acesso em: 17 out. 2013.

WIKIPEDIA (Org.). **Comparison of BPEL engines**. Disponível em:  
<[http://en.wikipedia.org/wiki/Comparison\\_of\\_BPEL\\_engines](http://en.wikipedia.org/wiki/Comparison_of_BPEL_engines)>. Acesso em: 10  
out. 2013.

## **APÊNDICE I – Artigo**

# Ambiente para Execução Automática de Composições de Web Services

Augusto C. Ferreira<sup>1</sup>

<sup>1</sup>Departamento de Informática e Estatística  
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

augustof@inf.ufsc.br

**Abstract.** *This work aims to address the tasks related to permit, automatically, that a composition of Web services discovery previously run as a single service, in a transparent manner to the requester, following the proposal by Hobold and Siqueira (2012) architecture. An approach is presented for representing a composition of Web Services following the standard WS-BPEL generation of deployment artifacts required for deployment and execution of the composition in a running engine, both automatically. A prototype was developed in order to verify the operation of the proposed approach.*

**Resumo.** *Este trabalho visa abordar as tarefas relacionadas a permitir, de forma automática, que uma composição de Web Services previamente descoberta seja executada como um serviço único, de forma transparente ao solicitante, seguindo a arquitetura proposta por Hobold e Siquera (2012). É apresentada uma abordagem para representação de uma composição de Web Services seguindo o padrão WS-BPEL, geração dos artefatos de implantação necessários para implantação e execução da composição em um engine de execução, ambos de forma automática. Um protótipo foi desenvolvido a fim de verificar o funcionamento da abordagem proposta.*

## 1. Introdução

Na Arquitetura Orientada a Serviços (SOA – *Service Oriented Architecture*), os serviços, funções bem definidas, autônomas e independentes entre si, são distribuídos como componentes de *software* através de uma rede de computadores. Com a adoção de *Web Services* é possível implementar a Arquitetura Orientada a Serviços através da criação de uma camada adicional entre os serviços, facilitando a integração entre estes.

*Web Services* podem ser compostos a fim de criar novos serviços que atendam a requisições que não poderiam ser atendidas por um *Web Service* individualmente, porém a tarefa de compô-los manualmente é complexa. Com a descrição semântica dos serviços possibilita-se a composição automática de *Web Services*.

Hobold e Siqueira (2012) descrevem uma plataforma para descoberta e execução de composições de *web services* semânticos. Nesta plataforma as composições são descobertas a partir da descrição semântica dos serviços e após convertidas para o padrão WS-BPEL (*Web Services Business Process Execution Language*), com isso permitindo a execução da composição como um serviço único. A fase de descoberta das composições descrita na plataforma foi abordada por Hobold (2012).

## 1.1. Objetivos

O objetivo principal deste trabalho é abordar a fase de execução da plataforma proposta por Hobold e Siqueira (2012), permitindo a execução das composições descobertas como um serviço único de maneira automática. Os objetivos específicos abrangem a conversão das composições para o padrão WS-BPEL, seleção de um *engine* de execução de processos e implantação automática das composições descritas no padrão WS-BPEL no *engine* selecionado.

## 1.2. Justificativa

A justificativa para realização deste trabalho se baseia na complexidade para realização das tarefas relacionadas a descoberta e execução de composições de *Web Services* de modo manual.

## 1.3. Delimitação do Escopo

Não estão incluídos no escopo deste trabalho o tratamento e recuperação de falhas, controle transacional, coleta de métricas de qualidade de serviço, nem estruturas de controle de paralelismo.

## 1.4. Metodologia

Primeiramente será realizada a revisão bibliográfica a fim de se obter a fundamentação teórica sobre a descoberta e execução de composições de *web services* semânticos. Após, a partir dos resultados obtidos por Hobold (2012), definir a abordagem a fim de permitir de maneira automática a execução das composições descobertas e com isto desenvolver um protótipo.

## 2. Serviços Web

### 2.1. Arquitetura Orientada a Serviços

A Arquitetura Orientada a Serviços apresenta o conceito onde os serviços fornecidos por softwares são disponibilizados através de uma rede de computadores, na internet ou intranet, de forma a reduzir o acoplamento e aumentar o reuso. Estes serviços se comunicam através de simples troca de mensagens ou pela interação de dois ou mais serviços para execução de determinada atividade.

### 2.2. Web Services

Os *Web Services* fornecem uma forma alternativa à interação tradicional da Web, onde um ser humano por meio de um navegador de internet acessa documentos através de *links* em uma página escrita em HTML (*Hyper Text Markup Language*), disponibilizadas por um servidor remoto. A interação através de Web Services permite que aplicações façam o intercâmbio de informações estruturadas, através do uso da linguagem XML

#### 2.2.1. XML

Segundo HURWITZ et al. (2007), XML (*eXtensible Markup Language*) é uma meta-linguagem de marcação padrão do W3C (*World Wide Web Consortium*) derivada da



SGML (*Standard Generalized Markup Language*), utilizada para descrever estruturas de dados.

### **2.2.2. SOAP**

SOAP (*Simple Object Access Protocol*) é um protocolo baseado em XML que descreve um formato de mensagens para comunicação entre aplicações. SOAP fornece o mecanismo de comunicação para interação com Web Services (NEWCOMER, 2002).

### **2.2.3. WSDL**

Conforme Newcomer (2002), *Web Service Description Language* (Linguagem de Descrição de Web Services) é uma linguagem padrão, baseada em XML, para descrever e publicar os formatos e protocolos de um *Web Service* de uma maneira padrão.

### **2.2.4. UDDI**

UDDI (*Universal Description, Discovery, and Integration*) segundo AYALA et al. (2002), é um registro baseado em XML para entidades globais se registrarem na Internet, que tem como objetivo principal permitir que empresas encontrem umas as outras e tornem seus sistemas interoperáveis para comércio eletrônico.

### **2.2.5. Orquestração e Coreografia de Web Services**

Conforme Juric, Mathew e Sarang (2006), dependendo dos requisitos, composições de *Web Services* podem tratar de processos privados ou públicos, para isso são empregados dois termos: Orquestração e Coreografia, respectivamente.

A Orquestração de *Web Services* é caracterizada pela existência de um coordenador central, que é responsável por controlar a execução de um processo de negócio pela invocação de operações dos *Web Services* participantes deste processo.

Coreografia por outro lado, não possui um coordenador central, cada Web Service participante envolvido na coreografia sabe exatamente quando atuar e com quem interagir.

### **2.2.4. WS-BPEL**

A especificação WS-BPEL é uma extensão do modelo de interação de *Web Services* para suporte a transações de negócios. A linguagem definida por WS-BPEL fornece meios para descrever formalmente processos de negócio e interação de protocolos de negócios.

### **2.2.4. Web Services Semânticos**

Os Web Services Semânticos (SWS - *Semantic Web Services*) são serviços, bem como suas mensagens de entrada e saída, descritos semanticamente através da adoção de ontologias, com isso possibilitando que estes sejam descobertos, invocados, compostos e executados de forma automática.

### 2.3. Composição e Execução de Serviços

Quando um único *Web Service* é incapaz de atender as necessidades de um consumidor de serviços, é possível combinar um conjunto de *Web Services*, desta forma criando um novo serviço a fim de satisfazer os requisitos de uma solicitação de um usuário. A composição é o processo de seleção, combinação e execução de *Web Services* para atingir o objetivo de um usuário (MARTIN et al., 2007, p. 36).

Combinações de *Web Services* podem ser definidas manualmente, onde o usuário define o fluxo de informações através dos diversos *Web Services* selecionados para, em conjunto, executar determinada operação ou ainda através de métodos automatizados, que são baseados, principalmente, em técnicas de *workflow* e Inteligência Artificial (RAO; SU, 2004).

### 3. Plataforma para Descoberta e Execução de Composições de Web Services Semânticos

Hobold e Siqueira (2012) propõem uma plataforma que permite a descoberta automática e a execução de composições de *Web Services* semânticos. Nesta plataforma, anotações SAWSDL (*Semantic Annotations for WSDL*) são extraídas de documentos WSDL e empregadas na construção de grafos de composição, ou seja, as composições são descobertas com base na semântica dos serviços. Após a descoberta das composições estas são convertidas para o padrão WS-BPEL e podem ser invocadas como um serviço único.

A fase de descoberta das composições foi desenvolvida por Hobold (2012) e a fase de execução das composições será abordada no presente trabalho.

A plataforma proposta tem como característica ser independente de linguagem de programação e plataforma de execução e é composta por nove componentes, apresentados na Figura 1.

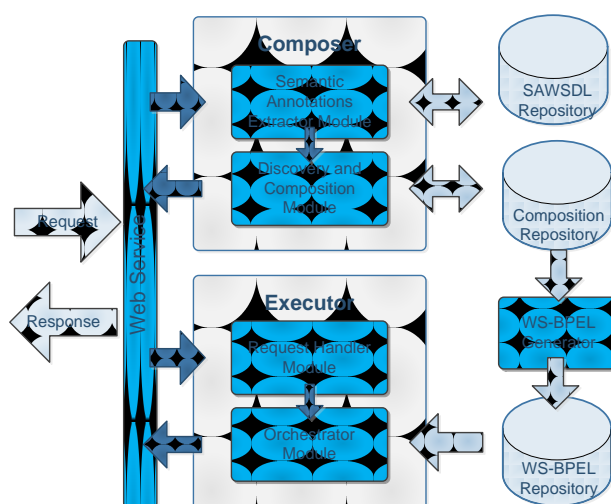


Figura 1. Arquitetura da Plataforma de Composição e Execução de Web Services (HOBOLD; SIQUEIRA, 2012)

### 3.1. Composição de Web Services

O trabalho de Hobold (2012), que tem como objetivo analisar e desenvolver uma plataforma para a composição automática de *Web Services* Semânticos, segue a arquitetura proposta por Hobold e Siqueira (2012), cobrindo a descoberta das composições, porém não abrangendo a execução das composições encontradas.

### 3.2. Execução das Composições de Web Services

Para a execução das composições, de acordo com a arquitetura proposta por Hobold e Siqueira (2012), os componentes envolvidos são o *WS-BPEL Generator*, o *WS-BPEL Repository*, o *Request Handler Component* e o *Orchestrator Module*.

Durante a fase de descoberta das composições pelo protótipo desenvolvido por Hobold (2012), para cada uma das saídas desejadas em uma composição é gerado um caminho. Este caminho é composto por uma sequência de operações associadas aos seus parâmetros de entrada e saída.

A abordagem para transformação de uma composição de *Web Services* em um processo executável descrito no padrão WS-BPEL é realizada com base nos caminhos obtidos na descoberta da composição, que são mapeados diretamente para uma sequência de invocações de serviços. Os caminhos que formam uma composição são independentes entre si, permitindo a execução paralela destes.

### 3.3. Arquitetura

As operações que devem ser realizadas na fase de execução, conforme a arquitetura proposta por Hobold e Siqueira (2012), são a implantação e a execução de uma composição de serviços.

Em razão da necessidade da geração de outros arquivos, chamados artefatos de implantação, além do WS-BPEL a arquitetura proposta inicialmente por Hobold e Siqueira (2012) para a fase de execução das composições foi estendida como pode ser visto na Figura 2.

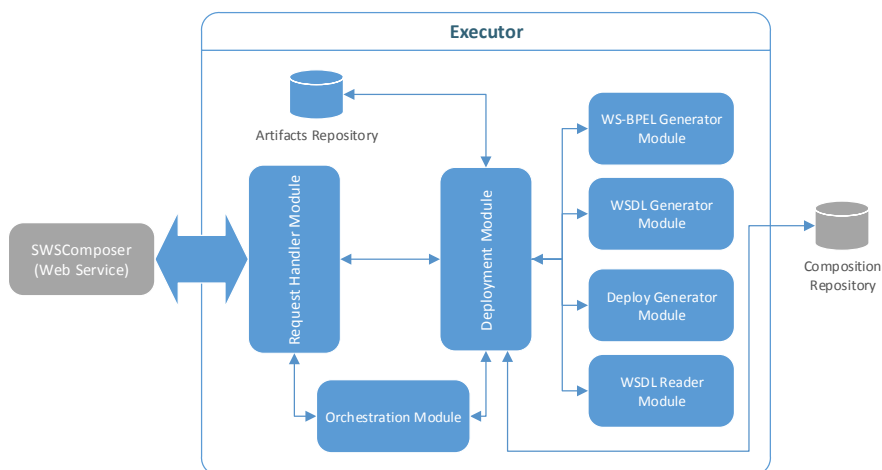


Figura 2. Arquitetura proposta para execução de composições de Web Services

O *Request Handler Module* recebe requisições para execução e implantação de composições, realizando a comunicação entre as camadas de descoberta e execução da arquitetura.

No *Deployment Module* as composições são obtidas a partir do *Composition Repository* e encaminhadas aos geradores dos artefatos de implantação. A seguir, estes artefatos são empacotados e encaminhados ao *Orchestration Module* para implantação da composição.

O *Artifacts Repository* é responsável pelo armazenamento dos artefatos de implantação gerados para cada composição e das cópias dos documentos WSDL de cada um dos serviços participantes de uma composição.

O *WS-BPEL Generator*, *WSDL Generator* e *Deploy Generator* são os módulos geradores dos artefatos de implantação, os arquivos WS-BPEL, WSDL e descritor de implantação respectivamente.

O *WSDL Reader Module* é responsável por adquirir uma cópia do arquivo WSDL de cada serviço pertencente à composição.

### 3.3.1. Implantação de uma Composição

Quando o *Request Handler Module* recebe uma solicitação para implantação de uma composição, o *Deployment Module* é acionado e primeiramente este recupera a composição informada pelo solicitante a partir do *Composition Repository*. A partir deste momento, cópias dos arquivos WSDL são obtidas pelo *WSDL Reader Module*. Em seguida, os geradores dos artefatos de implantação são invocados e cada um deles, após a geração do respectivo artefato, os armazena no *Artifacts Repository*. Com os artefatos gerados e cópias dos arquivos WSDL dos serviços participantes obtidas, o *Deployment Module* deve gerar o pacote de implantação e encaminhá-lo ao *Orchestration Module*, finalizando com isto a operação.

### 3.3.2. Execução de uma Composição

Para a execução de uma composição, assim que o *Request Handler Module* recebe uma requisição para tal, este primeiramente deve obter a composição a partir do *Composition Repository*. Estando a composição implantada, o *Request Handler Module* deve solicitar a execução ao *Orchestration Module*. Este deverá efetuar a execução da composição, invocando os *Web Services* participantes, de acordo com o processo definido no respectivo documento WS-BPEL. Por fim, o resultado da execução é retornado ao solicitante da execução.

## 3.4. Composition Repository

O modelo entidade-relacionamento do banco de dados para armazenamento das composições definido por Hobold (2012), por ter sido planejado para atender a fase de descoberta das composições cobrindo o aspecto semântico, necessitou alterações a fim de permitir a execução destas. Para abranger os aspectos referentes à execução das composições foram incluídas colunas para definição dos tipos de dados das entradas e saídas e também definidas relações para indicação da origem dos parâmetros de entrada das operações e origem da saída dos caminhos. Este modelo é apresentado na Figura 3.

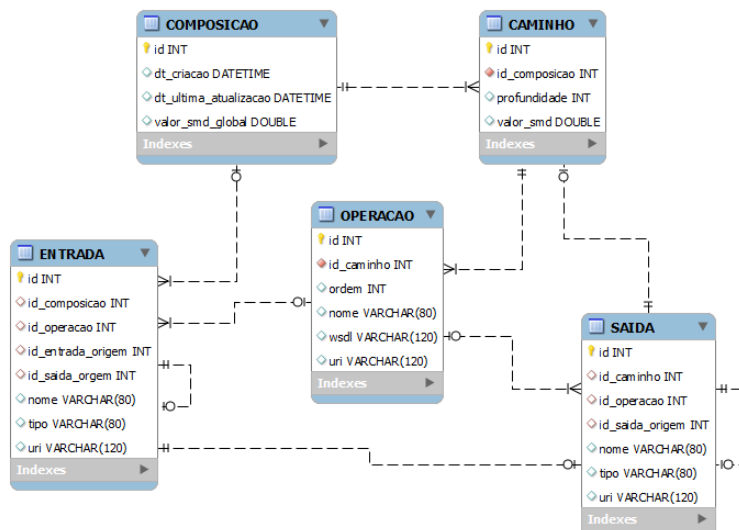


Figura 3. Modelo entidade-relacionamento do banco de dados de composições adequado

#### 4. Implementação e Avaliação do Protótipo

O protótipo foi implementado tendo os componentes previstos na arquitetura escritos em linguagem PHP (*PHP: Hypertext Preprocessor*), baseados no *framework* CakePHP<sup>1</sup> e executando no Apache HTTP Server, exceto pelo *Orchestration Module*, que será representado pelo Apache ODE executando sobre o Apache Tomcat. Os servidores Apache HTTP Server e Apache Tomcat são desenvolvidos e distribuídos pela licença Apache<sup>2</sup> pela Apache Software Foundation.

As composições serão carregadas diretamente do *Composition Repository*, implementado como um banco de dados no MySQL, através de consultas SQL (*Structured Query Language*). O MySQL é um DBMS (*Data Base Management System*) *open source* desenvolvido e distribuído pela Oracle Corporation.

Os componentes de *software* do ambiente de execução, descritos anteriormente, são apresentados através de diagrama de implantação na Figura 4.

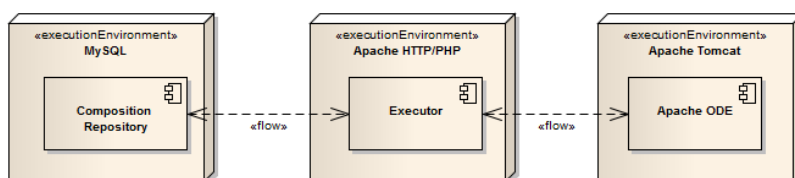


Figura 4. Diagrama de Implantação do protótipo

<sup>1</sup> <http://www.cakephp.org>

<sup>2</sup> <http://www.apache.org/licenses/>

#### 4.1. Linguagem de Programação e Framework de Desenvolvimento

Segundo The PHP Group (2013), PHP é uma linguagem de script *open-source* de uso geral, amplamente utilizada no desenvolvimento de aplicações Web.

Cake Software Foundation (2013) define o CakePHP como um *framework* de desenvolvimento rápido de aplicações *web*. O CakePHP fornece ferramentas para geração de código em tempo de desenvolvimento e execução além de bibliotecas embutidas para acesso a banco de dados, *caching*, validação, autenticação e segurança.

A adoção da linguagem de programação PHP e do *framework* CakePHP para implementação do protótipo se deve à experiência do autor.

