Rodrigo Farias Herculano Mendes

Uma abordagem orientada a modelos para a geração automática de composições de serviços em WS-BPEL

Natal

Agosto de 2013

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

Uma abordagem orientada a modelos para a geração automática de composições de serviços em WS-BPEL

Rodrigo Farias Herculano Mendes

Dissertação submetida ao Programa de Pós-graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Prof. Umberto Souza da Costa
Orientador

Natal, Agosto de 2013

Catalogação da Publicação na Fonte. UFRN / SISBI / Biblioteca Setorial Especializada do Centro de Ciências Exatas e da Terra – CCET.

HERCULANO MENDES, Rodrigo Farias.
Uma abordagem orientada a modelos para a geração automática de composições de serviços em WS-BPEL. – Natal, RN, 2013.

120 f. : il.

Orientador: Prof. Umberto Souza da Costa

Dissertação (Mestre) – Universidade Federal do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Departamento de Informática e Matemática Aplicada. Programa de Pós-graduação em Sistemas e Computação.

1. Engenharia de Software – Dissertação. 2. Orientação a serviços. 3. desenvolvimento dirigido a modelos. 4. Desenvolvimento baseado em modelos. 5. Avaliação de usabilidade. I. Costa, Umberto Souza. II. Título.

RN/UF/BSE-CCET CDU 004.4

Rodrigo Farias Herculano Mendes

Uma abordagem orientada a modelos para a geração automática de composições de serviços em WS-BPEL

Esta Dissertação foi defendida e julgada adequada para a obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte, em 26 de agosto de 2013.

Prof. Umberto Souza da Costa
Orientador

Prof. David Boris Paul Déharbe
Coordenador do Programa

Banca Examinadora:

Prof. Umberto Souza da Costa – UFRN
Presidente

Prof. Martin Alejandro Musicante – UFRN

Prof. Placido Neto – IFRN

Prof[a]. A definir

Resumo

Orientador: Prof. Umberto Souza da Costa
Área de concentração: Engenharia de Software
Palavras-chave: Orientação a serviços, desenvolvimento dirigido a modelos, metodologia, SOD-M, WS-BPEL.
Número de páginas: xi + 41

Abstract

Advisor: Prof. Umberto Souza da Costa
Area of concentration: Software Engineering
Keywords: Service orientation, driven development models, methodology, SOD-M, WS-BPEL.
Number of pages: xi + 41

Sumário

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

1 Introdução

A Computação Orientada a Serviços (SOC[1]) define um conjunto de conceitos, princípios, *frameworks* e métodos para auxiliar organizações no desenvolvimento de aplicações distribuídas. O objetivo central da SOC é a construção de aplicações através da cooperação entre serviços preexistentes, onde os processos de negócios são montados com pouco esforço, em uma rede de serviços que podem ser acoplados de forma flexível, dinâmica, ágil e independente de plataformas específicas (PAPAZOGLOU *et al.*, 2006).

Serviços são aplicações autônomas, interoperáveis que podem ser descritas, publicadas, descobertas, orquestradas e implementadas utilizando protocolos padronizados, promovendo a colaboração entre aplicações distribuídas (DUSTDAR; KRäMER, 2008). Desta maneira serviços definem as características e o comportamento de suas funcionalidades através de uma descrição de serviços. Uma descrição de serviço é um arquivo XML(colocar o que é XML), com a finalidade de descrever as operações que compõe o serviço. Esta descrição define detalhadamente qual o formato das entradas e saídas dos dados que serão processados por cada operação deste serviço.

O desenvolvimento de aplicações compostas por serviços, baseia-se na descrição e integração de serviços antes ou durante sua execução. De posse da descrição de serviços a aplicação busca provedores de serviços pela rede, e integra os serviços necessários, compondo a aplicação(Procurando referência). Esta abordagem de desenvolver aplicações consiste simplesmente em construir aplicações através da descoberta e invocação de serviços pela rede, ao invés de construir novas aplicações (PAPAZOGLOU *et al.*, 2006).

SOC utiliza Web Services (WS) como principal tecnologia para a integração entre sistemas (Procurando referência). Web Services descrevem funcionalidades que servem como base para construção e execução de processos de negócios que são previamente disponibilizados pela rede e acessíveis por interfaces e protocolos padrões. Web Services podem ser definidos como sistemas de software projetados para dar suporte a interação entre máquinas pela rede, através de

[1] Acrônimo do inglês para Service-Oriented Computing

interfaces descritas em um formato padrão como XML, processável especificamente em WSDL (*Web Service Definition Language*) (BOOTH *et al.*, 2004). A interação entre maquinas é feita através da descrição dos serviços que compõem a aplicação, utilizando protocolos de troca de mensagens como: SOAP (*Simple Object Access Protocol*), e são transportados por protocolos como HTTP (*Hiper Text Transfer Protocol*).

A construção de aplicações baseadas em Web Services é uma área que esta apenas no início de sua evolução, e tem como principal desafio manter o alinhamento entre os processos de negócio das organizações e Tecnologias de Informação (TI). A literatura aponta (WATSON, 2008), que metodologias e técnicas como: análise e design, o uso de modelos, melhores praticas e arquiteturas de referencias são necessárias, fundamentando que tais maneiras de modelar serviços são o caminho para o desenvolvimento de aplicações baseadas em serviços com qualidade(WESLEY, 2002).

Neste contexto, alguns autores como (ARSANJANI *et al.*, 2008; BROWN *et al.*, 2005; PAPAZOGLOU; HEUVEL, 2006a; CASTRO; MARCOS; WIERINGA, 2009), afirmam que técnicas como desenvolvimento orientado a modelos, orientação a serviço, técnicas para documentação e melhoria de processos de negócios, são a chave para garantir um desenvolvimento de software rápido e preciso de acordo com a necessidade das empresas em seu domínio de negócio (PAPAZOGLOU *et al.*, 2007).

Trabalhos recentes produzidos por (ZHANG; JIANG, 2008), dedicam-se à transformação de modelos. Este trabalho propõe uma abordagem baseada em MDA para modelar requisitos funcionais de processos de negócios, promovendo a flexibilidade da arquitetura de negócios a requisitos mutáveis.

O trabalho proposto por (NETO, 2012), propõe uma abordagem para modelar composições de serviços e suas restrições não-funcionais, partindo de abstrações de alto nível, e direcionando as transformações de modelos para a linguagem de orquestração de serviços PEWS (*Path Expression Web Services*). Este trabalho também define um meta-modelos específicos para a linguagem de orquestração PEWS e um conjunto de regras de transformação dos elementos deste meta-modelo (as composições de serviços, e *A-policies* associadas) para elementos específicos do modelo PEWS. Este trabalho considera os três níveis de abstração da Arquitetura Dirigida por Modelos (MDA [2]) o CIM [3], PIM [4] e PSM [5].

[2] Acrônimo do inglês para Model Driven Architecture
[3] Computation Independent Model
[4] Platform Independent Model
[5] Platform Specific Model

1.1 Objetivos

Este trabalho tem como objetivo geral propor um método de desenvolvimento que expresse requisitos funcionais de aplicações baseadas em serviços para a linguagem WS-BPEL. Desta maneira este trabalho complementa o trabalho realizado por (NETO, 2012): (i) expandindo a modelagem de plataformas específicas da metodologia π-SOD-M para a linguagem WS-BPEL (*Web Services - Business Process Execution Language*)(ii) transformando modelos da plataforma WS-BPEL para código de máquina processável. Neste sentido, uma das contribuições desta pesquisa é a proposição de um meta-modelo para a plataforma WS-BPEL, capaz de captar aspectos funcionais, para que estes forneçam diretrizes para expressar processos de negócio. (iii) Possibilitar que projetos sejam bem documentados e gerenciados;

1.1.1 Objetivos específicos

Os objetivos específicos deste trabalho são:

1. Definir um meta-modelo da plataforma especifica WS-BPEL, este metamodelo será nomeado por WS-BPEL Model;
2. Especificar regras de transformação em ATL para transformar elementos do modelo π-*Service Composition* para elementos do modelo WS-BPEL.
3. Definir regras de transformação do metamodelo WS-BPEL para gerar um esqueleto da especificação do processo de composição de serviços em WS-BPEL;
4. Validar das transformações entre modelos através do estudo de caso.
5. Análise comparativa entre a especificação de uma aplicação construída pelo método π-SOD-M com modelos π-*PEWS* e uma construída com modelos WS-BPEL, verificar a conformidade entre as duas instâncias.

1.2 Organização do Documento

Este trabalho está estruturado em cinco capítulos: no Capítulo 2, descrevemos a fundamentação teórica de nossa pesquisa, no Capítulo 3, contextualizamos alguns trabalhos relacionados ao desenvolvimento de nossa pesquisa; no Capítulo 4, descrevemos o nosso ambiente de estudo, a nossa implementação e a avaliação do mosso estudo de caso; e finalmente no Capítulo

5, descrevemos a avaliação sobre o nosso estudo de caso, as considerações finais deste trabalho e nossa perspectiva de trabalhos futuros.

2 *Fundamentação*

O processo de analise e designer de software envolvem vários conceitos fundamentais em torno de sua execução. Por isso, neste capítulo, são apresentados conceitos importantes na fundamentação deste trabalho. Dentre os conceitos aqui descritos abordamos: *Service-oriented computing* (SOC); *Model Driven Architecture* MDA; π-*Service-Oriented Development Method* (π-SOD-M); Web *Services-Business Process Execution Language* (WS-BPEL); *Web Service-Policy* (WS-Policy).

2.1 Computação Orientada a Serviços - SOC

A Computação Orientada a Serviço (SOC), é um paradigma computacional emergente que define um conjunto de conceitos, princípios, *frameworks* e métodos para auxiliar organizações no desenvolvimento de aplicações distribuídas (PAPAZOGLOU *et al.*, 2006). Este paradigma vem mudando a forma como as aplicações distribuídas vem sendo desenvolvidas, projetadas, disponibilizadas e "consumidas". (PAPAZOGLOU; GEORGAKOPOULOS, 2003) considera SOC o paradigma computacional que utiliza serviços como elementos fundamentais para o desenvolvimento de aplicações.

Serviços fornecem uma interface bem definida e estão associados à implementação de uma determinada funcionalidade de uma aplicação de acordo com as necessidades organizacionais. (PAPAZOGLOU; HEUVEL, 2006b) considera que o desenvolvimento de serviços está diretamente associado ao processo de negócio. Logo se faz necessário derivar o modelo de processo de negócio para efetivamente construir os serviços da organização considerando, assim, fatores como: reusabilidade, independência de aplicação e plataformas aos quais os mesmos serão executados.

A tecnologia de serviço mais difundida é o serviço Web ou *Web Services*. Serviços web são disponibilizados em um registro de serviços, sendo oferecidos por provedores de serviços e utilizados por consumidores (clientes). (PAPAZOGLOU; HEUVEL, 2007) define que provedo-

res e consumidores são fracamente acoplados no contexto da computação orientada a serviço. Serviços web fornecem a interoperabilidade entre aplicações através de troca de mensagens. A comunicação é realizada pela Web por meio de padrões preestabelecidos, o que inclui o SOAP (*Simple Object AccessProtocol*) para transmissão de dados e o WSDL (*Web Services Description Language*) para a definição dos serviços Web. E normalmente e não exclusivamente são transportados por protocolos como HTTP(*Hypertext Transfer Protocol*).

Serviços web são integrados para a construção de aplicações flexíveis, desacopladas e distribuídas dentro do ambiente organizacional e entre as mais diversas plataformas tecnológicas existentes (PAPAZOGLOU *et al.*, 2007). Para que a integração automática de serviços web aconteça é necessário o auxilio de elementos responsáveis pela integração dos serviços web, a composição e a coordenação de processos de negócio.

Os benefícios de uma aplicação orientada a serviços normalmente são alcançados através de mecanismos de integração que tornam possível a interação síncrona e assíncrona entre os serviços. Para facilitar esta integração, mecanismos como orquestração e coreografia são comumente utilizados para compor processos de negócio utilizando de *Web Services*. Na seção a seguir descrevemos estes dois mecanismos.

2.2 Orquestração X Coreografia

Para que aplicações orientadas a serviços sejam desenvolvidas, se faz necessário a implementação de mecanismos de composição, capazes de promover a integração entre os serviços que compõem tal aplicação. Estes mecanismos são responsáveis por intermediar a invocação e coordenação de serviços assim como também a execução de processos. Serviços web sem o auxilio de mecanismos de composição são limitados ao ponto de apenas fazer interações simples, já que a maioria dos padrões de interação como SOAP e HTTP, são essencialmente síncronos e sem a manutenção de seus estados (*stateless*).

O real potencial do Paradigma Orientado a Serviços está diretamente ligado ao auxilio de mecanismos de integração que tornem possíveis interações síncronas e assíncronas e com manutenção de estado (*stateful*) entre as partes envolvidas. Desta maneira, estes mecanismos são requisitos fundamentais para atender a processos de negócios complexos presentes nos sistemas atuais (DUSTDAR; KRäMER, 2008). Existem diversas técnicas para implementação de Composição de Serviços. Entre elas as mais comuns são a orquestração e coreografia.