#### 4.2. Componentes

Os módulos *Deployment Module*, *WS-BPEL Generator Module*, *WSDL Generator Module*, *Deploy Generator Module*, *WSDL Reader Module* e o *Artifacts Repository* foram implementados como *Components* do CakePHP, nomeados de acordo com a convenção de nomes<sup>1</sup> adotada pelo CakePHP. As requisições HTTP (*Hypertext Transfer Protocol*), necessárias para a obtenção das cópias dos arquivos WSDL dos serviços participantes pelo *WSDLReaderComponent*, foram encapsuladas na classe *HttpClientComponent*, também implementada em forma de *Component* e utilizando a biblioteca *cURL*<sup>2</sup> do PHP.

#### 4.3. Modelos

Conforme os padrões de desenvolvimento definidos pelo CakePHP foram definidas classes modelo para cada uma das tabelas presentes no banco de dados de composições (*Composition Repository*). Desta maneira, cada uma destas classes fornece métodos para recuperação e manipulação dos dados presentes nas tabelas referenciadas, seguindo o padrão de projeto *Active Record* (Flower et al.; 2002).

#### 4.4. Apache ODE

Para o *Orchestrator Module* neste protótipo foi escolhido o Apache ODE (*Orchestration Director Engine*), *engine* de orquestração capaz de executar processos de negócios descritos no padrão WS-BPEL.

#### 4.5. Artefatos de Implantação

Para implantação de uma composição no Apache ODE são necessários alguns artefatos que compõem o pacote de implantação. São eles: a composição descrita no padrão WS-BPEL, o arquivo WSDL que descreve como a composição poderá ser invocada, o arquivo de *deploy* do Apache ODE e cópias dos arquivos WSDL de cada um dos serviços participantes da composição.

#### 4.6. Avaliação do Protótipo

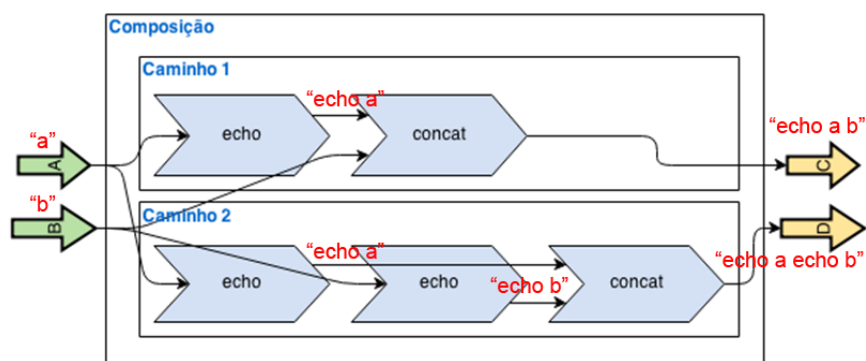
Para avaliar o funcionamento do protótipo foi definida uma composição fictícia contendo diversas situações que podem ocorrer em uma composição a fim de verificar

---

<sup>1</sup> <http://book.cakephp.org/2.0/en/getting-started/cakephp-conventions.html>

<sup>2</sup> <http://php.net/manual/en/book.curl.php>

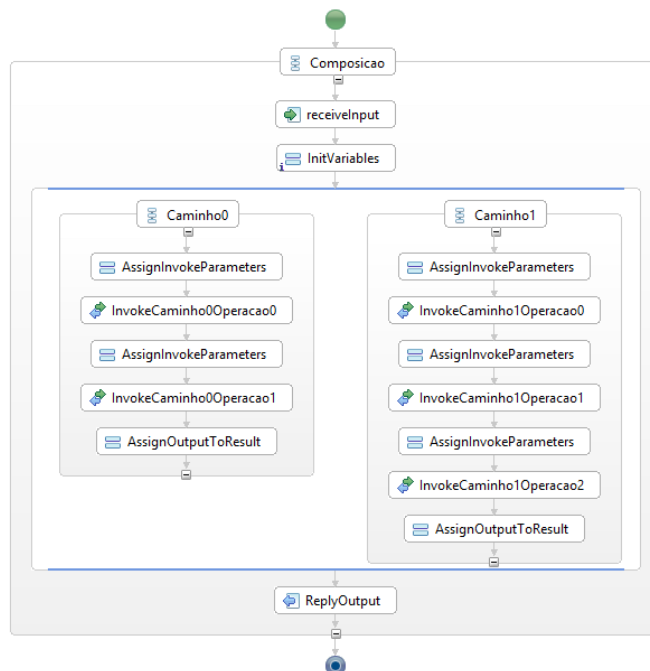
se o protótipo é capaz de gerar corretamente os artefatos de implantação de acordo com as definições da composição. O *workflow* desta composição, incluindo os valores para os parâmetros de entrada e resultados esperados, é apresentado na Figura 5.



**Figura 5 Workflow da composição exemplo**

O processo escrito no padrão WS-BPEL gerado pelo protótipo e visualizado no editor gráfico BPEL Designer<sup>1</sup> pode ser visto na Figura 6. É possível observar, além das atividades de recebimento das entradas, inicialização de variáveis e resposta para o solicitante, os dois caminhos gerados – Caminho0 e Caminho1 – cada qual com as operações *assign* e *invoke*, conforme definido pela abordagem proposta.

Para testar a execução da composição foram definidos os parâmetros de entrada desta com os valores “a” e “b”, e após a execução da composição, espera-se como saída o valor “echo a b” para a saída “c” e “echo a echo b” para a saída “d”. O protótipo apresentou os resultados em conformidade com os valores esperados.



**Figura 6. Visualização gráfica do documento WS-BPEL gerado pelo protótipo para a composição exemplo**

<sup>1</sup> <http://www.eclipse.org/bpel/>

## 5. Conclusão

A necessidade de compor *web services* surge no momento em que nenhum *web service* disponível é capaz de individualmente atender determinada solicitação. A tarefa de descobrir os serviços adequados, organizá-los de modo que os resultados esperados sejam gerados a partir das entradas disponíveis e executá-los é uma tarefa complexa e que demanda tempo por diversos fatores.

Com a adoção da semântica sobre a descrição de *web services* é possível compô-los de modo automático. A partir da definição destas composições, utilizando linguagens padronizadas para descrição de um processo executável que represente a composição, é possível também automatizar o processo de execução das composições.

Este trabalho abordou a etapa de execução das composições, prevista na plataforma proposta por Hobold e Siqueira (2012). A fase de descoberta das composições, abordada na dissertação de mestrado de Hobold (2012), foi definida com o princípio que para cada uma das saídas desejadas são gerados caminhos em que os serviços participantes são invocados sequencialmente. Com base nisso foi definida a abordagem para geração de um processo descrito no padrão WS-BPEL para possibilitar a execução de uma composição.

Após a definição da abordagem, foi implementado um protótipo com o intuito de validá-la. Os testes realizados com o protótipo demonstraram que ele é capaz de converter corretamente uma composição descoberta em um processo executável descrito no padrão WS-BPEL e torná-lo disponível de forma totalmente automática.

Neste contexto, é possível afirmar que o trabalho realizado alcançou os objetivos propostos em sua totalidade.

## Referências

- AYALA, Dietrich et al. Professional Open Source Web Services. Uk: Wrox Press, 2002.
- CAKE SOFTWARE FOUNDATION. The CakePHP Cookbook. 2013. Disponível em: <<http://book.cakephp.org/2.0/>>. Acesso em: 21 out. 2013.
- FLOWER, Martin et al. Patterns of Enterprise Application Architecture. Addison-Wesley, Boston, 2002. 560 p.
- HOBOLD, Guilherme Coan. Uma Abordagem para Descoberta Automática de Composições de Serviços Web Semânticos. 2012. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2012.
- HOBOLD, Guilherme Coan; SIQUEIRA, Frank. A Platform for Discovery and Execution of Semantic Web Services Compositions. In: INTERNATIONAL CONFERENCE ON SEMANTIC WEB AND WEB SERVICES, 2012., 2012, Las Vegas. Proceedings of The 2012 International Conference on Semantic Web and Web Services. Las Vegas: Csrea Press, 2012. p. 59 - 65. Disponível em: <<http://world-comp.org/p2012/SWW.html>>. Acesso em: 10 out. 2013.



- HURWITZ, Judith et al. Service Oriented Architecture For Dummies. Indianapolis: Wiley Publishing, Inc., 2007.
- MARTIN, David et al. Bringing Semantics to Web Services with OWL-S. World Wide Web, Hingham, Ma, Usa, p. 243-277. 3 jul. 2007.
- NEWCOMER, Eric. Understanding Web Services: XML, WSDL, SOAP, and UDDI. Boston, Ma: Addison-wesley, 2002. 332 p.
- RAO, Jinghai; SU, Xiaomeng. Survey of Automated Web Service Composition Methods. International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004.

## **APÊNDICE II – Código-fonte**

**Executor\Config\bootstrap.php**

```
<?php

    App::uses('Folder', 'Utility');

    /**
     * Define o nome do diretório para os arquivos WSDL e o cria caso
    não exista
     */
    $wsdl_cache_path = TMP . 'cache' . DS . 'wsdl' . DS;
    $folder = new Folder($wsdl_cache_path, true);
    unset($folder);
    /**
     * Configuração de cache para os arquivos WSDL
     */
    Cache::config('wsdl', array(
        'engine' => 'File',
        'duration' => '+1 day',
        'path' => $wsdl_cache_path,
        'prefix' => '',
    ));

    /**
     * Define o nome do diretório para os arquivos XSD e o cria caso
    não exista
     */
    $xsd_cache_path = TMP . 'cache' . DS . 'xsd' . DS;
    $folder = new Folder($xsd_cache_path, true);
    unset($folder);
    /**
     * Configuração de cache para os arquivos XSD
     */
    Cache::config('xsd', array(
        'engine' => 'File',
        'duration' => '+1 day',
        'path' => $xsd_cache_path,
        'prefix' => '',
    ));

?>
```

Executor\Controller\Component\AbstractXmlGeneratorComponent.php

<?php

```

/**
 * WsdlGeneratorComponent
 *
 * Classe abstrata que serve como base para os geradores do
Processo e do
 * WSDL
 *
 * @author Augusto Cesar Ferreira
 *
 */
abstract class AbstractXmlGeneratorComponent extends Component {

    /**
     * @var DOMDocument
     */
    private $DOMDocument;

    /**
     * Processo
     * @var Process
     */
    private $Process;

    /**
     * Comprimento da linha de comentário
     * @var int
     */
    private $commentLineLenght = 50;

    /**
     * Inicialização do componente
     * @param Controller $controller
     */
    public function initialize(Controller $controller) {
        parent::initialize($controller);
        $this->DOMDocument = null;
        $this->Process = null;
    }

    /**
     * Classes filhas devem implementar o método generate
     * @param Process $Process
     */
    abstract public function generate(Process $Process);

    /**
     * Define o atributo DOMDocument
     * @param DOMDocument $DOMDocument
     */
    public function setDOMDocument(DOMDocument $DOMDocument) {
        $this->DOMDocument = $DOMDocument;
    }

    /**
     * Le o atributo DOMDocument
     * @return DOMDocument
     */

```

```

public function getDOMDocument() {
    return $this->DOMDocument;
}

/**
 * Define o atributo Process
 * @param Process $Process
 */
public function setProcess(Process $Process) {
    $this->Process = $Process;
}

/**
 * Le o atributo Process
 * @return Process
 */
public function getProcess() {
    return $this->Process;
}

/**
 * Criar um DOMElement
 * @param string $element_name Nome do elemento
 * @param array $options Array associativo indicado os
atributos do
 * elemento 'nome' => 'valor'
 */
protected function createElement($element_name, array $options
= array()) {
    $element = $this->DOMDocument-
>createElement($element_name);
    foreach ($options as $attr => $value) {
        $element->setAttribute($attr, $value);
    }
    return $element;
}

}

?>

```

Executor\Controllor\Component\ArtifactsRepositoryComponent.php

<?php

```

/**
 * Description of ArtifactsRepositoryComponent
 *
 * @author Augusto
 */
class ArtifactsRepositoryComponent extends Component {

    public function generatePackage($composicao_id, $bpel, $wsdl,
$deploy, array $wsdl_partners) {
        $packageFiles = array();
        $packageFiles[] = $this->writeBpel($composicao_id, $bpel);
        $packageFiles[] = $this->writeWsd($composicao_id, $wsdl);
        $packageFiles[] = $this->writeDeploy($composicao_id,
$deploy);
        foreach ($wsdl_partners as $filename => $wsdl) {
            $packageFiles[] = $this->
writeWsdPartner($composicao_id, $filename, $wsdl);
        }
        //Nome do arquivo zip
        $packageFilename = $this->getFolder($composicao_id)->path
. DS . sprintf('Composicao%05d.zip', $composicao_id);
        $file = new File($packageFilename);
        $file->delete();
        $file->close();
        $zip = new ZipArchive();
        $zip->open($packageFilename, ZIPARCHIVE::CREATE);
        foreach ($packageFiles as $filename) {
            $pathinfo = pathinfo($filename);
            $zip->addFile($filename, $pathinfo['basename']);
        }
        $zip->close();
        return $packageFilename;
    }

    /**
     * @param type $composition_id
     * @return Folder
     */
    private function getFolder($composition_id) {
        $path = ROOT . DS . 'files' . DS . 'ArtifactsRepository' .
DS . sprintf('%05d', $composition_id);
        return new Folder($path, true);
    }

    /**
     *
     * @param type $filename
     * @param type $content
     * @return type
     */
    private function writeFile($composition_id, $filename,
$content) {
        $folder = $this->getFolder($composition_id);
        $path = $folder->path . DS . $filename;
        $file = new File($path, true, '0777');
        $file->delete();
        $file->write($content, 'w', true);
    }
}

```

```

        return $file->path;
    }

    /**
     * @param int $composicao_id ID da Composição
     * @param string $xml Conteúdo do documento WSBPEL
     * @return boolean
     */
    private function writeBpel($composicao_id, $xml) {
        $filename = sprintf('Composicao%05d.bpel',
$composicao_id);
        return $this->writeFile($composicao_id, $filename, $xml);
    }

    /**
     * @param int $composicao_id ID da Composição
     * @param string $xml Conteúdo do documento WSDL
     * @return boolean
     */
    private function writeWsd($composicao_id, $xml) {
        $filename = sprintf('Composicao%05dArtifacts.wsdl',
$composicao_id);
        return $this->writeFile($composicao_id, $filename, $xml);
    }

    /**
     * @param int $composicao_id ID da Composição
     * @param string $xml Conteúdo do documento WSDL
     * @return boolean
     */
    private function writeDeploy($composicao_id, $xml) {
        $filename = 'deploy.xml';
        return $this->writeFile($composicao_id, $filename, $xml);
    }

    private function writeWsdPartner($composicao_id, $filename,
$xml) {
        return $this->writeFile($composicao_id, $filename, $xml);
    }
}

?>

```

Executor\Controller\Component\DeployGeneratorComponent.php

<?php

```

/**
 * @author Augusto
 * @version 1.0
 * @created 11-nov-2013 18:50:07
 */
class DeployGeneratorComponent extends
AbstractXmlGeneratorComponent {

    /**
     * Lista de nomes dos components utilizados
     * @var array
     */
    public $components = array(
        'Executor.WsdlReader',
    );

    public function generate(Process $Process) {
        //Cria o DOMDocument
        $this->setDOMDocument(new DOMDocument());
        //Define o Process
        $this->setProcess($Process);
        //Deploy
        $DOMEltDeploy = $this->createDeploy();
        //Active
        $this->createActive($DOMEltDeploy);
        //Retired
        $this->createRetired($DOMEltDeploy);
        //Process
        $this->createProcess($DOMEltDeploy);
        //Anexa o elemento deploy ao documento
        $this->getDOMDocument()->appendChild($DOMEltDeploy);
        //Retorna o xml
        return $this->getDOMDocument()->saveXML();
    }

    /**
     * Cria o elemento process e o anexo ao elemento deploy,
    também chama
     * os métodos que geram os elementos contidos em process
     * @param DOMElement $DOMEltDeploy
     */
    private function createProcess(DOMElement $DOMEltDeploy) {
        $options = array(
            'name' => $this->getProcess()->getQNameDeploy(),
        );
        $DOMEltProcess = $this->createElement('process',
$options);
        $DOMEltDeploy->appendChild($DOMEltProcess);
        //ProcessEvents
        $this->createProcessEvents($DOMEltProcess);
        //Provide
        $this->createProvide($DOMEltProcess);
        //Invoke
        $this->createInvokes($DOMEltProcess);
    }

    /**

```



```

*
* @param DOMElement $DOMEltProcess
*/
private function createInvokes (DOMElement $DOMEltProcess) {
    foreach ($this->getProcess()-
>getAllPartnerLinkByType(RoleType::PARTNER) as $PartnerLink) {
        //Cria o elemento Invoke para o partner link
        $options = array(
            'partnerLink' => $PartnerLink->getName(),
        );
        $DOMEltInvoke = $this->createElement('invoke',
$options);
        $DOMEltProcess->appendChild($DOMEltInvoke);
        //Namespace
        $XmlNamespace = $PartnerLink->getPartnerLinkType()-
>getRole()->getPortType()->getXmlNamespace();
        //Service
        $Service_name = $this->WsdReader-
>getService($PartnerLink->getUrlWsd());
        //Port
        $Port_name = $this->WsdReader->getPort($PartnerLink-
>getUrlWsd());
        $options = array(
            'name' => sprintf('%s:%s', $XmlNamespace-
>getPrefixDeploy(), $Service_name),
            'port' => $Port_name,
        );
        $DOMEltService = $this->createElement('service',
$options);
        $DOMEltInvoke->appendChild($DOMEltService);
    }
}

/**
* Cria o elemento provide e o anexo ao elemento process
* @param DOMElement $DOMEltProcess
*/
private function createProvide (DOMElement $DOMEltProcess) {
    $options = array(
        'partnerLink' => 'client'
    );
    $DOMEltProvide = $this->createElement('provide',
$options);
    $DOMEltProcess->appendChild($DOMEltProvide);
    //Service
    $Service = $this->getProcess()->getDefinitions()-
>getService();
    $options = array(
        'name' => sprintf('%s:%s', $this->getProcess()-
>getNamespacePrefixDeploy(), $Service->getName()),
        'port' => $Service->getPort()->getName(),
    );
    $DOMEltService = $this->createElement('service',
$options);
    $DOMEltProvide->appendChild($DOMEltService);
}

/**
* Cria o elemento process-events e o anexa ao elemento
process
* @param DOMElement $DOMEltProcess