Segundo (PAPAZOGLOU; HEUVEL, 2006c) orquestração de serviços é uma composição de processos de negócio (através de Web Services) onde existe a figura de um processo central

(processo mestre) que controla e coordena os demais processos. Neste tipo de composição, cada processo participante não tem conhecimento de que faz parte de uma composição de processos, com exceção do processo mestre. Somente o processo mestre detém a inteligência sobre o processo completo, e a execução é então centralizada. Outra maneira de definir orquestração é como um processo de negócio executável, que interage através de mensagens com serviços web internos e externos que compõe o processo.

A Figura 2.1 ilustra um exemplo de um processo de orquestração onde, o controlador central invoca e recebe uma solicitação de um consumidor (passao 1). Em seguida, ele invoca um serviço (passo 2). Ao receber esta resposta, ele invoca outros serviços (passos 3 e 4). Ao concluir o processamento, o controlador central retorna uma resposta para o consumidor (passo 5).

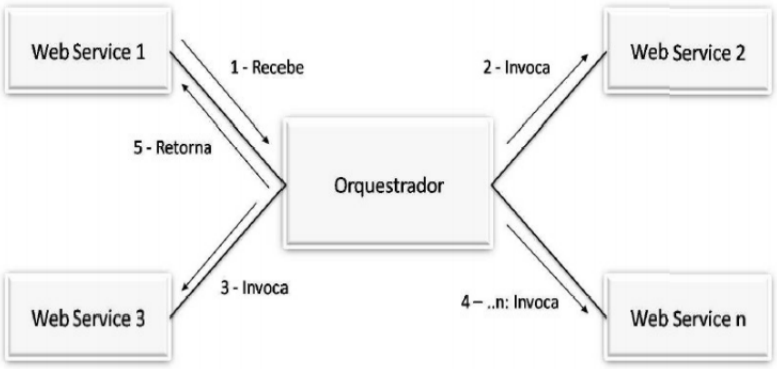

Figura 2.1: Funcionamento básico da orquestração.

A coreografia, diferentemente da orquestração, pode ser definida como um processo colaborativo onde cada parte envolvida descreve sua função na interação entre os serviços que compõe o processo de negócio. Segundo (PELTZ, 2003) coreografia é uma composição de processos de negócio (através de Web Services) onde não existe a figura de um processo central (processo mestre) que controla e coordena os demais processos. Neste tipo de composição, cada processo envolvido tem conhecimento de que faz parte de uma composição de processos e que precisa interagir com outros processos, de maneira ordenada, para que a composição resultante tenha sucesso. Cada processo participante sabe exatamente quando atuar, com quais outros processos participantes interagir, quais operações deve executar, quais mensagens deve trocar e até mesmo o momento adequado de troca de mensagens.

A coreografia tem como característica principal uma maior colaboração entre os Serviços Web que o constituem, o que induz cada um dos participantes a descrever seu papel nas interações através de trocas de mensagens entre si. A lógica do processo de negócio corresponde

à união das informações conhecidas por cada participante. A Figura 2.2 a seguir ilustra um processo de negócio coreografia.

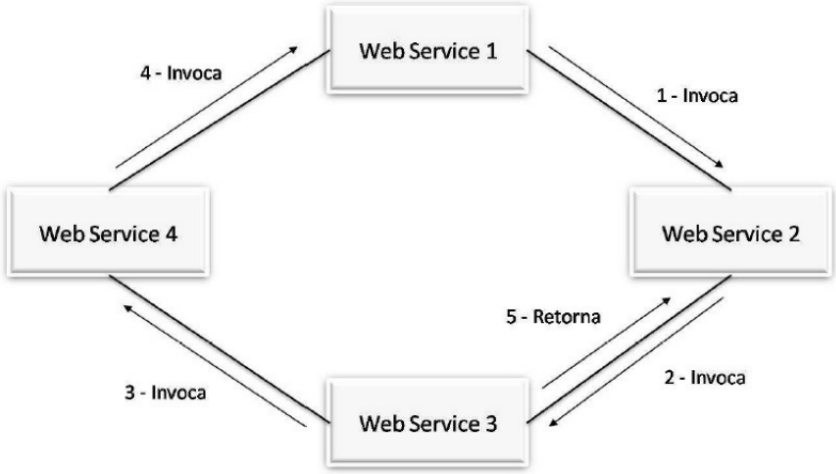

Figura 2.2: Funcionamento básico da coreografia.

O processo inicia com a invocação do serviço 1. Após a conclusão desta tarefa, o serviço 2 é invocado para realizar outra tarefa. Em seguida os serviços 3 e 4 e assim sucessivamente. Cada serviço realiza uma tarefa ou atividade especifica e inicia um ou mais serviços consecutivos que realizam outras tarefas ou atividades, de acordo com a necessidade do processo de negócio.

A coreografia descreve a colaboração dos serviços no sistema como um todo independente de um elemento controlador que gerencia as partes envolvidas. Por tanto, linguagens de coreografia não possuem a finalidade de serem executadas, mas sim de descrever regras de interações entre diversos participantes em um processo de negócio. A comunicação é modelada através de conexões permanentes e com manutenção de estado, visando os detalhes de interação entre os participantes do processo executado colaborativamente, descrevendo e restringindo as mensagens que cada participante pode enviar e quais respostas são esperadas. Normalmente as linguagens que descrevem coreografia definem os aspectos dinâmicos da interação entre serviços que envolvem o processo de negócio (PAPAZOGLOU; HEUVEL, 2006c).

2.2.1 Padrão de interação automática

O processo de desenvolvimento de aplicações composta por serviços, baseia-se somente na descrição e na integração automática de serviços. Uma vez de posse da descrição dos serviços, a aplicação busca os provedores de serviços necessários, e integra os serviços compondo a aplicação, isto acontece antes ou durante a execução. O foco do desenvolvimento orientado

a serviços concentra-se na maneira como as aplicações são descritas e organizadas. Como resultado, as aplicações tornam-se dinamicamente mutáveis, ou seja, são capazes de descobrir serviços em tempo de execução de acordo com a suas necessidades.

Para fornecer detecção dinâmica para aplicações baseadas em serviços, a orientação a serviços oferece uma padronização entre as entidades que atuam na formalização da descrição de serviços. A seguir, a Figura 2.3 ilustra o padrão de interação da Computação Orientada a Serviços.