```

```

    */
    private function createProcessEvents(DOMElement
$DOMEltProcess) {
        $options = array(
            'generate' => 'all',
        );
        $DOMEltProcessEvents = $this->createElement('process-
events', $options);
        $DOMEltProcess->appendChild($DOMEltProcessEvents);
    }

    /**
     * Cria o elemento retired e o anexo ao elemento deploy
     * @param DOMElement $DOMEltDeploy
     */
    private function createRetired(DOMElement $DOMEltDeploy) {
        $DOMEltRetired = $this->createElement('retired');
        $DOMEltRetired->appendChild($this->getDOMDocument()-
>createTextNode('false'));
        $DOMEltDeploy->appendChild($DOMEltRetired);
    }

    /**
     * Cria o elemento active e o anexo ao elemento deploy
     * @param DOMElement $DOMEltDeploy
     */
    private function createActive(DOMElement $DOMEltDeploy) {
        $DOMEltActive = $this->createElement('active');
        $DOMEltActive->appendChild($this->getDOMDocument()-
>createTextNode('true'));
        $DOMEltDeploy->appendChild($DOMEltActive);
    }

    /**
     * Cria o elemento deploy, raiz do documento
     * @return DOMElement
     */
    private function createDeploy() {
        $options = array(
            'xmlns' =>
'http://www.apache.org/ode/schemas/dd/2007/03',
            sprintf('xmlns:%s', $this->getProcess()-
>getNamespacePrefixDeploy()) => $this->getProcess()-
>getTargetNamespace(),
        );
        //Declara os namespaces dos serviços invocados
        foreach ($this->getProcess()-
>getAllPartnerLinkByType(RoleType::PARTNER) as $PartnerLink) {
            $XmlNamespace = $PartnerLink->getPartnerLinkType()-
>getRole()->getPortType()->getXmlNamespace();
            $options[sprintf('xmlns:%s', $XmlNamespace-
>getPrefixDeploy())] = $XmlNamespace->getUri();
        }
        return $this->createElement('deploy', $options);
    }
}

?>

```

**Executor\Controllor\Component\DeploymentComponent.php**

```

<?php

    /**
     * Componente responsável por efetuar a implantação das
     composições no
     * Orchestration Module.
     *
     * Para isso executa as seguintes tarefas:
     * - Converte uma composição armazenada no Composition Repository
     em um processo
     * (classe Process e suas associações);
     * - Invoca os geradores de arquivo (WSBpel, WSDL e deploy);
     * - Gera o pacote para implantação contendo os arquivos gerados
     no passo anterior;
     *
     * - Implanta o pacote gerado no Orchestration Module.
     * @author Augusto
     * @version 1.0
     * @updated 03-out-2013 09:53:20
     */
    class DeploymentComponent extends Component {

        /**
         * Lista dos nomes de componentes
         * @var array
         */
        public $components = array(
            'Executor.WsdlGenerator',
            'Executor.WsBpelGenerator',
            'Executor.WsdlReader',
            'Executor.DeployGenerator',
            'Executor.ArtifactsRepository',
        );

        /**
         * @var Composicao
         */
        private $m_Composicao;

        /**
         * @var Process
         */
        private $m_Process;

        /**
         *
         * @param composition_id
         */
        public function deploy($composicao_id) {
            return $this->generatePackage($composicao_id);
        }

        /**
         *
         * @param composition_id
         */
        public function rebuild($composition_id) {
    
```

```

    }

    /**
     * Gera o pacote zip para implantação do serviço no Apache Ode
     contendo o
     * deployment descriptor (deploy.xml), wsbpel descrevendo o
     processo (composição),
     * wsdl do processo e arquivos wsdl dos partners invocados
     pelo processo (serviços
     * que compõem a composição). Deve retornar o caminho para o
     arquivo .zip gerado
     * pela execução do método.
     *
     * @param deployment_descriptor
     * @param wsdl_partners
     * @param wsdl_composicao
     * @param wsbpel
     */
    public function generatePackage($composicao_id) {
        $start = microtime(true);
        $Process = $this->
>createProcessFromComposition($composicao_id);
        $this->log(microtime(true) - $start, 'convert');

        //Carrega os arquivos WSDL dos parceiros
        $start = microtime(true);
        $composicao = $this->loadComposition($composicao_id);
        $wsdls = Set::extract('/Caminho/Operacao/wsdl',
$composicao);
        $wsdl_partners = array();
        if (!empty($wsdls)) {
            foreach ($wsdls as $wsdl) {
                $filename = $this->WsdReader->filename($wsdl);
                $xml = $this->WsdReader->readWsd($wsdl);
                $wsdl_partners[$filename] = $xml;
            }
        }
        $this->log(microtime(true) - $start, 'wsdl_extract');

        $start = microtime(true);
        $bpel = $this->WsBpelGenerator->generate($Process);
        $this->log(microtime(true) - $start, 'bpel');

        $start = microtime(true);
        $wsdl = $this->WsdGenerator->generate($Process);
        $this->log(microtime(true) - $start, 'wsdl');

        $start = microtime(true);
        $deploy = $this->DeployGenerator->generate($Process);
        $this->log(microtime(true) - $start, 'deploy');

        $start = microtime(true);
        $package = $this->ArtifactsRepository->
>generatePackage($composicao_id, $bpel, $wsdl, $deploy,
$wsdl_partners);
        $this->log(microtime(true) - $start, 'package');

        return $package;
    }

    /**

```

```

        * A partir de uma composição gerada pelo SWSComposer e
armazenada no
        * CompositionRepository efetua a conversão desta para um
processo de acordo com a
        * estrutura definida pela classe Process e suas associações
para posterior
        * geração dos arquivos WSBpel, WSDL e deploy.
        *
        * @param composicao_id ID da Composicao
        * @return Process Composição gerada pelo SWSComposer
convertida em um Process
    */
    public function createProcessFromComposition($composicao_id) {
        App::uses('Process', 'Executor.Lib/Bpel');

        //Carregar a composicao
        $composicao = $this->loadComposition($composicao_id);

        //Mapeamento dos serviços envolvidos na composição
        $servicesMap = $this->getServicesMap($composicao);

        $Process_name = $composicao['Composicao']['uuid'];
        $targetNamespaceUri =
sprintf('http://executor.guto.me/%s', $Process_name);
        $this->m_Process = new Process($Process_name,
$targetNamespaceUri, 'yes');

        $TargetNamespace = new XmlNamespace('tns',
$targetNamespaceUri);
        $this->m_Process->addXmlNamespace($TargetNamespace);
        for ($i = 0; $i < count($servicesMap); $i++) {
            for ($j = 0; $j < count($servicesMap[$i]); $j++) {
                $operation = $servicesMap[$i][$j];
                $this->m_Process->addXmlNamespace(new
XmlNamespace($operation['prefix_ns'], $operation['namespace']));
            }
        }

        //Criar os imports para cada arquivo WSDL de serviços
invocados
        $importType = 'http://schemas.xmlsoap.org/wsdl/';
        for ($i = 0; $i < count($servicesMap); $i++) {
            for ($j = 0; $j < count($servicesMap[$i]); $j++) {
                $operation = $servicesMap[$i][$j];
                $location = $this->WsdLReader-
>filename($operation['wsdl']);
                $namespace = $this->WsdLReader-
>getTargetnamespace($operation['wsdl']);
                $this->m_Process->addImport(new Import($location,
$importType, $namespace));
            }
        }

        //Criar o import para o WSDL que sera gerado para o
processo
        $location = sprintf('%sArtifacts.wsdl', $Process_name);
        //@todo transferir para WsdLGenerator
        $this->m_Process->addImport(new Import($location,
$importType, $targetNamespaceUri));

        $Definitions = $this->m_Process->getDefinitions();
    }
}

```

```

//Target namespace
$Definitions->addXmlNamespace($TargetNamespace);
//Namespaces definidos para cada operação envolvida
for ($i = 0; $i < count($servicesMap); $i++) {
    for ($j = 0; $j < count($servicesMap[$i]); $j++) {
        $operation = $servicesMap[$i][$j];
        $Definitions->addXmlNamespace(new
XmlNamespace($operation['prefix'], $operation['namespace']));
    }
}
for ($i = 0; $i < count($servicesMap); $i++) {
    for ($j = 0; $j < count($servicesMap[$i]); $j++) {
        $operation = $servicesMap[$i][$j];
        $location = $this->WsdLReader-
>filename($operation['wsdl']);
        $namespace = $this->WsdLReader-
>getTargetnamespace($operation['wsdl']);
        $Definitions->addImportWsdL(new
ImportWsdL($location, $namespace));
    }
}

//Cria o ElementRequest que representa o tipo de entrada
do processo
$ElementRequest = new Element(sprintf('%sRequest',
$Process_name), null, $TargetNamespace);
//Para cada entrada disponível gera um subelemento no
ElementRequest
$entradas = Hash::extract($composicao, 'Entrada');
foreach ($entradas as $entrada) {
    $ElementRequest->addElement(new
ElementInput($entrada['nome'], $entrada['tipo']));
}
//Define o ElementRequest em Definitions
$Definitions->addElement($ElementRequest);

//Cria o ElementResponse que representa o tipo de saída do
processo
$ElementResponse = new Element(sprintf('%sResponse',
$Process_name), null, $TargetNamespace);
//Para a saída produzida de cada caminho gera um sublement
no ElementResponse
$saidas = Hash::extract($composicao, 'Caminho.{n}.Saida');
foreach ($saidas as $saida) {
    $ElementResponse->addElement(new
ElementOutput($saida['nome'], 'string'));
}
//Define o ElementRequest em Definitions
$Definitions->addElement($ElementResponse);
//Request
$requestPart = new Part('payload', $ElementRequest);
$inputMessage = new Message(sprintf('%sRequestMessage',
$Process_name), $requestPart, $TargetNamespace);
$Definitions->addMessage($inputMessage);

//Response
$responsePart = new Part('payload', $ElementResponse);
$outputMessage = new Message(sprintf('%sResponseMessage',
$Process_name), $responsePart, $TargetNamespace);
$Definitions->addMessage($outputMessage);

```

```

        $Operation = new Operation('process', $InputMessage,
$OutputMessage);
        $PortType = new PortType($Process_name, $Operation,
$TargetNamespace);
        $Definitions->setPortType($PortType);

        $Role = new Role(sprintf('%sProvider', $Process_name),
$PortType);
        $PartnerLinkTypeClient = new
PartnerLinkType($Process_name, $Role);
        $Definitions->setPartnerLinkType($PartnerLinkTypeClient);

        $Binding = new Binding(sprintf('%sBinding',
$Process_name), $PortType, $TargetNamespace);
        $Definitions->setBinding($Binding);

        $Address = new
Address(sprintf('http://localhost:8080/ode/processes/%s',
$Process_name));
        $Port = new Port(sprintf('%sPort', $Process_name),
$Binding, $Address);
        $Service = new Service(sprintf('%sService',
$Process_name), $Port);
        $Definitions->setService($Service);

        //Variavel de entrada
        $VariableInput = new Variable('input', $InputMessage);
        $this->m_Process->addVariable($VariableInput);

        //Variavel de saida
        $VariableOutput = new Variable('output', $OutputMessage);
        $this->m_Process->addVariable($VariableOutput);

        //Cria o partnerLink client que representa o solicitante
deste serviço
        $clientPL = new PartnerLink('client', RoleType::MY,
$PartnerLinkTypeClient);
        $this->m_Process->addPartnerLink($clientPL);

        //Criar os partnerLinks para cada serviço utilizado na
composição
        for ($i = 0; $i < count($servicesMap); $i++) {
            $Sequence = new Sequence(sprintf('Caminho%d', $i));
            $this->m_Process->addSequence($Sequence);
            //Define a variavel e elemento da saida gerada por
este caminho
            $Sequence->setVariableOutput($VariableOutput);
            $Sequence->setElementOutput($VariableOutput-
>getMessageType()->getPart()->getElement()-
>getElementByName($composicao['Caminho'][$i]['Saida']['nome']));
            //Cache de inputs e outputs do caminho
            $caminhoInputCache = array();
            $caminhoOutputCache = array();
            for ($j = 0; $j < count($servicesMap[$i]); $j++) {
                $operacaoGeraSaida = false;
                $operation = $servicesMap[$i][$j];
                //Criação do Operation
                $operation = new Operation($operation['name']);
                //Criação do PortType - Pega no arquivo WSDL o
nome do PortType que contém a operação atual

```

```

        $PortTypeNamespace = $Definitions-
>getXmlNamespaceByUri($operation['namespace']);
        $PortType_name = $this->WsdReader-
>getPortTypeByOperation($operation['name'], $operation['wsdl']);
        $PortType = new PortType($PortType_name,
$Operation, $PortTypeNamespace);
        //Criação do Role
        $Role = new Role($this->WsdReader-
>createNameByWsd($operation['wsdl'], 'PLRole'), $PortType);
        //Criação do PartnerLinkType
        $PartnerLinkType = new PartnerLinkType($this-
>WsdReader->createNameByWsd($operation['wsdl'], 'PLT'), $Role);
        //Criação do PartnerLink
        $PartnerLink = new PartnerLink($this->WsdReader-
>createNameByWsd($operation['wsdl'], 'PL'), RoleType::PARTNER,
$PartnerLinkType);
        $PartnerLink->setUrlWsd($operation['wsdl']);
        //Inclui o PartnerLink no Bpel::Process
        $this->m_Process->addPartnerLink($PartnerLink);
        //Inclui o PartnerLinkType no Wsdl::Definitions
        $Definitions-
>addPartnerLinkType($PartnerLinkType);
        //Variavel de entrada para este PartnerLink
        $InputMessage_name = $this->WsdReader-
>getOperationInputMessageType($operation['name'], $operation['wsdl']);
        $InputMessagePart = new Part('parameters', new
Element($InputMessage_name, null));
        //Cada parametro (elemento element) é incluido na
mensagem de entrada
        foreach
($composicao['Caminho'][$i]['Operacao'][$j]['Entrada'] as $entrada) {
            $caminhoInputCache[$entrada['id']] = new
ElementInput($entrada['nome'], $entrada['tipo']);
            $InputMessagePart->getElement()-
>addElement($caminhoInputCache[$entrada['id']]);
        }
        $InputMessage = new Message($InputMessage_name,
$InputMessagePart, $this->m_Process-
>getXmlNamespaceByUri($operation['namespace']));
        $InputVariable = new
Variable(sprintf('%sC%dO%dRequest', $PartnerLink->getName(), $i, $j),
$InputMessage);
        $this->m_Process->addVariable($InputVariable);

        //Variavel de saida para este PartnerLink
        $OutputMessage_name = $this->WsdReader-
>getOperationOutputMessageType($operation['name'],
$operation['wsdl']);
        $OutputMessagePart = new Part('parameters', new
Element($OutputMessage_name, null));
        foreach
($composicao['Caminho'][$i]['Operacao'][$j]['Saida'] as $saida) {
            $caminhoOutputCache[$saida['id']] = array(
                'variable' => null,
                'element' => new
ElementOutput($saida['nome'], $saida['tipo']),
            );
            $OutputMessagePart->getElement()-
>addElement($caminhoOutputCache[$saida['id']]['element']);
        }
        //Testa se esta saida é a saida do caminho

```



```

        if ($saida['id'] ==
$composicao['Caminho'][$i]['Saida']['OrigemSaida']['id']) {
            $Sequence-
>setElementSource($caminhoOutputCache[$saida['id']]['element']);
            $operacaoGeraSaida = true;
        }
    }
    $OutputMessage = new Message($OutputMessage_name,
$OutputMessagePart, $this->m_Process-
>getXmlNamespaceByUri($operation['namespace']));
    $OutputVariable = new
Variable(sprintf('%sC%dO%dResponse', $PartnerLink->getName(), $i, $j),
$OutputMessage);
    $this->m_Process->addVariable($OutputVariable);

    //Preenche a variavel para os elementos incluidos
foreach ($caminhoOutputCache as &$item) {
        if (is_null($item['variable'])) {
            $item['variable'] = $OutputVariable;
        }
    }

    //Se esta operação gera a saida do caminho define
a variavel
    //de saida como variable de origem da saida da
composicao
    if ($operacaoGeraSaida) {
        $Sequence->setVariableSource($OutputVariable);
    }
    //Define as mensagens de entrada e saida da
operação
    $Operation->setInputMessage($InputMessage);
    $Operation->setOutputMessage($OutputMessage);
    //Cria um Invoke para esta operação
    $Invoke = new
Invoke(sprintf('Caminho%dOperacao%d', $i, $j), $PartnerLink,
$InputVariable, $OutputVariable);
    $Sequence->addInvoke($Invoke);
    //Para este invoke faz a vinculação dos parametros
de entrada com sua origem
    $_operacao =
$composicao['Caminho'][$i]['Operacao'][$j];
    foreach ($Invoke->getInputVariable()-
>getMessageType()->getPart()->getElement()->getAllElements() as
$Element) {
        $entrada =
array_shift(Set::extract('/Entrada[nome=' . $Element->getName() . ']',
$operacao));
        switch (true) {
            //Origem do parametro em entrada da
composição
            case (!empty($entrada['Entrada']['OrigemEntradaComposicao'])):
                //Define a variavel de origem como a
variavel de
                //entrada da composição
                $Element-
>setVariableSource($VariableInput);
                //Define o elemento de origem como o
elemento de

```

```

//mesmo nome contido na variavel de
entrada da composição
$ElementSource_name =
$entrada['Entrada']['OrigemEntradaComposicao']['nome'];
$ElementSource = $VariableInput-
>getMessageType()->getPart()->getElement()-
>getElementByName($ElementSource_name);
$Element-
>setElementSource($ElementSource);
break;
//Origem do parametro em saida de operação
executada anteriormente

case(!empty($entrada['Entrada']['OrigemSaidaOperacao'])):
$Element-
>setVariableSource($caminhoOutputCache[$entrada['Entrada']['OrigemSaid
aOperacao']['id']]['variable']);
$Element-
>setElementSource($caminhoOutputCache[$entrada['Entrada']['OrigemSaida
Operacao']['id']]['element']);
break;
}
}
}

return $this->m_Process;
}

/**
 * Carrega uma composição do Composition Repository
 *
 * @param composicao_id ID da Composicao
 */
public function loadComposition($composicao_id = null) {
$this->m_Composicao =
ClassRegistry::init('Executor.Composicao');
$conditions = array('Composicao.id' => $composicao_id);
$composicao = $this->m_Composicao->find('full',
compact('conditions'));
if (!$composicao) {
throw new NotFoundException(sprintf('Composição #%d
não encontrada', $composicao_id));
}
return $composicao;
}

/**
 * Para cada operação de cada caminho invocadas pelo
composição é gerado
 * um array associativo, por caminho, contendo o nome,
prefixo, namespace
 * e url do wsdl do serviço invocado
 * @param array $composicao
 * @return array
 * @todo Extrair o targetNamespace
 */
public function getServicesMap(array $composicao) {
$map = array();