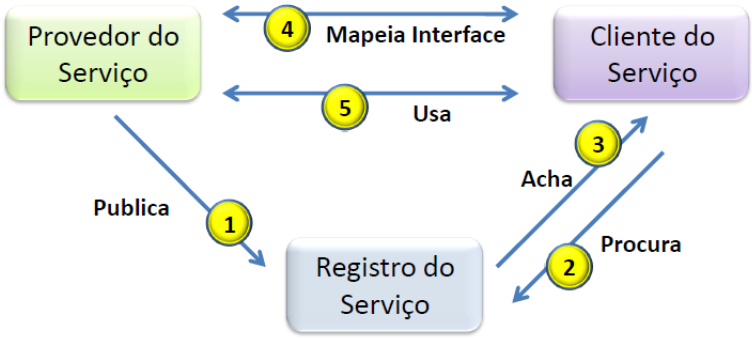

Figura 2.3: Diagrama de interação SOC.

As seguintes entidades atuam no processo padrão de integração automática de serviços:

- *Service providers* (provedores de serviço): representa a plataforma que hospeda e disponibiliza o serviço web, permitindo que os clientes acessem o serviço. Organizações disponibilizam seus serviços implementados em uma descrição de serviços, para que os mesmos sejam visualizados e acessados por outras organizações e mais tarde integrados a suas aplicações. (1) Um prestador de serviço publica a descrição de seu serviço em um registro de serviço, para que aplicações possam encontrar um dado serviço a partir de sua descrição.

- *Service requesters* (consumidores de serviços): é a aplicação cliente, que está procurando, invocando ou iniciando uma interação entre os serviços web para a composição da aplicação. O cliente do serviço pode ser uma pessoa acessando a aplicação através de um *browser* ou uma aplicação realizando uma invocação aos métodos descritos na interface de algum serviço web que compõe a aplicação de fato. (2) Os consumidores utilizam o registro de serviços para obter os serviços necessários para realizar suas tarefas.

- Um ou vários *service registry* (registradores de serviços): Os clientes, ou seja, os (service requesters) buscam por serviços necessários de acordo com suas necessidades nos regis-

tradores de serviços e recuperam informações referentes a interface de comunicação para que a aplicação ligue-se aos serviços web necessários, durante a fase de desenvolvimento ou durante a execução da aplicação cliente. (3) Se os provedores de serviço adequados encontram-se no registro de serviços no momento da solicitação, o registrador de serviço retorna uma referência do serviço solicitado, para que, desta forma, a aplicação possa ligar-se a um destes provedores (FEIG *et al.*, 2007). (4) Uma vez que esta ligação seja concretizada, o provedor de serviço retorna uma instancia deste serviço. E finalmente (5) uma instância dos serviços podem ser utilizados pela aplicação cliente.

2.2.2 Serviços Web

SOC utiliza serviços web como elemento essencial para a construção de aplicações distribuídas. Neste contexto serviços são aplicações autônomas interoperáveis que podem ser descritas, publicadas, descobertas, orquestradas e implementadas utilizando protocolos padronizados, promovendo a colaboração entre aplicações distribuídas de diversas organizações (DUSTDAR; KRäMER, 2008). Além disso, são projetados de forma independe e contratualmente definidos em uma descrição de serviços, que contém uma combinação sintática, semântica e informações comportamentais, que expressam suas capacidades através de sua descrição.

Web services é um padrão aberto mantido pela *World Wide Web Consortium* (W3C), para padronizar a interação entre aplicações distribuídas. Utilizar este padrão, para implementar serviços, torna-se cada vez mais comum. O uso deste padrão por todos os fornecedores de serviços é um meio para promover a integração entre sistemas díspares.

Web services podem ser disponibilizados através da Internet, usando padrões e protocolos baseados em linguagem XML(*eXtensible Markup Language*). Desta maneira podemos definilos como: um sistema de software projetado para dar suporte interações máquina-a-máquina interoperáveis sobre uma rede. *Web services* são descritos através de uma descrição de serviços em um formato processável por linguagem de máquina (mais especificamente WSDL) (BOOTH; MCCABE; CHAMPION,). Outros sistemas interagem com *web services* através de sua descrição de serviço por meio de troca de mensagens utilizando protocolos web como SOAP, normalmente e não exclusivamente transmitidas através de HTTP, com serialização XML em conjunto com outros padrões Web de comunicação. Por englobar todos estes padrões em seu desenvolvimento, *web Service* pode ser considerado como elemento chave dá Computação Orientada a Serviço.

Por padrão, a arquitetura desta tecnologia é descrita em camadas, para facilitar o entendimento e compreensão das operações de interação entre *web services*, estas camadas são deno-

minadas Pilha de protocolos (KREGER, 2001). Para que seja possível as operações de publicar, descobrir e invocar remotamente por parte de clientes e contratantes os *web services* devem ser disponibilizados através da rede. A Figura 2.4 ilustra a pilha de protocolos utilizada na interação entre *web services*.

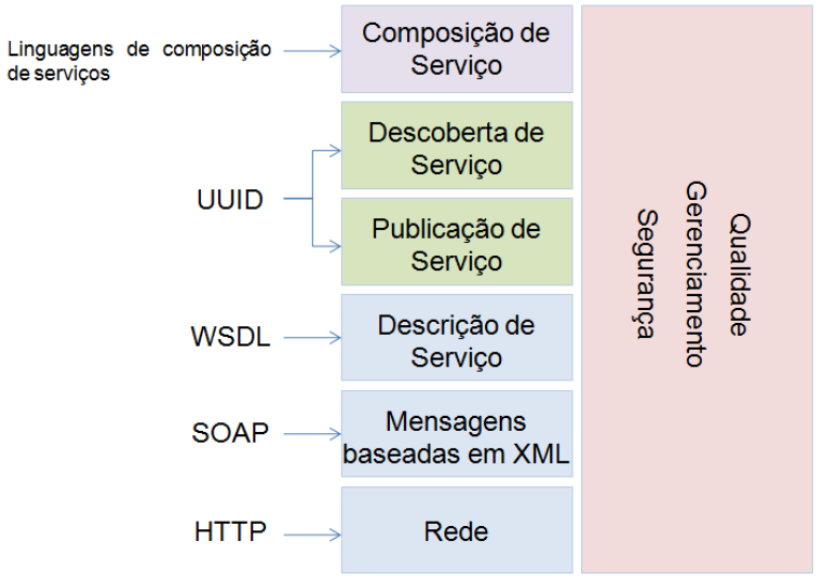

Figura 2.4: Pilha de protocolos para interação entre web services.

Na base da pilha de protocolos, encontra-se a camada mais próximo a rede, esta camada é responsável por fazer a comunicação entre serviços através da internet. Para que esta comunicação seja possível, é necessário o uso de protocolos específicos para a comunicação entre maquinas através da rede. Protocolos como HTTP (*Hypertext Transfer Protocol*) que são amplamente difundido através da rede mundial de computadores. Este é o principal protocolo a nível de rede utilizado na implementação de web services. Entretanto, outros protocolos como FTP, REST e SMTP, podem ser utilizados para realizar a comunicação entre *web services* partindo da camada mais baixa.