```

```

    $caminho_index = 0;
    $operacao_index = 0;
    foreach ($composicao['Caminho'] as $caminho) {
        $map[$caminho_index] = array();
        foreach ($caminho['Operacao'] as $operacao) {
            $prefix_ns = sprintf('ns%s', $operacao_index ?
$operacao_index : '');
            $prefix = sprintf('wsdl%s', $operacao_index ?
$operacao_index : '');
            $namespace = $this->WsdReader-
>getTargetnamespace($operacao['wsdl']);
            $pathinfo = pathinfo($operacao['wsdl']);
            $map[$caminho_index][] = array(
                'id' => $operacao['id'],
                'name' => $operacao['nome'],
                'prefix_ns' => $prefix_ns,
                'prefix' => $prefix,
                'namespace' => $namespace,
                'wsdl' => $operacao['wsdl']
            );
        }
        $caminho_index++;
    }
    return $map;
}
}
?>

```

**Executor\Controller\Component\HttpClientComponent.php**

```

<?php

    /**
     * Componente responsável por fazer requisições HTTP e capturar
     seu conteúdo.
     *
     * Implementada com base na biblioteca cUrl.
     *
     * @author Augusto
     */
App::uses('Component', 'Controller');

class HttpClientComponent extends Component {

    /**
     * Inicialização do componente. Efetua teste se a biblioteca
     cURL está disponível
     * no sistema.
     * @param Controller $controller
     * @throws CakeException
     */
public function initialize(Controller $controller) {
    parent::initialize($controller);
    if (!function_exists('curl_init')) {
        throw new CakeException('A biblioteca cURL
indisponível.');
```

Executor\Controller\Component\WsBpelGeneratorComponent.php

```
<?php

    /**
     * WsBpelGeneratorComponent - Implementação do WsBpel Generator
Module
     *
     * Componente responsável por gerar o arquivo no formato BPEL que
descreve
     * o processo (uma composição) para implantação no Orchestration
Module
     *
     * @author Augusto Cesar Ferreira
     */
    App::uses('AbstractXmlGeneratorComponent',
'Executor.Controller/Component');

    class WsBpelGeneratorComponent extends
AbstractXmlGeneratorComponent {

        /**
         * Implementação do método generate definido pela classe
         * AbstractXmlGeneratorComponent
         * @param Process $Process
         */
        public function generate(Process $Process) {
            //Cria o DOMDocument
            $this->setDOMDocument(new DOMDocument());
            //Define o Process
            $this->setProcess($Process);
            //process (root do wsbpel)
            $DOMElmtProcess = $this->createProcess();
            //Imports
            $this->createImports($DOMElmtProcess);
            //PartnerLinks
            $this->createPartnerLinks($DOMElmtProcess);
            //Variables
            $this->createVariables($DOMElmtProcess);
            //Sequence
            $this->createMainSequence($DOMElmtProcess);
            //Inlcui o elementDefinitions no DOMDocument
            $this->getDOMDocument()->appendChild($DOMElmtProcess);
            //Retorna o xml
            return $this->getDOMDocument()->saveXML();
        }

        /**
         * Cria o elemento reply responsável por responder com o
resultado da
         * execução da composição pelo cliente
         * @param DOMElement $DOMElmtMainSequence
         */
        private function createReply(DOMElement $DOMElmtMainSequence)
        {
            $options = array(
                'name' => 'ReplyOutput',
                'partnerLink' => $this->getProcess()-
>getPartnerLinkByName('client')->getName(),
```

```

        'portType' => $this->getProcess()-
>getPartnerLinkByName('client')->getPartnerLinkType()->getRole()-
>getPortType()->getQName(),
        'operation' => 'process',
        'variable' => 'output',
    );
    $DOMEltReply = $this->createElement('bpel:reply',
$options);
    $DOMEltMainSequence->appendChild($DOMEltReply);
}

/**
 * Cria o elemento sequence que representa a sequencia
principal, que
 * contém o elemento flow onde os caminhos da composição
estarão representados
 * por elementos sequence.
 * Cria e anexa ao elemento process o elemento variables e
seus subelementos
 * @param DOMElement $$DOMEltProcess Element Process
 */
private function createMainSequence(DOMElement
$DOMEltProcess) {
    $DOMEltMainSequence = $this-
>createElement('bpel:sequence', array('name' => 'Composicao'));
    //Cria a atividade receive
    $this->createReceiveActivity($DOMEltMainSequence);
    //Iniciar as variaveis
    $this->createInitVariables($DOMEltMainSequence);
    //Criar o Flow (processamento paralelo dos caminhos
    $this->createFlow($DOMEltMainSequence);
    //Reply
    $this->createReply($DOMEltMainSequence);
    //Anexa o elemento sequence ao elemento process
    $DOMEltProcess->appendChild($DOMEltMainSequence);
}

/**
 * Criar o elemento flow, onde serão definidos os elementos
sequence
 * para cada caminho, permitindo que estes sejam executados em
paralelo
 * @param DOMElement $DOMEltMainSequence Sequencia principal
do processo
 */
private function createFlow(DOMElement $DOMEltMainSequence) {
    //Cria o elemento flow
    $DOMEltFlow = $this->createElement('bpel:flow',
array('name' => 'Flow'));
    //Para cada sequence cria um elemento sequence e o anexo
ao flow
    $Sequences = $this->getProcess()->getAllSequence();
    foreach ($Sequences as $Sequence) {
        //Cria o elemento sequence
        $DOMEltSequence = $this-
>createElement('bpel:sequence', array('name' => $Sequence-
>getName()));
        $Invokes = $Sequence->getAllInvoke();
        //Cria os elementos invokes que compoem este sequence
        foreach ($Invokes as $Invoke) {

```

```

//Cria e anexa o(s) elemento(s) assign que faz(em)
a definição do(s)
//parametro(s) de entrada do proximo serviço que
será invocado
    $DOMEltAssign = $this-
>createAssignInvokeParameters($Invoke->getInputVariable());
    $DOMEltSequence->appendChild($DOMEltAssign);
//Cria e anexa o elemento invoke ao elemento
sequence
    $DOMEltInvoke = $this->createInvoke($Invoke);
    $DOMEltSequence->appendChild($DOMEltInvoke);
}
//Cria o elemento assign que associa a saida do
caminho a saida da composicao
    $DOMEltAssign = $this-
>createAssignSequenceToOutput($Sequence);
    $DOMEltSequence->appendChild($DOMEltAssign);
//Anexo o elemento sequence ao elemento flow
    $DOMEltFlow->appendChild($DOMEltSequence);
}
//Anexo o elemento flow ao elemento sequence referente a
sequencia principal
    $DOMEltMainSequence->appendChild($DOMEltFlow);
}

/**
 * Define a saida gerada pelo caminha na saida da composicao.
 * @param Sequence $Sequence
 * @return DOMElement
 */
public function createAssignSequenceToOutput(Sequence
$Sequence) {
    //Cria o elemento assign
    $options = array(
        'validate' => 'no',
        'name' => 'AssignOutputToResult'
    );
    $DOMEltAssign = $this->createElement('bpel:assign',
$options);
    //Cria o elemento copy e anexa ao elemento assign
    $DOMEltCopy = $this->createElement('bpel:copy');
    $DOMEltAssign->appendChild($DOMEltCopy);
    //Cria o elemento from e o anexa ao elemento copy
    $options = array(
        'part' => $Sequence->getVariableSource() -
>getMessageType()->getPart()->getName(),
        'variable' => $Sequence->getVariableSource()-
>getName(),
    );
    $DOMEltFrom = $this->createElement('bpel:from',
$options);
    $DOMEltCopy->appendChild($DOMEltFrom);
    //Cria o elemento query e anexa ao elemento from
    $options = array(
        'queryLanguage' =>
'urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0',
    );
    $DOMEltQuery = $this->createElement('bpel:query',
$options);
    $DOMEltQuery->appendChild($this->getDOMDocument()-
>createCDATASection($Sequence->getElementSource()->getName()));
}

```

```

        $DOMEltFrom->appendChild($DOMEltQuery);
        //Cria o elemento to e anexa ao elemento copy
        $options = array(
            'part' => $Sequence->getVariableOutput()-
>getMessageType()->getPart()->getName(),
            'variable' => $Sequence->getVariableOutput()-
>getName(),
        );
        $DOMEltTo = $this->createElement('bpel:to', $options);
        $DOMEltCopy->appendChild($DOMEltTo);
        //Cria o elemento query e anexa ao elemento to
        $options = array(
            'queryLanguage' =>
'urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0',
        );
        $DOMEltQuery = $this->createElement('bpel:query',
$options);
        $DOMEltQuery->appendChild($this->getDOMDocument()-
>createCDATASection('tns:' . $Sequence->getElementOutput()-
>getName()));
        $DOMEltTo->appendChild($DOMEltQuery);
        return $DOMEltAssign;
    }

/**
 * Faz a inicialização das variáveis de entrada de um serviço
que será
 * realizado em seguida. Define um elemento assign que contém
elementos
 * copy, um para cada parametro do serviço.
 * @param Variable $InputVariable Variável de entrada do
serviço que será
 * chamado pela operação invoke subsequente
 * @return DOMElement DOMElement representando a atividade
assign
 */
    public function createAssignInvokeParameters(Variable
$InputVariable) {
        //Cria o elemento assign
        $options = array(
            'validate' => 'no',
            'name' => 'AssignInvokeParameters',
        );
        $DOMEltAssign = $this->createElement('bpel:assign',
$options);
        foreach ($InputVariable->getMessageType()->getPart()-
>getElement()->getAllElements() as $Element) {
            //Variavel de origem
            $VariableSource = $Element->getVariableSource();
            //Elemento de origem
            $ElementSource = $Element->getElementSource();
            //Cria o elemento copy e anexa ao elemento assign
            $DOMEltCopy = $this->createElement('bpel:copy');
            $DOMEltAssign->appendChild($DOMEltCopy);
            //Cria o elemento from e o anexa ao elemento copy
            $options = array(
                'part' => $VariableSource->getMessageType()-
>getPart()->getName(),
                'variable' => $VariableSource->getName(),
            );

```



```

        $DOMEltFrom = $this->createElement('bpel:from',
$options);
        $DOMEltCopy->appendChild($DOMEltFrom);
        //Cria o elemento query e anexa ao elemento from
        $options = array('queryLanguage' =>
'urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0');
        $DOMEltQuery = $this->createElement('bpel:query',
$options);
        if ($VariableSource->getName() == 'input') {
            $DOMEltQuery->appendChild($this-
>getDOMDocument()->createCDATASection('tns:' . $ElementSource-
>getName()));
        } else {
            $DOMEltQuery->appendChild($this-
>getDOMDocument()->createCDATASection($ElementSource->getName()));
        }
        $DOMEltFrom->appendChild($DOMEltQuery);
        //Cria o elemento to e anexa ao elemento copy
        $options = array(
'part' => $InputVariable->getMessageType()-
>getPart()->getName(),
'variable' => $InputVariable->getName(),
);
        $DOMEltTo = $this->createElement('bpel:to',
$options);
        $DOMEltCopy->appendChild($DOMEltTo);
        //Cria o elemento query e anexa ao elemento to
        $options = array('queryLanguage' =>
'urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0');
        $DOMEltQuery = $this->createElement('bpel:query',
$options);
        $DOMEltQuery->appendChild($this->getDOMDocument()-
>createCDATASection($Element->getName()));
        $DOMEltTo->appendChild($DOMEltQuery);
    }
    return $DOMEltAssign;
}

/**
 * Cria um elemento invoke
 * @param Invoke $Invoke
 * @return DOMElement Elemento invoke
 */
private function createInvoke(Invoke $Invoke) {
    //Cria o elemento
    $options = array(
        'name' => sprintf('Invoke%s', $Invoke->getName()),
        'partnerLink' => $Invoke->getPartnerLink()->getName(),
        'operation' => $Invoke->getPartnerLink()-
>getPartnerLinkType()->getRole()->getPortType()->getOperation()-
>getName(),
        'portType' => 'ns:' . $Invoke->getPartnerLink()-
>getPartnerLinkType()->getRole()->getPortType()->getName(),
        'inputVariable' => $Invoke->getInputVariable()-
>getName(),
        'outputVariable' => $Invoke->getOutputVariable()-
>getName(),
    );
    return $this->createElement('bpel:invoke', $options);
}

```

```

/**
 * Cria um element assign responsavel por fazer a
inicialização das
 * variaveis definidas
 * @param DOMElement $DOMEltSequence Elemento bpel:sequence
 * @todo Verificar as variaveis que necessitam inicialização e
ler
 * a partir do wsdl que se referem seu tipo
 */
private function createInitVariables(DOMElement
$DOMEltSequence) {
    $options = array(
        'name' => 'InitVariables',
        'validate' => 'no',
    );
    $DOMEltAssign = $this->createElement('bpel:assign',
$options);
    $DOMEltSequence->appendChild($DOMEltAssign);
//Realiza a inicialização das variaveis de entrada dos
serviços
//participantes e da variável de saída da composição
foreach ($this->getProcess()->getAllVariable() as
$Variable) {
    //Somente para variáveis que contém o elemento Part e
excluindo
    //a variável input
    if (!is_null($Variable->getMessageType()->getPart())
&& $Variable->getName() != 'input') {
        $DOMEltCopy = $this-
>createInitVariable($Variable);
        $DOMEltAssign->appendChild($DOMEltCopy);
    }
}
}

/**
 * Cria a inicialização de uma variável
 * @param Variable $Variable
 * @return type
 */
private function createInitVariable(Variable $Variable) {
    //Elemento copy
    $DOMEltCopy = $this->createElement('bpel:copy');
    //Elemento from
    $DOMEltFrom = $this->createElement('bpel:from');
    //Anexa o elemento from ao elemento copy
    $DOMEltCopy->appendChild($DOMEltFrom);
    //Elemento Literal
    $DOMEltLiteral = $this->createElement('bpel:literal');
    //Cria o elemento
    $element_name = sprintf('tns:%s', $Variable-
>getMessageType()->getPart()->getElement()->getName());
    $options = array(
        'xmlns:tns' => $Variable->getMessageType()-
>getXmlNamespace()->getUri(),
        'xmlns:xsi' => 'http://www.w3.org/2001/XMLSchema-
instance'
    );
    $DOMEltElement = $this->createElement($element_name,
$options);
    $DOMEltLiteral->appendChild($DOMEltElement);
}

```

```

        foreach ($Variable->getMessageType()->getPart()-
>getElement()->getAllElements() as $Element) {
            //De acordo com o tipo do element define o valor
inicial
            switch ($Element->getType()) {
                case 'char':
                case 'string':
                    $value = '-';
                    break;
                case 'float':
                case 'int':
                case 'integer':
                case 'double':
                case 'long':
                    $value = 0;
                    break;
                default:
                    $value = '';
            }
            $options = array();
            if ($Variable->getName() == 'output') {
                $element_name = sprintf('tns:%s', $Element-
>getName());
            } else {
                $element_name = $Element->getName();
            }
            $DOMElmtElement2 = $this->createElement($element_name,
$options);
            $DOMElmtElement2->appendChild($this->getDOMDocument()-
>createTextNode($value));
            $DOMElmtElement->appendChild($DOMElmtElement2);
        }

        //Anexa o elemento literal ao elemento from
        $DOMElmtFrom->appendChild($DOMElmtLiteral);
        //Elemento To
        $options = array(
            'variable' => $Variable->getName(),
            'part' => $Variable->getMessageType()->getPart()-
>getName(),
        );
        $DOMElmtTo = $this->createElement('bpel:to', $options);
        //Anexo o elemento to ao elemento copy
        $DOMElmtCopy->appendChild($DOMElmtTo);
        return $DOMElmtCopy;
    }

    /**
     * Cria o elemento receive e o anexa ao elemento sequence.
     * @param DOMEElement $DOMElmtSequence Elemento bpel:sequence
     */
    private function createReceiveActivity(DOMEElement
$DOMElmtSequence) {
        $PartnerLink = $this->getProcess()-
>getPartnerLinkByName('client');
        $options = array(
            'name' => 'receiveInput',
            'partnerLink' => $PartnerLink->getName(),
            'portType' => $PartnerLink->getPartnerLinkType()-
>getRole()->getPortType()->getQName(),
            'operation' => 'process',

```

```

        'variable' => 'input',
        'createInstance' => 'yes',
    );
    $DOMEltReceive = $this->createElement('bpel:receive',
$options);
    $DOMEltSequence->appendChild($DOMEltReceive);
}

/**
 * Cria e anexa ao elemento process o elemento variables e
seus subelementos
 * @param DOMElement $$DOMEltProcess Element Process
 */
private function createVariables(DOMElement $DOMEltProcess) {
    //PartnerLinks
    $DOMEltVariables = $this-
>createElement('bpel:variables');
    //PartnerLink
    $Variables = $this->getProcess()->getAllVariable();
    foreach ($Variables as $Variable) {
        $options = array(
            'name' => $Variable->getName(),
            'messageType' => $Variable->getMessageType()-
>getQName(),
        );
        $DOMEltVariable = $this-
>createElement('bpel:variable', $options);
        //Anexa o elemento variable ao variables
        $DOMEltVariables->appendChild($DOMEltVariable);
    }
    //Anexa o elemento variables ao process
    $DOMEltProcess->appendChild($DOMEltVariables);
}

/**
 * Cria e anexa ao elemento process o elemento partnerLinks e
seus subelementos
 * @param DOMElement $$DOMEltProcess Element Process
 */
private function createPartnerLinks(DOMElement
$DOMEltProcess) {
    //PartnerLinks
    $DOMEltPartnerLinks = $this-
>createElement('bpel:partnerLinks');
    //PartnerLink
    $PartnerLinks = $this->getProcess()->getAllPartnerLink();
    foreach ($PartnerLinks as $PartnerLink) {
        $options = array(
            'name' => $PartnerLink->getName(),
            'partnerLinkType' => $PartnerLink-
>getPartnerLinkType()->getQName(),
            $PartnerLink->getRoleType() => $PartnerLink-
>getPartnerLinkType()->getRole()->getName(),
        );
        $DOMEltPartnerLink = $this-
>createElement('bpel:partnerLink', $options);
        //Anexa o partnerLink ao partnerLinks
        $DOMEltPartnerLinks-
>appendChild($DOMEltPartnerLink);
    }
    //Anexa o partnerLinks ao process

```

```

        $DOMEltProcess->appendChild($DOMEltPartnerLinks);
    }

    /**
     * Cria e anexa ao elemento process os elementos import
     * @param DOMElement $$DOMEltProcess Element Process
     */
    private function createImports(DOMElement $DOMEltProcess) {
        $imports = $this->getProcess()->getAllImport();
        foreach ($imports as $import) {
            $options = array(
                'location' => $import->getLocation(),
                'namespace' => $import->getNamespace(),
                'importType' => $import->getImportType(),
            );
            $DOMEltProcess->appendChild($this-
>createElement('bpel:import', $options));
        }

        /**
         * Cria o elemento definitions com seus atributos
         * @return DOMElement Process
         */
        private function createProcess() {
            $Process = $this->getProcess();
            $options = array(
                'name' => $Process->getName(),
                'targetNamespace' => $Process->getTargetNamespace(),
                'suppressJoinFailure' => $Process-
>getSuppressJoinFailure(),
            );
            //Declaração de namespaces
            foreach ($Process->getAllXmlNamespace() as $XmlNamespace) {
                $prefix = is_null($XmlNamespace->getPrefix()) ?
'xmlns' : sprintf('xmlns:%s', $XmlNamespace->getPrefix());
                $options[$prefix] = $XmlNamespace->getUri();
            }
            return $this->createElement('bpel:process', $options);
        }
    }

    ?>

```

**Executor\Controller\Component\WsdGeneratorComponent.php**

```

<?php

    /**
     * WsdGeneratorComponent - Implementação do Wsd Generator Module
     *
     * Componente responsável por gerar o arquivo wsdl para
    implantação do processo
     * bpel no Orchestration Module
     *
     * @author Augusto Cesar Ferreira
     */
    class WsdGeneratorComponent extends AbstractXmlGeneratorComponent
    {

        /**
         * Implementação do método generate definido pela classe
        AbstractXmlGeneratorComponent.
         * A partir de um objeto da classe Process e suas associações
        representando
         * uma composições gerada pelo SWSComposer gera o arquivo WSDL
        para
         * implantação do processo no Orchestration Module.
         * @param Process $Process
         */
        public function generate(Process $Process) {
            //Cria o DOMDocument
            $this->setDOMDocument(new DOMDocument());
            //Define o Process
            $this->setProcess($Process);
            //Definitions (root do wsdl)
            $DOMElmtDefinitions = $this->createDefinitions();
            //PartnerLinkTypes
            $this->createPartnerLinkTypes($DOMElmtDefinitions);
            //Imports
            $this->createImportWsd($DOMElmtDefinitions);
            //Types
            $this->createTypes($DOMElmtDefinitions);
            //Messages
            $this->createMessages($DOMElmtDefinitions);
            //Port Type
            $this->createPortType($DOMElmtDefinitions);
            //Partner Type (do processo)
            $this->createPartnerLinkType($DOMElmtDefinitions);
            //Binding
            $this->createBinding($DOMElmtDefinitions);
            //Service
            $this->createService($DOMElmtDefinitions);
            //Inlcui o elementDefinitions no DOMDocument
            $this->getDOMDocument()->appendChild($DOMElmtDefinitions);
            //Retorna o xml
            return $this->getDOMDocument()->saveXML();
        }

        /**
         * Cria e anexa ao elemento definitions o elemento service
         * @param DOMElement $DOMElmtDefinitions Definitions
         */
        private function createService(DOMElement $DOMElmtDefinitions)
    {

```

```

        //Service
        $Service = $this->getProcess()->getDefinitions()-
>getService();
        $DOMEltService = $this->createElement('service',
array('name' => $Service->getName()));
        //Anexa service ao definitions
        $DOMEltDefinitions->appendChild($DOMEltService);
        //Port
        $Port = $Service->getPort();
        $options = array(
            'name' => $Port->getName(),
            'binding' => $Port->getBinding()->getQName(),
        );
        $DOMEltPort = $this->createElement('port', $options);
        //Anexa port ao service
        $DOMEltService->appendChild($DOMEltPort);
        //SoapAddress
        $Address = $Port->getAddress();
        $DOMEltAddress = $this->createElement('soap:address',
array('location' => $Address->getLocation()));
        //Anexa soap:address ao port
        $DOMEltPort->appendChild($DOMEltAddress);
    }

/**
 * Cria e anexa ao elemento definitions o elemento binding
 * @param DOMElement $DOMEltDefinitions Definitions
 */
private function createBinding(DOMElement $DOMEltDefinitions)
{
    //Binding
    $Binding = $this->getProcess()->getDefinitions()-
>getBinding();
    $options = array(
        'name' => $Binding->getName(),
        'type' => $Binding->getPortType()->getQName(),
    );
    $DOMEltBinding = $this->createElement('binding',
    $options);
    //Anexa o binding ao definitions
    $DOMEltDefinitions->appendChild($DOMEltBinding);
    //SoapBinding
    $DOMEltSoapBinding = $this->createElement('soap:binding',
array('style' => 'document', 'transport' =>
'http://schemas.xmlsoap.org/soap/http'));
    //Anexa o soap:binding ao binding
    $DOMEltBinding->appendChild($DOMEltSoapBinding);
    //Operation
    $DOMEltOperation = $this->createElement('operation',
array('name' => 'process'));
    //Anexa o operation ao binding
    $DOMEltBinding->appendChild($DOMEltOperation);
    //Input
    $DOMEltInput = $this->createElement('input');
    //Anexa o input ao operation
    $DOMEltOperation->appendChild($DOMEltInput);
    //Output
    $DOMEltOutput = $this->createElement('output');
    //Anexa o output ao operation
    $DOMEltOperation->appendChild($DOMEltOutput);
    //SoapBody

```

```

        $DOMEltSoapBody = $this->createElement('soap:body',
array('use' => 'literal'));
        //Anexa soap:body ao input e output
        $DOMEltInput->appendChild($DOMEltSoapBody);
        $DOMEltOutput->appendChild($DOMEltSoapBody-
>cloneNode());
    }

    /**
     * Cria e anexa ao elemento definitions o elemento
partnerLinkType
     * referente ao próprio processo
     * @param DOMElement $DOMEltDefinitions Definitions
     */
    private function createPartnerLinkType(DOMElement
$DOMEltDefinitions) {
        //PartnerLinkType
        $PartnerLinkType = $this->getProcess()->getDefinitions()-
>getPartnerLinkType();
        $DOMEltPLT = $this->createElement('plnk:partnerLinkType',
array('name' => $PartnerLinkType->getName()));
        //Anexa o partnerLinkType ao definitions
        $DOMEltDefinitions->appendChild($DOMEltPLT);
        //Role ao partnerLinkType
        $Role = $PartnerLinkType->getRole();
        $options = array(
            'name' => $Role->getName(),
            'portType' => $Role->getPortType()->getQName(),
        );
        $DOMEltRole = $this->createElement('plnk:role',
$options);
        $DOMEltPLT->appendChild($DOMEltRole);
    }

    /**
     * Cria e anexa ao elemento definitions o elemento portType
     * @param DOMElement $DOMEltDefinitions Definitions
     */
    private function createPortType(DOMElement
$DOMEltDefinitions) {
        //PortType
        $PortType = $this->getProcess()->getDefinitions()-
>getPortType();
        $DOMEltPortType = $this->createElement('portType',
array('name' => $PortType->getName()));
        //Anexa o portType ao definitions
        $DOMEltDefinitions->appendChild($DOMEltPortType);
        //Operation
        $Operation = $PortType->getOperation();
        $DOMEltOperation = $this->createElement('operation',
array('name' => $Operation->getName()));
        //Anexa operation ao portType
        $DOMEltPortType->appendChild($DOMEltOperation);
        //InputMessage
        $InputMessage = $Operation->getInputMessage();
        $DOMEltInputMessage = $this->createElement('input',
array('message' => $InputMessage->getQName()));
        //Anexa input ao operation
        $DOMEltOperation->appendChild($DOMEltInputMessage);
        //OutputMessage
        $OutputMessage = $Operation->getOutputMessage();
    }

```



```

        $DOMEltOutputMessage = $this->createElement('output',
array('message' => $OutputMessage->getQName()));
        //Anexa input ao operation
        $DOMEltOperation->appendChild($DOMEltOutputMessage);
    }

    /**
     * Cria e anexa ao elemento definitions a seção onde são
     definidas os elementos Message
     * @param DOMElt $DOMEltDefinitions
     */
    private function createMessages(DOMElt $DOMEltDefinitions) {
        foreach ($this->getProcess()->getDefinitions()-
>getAllMessage() as $Message) {
            //Message
            $DOMEltMessage = $this->createElement('message',
array('name' => $Message->getName()));
            //Anexa message ao definitions
            $DOMEltDefinitions->appendChild($DOMEltMessage);
            //Part
            $options = array(
                'name' => $Message->getPart()->getName(),
                'element' => $Message->getPart()->getElement()-
>getQName(),
            );
            $DOMEltPart = $this->createElement('part', $options);
            //Anexa part ao Message
            $DOMEltMessage->appendChild($DOMEltPart);
        }
    }

    /**
     * Cria e anexa ao elemento definitions a seção types no
     documento wsdl
     * com todos os seus subelementos de acordo com os elements
     definidos
     * no processo
     * @param DOMElt $DOMEltDefinitions
     */
    private function createTypes(DOMElt $DOMEltDefinitions) {
        //Types
        $DOMEltTypes = $this->createElement('types', array());
        //Schema
        $options = array(
            'xmlns' => 'http://www.w3.org/2001/XMLSchema',
            'attributeFormDefault' => 'unqualified',
            'elementFormDefault' => 'qualified',
            'targetNamespace' => $this->getProcess()-
>getTargetNamespace(),
        );
        $DOMEltSchema = $this->createElement('schema', $options);
        $DOMEltTypes->appendChild($DOMEltSchema);
        //Elements
        foreach ($this->getProcess()->getDefinitions()-
>getAllMessage() as $Message) {
            //Cria o element
            $Element = $Message->getPart()->getElement();
            $DOMEltElement = $this->createElement('element',
array('name' => $Element->getName()));
            //Cria o complexType

```

```

        $DOMEltComplexType = $this-
>createElement('complexType');
        $DOMEltElement->appendChild($DOMEltComplexType);
        //Cria o sequence
        $DOMEltSequence = $this->createElement('sequence');
        $DOMEltComplexType->appendChild($DOMEltSequence);
        //Cria os elements que compoem o sequence
        foreach ($Element->getAllElements() as $SubElement) {
            $options = array(
                'name' => $SubElement->getName(),
                'type' => $SubElement->getType(),
            );
            $DOMEltSubElement = $this-
>createElement('element', $options);
            $DOMEltSequence->appendChild($DOMEltSubElement);
        }
        //Anexa o element ao schema
        $DOMEltSchema->appendChild($DOMEltElement);
    }
    //Anexa types em definitions
    $DOMEltDefinitions->appendChild($DOMEltTypes);
}

/**
 * Cria e anexa ao elemento definitions os elementos import
 * @param DOMElement $DOMEltDefinitions Definitions
 */
private function createImportWsdL(DOMElement
$DOMEltDefinitions) {
    $imports = $this->getProcess()->getDefinitions()-
>getAllImportWsdL();
    foreach ($imports as $import) {
        $options = array(
            'location' => $import->getLocation(),
            'namespace' => $import->getNamespace(),
        );
        $DOMEltDefinitions->appendChild($this-
>createElement('import', $options));
    }
}

/**
 * Cria e anexa ao elemento definitions os elementos
partnerLinkTypes referente
 * aos serviços invocados por este processo
 * @param DOMElement $DOMEltDefinitions Definitions
 */
private function createPartnerLinkTypes(DOMElement
$DOMEltDefinitions) {
    $partnerLinkTypes = $this->getProcess()->getDefinitions()-
>getAllPartnerLinkType();
    foreach ($partnerLinkTypes as $partnerLinkType) {
        //Elemento partnerLinkType
        $options = array(
            'name' => $partnerLinkType->getName(),
        );
        $DOMEltPLT = $this-
>createElement('plnk:partnerLinkType', $options);
        //Elemento partnerLinkType > Role
        $Role = $partnerLinkType->getRole();
        $options = array(

```

```

        'name' => $Role->getName(),
        'portType' => $Role->getPortType()->getQName(),
    );
    $DOMEltRole = $this->createElement('plnk:role',
$options);
    $DOMEltPLT->appendChild($DOMEltRole);
    //Anexa em definitions
    $DOMEltDefinitions->appendChild($DOMEltPLT);
}
}

/**
 * Cria o elemento definitions com seus atributos
 * @return DOMElement Definitions
 */
private function createDefinitions() {
    $Definitions = $this->getProcess()->getDefinitions();
    $options = array(
        'name' => $Definitions->getName(),
        'targetNamespace' => $Definitions-
>getTargetNamespace(),
    );
    //Declaração de namespaces
    foreach ($Definitions->getAllXmlNamespace() as
$xmlNamespace) {
        $prefix = is_null($xmlNamespace->getPrefix()) ?
'xmlns' : sprintf('xmlns:%s', $xmlNamespace->getPrefix());
        $options[$prefix] = $xmlNamespace->getUri();
    }
    return $this->createElement('definitions', $options);
}
}

?>

```

**Executor\Controller\Component\WsdReaderComponent.php**

```

<?php

    /**
     * O WsdReaderComponent faz a leitura e gravação em disco de um
    arquivo
     * WSDL identificado por sua URL.
     *
     * @author Augusto
     * @version 1.0
     * @created 03-out-2013 14:03:30
     */
    class WsdReaderComponent extends Component {

        /**
         * Lista de componentes associados
         */
        public $components = array(
            'Executor.HttpClient'
        );

        /**
         * Gera o nome baseado no WSDL, para definição dos nomes dos
    PartnerLinks,
         * PartnerLinkTypes e Roles
         * @param string $url_wsdL URL do WSDL
         * @param string $suffix Sufixo para ser colocado no nome
         * @return string Nome para o partner link
         */
        public function createNameByWsdL($url_wsdL, $suffix = null) {
            $pathinfo = pathinfo($url_wsdL);
            return ucfirst(preg_replace('/\?.*$/',' ',
    $pathinfo['filename'])) . $suffix;
        }

        /**
         * Gera o nome de arquivo baseado na url de um WSDL
         * @param string $url
         * @return string Nome do arquivo
         */
        public function filename($url, $extension = 'wsdl') {
            return Inflector::slug($url) . '.' . $extension;
        }

        /**
         * Faz a leitura de um arquivo remoto, através de sua url.
    Primeiro o arquivo
         * é verificado em cache e caso não exista é feita uma
    requisição HTTP, via
         * HttpClientComponent para obter o conteúdo do arquivo e ser
    gravado em disco.
         * @param string $url URL do arquivo remoto
         * @param string $type Tipo do arquivo a ser lido (wsdl ou
    xsd), também é utilizado como extensão do arquivo
         * @return string Conteúdo do WSDL
         */
        private function read($url, $type) {
            $filename = $this->filename($url, $type);
            $content = Cache::read($filename, $type);
            if ($content === false) {
                $content = $this->HttpClient->get($url);
            }
        }
    }

```

```

        Cache::write($filename, $content, $type);
    }
    return $content;
}

/**
 * Le um arquivo WSDL remoto identificado por sua URL
 * @param string $url URL do arquivo WSDL
 * @return string Conteúdo do arquivo WSDL
 */
public function readWsd($url) {
    return $this->read($url, 'wsdl');
}

/**
 * Le um arquivo XSD remoto identificado por sua URL
 * @param string $url URL do arquivo XSD
 * @return string Conteúdo do arquivo XSD
 */
public function readXsd($url) {
    return $this->read($url, 'xsd');
}

/**
 * Abre o document xml e le o atributo targetNamespace do tag
definitions
 * @param string $url_wsd
 * @return string targetNamespace definido no documento WSDL
 */
public function getTargetnamespace($url_wsd) {
    $wsdl = Xml::build($this->readWsd($url_wsd),
array('return' => 'domdocument'));
    $xpath = new DOMXPath($wsdl);
    $entries = $xpath->query('//*[local-
name()="definitions"]/@targetNamespace');
    return $entries->item(0)->nodeValue;
}

/**
 * Recupera o nome do serviço em um documento wsdl
 * @param string $url_wsd
 * @return string
 */
public function getService($url_wsd) {
    $wsdl = Xml::build($this->readWsd($url_wsd),
array('return' => 'domdocument'));
    $xpath = new DOMXPath($wsdl);
    $entries = $xpath->query('//*[local-
name()="service"]/@name');
    return $entries->item(0)->nodeValue;
}

/**
 * Recupera o nome do port que possui o elemento address
associado ao namespace soap
 * @param string $url_wsd
 * @return string
 */
public function getPort($url_wsd) {
    $wsdl = Xml::build($this->readWsd($url_wsd),
array('return' => 'domdocument'));

```

```

        $xpath = new DOMXPath($wsdl);
        $expression = '//*[local-name()="address" and namespace-
uri()="http://schemas.xmlsoap.org/wsdl/soap/"]/..';
        $entries = $xpath->query($expression);
        return $entries->item(0)->attributes-
>getNamedItem('name')->nodeValue;
    }

    /**
     * Analisa o WSDL localizado em $url_wsdl e retorna o nome do
portType
     * onde esta definida a operação denominada $operation
     * @param string $operation Nome da operação
     * @param string $url_wsdl URL do WSDL
     * @return string Nome do portType onde está definida a
operação informada
     * @todo refatorar método utilizando XPath
     */
    public function getPortTypeByOperation($operation, $url_wsdl)
    {
        $wsdl = Xml::build($this->readWsdl($url_wsdl),
array('return' => 'domdocument'));
        //Inicializa o retorno
        $portType_name = null;
        //Procura pelo elemento operation no namespace soap
        $listOperation = $wsdl-
>getElementsByTagNameNS('http://schemas.xmlsoap.org/wsdl/soap/',
'operation');
        for ($i = 0; $i < $listOperation->length; $i++) {
            $soapOperation = $listOperation->item($i);
            //Pega o type do binding que contém este
soap:operation
            $binding_type = $soapOperation->parentNode-
>parentNode->attributes->getNamedItem('type')->nodeValue;
            //Remove o prefixo do namespace do valor do node
            $binding_type = $this-
>removeNamespacePrefix($binding_type);
            //Procura o portType que possua um operation filho com
o nome da operação informada
            $listPortType = $wsdl->getElementsByTagNameNS('*',
'portType');
            for ($j = 0; $j < $listPortType->length; $j++) {
                $portType = $listPortType->item($j);
                //Se valor do atributo name do portType = Type do
Binding então encontrou
                if ($portType->attributes->getNamedItem('name')-
>nodeValue == $binding_type) {
                    //Pegar o atributo name do PortType
                    $portType_name = $binding_type;
                    break;
                }
            }
        }
        return $portType_name;
    }

    /**
     * Retorna o message type de entrada de uma operação
     * @param string $operation_name Nome da operação
     * @param string $url_wsdl URL do WSDL
     * @return string