A segunda camada é responsável por fornecer meios para trocas de mensagens baseadas em linguagem XML entre *web services*. Nesta camada o protocolo SOAP (*Simple Object Access Protocol*) é amplamente utilizado para a comunicação entre web services, devido principalmente por seu mecanismo padronizado de envelope de mensagens centradas e documentos ou chamadas remotas de procedimento (RPC) (BOOTH; MCCABE; CHAMPION,). Além disso, SOAP oferece um mecanismo padronizado de codificação de tipos, ou seja, um conjunto de regras de codificação para expressar e converter instâncias de tipos de dados definidos em aplicações em plataformas e linguagens de programação díspares.

A terceira camada é responsável pela descrição dos *web services*, aqui o padrão utilizado para descrever a combinação sintática, semântica e informações comportamentais das capacidades funcionais dos serviços web é o padrão WSDL. Este padrão por sua vez, define a interface de comunicação de *web services*, assim como também, a dinâmica de interação com o mesmo. Por meio da descrição de serviços é possível que provedores publiquem as especificações contratuais necessárias para que clientes possam invocar tal *web service*. A descrição do serviço é a chave para tornar aplicações compostas por *web services* fracamente acoplada bem como reduzir a quantidade necessária de conhecimento compartilhado entre o cliente e provedor de serviços.

As três camadas descritas anteriormente, formam a base minima para que seja possível que aplicações baseadas em serviços possam interopera entre os serviços web que a compõe. Enquanto estas camadas apresentam tecnologias de interoperabilidade entre serviços web, as duas próximas camadas, apresentam a forma como *web services* são publicados e descobertos, é possível implementar estas operações de interação através de uma variedade de soluções.

A quarta camada da pilha de protocolos é subdividida em duas, a camada de publicação de serviços e a camada de descoberta de serviços. Estas camadas são responsáveis por publicar e descobrir *web services*). A publicação de um *web sevice* consiste em disponibilizar o documento WSDL para que os mesmos possam ser descobertos através da rede. A publicação de um *web service* pode ser feita de duas formas: direta e indiretamente. A publicação direta, consiste simplesmente em enviar o documento WSDL a um cliente interessado em contratar um serviço.

Já a publicação indireta é feita quando provedores de serviços publicam documentos WSDL em repositórios para serem descobertos e acessados posteriormente por terceiros. A principal tecnologia utilizada para o desenvolvimento destes repositórios é o UDDI (*Universal Description, Discovery, Integration*) que utiliza uma base de registro compartilhada por diferentes clientes, permitindo descrever, registrar e localizar serviços (BOOTH; MCCABE; CHAMPION,). É importante frisar que a camada de descoberta esta fortemente ligada a camada de publicação, haja visto que serviços não podem ser descobertos pela rede, se não foram publicados anteriormente.

A camada superior da pilha de protocolos é responsável por descrever de fato como os serviços são compostos. Serviços implementam funcionalidades especificas e independente, logo a composição de vários serviços podem ser implementada gerando um serviço mais complexo ou para a construção de sistemas baseados em serviços. Linguagens como WS-BPEL, PEWS, WSFL, etc, podem ser usada nesta camada visto que ela permite a composição de serviços a partir da descrição de processos de negócios.

2.2.3 Linguagens de Orquestração

Para descrever orquestração de serviços em processos de negócio, é preciso fazer uso de linguagens especializadas para este fim. Tais linguagens, devem ser capazes de disponibilizar construções: condicional, controle de exceção, sequencia, laços de repetição, paralelismo e variáveis, assim como construções capazes de manipular e tratar eventos, controles de tempo e recuperação de exceções (ALVES *et al.*, 2006).

Como exemplo das características descritas anteriormente, podemos citar algumas linguagens de orquestração: WSFL (*Web Services Flow Language*) (TSALGATIDOU; PILIOURA, 2002), WS-BPEL (*Web Service - Business Process Execution Language WS-BPEL*) (ALVES *et al.*, 2006), PEWS (*Predicatepath Expression for Web Services*) (BA; FERRARI; MUSICANTE, 2005). A seguir definiremos de maneira sucinta cada uma destas linguagens.A linguagem de orquestração WS-BPEL terá uma seção própria por ser a linguagem em foco neste trabalho.

- *Web Service Flow Language*: WSFL é uma linguagem produzida pela IBM em meados de 2001, com sintaxe baseada em XML para descrever composição de serviços em processo de negócios públicos, privados ou sequencias. Com WSFL, também é possível descrever operações específicas, execução de processos e das trocas de dados entre os serviços web que compõe o processo de negócio. WSFL implementa interfaces WSDL, permitindo composição recursiva e tratamento de exceção (LEYMANN, 2001).

- *Predicatepath Expression for Web Services*: PEWS é uma linguagem de orquestração de serviços que destaca-se por promover suporte a descrição de comportamento em aplicações baseadas em serviços (BA; FERRARI; MUSICANTE, 2006). Tal comportamento, define a ordem em que os serviços de uma composição são executados, a definição de comportamento ocorre através da adição de *predicate path expressions* as descrições dos serviços envolvidos em uma composição. Como diferencial em comparação a outras linguagens de orquestração, PEWS dá suporte à representação de comportamento temporal e verificação dos dados em tempo de execução.

- *Web Services - Business Process Execution Language*: WS-BPEL é uma linguagem de programação com sintaxe baseada em XML, usada para composição, orquestração e coordenação de *Web Services*, capaz de descrever comportamento de processos de negócios (ALVES *et al.*, 2006). WS-BPEL pode ser utilizada para descrever aplicações baseadas em serviços, assim como também descrever aplicações que são serviços. Esta linguagem foi inspirada em linguagens anteriores como *Web Services Flow Language* (WSFL) da IBM e XLANG da Microsoft (ALVES *et al.*, 2006).

2.2.4 Model Dirven Architecture

Desenvolver sistemas que atendam aos requisitos das organizações, não é uma tarefa simples. Atualmente, organizações buscam mecanismos de desenvolvimentos, capazes de facilitar esta tarefa. A literatura aponta, que metodologias, técnicas de desenvolvimento, análise e designer são o ponto de partida para desenvolver sistemas que atendam aos requisitos organizacionais com rapidez, baixo custo e qualidade. Alguns autores, salientam que metodologias orientadas por modelos, técnicas para documentação, são a chave para tornar real a ideia de rapidez, maior produtividade, portabilidade, interoperabilidade, menor custo, facilidade na evolução do software e maior qualidade no desenvolvimento de softwares (NETO, 2012).