```

```

    */
    public function getOperationInputMessageType($operation_name,
$url_wsdl) {
        return $this->getOperationMessageType($operation_name,
$url_wsdl, 'input');
    }

    /**
     * Retorna o message type de saída de uma operação
     * @param string $operation_name Nome da operação
     * @param string $url_wsdl URL do WSDL
     * @return string
     */
    public function getOperationOutputMessageType($operation_name,
$url_wsdl) {
        return $this->getOperationMessageType($operation_name,
$url_wsdl, 'output');
    }

    /**
     * Retorna o message type de entrada/saída de uma operação
     * @param string $operation_name Nome da operação
     * @param string $url_wsdl URL do WSDL
     * @param string $type Tipo de mensagem entrada ou saída
     (input ou output)
     * @return string
     */
    private function getOperationMessageType($operation_name,
$url_wsdl, $type) {
        $wsdl = Xml::build($this->readWsdl($url_wsdl),
array('return' => 'domdocument'));
        //Nome do portType
        $portType_name = $this->getSoapPortTypeName($wsdl);
        //Busca o elemento operation com o nome informado
        $xpath = new DOMXPath($wsdl);
        $expression = '//*[local-name()="portType" and @name="' .
$portType_name . '"]/*[local-name()="operation" and @name="' .
$operation_name . '"]/*[local-name()="' . $type . '"]';
        $entries = $xpath->query($expression);
        return ($entries->length > 0) ? $this-
>removeNamespacePrefix($entries->item(0)->attributes-
>getNamedItem('message')->nodeValue) : null;
    }

    /**
     * Retorna o nome do portType associado ao elemento binding no
namespace soap
     * @param DOMDocument $wsdl
     */
    private function getSoapPortTypeName(DOMDocument $wsdl) {
        $xpath = new DOMXPath($wsdl);
        //Busca o element binding no namespace wsdl que possui um
element binding no namespace soap
        $expression = '//*[local-name()="binding" and namespace-
uri()="http://schemas.xmlsoap.org/wsdl/"]/*[local-name()="binding" and
namespace-uri()="http://schemas.xmlsoap.org/wsdl/soap"/]';
        $entries = $xpath->query($expression);
        //Extrai o atributo type, que faz referencia ao portType
associado a este binding
        return $this->removeNamespacePrefix($entries->item(0)-
>attributes->getNamedItem('type')->nodeValue);
    }

```

```

}

/**
 * Remove o prefixo do namespace de uma string
 * @param string $string
 */
private function removeNamespacePrefix($string) {
    return preg_replace('/^(.+:)/', '', $string);
}

/**
 * Encontrar os parametros para um elemento message
 * @param string $operation Nome da operação
 * @param string $url_wsdl URL do documento WSDL onde a
operação está declarada
 * @return array Array com a definição dos parametros para a
mensagem informada
 */
public function getOperationMessageTypeParameters($operation,
$url_wsdl) {
    //Inicializa o retorno
    $parameters = null;
    //Cria o document wsdl
    $wsdl = Xml::build($this->readWsdl($url_wsdl),
array('return' => 'domdocument'));
    $xpath = new DOMXPath($wsdl);
    //Busca o nome do message type para a operação informada
    $inputMessage_name = $this->
>getOperationInputMessageType($operation, $url_wsdl);
    //Procura o elemento part contido no elemento message com
o nome $inputMessageType e le o atributo element
    $expression = '//*[local-name()="message" and @name="' .
$inputMessage_name . '"]/*[local-name()="part"]';
    $entries = $xpath->query($expression);
    if ($entries->length === 1) {
        $Element_name = $this->removeNamespacePrefix($entries->
>item(0)->attributes->getNamedItem('element')->nodeValue);
        //Procura o elemento message com o nome encontrado no
documento WSDL
        $expression = '//*[local-name()="element" and @name="'
. $Element_name . '"]';
        $entries = $xpath->query($expression);
        if ($entries->length === 1) {
            //element está definido no documento WSDL
            $parameters = $this->
>_getOperationMessageTypeParameters($wsdl, $Element_name);
        } else {
            //element está definindo no arquivo XSD
            //Procura o import com o atributo schemaLocation
            $expression = '//*[local-name()="import" and
@schemaLocation]';
            $entries = $xpath->query($expression);
            if ($entries->length === 1) {
                $schemaLocation = $entries->item(0)-
>attributes->getNamedItem('schemaLocation')->nodeValue;
                $xsd = Xml::build($this->
>readXsd($schemaLocation), array('return' => 'domdocument'));
                $parameters = $this->
>_getOperationMessageTypeParameters($xsd, $Element_name);
            }
        }
    }
}

```



```

    }
    return $parameters;
}

/**
 * Monta o array associativo com parametro => tipo no
 documento informado (WSDL ou XSD)
 * e o nome do elemento definido por $Element name
 * @param DOMDocument $xml
 * @param string $Element_name
 * @return array Array associativo de parametro => tipo
 */
private function
_getOperationMessageTypeParameters(DOMDocument $xml, $Element_name) {
    $xpath = new DOMXPath($xml);
    //Verifica se o element informado possui o atributo type
    $expression = '//*[local-name()="element" and @name="' .
$Element_name . '"' and @type]';
    $entries = $xpath->query($expression);
    if ($entries->length === 1) {
        //Possui o atributo type, então le o valor deste e
 procura o elemento complexType com este nome
        $complexType_name = $this-
>removeNamespacePrefix($entries->item(0)->attributes-
>getNamedItem('type')->nodeValue);
        //Le o elemento complexType com o nome descoberto
        $expression = '//*[local-name()="complexType" and
@name="' . $complexType_name . '"]/*[local-name()="sequence"]/*[local-
name()="element"]';
    } else {
        //Não possui o atributo type, então o elemento
 complexType está contido no proprio element
        $expression = '//*[local-name()="element" and @name="'
 . $Element_name . '"]/*[local-name()="complexType"]/*[local-
name()="sequence"]/*[local-name()="element"]';
    }
    //Faz a query no documento
    $entries = $xpath->query($expression);
    //Extrai os parametros
    $parameters = array();
    for ($i = 0; $i < $entries->length; $i++) {
        $name = $entries->item($i)->attributes-
>getNamedItem('name')->nodeValue;
        $type = $this->removeNamespacePrefix($entries-
>item($i)->attributes->getNamedItem('type')->nodeValue);
        $parameters[$name] = $type;
    }
    return $parameters;
}

}

?>

```

Executor\Lib\Bpel\Import.php

```
<?php
```

```

/**
 * Description of Import
 * @author Augusto
 * @version 1.0
 * @updated 08-out-2013 08:52:07
 */
class Import {

    /**
     * Nome do arquivo local onde está armazenado o wsdl
     * @var string
     */
    private $location;

    /**
     * Import type
     * @var string
     */
    private $importType;

    /**
     * Namespace
     * @var string
     */
    private $namespace;

    /**
     * Cria um novo import definido url, importType e namespace
     * @param string $url
     * @param string $importType
     * @param string $namespace
     */
    public function __construct($location, $importType,
$namespace) {
        $this->location = $location;
        $this->importType = $importType;
        $this->namespace = $namespace;
    }

    /**
     * Define o atributo location
     * @param string $location
     */
    public function setLocation($location) {
        $this->location = $location;
    }

    /**
     * Le o atributo location
     * @return string
     */
    public function getLocation() {
        return $this->location;
    }

    /**
     * Define o atributo importType
     * @param string $importType

```

```
    */
    public function setImportType($importType) {
        $this->importType = $importType;
    }

    /**
     * Le o atributo importType
     * @return string
     */
    public function getImportType() {
        return $this->importType;
    }

    /**
     * Define o atributo namespace
     * @param string $namespace
     */
    public function setNamespace($namespace) {
        $this->namespace = $namespace;
    }

    /**
     * Le o atributo namespace
     * @return string
     */
    public function getNamespace() {
        return $this->namespace;
    }
}

```

?>

Executor\Lib\Bpel\Invoke.php

```
<?php
```

```

/**
 * @author Augusto
 * @version 1.0
 * @created 16-out-2013 15:52:11
 */
class Invoke {

    /**
     * Nome
     * @var string
     */
    private $name;

    /**
     * @var PartnerLink
     */
    private $m_PartnerLink;

    /**
     * Variavel de entrada
     * @var Variable
     */
    private $m_InputVariable;

    /**
     * Variavel de saída
     * @var Variable
     */
    private $m_OutputVariable;

    function __construct($name, PartnerLink $PartnerLink, Variable
    $InputVariable = null, Variable $OutputVariable = null) {
        $this->name = $name;
        $this->m_PartnerLink = $PartnerLink;
        $this->m_InputVariable = $InputVariable;
        $this->m_OutputVariable = $OutputVariable;
    }

    /**
     * Define o atributo name
     * @param name Nome
     */
    public function setName(string $name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * Define o atributo PartnerLink
     *
     * @param PartnerLink

```

```

    */
    public function setPartnerLink(PartnerLink $PartnerLink) {
        $this->m_PartnerLink = $PartnerLink;
    }

    /**
     * Le o atributo PartnerLink
     * @return PartnerLink
     */
    public function getPartnerLink() {
        return $this->m_PartnerLink;
    }

    /**
     * Define o atributo m_InputVariable
     *
     * @param Variable $Variable
     */
    public function setInputVariable(Variable $Variable) {
        $this->m_InputVariable = $Variable;
    }

    /**
     * Le o atributo m_InputVariable
     */
    public function getInputVariable() {
        return $this->m_InputVariable;
    }

    /**
     * Define o atributo m_OutputVariable
     *
     * @param Variable
     */
    public function setOutputVariable(Variable $Variable) {
        $this->m_OutputVariable = $Variable;
    }

    /**
     * Le o atributo m_OutputVariable
     */
    public function getOutputVariable() {
        return $this->m_OutputVariable;
    }
}

```

?>

**Executor\Lib\Bpel\PartnerLink.php**

```

<?php

/**
 * Description of PartnerLink
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:46
 */
class PartnerLink {

    /**
     * Name
     * @var string
     */
    private $name;

    /**
     * Role tipo (myRole ou partnerRole)
     * @var string
     */
    private $roleType;

    /**
     * @var PartnerLinkType
     */
    private $m_PartnerLinkType;

    /**
     * URL do documento WSDL
     * @var string
     */
    private $url_wsdl;

    /**
     * Cria um objeto PartnerLink definindo o atributo name
     * @param string $name Nome do PartnerLink
     * @param string $roleType Tipo de Role (myRole ou
partnerRole)
     * @param PartnerLinkType $PartnerLinkType
     */
    public function __construct($name, $roleType =
RoleType::PARTNER, PartnerLinkType $PartnerLinkType = null) {
        $this->name = $name;
        $this->roleType = $roleType;
        $this->m_PartnerLinkType = $PartnerLinkType;
    }

    /**
     * Define o atributo name
     * @param string $name
     */
    public function setName($name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {

```

```

        return $this->name;
    }

    /**
     * Define o atributo roleType
     * @param string $roleType
     */
    public function setRoleType($roleType) {
        $this->roleType = $roleType;
    }

    /**
     * Le o atributo roleType
     * @return string
     */
    public function getRoleType() {
        return $this->roleType;
    }

    /**
     * Define o atributo partnerLinkType
     * @param PartnerLinkType $PartnerLinkType
     */
    public function setPartnerLinkType($PartnerLinkType) {
        $this->m_PartnerLinkType = $PartnerLinkType;
    }

    /**
     * Le o atributo partnerLinkType
     * @return PartnerLinkType
     */
    public function getPartnerLinkType() {
        return $this->m_PartnerLinkType;
    }

    /**
     * Le o atributo url_wsdl
     * @return string
     */
    public function getUrlWsdl() {
        return $this->url_wsdl;
    }

    /**
     * Define o atributo url_wsdl
     * @param string $url_wsdl
     */
    public function setUrlWsdl($url_wsdl) {
        $this->url_wsdl = $url_wsdl;
    }
}

```

?>

Executor\Lib\Bpel\PartnerLinkType.php

```
<?php
```

```

/**
 * PartnerLinkType
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:46
 */
class PartnerLinkType {

    /**
     * Name
     * @var string
     */
    private $name;

    /**
     * @var Role
     */
    private $m_Role;

    /**
     * Cria um novo PartnerLinkType definindo name e role
     * @param string $name
     * @param Role $Role
     */
    public function __construct($name, Role $Role) {
        $this->name = $name;
        $this->m_Role = $Role;
    }

    /**
     * Define o atributo name
     * @param string $name
     */
    public function setName($name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * Retorna o nome qualificado (<prefixo namespace>:<nome
element>)
     * @return string Nome qualificado
     * @todo definir o namespace para o partnerlinktype
     */
    public function getQName() {
        return sprintf('%s:%s', 'tns', $this->getName());
    }

    /**
     * Define o atributo role
     * @param Role $Role

```



```
    */  
    public function setRole($Role) {  
        $this->m_Role = $Role;  
    }  
  
    /**  
     * Le o atributo role  
     * @return Role  
     */  
    public function getRole() {  
        return $this->m_Role;  
    }  
}  
  
?>
```

**Executor\Lib\Bpel\Process.php**

&lt;?php

```

App::uses('PartnerLink', 'Executor.Lib/Bpel');
App::uses('PartnerLinkType', 'Executor.Lib/Bpel');
App::uses('Role', 'Executor.Lib/Bpel');
App::uses('Import', 'Executor.Lib/Bpel');
App::uses('RoleType', 'Executor.Lib/Bpel');
App::uses('PortType', 'Executor.Lib/Bpel');
App::uses('Variable', 'Executor.Lib/Bpel');
App::uses('Sequence', 'Executor.Lib/Bpel');
App::uses('Invoke', 'Executor.Lib/Bpel');
App::uses('Definitions', 'Executor.Lib/WSDL');
App::uses('PortType', 'Executor.Lib/WSDL');
App::uses('Message', 'Executor.Lib/WSDL');
App::uses('Part', 'Executor.Lib/WSDL');
App::uses('Element', 'Executor.Lib/WSDL');
App::uses('ElementInput', 'Executor.Lib/WSDL');
App::uses('ElementOutput', 'Executor.Lib/WSDL');
App::uses('Operation', 'Executor.Lib/WSDL');
App::uses('ImportWSDL', 'Executor.Lib/WSDL');
App::uses('Binding', 'Executor.Lib/WSDL');
App::uses('Service', 'Executor.Lib/WSDL');
App::uses('Address', 'Executor.Lib/WSDL');
App::uses('Port', 'Executor.Lib/WSDL');
App::uses('XmlNamespace', 'Executor.Lib/Xml');

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:46
 */
class Process {

    /**
     * @var string
     */
    private $name;

    /**
     * @var string
     */
    private $targetNamespace;

    /**
     * @var string
     */
    private $suppressJoinFailure;

    /**
     * @var Definitions
     */
    private $m_Definitions;

    /**
     * Lista de XmlNamespace
     * @var array<XmlNamespace>
     */
    private $arXmlNamespace = array();

    /**

```

```

* Lista de PartnerLink
* @var array<PartnerLink>
*/
private $arPartnerLink = array();

/**
* Lista de Import
* @var array<Import>
*/
private $arImport = array();

/**
* Lista de Variable
* @var array<Variable>
*/
private $arVariable = array();

/**
* Lista de Sequence
* @var array<Sequence>
*/
private $arSequence = array();

/**
* Cria um novo Process definido nome e targetNamespace
* @param string $name
* @param string $targetNamespace
* @param string $suppressJoinFailure
*/
public function __construct($name, $targetNamespace,
$suppressJoinFailure) {
    $this->name = $name;
    $this->targetNamespace = $targetNamespace;
    $this->suppressJoinFailure = $suppressJoinFailure;
    $this->addXmlNamespace(new XmlNamespace('bpel',
'http://docs.oasis-open.org/wsbpel/2.0/process/executable'));
    $this->m_Definitions = new Definitions($name,
$targetNamespace);
}

/**
* Definir atributo name
* @param string $name
*/
public function setName($name) {
    $this->name = $name;
}

/**
* Ler atributo name
* @return string
*/
public function getName() {
    return $this->name;
}

/**
* Retorna o Qualified Name do Process se estiver definido o
Namespace
* para tal, caso contrário retorna o próprio nome do Element
não qualificado

```

```

    * @return string
    */
    public function getQNameDeploy() {
        return sprintf('%s:%s', $this->getNamespacePrefixDeploy(),
$this->getName());
    }

    public function getNamespacePrefixDeploy() {
        $url = parse_url($this->targetNamespace);
        return $url['host'];
    }

    /**
     * Define o atributo supress join failure
     * @param string $suppressJoinFailure
     */
    public function setSuppressJoinFailure($suppressJoinFailure) {
        $this->suppressJoinFailure = $suppressJoinFailure;
    }

    /**
     * Le o atributo supress join failure
     * @return string
     */
    public function getSuppressJoinFailure() {
        return $this->suppressJoinFailure;
    }

    /**
     * Definir atributo targetNamespace
     * @param string $targetNamespace
     */
    public function setTargetNamespace($targetNamespace) {
        $this->targetNamespace = $targetNamespace;
    }

    /**
     * Ler atributo targetNamespace
     * @return string
     */
    public function getTargetNamespace() {
        return $this->targetNamespace;
    }

    /**
     * Incluir um namespace na lista de namespaces do processo
     * @param XmlNamespace $XmlNamespace
     */
    public function addXmlNamespace(XmlNamespace $XmlNamespace) {
        if (!in_array($XmlNamespace, $this->arXmlNamespace)) {
            $this->arXmlNamespace[] = $XmlNamespace;
        }
    }

    /**
     * Ler os namespaces deste processo
     * @return array<XmlNamespace>
     */
    public function getAllXmlNamespace() {
        return array_values($this->arXmlNamespace);
    }
}

```

```

/**
 * Busca um XmlNamespace pela uri
 * @param string $uri URI do XmlNamespace
 * @return XmlNamespace Objeto da classe XmlNamespace com a
uri
 * informada ou null caso contrário
 */
public function getXmlNamespaceByUri($uri) {
    $XmlNamespace = null;
    if (!empty($this->arXmlNamespace)) {
        foreach ($this->arXmlNamespace as $_XmlNamespace) {
            if ($_XmlNamespace->getUri() == $uri) {
                $XmlNamespace = $_XmlNamespace;
                break;
            }
        }
    }
    return $XmlNamespace;
}

/**
 * Incluir um partnerLink na lista de partnerLinks do processo
 * @param PartnerLink $partnerLink
 */
public function addPartnerLink(PartnerLink $PartnerLink) {
    $exists = false;
    foreach ($this->arPartnerLink as $_PartnerLink) {
        if ($PartnerLink->getName() == $_PartnerLink-
>getName()) {
            $exists = true;
            break;
        }
    }
    if (!$exists) {
        $this->arPartnerLink[] = $PartnerLink;
    }
}

/**
 * Retorna a lista de partner links
 * @return array<PartnerLink>
 */
public function getAllPartnerLink() {
    return $this->arPartnerLink;
}

/**
 * Procura um PartnerLink pelo nome
 * @param string $name Nome do PartnerLink
 * @return PartnerLink O PartnerLink cujo atributo name
igual ao informado ou null caso não exista
 */
public function getPartnerLinkByName($name) {
    $PartnerLink = null;
    foreach ($this->arPartnerLink as $_PartnerLink) {
        if ($_PartnerLink->getName() == $name) {
            $PartnerLink = $_PartnerLink;
            break;
        }
    }
}

```

```

        return $PartnerLink;
    }

    /**
     * Procura um PartnerLink pelo tipo
     * @param string $type Nome do PartnerLink
     * @return PartnerLink O PartnerLink cujo atributo name
    igual ao informado ou null caso n?o exista
     */
    public function getAllPartnerLinkByType($type) {
        $PartnerLinks = array();
        foreach ($this->arPartnerLink as $_PartnerLink) {
            if ($_PartnerLink->getRoleType() == $type) {
                $PartnerLinks[] = $_PartnerLink;
            }
        }
        return $PartnerLinks;
    }

    /**
     * Incluir um import na lista de imports do processo
     * @param Import $Import
     */
    public function addImport(Import $Import) {
        if (!in_array($Import, $this->arImport)) {
            $this->arImport[] = $Import;
        }
    }

    /**
     * Retorna a lista de imports
     * @return array<Import>
     */
    public function getAllImport() {
        return array_values($this->arImport);
    }

    /**
     * Define o atributo definitions
     * @para Definitions $Definitions
     */
    public function setDefinitions($Definitions) {
        $this->m_Definitions = $Definitions;
    }

    /**
     * Le o atributo definitions
     * @return Definitions
     */
    public function getDefinitions() {
        return $this->m_Definitions;
    }

    /**
     * Incluir uma vari?vel (objeto da classe Variable) na lista
    de vari?veis do
     * processo
     * @param Variable $Variable
     */
    public function addVariable(Variable $Variable) {
        $exists = false;

```

```

        foreach ($this->arVariable as $_Variable) {
            if ($Variable->getName() == $_Variable->getName()) {
                $exists = true;
                break;
            }
        }
        if (!$exists) {
            $this->arVariable[] = $Variable;
        }
    }

    /**
     * Retorna a lista de Variable
     * @return array<Variable>
     */
    public function getAllVariable() {
        return $this->arVariable;
    }

    /**
     * Busca um Variable pelo nome
     * @param string $name Nome da Variable
     * @return Variable Objeto da classe Variable com o nome ou
null
     * caso contrario
     */
    public function getVariableByName($name) {
        $Variable = null;
        if (!empty($this->arVariable)) {
            foreach ($this->arVariable as $_Variable) {
                if ($_Variable->getName() == $name) {
                    $Variable = $_Variable;
                    break;
                }
            }
        }
        return $Variable;
    }

    /**
     * Incluir uma sequencia (objeto da classe Sequence) na lista
de sequencias do
     * processo
     * @param Sequence $Sequence
     */
    public function addSequence(Sequence $Sequence) {
        if (!in_array($Sequence, $this->arSequence)) {
            $this->arSequence[] = $Sequence;
        }
    }

    /**
     * Retorna a lista de Sequence
     * @return array<Sequence>
     */
    public function getAllSequence() {
        return $this->arSequence;
    }
}

?>

```

## Executor\Lib\Bpel\Role.php

```

<?php

/**
 * Role
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:47
 */
class Role {

    /**
     * Name
     * @var string
     */
    private $name;

    /**
     * @var PortType
     */
    private $m_PortType;

    /**
     * Cria um novo Role definindo name e portType
     * @param string $name
     * @param PortType $PortType
     */
    public function __construct($name, PortType $PortType) {
        $this->name = $name;
        $this->m_PortType = $PortType;
    }

    /**
     * Define o atributo name
     * @param string $name
     */
    public function setName($name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * Define o atributo portType
     * @param string $PortType
     */
    public function setPortType($PortType) {
        $this->m_PortType = $PortType;
    }

    /**
     * Le o atributo portType
     * @return string
     */
}

```



```
public function getPortType () {  
    return $this->m_PortType;  
}  
  
?>
```

Executor\Lib\Bpel\RoleType.php

```
<?php

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 15:18:01
 */
abstract class RoleType {

    const MY = 'myRole';
    const PARTNER = 'partnerRole';

}

?>
```

Executor\Lib\Bpel\Import.php

```
<?php
```

```

/**
 * Representa um elemento sequence dentro de um processo bpel.
 * @author Augusto
 * @version 1.0
 * @created 16-out-2013 15:59:28
 */
class Sequence {

    /**
     * Nome da Sequence
     * @var string
     */
    private $name;

    /**
     * Lista de Invoke
     * @var array<Invoke>
     */
    private $arInvoke = array();

    /**
     * @var Variable
     */
    private $VariableSource = null;

    /**
     * @var Element
     */
    private $ElementSource = null;

    /**
     * @var Variable
     */
    private $VariableOutput = null;

    /**
     * @var Element
     */
    private $ElementOutput = null;

    function __construct($name) {
        $this->name = $name;
    }

    /**
     * Define o atributo name
     *
     * @param name    Nome da Sequence
     */
    public function setName(string $name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {

```

```

        return $this->name;
    }

    /**
     * Incluir um Invoke na lista de Invoke de Sequence
     *
     * @param Invoke
     */
    public function addInvoke(Invoke $Invoke) {
        if (!in_array($Invoke, $this->arInvoke)) {
            $this->arInvoke[] = $Invoke;
        }
    }

    /**
     * Retorna a lista de Invoke
     * @return array<Invoke>
     */
    public function getAllInvoke() {
        return $this->arInvoke;
    }

    /**
     * Le o atributo VariableSource
     * @return Variable
     */
    public function getVariableSource() {
        return $this->VariableSource;
    }

    /**
     * Define o atributo VariableSource
     * @param Variable $VariableSource
     */
    public function setVariableSource(Variable $VariableSource) {
        $this->VariableSource = $VariableSource;
    }

    /**
     * Define o atributo ElementSource
     * @return Element
     */
    public function getElementSource() {
        return $this->ElementSource;
    }

    /**
     * Le o atributo ElementSource
     * @param Element $ElementSource
     */
    public function setElementSource(Element $ElementSource) {
        $this->ElementSource = $ElementSource;
    }

    /**
     * Le o atributo VariableOutput
     * @return Variable
     */
    public function getVariableOutput() {
        return $this->VariableOutput;
    }
}

```

```
/**
 * Define o atributo VariableOutput
 * @param Variable $VariableOutput
 */
public function setVariableOutput(Variable $VariableOutput) {
    $this->VariableOutput = $VariableOutput;
}

/**
 * Le o atributo ElementOutput
 * @return Element
 */
public function getElementOutput() {
    return $this->ElementOutput;
}

/**
 * Define o atributo ElementOutput
 * @param Element $ElementOutput
 */
public function setElementOutput(Element $ElementOutput) {
    $this->ElementOutput = $ElementOutput;
}
}

?>
```

Executor\Lib\Bpel\Variable.php

```

<?php

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:47
 */
class Variable {

    /**
     * @var string
     */
    private $name;

    /**
     * @var Message
     */
    private $m_Message;

    /**
     * Cria um novo objeto da classe Variable
     * @param string $name Nome
     * @param Message $m_Message
     */
    public function __construct($name, Message $m_Message = null)
    {
        $this->name = $name;
        $this->m_Message = $m_Message;
    }

    /**
     * Define o atributo name
     * @param string $name
     */
    public function setName($name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * Define o atributo MessageType
     * @param Message $MessageType
     */
    public function setMessageType(Message $MessageType) {
        $this->m_Message = $MessageType;
    }

    /**
     * Le o atributo MessageType
     * @return Message
     */
    public function getMessageType() {
        return $this->m_Message; } }?>

```

Executor\Lib\Wsd1\Address.php

```
<?php
```

```
/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:45
 */
class Address {

    /**
     * URL do Servi 篠
     * @var string
     */
    private $location;

    /**
     * Cria um novo objeto Address
     * @param string $location
     */
    public function __construct($location) {
        $this->location = $location;
    }

    /**
     * Define o atributo location
     * @param string $location
     */
    public function setLocation($location) {
        $this->location = $location;
    }

    /**
     * Le o atributo location
     * @return string
     */
    public function getLocation() {
        return $this->location;
    }

}
```

```
?>
```

Executor\Lib\WsdL\Binding.php

```

<?php

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:45
 */
class Binding {

    /**
     * Name
     * @var string
     */
    private $name;

    /**
     * Referencia ao portType descrito como o atributo type do
element
     * binding no wsdl
     * @var PortType
     */
    private $m_PortType;

    /**
     * @var XmlNamespace
     */
    private $m_XmlNamespace;

    /**
     * Cria um novo objeto Binding
     * @param type $name
     * @param PortType $m_PortType
     * @param XmlNamespace $XmlNamespace
     */
    public function __construct($name, PortType $m_PortType,
XmlNamespace $XmlNamespace) {
        $this->name = $name;
        $this->m_PortType = $m_PortType;
        $this->m_XmlNamespace = $XmlNamespace;
    }

    /**
     * Define o atributo name
     * @param string name
     */
    public function setName($name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * Retorna o nome qualificado (<prefixo namespace>:<nome
element>)

```



```
    * @return string Nome qualificado
    */
    public function getQName() {
        return sprintf('%s:%s', $this->m_XmlNamespace-
>getPrefix(), $this->getName());
    }

    /**
     * Define o atributo PortType
     * @param PortType
     */
    public function setPortType($PortType) {
        $this->m_PortType = $PortType;
    }

    /**
     * Le o atributo PortType
     * @return PortType
     */
    public function getPortType() {
        return $this->m_PortType;
    }

    /**
     * Define o atributo XmlNamespace
     * @param XmlNamespace
     */
    public function setXmlNamespace($XmlNamespace) {
        $this->m_XmlNamespace = $XmlNamespace;
    }

    /**
     * Le o atributo XmlNamespace
     * @return XmlNamespace
     */
    public function getXmlNamespace() {
        return $this->m_XmlNamespace;
    }
}

?>
```

Executor\Lib\Wsd\Definitions.php

```
<?php
```

```

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:45
 */
class Definitions {

    /**
     * Name
     * @var string
     */
    private $name;

    /**
     * target namespace
     * @var string
     */
    private $targetNamespace;

    /**
     * @var array<Message>
     */
    private $arMessage = array();

    /**
     * @var array<Element>
     */
    private $arElement = array();

    /**
     * @var PortType
     */
    private $m_PortType;

    /**
     * @var Binding
     */
    private $m_Binding;

    /**
     * @var array<XmlNamespace>
     */
    private $arXmlNamespace = array();

    /**
     * @var array<ImportWsdL>
     */
    private $arImportWsdL = array();

    /**
     * @var Service
     */
    private $m_Service;

    /**
     * PartnerLinkType definido para o prºo processo
     * @var PartnerLinkType

```

```

    */
    private $m_PartnerLinkType;

    /**
     * Lista de PartnerLinkType para os PartnerLink definidos para
     os parceiros
     * @var array<PartnerLinkType>
     */
    private $arPartnerLinkType = array();

    /**
     * Cria um novo objeto Definitions definindo nome e fazendo a
     cria 磯 e associa 磯
     * aos namespace padr 磯
     * @param string $name
     * @param string $targetNamespace
     */
    public function __construct($name, $targetNamespace) {
        $this->name = $name;
        $this->targetNamespace = $targetNamespace;
        $this->addXmlNamespace(new XmlNamespace(null,
'http://schemas.xmlsoap.org/wsdl/'));
        $this->addXmlNamespace(new XmlNamespace('soap',
'http://schemas.xmlsoap.org/wsdl/soap/'));
        $this->addXmlNamespace(new XmlNamespace('vprop',
'http://docs.oasis-open.org/wsbpel/2.0/varprop/'));
        $this->addXmlNamespace(new XmlNamespace('plnk',
'http://docs.oasis-open.org/wsbpel/2.0/plnktype/'));
    }

    /**
     * Define o atributo name
     * @param string $name
     */
    public function setName($name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * Define o atributo targetNamespace
     * @param string $targetNamespace
     */
    public function setTargetNamespace($targetNamespace) {
        $this->targetNamespace = $targetNamespace;
    }

    /**
     * Le o atributo targetNamespace
     * @return string
     */
    public function getTargetNamespace() {
        return $this->targetNamespace;
    }
}

```

```

/**
 * Define o atributo Binding
 * @param Binding $Binding
 */
public function setBinding($Binding) {
    $this->m_Binding = $Binding;
}

/**
 * Le o atributo Binding
 * @return Binding
 */
public function getBinding() {
    return $this->m_Binding;
}

/**
 * Define o atributo PortType
 * @param PortType $PortType
 */
public function setPortType($PortType) {
    $this->m_PortType = $PortType;
}

/**
 * Le o atributo PortType
 * @return PortType
 */
public function getPortType() {
    return $this->m_PortType;
}

/**
 * Incluir um ImportWsdL na lista de ImportWsdL do processo
 * @param ImportWsdL $ImportWsdL
 */
public function addImportWsdL(ImportWsdL $ImportWsdL) {
    if (!in_array($ImportWsdL, $this->arImportWsdL)) {
        $this->arImportWsdL[] = $ImportWsdL;
    }
}

/**
 * Retorna a lista de ImportWsdL
 * @return array<ImportWsdL>
 */
public function getAllImportWsdL() {
    return array_values($this->arImportWsdL);
}

/**
 * Incluir um namespace na lista de namespaces do processo
 * @param XmlNamespace $XmlNamespace
 */
public function addXmlNamespace(XmlNamespace $XmlNamespace) {
    if (is_null($this->getXmlNamespaceByUri($XmlNamespace-
>getUri())) {
        $this->arXmlNamespace[] = $XmlNamespace;
    }
}

```

```

/**
 * Retorna todos os XmlNamespace
 * @return array<XmlNamespace>
 */
public function getAllXmlNamespace() {
    return array_values($this->arXmlNamespace);
}

/**
 * Busca um XmlNamespace pela uri
 * @param string $uri URI do XmlNamespace
 * @return XmlNamespace Objeto da classe XmlNamespace com a
uri
 * informada ou null caso contrário
 */
public function getXmlNamespaceByUri($uri) {
    $XmlNamespace = null;
    if (!empty($this->arXmlNamespace)) {
        foreach ($this->arXmlNamespace as $_XmlNamespace) {
            if ($_XmlNamespace->getUri() == $uri) {
                $XmlNamespace = $_XmlNamespace;
                break;
            }
        }
    }
    return $XmlNamespace;
}

/**
 * Inclui um elemento (objeto da classe Element) na lista de
elementos
 * @param Element $Element
 */
public function addElement($Element) {
    if (!in_array($Element, $this->arElement)) {
        $this->arElement[] = $Element;
    }
}

/**
 * Incluir uma mensagem (objeto da classe Message) na lista de
mensagens
 * @param Message $Message
 *
 * @param Message
 */
public function addMessage($Message) {
    if (!in_array($Message, $this->arMessage)) {
        $this->arMessage[] = $Message;
    }
}

/**
 * Retorna a lista de Message
 * @return array<Message>
 */
public function getAllMessage() {
    return $this->arMessage;
}

```

```

/**
 * Define o atributo Service
 * @param Service $Service
 */
public function setService($Service) {
    $this->m_Service = $Service;
}

/**
 * Le o atributo Service
 * @return Service
 */
public function getService() {
    return $this->m_Service;
}

/**
 * Le o atributo PartnerLinkType
 * @return PartnerLinkType
 */
public function getPartnerLinkType() {
    return $this->m_PartnerLinkType;
}

/**
 * Define o atributo PartnerLinkType
 * @param PartnerLinkType $PartnerLinkType
 */
public function setPartnerLinkType(PartnerLinkType
$PartnerLinkType) {
    $this->m_PartnerLinkType = $PartnerLinkType;
}

/**
 * Incluir um namespace na lista de namespaces do processo
 * @param PartnerLinkType $PartnerLinkType
 */
public function addPartnerLinkType(PartnerLinkType
$PartnerLinkType) {
    if (!in_array($PartnerLinkType, $this->arPartnerLinkType))
    {
        $this->arPartnerLinkType[] = $PartnerLinkType;
    }
}

/**
 * Retorna todos os arPartnerLinkType
 * @return array<arPartnerLinkType>
 */
public function getAllPartnerLinkType() {
    return array_values($this->arPartnerLinkType);
}
}

?>

```

Executor\Lib\Wsd\Element.php

```
<?php

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:45
 */
class Element {

    /**
     * Name
     * @var string
     */
    protected $name;

    /**
     * Type
     * @var string
     */
    protected $type;

    /**
     * @var XmlNamespace
     */
    protected $m_XmlNamespace;

    /**
     * Lista de Element
     * @var array<Element>
     */
    protected $arElement = array();

    /**
     * @var Variable
     */
    protected $VariableSource = null;

    /**
     * @var Element
     */
    protected $ElementSource = null;

    /**
     *
     * @param int $source_id
     * @param string $name
     * @param string $type
     * @param XmlNamespace $XmlNamespace
     */
    public function __construct($name, $type, XmlNamespace
$xmlNamespace = null) {
        $this->name = $name;
        $this->type = $type;
        $this->m_XmlNamespace = $xmlNamespace;
    }

    /**
     * Define o atributo name
     * @param string $name
     */
}
```

```

public function setName($name) {
    $this->name = $name;
}

/**
 * Le o atributo name
 * @return string
 */
public function getName() {
    return $this->name;
}

/**
 * Retorna o Qualified Name do Element se estiver definido o
Namespace
 * para tal, caso contrário retorna o próprio nome do Element
não qualificado
 * @return string
 */
public function getQName() {
    return is_null($this->m_XmlNamespace) ? $this->getName() :
sprintf('%s:%s', $this->m_XmlNamespace->getPrefix(), $this-
>getName());
}

/**
 * Define o atributo XmlNamespace
 * @param XmlNamespace
 */
public function setXmlNamespace($XmlNamespace) {
    $this->m_XmlNamespace = $XmlNamespace;
}

/**
 * Le o atributo XmlNamespace
 * @return XmlNamespace;
 */
public function getXmlNamespace() {
    return $this->m_XmlNamespace;
}

/**
 * Define o atributo type
 * @param string $type
 */
public function setType($type) {
    $this->type = $type;
}

/**
 * Le o atributo type
 * @return string
 */
public function getType() {
    return $this->type;
}

/**
 * Incluir um Element na lista de Elements
 *
 * @param Element