Neste contexto, o processo de desenvolvimento de software dirigido a modelos (*Model Driving Development*), padronizado pela *Model Driven Architecture* (MDA) é uma nova abordagem para o desenvolvimento de software, definida pelo *Object Management Group* (OMG), que vem recebendo atenção da comunidade acadêmica e industrial. MDA visa separar a lógica do negócio da lógica da aplicação, mantendo-as em diferentes níveis de abstração (MILLER; MUKERJI, 2003). Esta metodologia coloca a criação de modelos como elemento central do processo de desenvolvimento de software.

MDA prega a construção de modelos desde o início do processo de desenvolvimento de software, ou seja, a partir da captação dos requisitos do sistema (domínio do problema), gerando modelos com maior nível de abstração. Tais modelos podem ser incrementados com características especificas de uma plataforma, tornando-se modelos mais concretos (domínio solução), para que mais tarde estes modelos sejam transformados em código. Desta maneira o ciclo de vida MDA é baseado em transformações automáticas de modelos de mais alto nível pro meio de regras de mapeamento, em modelos mais concretos, ate que por fim estes modelos seja transformados automaticamente em código do sistema. A Figura 2.6 ilustra o cilo de vida do MDA.

Em um processo MDA automatizado, os modelos mais concretos do sistema, devem representar com precisão o código, ou seja, ele deve ser executável e ter uma equivalência funcional com todos os outros modelos mais abstratos (MILLER; MUKERJI, 2003). Desta maneira, modificações no modelo de mais alto nível de abstração são refletidas automaticamente nos modelos de mais baixo nível, tornando a atividade de modelar no nível mais abstrato, o centro de todo processo de desenvolvimento do software, dispensando completamente atividades manuais nos modelos de mais baixos níveis de abstração. É possível observar que MDA faz uma separação clara dos requisitos do sistema, através da representação de modelos com abstração mais elevada, das características especificas de implementação do sistema, que formalizam os

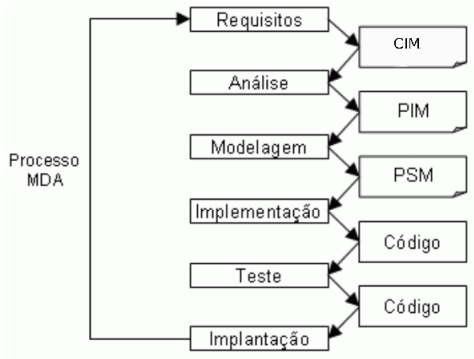

Figura 2.5: Ciclo de vida do MDA.

modelos mais concretos (MILLER; MUKERJI, 2003).

2.2.5 Modelos

O processo de modelagem de um sistema, parte de um conjunto de requisitos de domínio específico para a obtenção de uma abstração apropriada deste sistema, não é uma tarefa simples. O desenvolvimento de sistemas, independente de sua complexidade, levam ao uso de modelos. Modelos são a representação (especificação) funcional, estrutural e comportamental de sistemas de software (MILLER; MUKERJI, 2003). Modelos facilitam a comunicação entre os *stakeholders* do sistema, são mais baratos de construir do que protótipos do sistema em se, servem como documentação e aumentam a produtividade no domínio da aplicação do sistema. Tornando -se uma ferramenta poderosa em mãos da equipe de desenvolvimento, para diminuir a distancia entre os requisitos e as funcionalidades implementadas.

MDA é uma abordagem que faz uso de modelos em vários níveis de abstração, juntos estes modelos formalizam um sistema modelado de fato. Alguns destes modelos serão construídos independente de plataformas especificas, enquanto outros serão modelados de acordo com a plataforma de software escolhida (MILLER; MUKERJI, 2003). Os níveis de abstração de modelos MDA são: Computação Independente de Modelo (CIM); Modelo Independente de Plataforma (PIM) e Modelo de Plataforma Especifica (PSM). A seguir os três níveis de abstração citados anteriormente serão descritos.

2.2.6 *Computation Independent Model* (CIM)

O modelo CIM mostra o sistema do ponto de vista independente de computação. Este modelo, não se preocupa com detalhes específicos da estrutura dos sistemas, sendo considerado

o modelo de domínio de negócio.

Este modelo, é construído com um vocabulário simples para que usuários do sistema possam entender facilmente as ideias especificadas no mesmo, pois usuários dos sistemas geralmente não têm conhecimento sobre modelos ou artefatos usados para modelar requisitos de sistemas (MILLER; MUKERJI, 2003).

A construção do modelo CIM é importante, pois este modelo é a ponte entre especialistas no domínio do problema e seus requisitos, e a equipe de desenvolvimento que são especialistas em projetar (arquitetura) e construir artefatos que juntos vão satisfazer aos requisitos do domínio, elicitados junto aos usuários (MILLER; MUKERJI, 2003). O CIM é obtido no processo de documentação e especificação dos requisitos.

2.2.7 *Platform Independent Mode* - PIM

O modelo PIM representa as operações do sistema, ou seja, o modelo computacional do sistema que sera produzido, mas escondendo os detalhes necessários para implantar esse modelo numa plataforma específica. Este modelo é único e não há variação deste de uma plataforma para outra.

O modelo PIM pode ser especificado usando uma linguagem de modelagem de proposito geral como *Unifield Modeling Language* (UML) ou especifico como o método PI-SOD-M, de maneira que este modelo pode ser considerado o modelo conceitual do sistema (MILLER; MUKERJI, 2003).

Este modelo é uma matéria de sujeito como um sistema bancário ou controle de estoque. Um modelo PIM representa um ou mais modelos de plataforma, não preocupando-se com detalhes específicos da mesma, como por exemplo, um PIM de um sistema de controle de estoque, não precisa ser capturados aspectos da persistência dos dados do sistema, ou tratamento de exceção do mesmo (MILLER; MUKERJI, 2003).

2.2.8 *Platform Specific Mode* - PSM

O modelo PSM mostra a visão do sistema que agrega características e elementos da plataforma específica, contendo informações da tecnologia utilizada na aplicação, como por exemplo a linguagem de programação, os componentes de *middleware*, a arquitetura de hardware e de software. Para que isso seja possível é necessário o suporte de ferramentas que façam o mapeamento adequado de uma especificação abstrata (PIM) para uma determinada plataforma

(MILLER; MUKERJI, 2003).

O PSM, por sua vez, passa por processo(s) de refinamento(s) para obtenção de um alto nível de detalhes que especificam a plataforma de destino. Uma vez capturado todo este detalhamento da plataforma de destino, é possível efetuar um processo de transformação do mesmo em código (implementação) da aplicação (MILLER; MUKERJI, 2003). O modelo PSM é o responsável por lidar com toda heterogeneidade e complexidade dos diversos tipos de plataformas existentes.