```



```

*/
public function addElement($Element) {
    if (!in_array($Element, $this->arElement)) {
        $this->arElement[] = $Element;
    }
}

/**
 * Retorna a lista de Elements que compõe este Element
 * @return array<Element>
 */
public function getAllElements() {
    return $this->arElement;
}

/**
 * Procura um Element pelo nome
 * @param string $name Nome do $Element
 * @return $Element O Element cujo atributo name é igual ao
informado ou null caso não exista
 */
public function getElementByName($name) {
    $Element = null;
    foreach ($this->arElement as $_Element) {
        if ($_Element->getName() == $name) {
            $Element = $_Element;
            break;
        }
    }
    return $Element;
}

/**
 * Le o atributo VariableSource
 * @return Variable
 */
public function getVariableSource() {
    return $this->VariableSource;
}

/**
 * Define o atributo VariableSource
 * @param Variable $VariableSource
 */
public function setVariableSource(Variable $VariableSource) {
    $this->VariableSource = $VariableSource;
}

/**
 * Le o atributo ElementSource
 * @return type
 */
public function getElementSource() {
    return $this->ElementSource;
}

/**
 * Define o atributo ElementSource
 * @param Element $ElementSource
 */
public function setElementSource(Element $ElementSource) {

```

```
        $this->ElementSource = $ElementSource;
    }
}
?>
```

Executor\Lib\Wsd1\ElementInput.php

```
<?php
```

```
    /**
     * Description of ElementInput
     *
     * @author Augusto
     */
    class ElementInput extends Element {
    }
```

```
?>
```

Executor\Lib\Wsd1\ElementOutput.php

```
<?php

    /**
     * Description of ElementOutput
     *
     * @author Augusto
     */
    class ElementOutput extends Element {

    }

?>
```

Executor\Lib\WsdL\ImportWsdL.php

<?php

```

/**
 * Representa um elemento import dentro de um documento WSDL
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 17:48:25
 */
class ImportWsdL {

    /**
     * Localiza 機 do arquivo WSDL
     * @var string
     */
    private $location;

    /**
     * Namespace definido no arquivo WSDL
     * @var string
     */
    private $namespace;

    /**
     * Cria um novo ImportWsdL definindo location e namespace
     * @param string $location
     * @param string $namespace
     */
    public function __construct($location, $namespace) {
        $this->location = $location;
        $this->namespace = $namespace;
    }

    /**
     * Le o atributo location
     * @return string
     */
    public function getLocation() {
        return $this->location;
    }

    /**
     * Define o atributo location
     * @param string $location
     */
    public function setLocation($location) {
        $this->location = $location;
    }

    /**
     * Le o atributo namespace
     * @return string
     */
    public function getNamespace() {
        return $this->namespace;
    }

    /**
     * Define o atributo namespace
     * @param string $namespace
     */
}

```

```
public function setNamespace($namespace) {  
    $this->namespace = $namespace;  
}  
  
}  
  
?>
```

Executor\Lib\Wsd\Message.php

```
<?php

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:46
 */
class Message {

    /**
     * Name
     * @var string
     */
    private $name;

    /**
     * @var XmlNamespace
     */
    private $m_XmlNamespace;

    /**
     * @var Part
     */
    private $m_Part;

    /**
     * Cria um novo objeto Message
     * @param type $name
     * @param Part $Part
     * @param XmlNamespace $XmlNamespace
     */
    public function __construct($name, Part $Part = null,
    XmlNamespace $XmlNamespace = null) {
        $this->name = $name;
        $this->m_Part = $Part;
        $this->m_XmlNamespace = $XmlNamespace;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * Retorna o nome qualificado (<prefixo namespace>:<nome
    element>)
     * @return string Nome qualificado
     */
    public function getQName() {
        return sprintf('%s:%s', $this->m_XmlNamespace-
    >getPrefix(), $this->getName());
    }

    /**
     * Define o atributo name
     * @param string $name
     */

```

```
public function setName($name) {
    $this->name = $name;
}

/**
 * Define o atributo Part
 * @param Part $Part
 */
public function setPart($Part) {
    $this->m_Part = $Part;
}

/**
 * Le o atributo Part
 */
public function getPart() {
    return $this->m_Part;
}

/**
 * Define o atributo XmlNamespace
 * @param XmlNamespace
 */
public function setXmlNamespace($XmlNamespace) {
    $this->m_XmlNamespace = $XmlNamespace;
}

/**
 * Le o atributo XmlNamespace
 */
public function getXmlNamespace() {
    return $this->m_XmlNamespace;
}
}
```

?>



Executor\Lib\Wsd1\Operation.php

```
<?php
```

```

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:46
 */
class Operation {

    /**
     * Name
     * @var string
     */
    private $name;

    /**
     * Input message
     * @var Message
     */
    private $m_InputMessage;

    /**
     * Output message
     * @var Message
     */
    private $m_OutputMessage;

    /**
     * Cria um novo Operation definindo name, InputMessage e
     OutputMessage.
     * @param string $name
     * @param Message $InputMessage
     * @param Message $OutputMessage
     */
    public function __construct($name, Message $InputMessage =
    null, Message $OutputMessage = null) {
        $this->name = $name;
        $this->m_InputMessage = $InputMessage;
        $this->m_OutputMessage = $OutputMessage;
    }

    /**
     * Define o atributo name
     * @param string $name
     */
    public function setName($name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * Define o atributo InputMessage
     * @param Message $InputMessage

```

```
 */
public function setInputMessage($InputMessage) {
    $this->m_InputMessage = $InputMessage;
}

/**
 * Le o parametro InputMessage
 * @return Message
 */
public function getInputMessage() {
    return $this->m_InputMessage;
}

/**
 * Define o atributo OutputMessage
 * @param Message $OutputMessage
 */
public function setOutputMessage($OutputMessage) {
    $this->m_OutputMessage = $OutputMessage;
}

/**
 * Le o atributo OutputMessage
 * @return Message
 */
public function getOutputMessage() {
    return $this->m_OutputMessage;
}
}

```

?>

Executor\Lib\Wsd1\Part.php

```
<?php
```

```

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:46
 */
class Part {

    /**
     * Name
     * @var string
     */
    var $name;

    /**
     * @var Element
     */
    var $m_Element;

    /**
     *
     * @param name $name
     * @param Element $Element
     */
    public function __construct($name, Element $Element = null) {
        $this->name = $name;
        $this->m_Element = $Element;
    }

    /**
     * Define o atributo name
     * @param string $name
     */
    function setName($name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    function getName() {
        return $this->name;
    }

    /**
     * Define o atributo Element
     * @param Part $Element
     *
     * @param Part
     */
    function setElement($Element) {
        $this->m_Element = $Element;
    }

    /**
     * Le o atributo Element
     * @return Element
     */

```

```
function getElement () {  
    return $this->m_Element;  
}  
  
}  
  
?>
```

Executor\Lib\Wsd1\Port.php

```
<?php
```

```

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:46
 */
class Port {

    /**
     * Nome deste Port
     * @var string
     */
    private $name;

    /**
     * @var Binding
     */
    private $m_Binding;

    /**
     *
     * @var Address
     */
    private $m_Address;

    /**
     * Cria um novo objeto Port
     * @param string $name
     * @param Binding $m_Binding
     * @param Address $m_Address
     */
    public function __construct($name, Binding $m_Binding = null,
Address $m_Address = null) {
        $this->name = $name;
        $this->m_Binding = $m_Binding;
        $this->m_Address = $m_Address;
    }

    /**
     * Define o atributo name
     * @param string $name
     */
    public function setName($name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * Define o atributo Binding
     * @param Binding $Binding
     */
    public function setBinding(Binding $Binding) {

```

```
        $this->m_Binding = $Binding;
    }

    /**
     * Le o atributo Binding
     * @return Binding
     */
    public function getBinding() {
        return $this->m_Binding;
    }

    /**
     * Define o atributo Address
     * @param Address $Address
     */
    public function setAddress(Address $Address) {
        $this->m_Address = $Address;
    }

    /**
     * Le o atributo Address
     * @return Address
     */
    public function getAddress() {
        return $this->m_Address;
    }
}

?>
```

Executor\Lib\Wsd\PortType.php

<?php

```

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:46
 */
class PortType {

    /**
     * Name
     * @var string
     */
    private $name;

    /**
     * @var Operation
     */
    private $m_Operation;

    /**
     * @var XmlNamespace
     */
    private $m_XmlNamespace;

    /**
     * Cria um novo PortType definindo name e Operation
     * @param string $name
     * @param Operation $Operation
     * @param XmlNamespace $XmlNamespace
     */
    public function __construct($name, Operation $Operation =
null, XmlNamespace $XmlNamespace = null) {
        $this->name = $name;
        $this->m_Operation = $Operation;
        $this->m_XmlNamespace = $XmlNamespace;
    }

    /**
     * Define o atributo name
     * @param string $name
     */
    public function setName($name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * Retorna o nome qualificado (<prefixo namespace>:<nome
element>)
     * @return string Nome qualificado
     */
    public function getQName() {

```

```
        return sprintf('%s:%s', $this->m_XmlNamespace-
>getPrefix(), $this->getName());
    }

    /**
     * Define o atributo XmlNamespace
     * @param XmlNamespace $XmlNamespace
     */
    public function setXmlNamespace($XmlNamespace) {
        $this->m_XmlNamespace = $XmlNamespace;
    }

    /**
     * Le o atributo XmlNamespace
     * @return XmlNamespace
     */
    public function getXmlNamespace() {
        return $this->m_XmlNamespace;
    }

    /**
     * Define o atributo Operation
     * @param Operation $Operation
     */
    public function setOperation($Operation) {
        $this->m_Operation = $Operation;
    }

    /**
     * Le o atributo Operation
     * @return Operation
     */
    public function getOperation() {
        return $this->m_Operation;
    }
}

?>
```



Executor\Lib\Wsd\Service.php

```
<?php
```

```

/**
 * @author Augusto
 * @version 1.0
 * @created 08-out-2013 07:24:47
 */
class Service {

    /**
     * Nome deste serviço
     * @var string
     */
    private $name;

    /**
     * @var Port
     */
    private $m_Port;

    /**
     * Cria um novo objeto Service
     * @param string $name Nome do Service
     * @param Port $m_Port Port associado
     */
    public function __construct($name, Port $m_Port) {
        $this->name = $name;
        $this->m_Port = $m_Port;
    }

    /**
     * Define o atributo name
     * @param string name
     */
    public function setName($name) {
        $this->name = $name;
    }

    /**
     * Le o atributo name
     * @return string
     */
    public function getName() {
        return $this->name;
    }

    /**
     * Define o atributo Port
     * @param Port $Port
     */
    public function setPort(Port $Port) {
        $this->m_Port = $Port;
    }

    /**
     * Le o atributo Port
     * @return Port
     */
    public function getPort() {
        return $this->m_Port;}}?>

```

Executor\Lib\Xml\XmlNamespace.php

```
<?php
```

```

    /**
     * Representa a declaração de um namespace em um documento Xml
     contendo prefixo e uri
     * @link http://www.w3schools.com/xml/xml_namespaces.asp
     Documentação sobre XML Namespaces da W3Schools
     * @author Augusto
     * @version 1.0
     * @created 08-out-2013 07:24:47
     */
    class XmlNamespace {

        /**
         * Prefix do namespace
         * @var string
         */
        private $prefix;

        /**
         * URI do namespace
         * @var string
         */
        private $uri;

        /**
         * Cria um novo XmlNamespace definindo prefix e uri
         * @param string $prefix
         * @param string $uri
         */
        public function __construct($prefix, $uri) {
            $this->setPrefix($prefix);
            $this->setUri($uri);
        }

        /**
         * Define o atributo prefix
         * @param string $prefix
         */
        public function setPrefix($prefix) {
            $this->prefix = $prefix;
        }

        /**
         * Le o atributo prefix
         * @return string
         */
        public function getPrefix() {
            return $this->prefix;
        }

        public function getPrefixDeploy() {
            $url = parse_url($this->getUri());
            return $url['host'];
        }

        /**
         * Define o atributo url
         * @param string $uri
         */
    }

```

```
public function setUri($uri) {
    $this->uri = $uri;
}

/**
 * Le o atributo uri
 * @return string
 */
public function getUri() {
    return $this->uri;
}
}

?>
```

## Executor\Model\Caminho.php

```

<?php

/**
 * Description of Caminho
 * @author Augusto Cesar Ferreira
 * @version 1.0
 * @updated 01-out-2013 14:01:24
 */
class Caminho extends ExecutorAppModel {

    /**
     * Nome da tabela
     * @var string
     */
    public $useTable = 'caminho';

    /**
     * Campos virtuais
     * @var array
     */
    public $virtualFields = array(
        'custo' => '`Caminho`.`valor_smd` +
`Caminho`.`profundidade`, //CUSTO(P) = SMD(P) + PROFUNDIDADE(P)
    );

    /**
     * @var array
     */
    public $belongsTo = array(
        'Composicao' => array(
            'className' => 'Executor.Composicao',
            'foreignKey' => 'id_composicao',
        ),
    );

    /**
     * @var array
     */
    public $hasMany = array(
        'Operacao' => array(
            'className' => 'Executor.Operacao',
            'foreignKey' => 'id_caminho',
        ),
    );

    /**
     * @var array
     */
    public $hasOne = array(
        'Saida' => array(
            'className' => 'Executor.Saida',
            'foreignKey' => 'id_caminho',
        ),
    );
}

?>

```

Executor\Model\Composicao.php

<?php

```

/**
 * Description of Caminho
 * @author Augusto Cesar Ferreira
 * @version 1.0
 * @updated 01-out-2013 14:01:24
 */
class Composicao extends ExecutorAppModel {

    /**
     * Nome da tabela
     * @var string
     */
    public $useTable = 'composicao';

    /**
     * Métodos find personalizados
     * @var array
     */
    public $findMethods = array(
        'full' => true
    );

    /**
     * Campos virtuais
     * @var array
     */
    public $virtualFields = array(
        'uuid' => 'CONCAT(\'Composicao\', LPAD(Composicao.id, 5,
0))',
    );

    /**
     * @var array
     */
    public $hasMany = array(
        'Entrada' => array(
            'className' => 'Executor.Entrada',
            'foreignKey' => 'id_composicao',
        ),
        'Caminho' => array(
            'className' => 'Executor.Caminho',
            'foreignKey' => 'id_composicao',
        ),
    );

    /**
     * Implementa o método find personalizado 'full'.
     * Retorna a primeira composição, com seus models associados,
     que satisfaz
     * as condições definidas.
     * @param type $state
     * @param string $query
     * @param type $results
     * @return string
     */
    public function _findFull($state, $query, $results = array())
{

```

```

        $method = sprintf('_findFull%s', ucfirst($state));
        return $this->{$method}($query, $results);
    }

    /**
     * Método que executa o estado 'before' da implementação do
find 'full'
     * @param array $query
     * @param type $results
     * @return string
     */
    private function _findFullBefore($query, $results) {
        $this->Behaviors->attach('Containable');
        $contain = array(
            'Entrada',
            'Caminho' => array(
                'Operacao' => array(
                    'order' => 'Operacao.ordem ASC',
                    'Entrada' => array(
                        'OrigemEntradaComposicao',
                        'OrigemSaidaOperacao',
                    ),
                    'Saida' => array(),
                ),
                'Saida' => array(
                    'OrigemSaida',
                ),
            ),
        );
        $query['contain'] = $contain;
        return $query;
    }

    /**
     * Método que executa o estado 'after' da implementação do
find 'full'
     * @param array $query
     * @param type $results
     * @return string
     */
    private function _findFullAfter($query, $results) {
        return empty($results[0]) ? array() : $results[0];
    }

    /**
     * Informa se a Composicao identificada por $id esta
implantada no
     * Orchestration Module
     * @param type $id
     * @return boolean
     */
    public function isDeployed($id) {
        return true;
    }
}
}

```

?>

**Executor\Model\Entrada.php****<?php**

```
/**
 * Description of Caminho
 *
 * @author Augusto Cesar Ferreira
 */
class Entrada extends ExecutorAppModel {

    /**
     * Nome da tabela
     * @var string
     */
    public $useTable = 'entrada';

    /**
     * @var array
     */
    public $belongsTo = array(
        'Composicao' => array(
            'className' => 'Executor.Composicao',
            'foreignKey' => 'id_composicao',
        ),
        'Operacao' => array(
            'className' => 'Executor.Operacao',
            'foreignKey' => 'id_operacao',
        ),
        'OrigemEntradaComposicao' => array(
            'className' => 'Executor.Entrada',
            'foreignKey' => 'id_entrada_origem',
        ),
        'OrigemSaidaOperacao' => array(
            'className' => 'Executor.Saida',
            'foreignKey' => 'id_saida_origem',
        ),
    );
}

?>
```

Executor\Model\Operacao.php

```
<?php

/**
 * Description of Caminho
 *
 * @author Augusto Cesar Ferreira
 */
class Operacao extends ExecutorAppModel {

    /**
     * Nome da tabela
     * @var string
     */
    public $useTable = 'operacao';

    /**
     * @var array
     */
    public $belongsTo = array(
        'Caminho' => array(
            'className' => 'Executor.Caminho',
            'foreignKey' => 'id_caminho',
        ),
    );

    /**
     * @var array
     */
    public $hasMany = array(
        'Entrada' => array(
            'className' => 'Executor.Entrada',
            'foreignKey' => 'id_operacao',
        ),
        'Saida' => array(
            'className' => 'Executor.Saida',
            'foreignKey' => 'id_operacao',
        ),
    );
}

?>
```



## Executor\Model\Saida.php

&lt;?php

```
/**
 * Description of Caminho
 *
 * @author Augusto Cesar Ferreira
 */
class Saida extends ExecutorAppModel {

    /**
     * Nome da tabela
     * @var string
     */
    public $useTable = 'saida';

    /**
     * @var array
     */
    public $belongsToMany = array(
        'Operacao' => array(
            'className' => 'Executor.Operacao',
            'foreignKey' => 'id_operacao',
        ),
        'Caminho' => array(
            'className' => 'Executor.Caminho',
            'foreignKey' => 'id_caminho',
        ),
        'OrigemSaida' => array(
            'className' => 'Executor.Saida',
            'foreignKey' => 'id_saida_origem',
        ),
    );
}

?>
```