3 *Trabalhos relacionados*

Neste capítulo apresentamos uma descrição geral dos trabalhos relacionados com a pesquisa, além de promover comparações entre as abordagens correlatas e a abordagem proposta por nós.

3.1 *Transforming Business Requirements into BPEL: a MDA-Based Approach to Web Application Development*

O trabalho produzido por (ZHANG; JIANG, 2008), propõe uma abordagem baseada em MDA para modelar requisitos funcionais (RF) de processos de negócios, promovendo a flexibilidade da arquitetura de negócios a requisitos mutáveis. A abordagem utiliza o VIP-*framework* para auxiliar a modelagem dos níveis de abstração CIM e PIM intrínsecos ao MDA.

A abordagem inicia dividindo o modelo CIM em dois níveis conceituais. Em um nível superior o modelo BVM-GAV (*Global Actor Viewpoint*), é construído sobre os valores ao qual a organização lucra ou aumenta sua utilidade econômica. Em um nível inferior, dois modelos são propostos para expressar a lógica de negócio atual, o BIM e o BPM. O BIM-SV (*System Viewpoint*) é responsável por expressar as trocas de informações entre as entidades que circunscreve o sistema, mais tarde este modelo é estendido para descrever maiores detalhes sobre as informações e seus relacionamentos especializando o BIM-SV para o BIM-OV (*Organization Viewpoint*). O BPM-BV (*Business Viewpoint*) é responsável por expressar o fluxo de controle do processo de negócio, mais tarde este modelo é estendido para expressar a rastreabilidade e consistência do modelo de negócios especializando o BPM-BV (*Business Viewpoint*) para o BPM-SV (*Sistem Viewpoint*).

Como modelo PIM a abordagem utiliza o metamodelo BPEL, pois, ele promove uma solução flexível para descrever a arquitetura de aplicações orientadas a serviços, oferecendo um refinamento do modelo de negócio através da orquestração de serviços.

3.2 *Towards a Service-Oriented MDA-Based Approach to the Aalignment of Business Processes With it Systems: From The Business Model to a Web Service Compositon Model*

Este trabalho apresenta uma abordagem para desenvolvimento de Sistemas baseados em serviços, chamada SOD-M (*Service Oriented Development Method*), esta metodologia é integrada ao *framework* MDA (*Model Driven Architecture*). O método SOD-M define um processo guiado por modelos que inicia a modelagem do ambiente de negócio partindo do alto nível de abstração obtendo um design de composição de serviços.

O método tem como característica fazer uso de serviços identificados na modelagem de alto nível de processos como elementos para a construção de aplicações. O método SOD-M usa a técnica de elicitação de requisitos chamada e3value como uma abordagem de modelagem de negócios, ao qual nos permite compreender o ambiente de negócios ao qual a aplicação será usado, além de identificar os serviços que serão oferecidos pelo sistema para satisfazer as necessidades dos clientes.

SOD-M também modela o comportamento das aplicações baseadas em serviços, incluindo a modelagem de novos elementos para que seja possível representar o processo de negócio e como eles serão providos por meio da composição de *Web Services* (WS). Assim, é possível explicitar como o design da composição de *Web Services* pode ser derivado da modelagem de alto nível de negócio.

O trabalho realizado por (NETO, 2012) é uma extensão da metodologia SOD-M para a construção de composições de serviços e suas restrições não funcionais associadas, esta extensão denomina-se π-SOD-M. Nossa proposta é um complemento a extensão feita por (NETO, 2012). Onde propomos a extensão do número de PSM's, gerando um modelo voltado para a plataforma especifica WS-BPEL.

3.3 *A Methodology for Building Reliable Service-Based Applications*

π-SOD-M (*Policy-based Service Oriented Development Methodology*), é uma metodologia para a modelagem de aplicações orientadas a serviços a qual usa Políticas de qualidade. O trabalho propõe um método orientado a modelos para desenvolvimento de aplicações confiáveis.

π-SOD-M consiste de: (i) um conjunto de meta-modelos para representação de requisitos

não-funcionais associados a serviços em diferentes níveis de modelagem, a partir de um modelo de caso de uso até um modelo de composição de serviço, (ii) um meta-modelo de plataforma específica que representa a especificação das composições e as políticas, (iii) regras de transformação de modelo para modelo e de modelo para texto para semi-automatizar a implementação de composições de serviços, e (iv) um ambiente que implementa estes meta-modelos e regras.

Nossa proposta complementa este trabalho aumentando o número de PSM's da metodologia π-SOD-M. Este aumento se dá através da geração de um meta-modelo de plataforma específica capaz de representar composições de serviços voltada para a linguagem de orquestração de serviços WS-BPEL. Propomos também regras de transformação de modelo para modelo e de modelo para texto para semi-automatizar a implementação de código de maquina das composições de serviços.

4 *Estendendo π-SOD-M para a plataforma WS-BPEL 2.0*

Este capítulo descreve a extensão proposta para aumentar o número de plataformas especificas do método π-SOD-M, abrangendo o desenvolvimento de sistemas orientados a serviços para a linguagem de orquestração WS-BPEL 2.0. A construção de aplicações com o método π-SOD-M inicia-se na modelagem de requisitos funcionais e não funcionais a partir de um modelo π-Use Case, em seguida estes casos de uso são transformados em uma serie de modelos PIM e PSM antes de finalmente gerar o código que implementa a aplicação.

O ambiente π-SOD-M foi construído através da IDE Eclipse, mais especificamente utilizando *Eclipse Modelling Framework* (EMF) para gerar o meta-modelo WS-BPEL 2.0, este *framework* fornece utilitários necessários para definir, editar e manipular modelos. Para automatizar as transformações entre modelos utilizamos a linguagem ATL, ao qual é suportada pela IDE Eclipse em sua suíte de desenvolvimento. π-SOD-M Utiliza este ambiente para gerar composições de serviços em π-PEWS (NETO, 2012). Após nossa extensão do método π-SOD-M também construirá composições de serviços em WS-BPEL2.0.

O restante deste capítulo está organizado da seguinte forma. Seção 4.1 descreve detalhadamente as entidades e suas funções da linguagem WS-BPEL 2.0. A seção 4.2 descreve como WS-BPEL 2.0 será incorporado ao metodologia π-SOD-M. Seção 4.3 descreve as alterações feitas para estender o ambiente π-SOD-M, para a adição de novos componentes para geração de especificação de sistemas em diferentes linguagens. Seção 4.4 descreve as transformações entre modelos citadas anteriormente. Seção 4.5 faz uma comparação entre códigos gerados pela metodologia π-SOD-M para as plataformas π-PEWS e WS-BPEL, e por fim conclui capítulo.

4.1 Meta-modelo WS-BPEL

Em construção!!!

4.1.1 Web Services-Business Process Executation Language

Web Services-Business Process Execution Language (WS-BPEL) é uma linguagem de composição com sintaxe baseada em XML, usada para descrever composição, orquestração, coordenação e processos de negócios composto por *Web Services* (WS)(ALVES *et al.*, 2006). Assim como também, pode ser utilizada para criar serviços a partir da coordenação de outros serviços pré-existentes.

Um processo BPEL, descreve como ocorre o relacionamento entre diversos *web services* participantes de uma composição. Para descrever tais relacionamentos, BPEL oferece determinados tipos de construções similares as das linguagens de programação tradicionalmente conhecidas, como por exemplo: estrutura de repetição, condicionais, variáveis e atribuição, tratamento de exceções, parceiros de negócio e a coordenação destes parceiros.

Para que seja possível construir um processo de negócio BPEL é necessário uma série de elementos fundamentais para especificação de um processo de negócio. A estrutura básica de um processo BPEL é formada por três seções: *PartnerLinks, Variables e Activities*. Desta maneira um esqueleto de um processo BPEL assemelha-se a um documento XML tradicional. A listagem de código ilustra o esqueleto base de um processo descrito WS-BPEL.

```
<process name="NCName" targetNamespace="anyURI"
 queryLanguage="anyURI"?
 expressionLanguage="anyURI"?
 suppressJoinFailure="yes|no"?
 exitOnStandardFault="yes|no"?
 xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"">
 <partnerLinks>
   ...
 </partnerLinks>
 <variables>
   ...
 </variables>
 <sequence>
   ...
 </sequence>
  activity
</process>
```

Listing 4.1: Esqueleto de um processo BPEL.

A seguir descreveremos cada uma das seções demonstradas na listagem de código acima.

<partnerLinks>: Esta seção define os elementos responsáveis por definir o conjunto de parceiros de negócios (*PartnerLink*) utilizados em uma descrição de processo BPEL, identificando desta maneira qual funcionalidade deve ser oferecida por cada serviço parceiro (ALVES *et al.*, 2006).

O elemento *PartnerLink* ilustrado na , estabelece um canal de comunicação direto entre os parceiros de serviço internos ou externos que atuam na execução de um processo BPEL. As ligações dos serviços parceiros (*partner link*) devem estar associadas a um tipo de ligação entre parceiros (*partner link type*) definidos na especificação do *web services* no WSDL. Dependendo da necessidade da comunicação entre parceiros de serviços, o papel de um parceiro pode variar(ALVES *et al.*, 2006).

Um parceiro de serviço pode ser invocado por um serviço do processo e atuar com o papel de provedor de serviços (*service provider*). Entretanto, se o mesmo serviço do processo invoca um serviço diferente, ele atuará como solicitante de um serviço (*service requesters*). A definição do papel de um *partnerLink* é definida através dos atributos *myRole* e *partnerRole*, que estabelecem o papel de provedor de serviço ou serviço associado respectivamente. A listagem de código a seguir demonstra a sintaxe de uma atividade *partnerLink* (ALVES *et al.*, 2006).

```
1 <partnerLinks>
2  <partnerLink name="NCName"
3   partnerLinkType="QName"
4   myRole="NCName"?
5   partnerRole="NCName"?
6   initializePartnerRole="yes|no"? />+
7 </partnerLinks>
```

Listing 4.2: Sintaxe da atividade patnerLink.

<variables>: Esta seção define as variáveis dos dados usadas pelo processo de negócio, para manter as mensagens que constituem uma parte do estado de um processo negócio. As mensagens que tem seu estado capturados pelas variáveis são na grande maioria das vezes as mensagens trocadas entre os parceiros de negócio (*patnerlink*). As definições são feitas em termos de tipos de mensagem WSDL, elementos ou tipos simples de esquemas XML (ALVES *et al.*, 2006). Elas são usadas para manter os dados de estado e o histórico do processo com base nas mensagens trocadas.

As variáveis devem estar associadas a tipos de mensagens (*menssages*) definidos na especificação WSDL. Variáveis podem conter dados que são necessários para a realização de estado relacionado com o processo e nunca trocada com parceiros. WS-BPEL usa três tipos de de-

clarações de variáveis: tipo de mensagem WSDL, XML Schema tipo (simples ou complexa) e XML *Schema* elemento. A listagem a segui demonstra a sintaxe da atividade <variables>.

```
<variables>
 <variable name="BPELVariableName"
  messageType="QName"
  type="QName"
  element="QName">
  from-spec
 </variable>
</variables>
```

Listing 4.3: Sintaxe da atividade variables.

<faultHandlers>: Esta seção é responsável por declarar todos os manipuladores de falhas do processo, os manipuladores de falha definem as atividades que devem ser executadas em resposta a falhas resultantes da invocação dos serviços de avaliação e aprovação. Em WS-BPEL, todas as falhas, seja ele interno ou resultantes de uma chamada de serviço, são identificados por um nome qualificado. Em particular, cada falha WSDL é identificada em WS-BPEL por um nome especifico formado pelo *namespace* de destino do documento WSDL em que o tipo de porta relevante e o motivo da falha são definidos (ALVES *et al.*, 2006). A listagem a seguir demonstra a sintaxe de um <faultHandlers>.

```
<faultHandlers>
 <catch faultName="QName"
 faultVariable="BPELVariableName">
 ...
 <catch>
  <catchAll>
  ...
  </catchAll>
 </catch>
</faultHandlers>
```

Listing 4.4: Sintaxe da atividade faultHandlers.

activities: Esta seção contém a descrição de todas as atividades que descrevem o comportamento geral do processo de negócio. O BPEL possui uma sintaxe com vários tipos de elementos, cuja conjunção consegue descrever todo o tipo de processo de negócio que um utilizador necessite. Assim como em qualquer outra linguagem de programação, a sintaxe da linguagem WS-BPEL, dispõe de primitivas básicas para declarar seus elementos em descrições de processos. Estas primitivas são chamadas de atividades(ALVES *et al.*, 2006). As atividades de um

processo BPEL são dividas em duas categorias: as atividades básicas e atividades estruturadas.

Atividades Básicas

Atividades básicas são atividades simples que não exige nenhuma lógica mais elaborada que são responsáveis por descreverem os passos elementares do comportamento de um processo (ALVES *et al.*, 2006). Estas atividades são vistas como um componente que interage com algo externo ao processo, como por exemplo a invocação de um serviço externo ao processo através de uma de suas construções.

Os comandos para definir uma atividade em uma descrição BPEL, são similares as *tags* da linguagem XML, onde são denotados por um único caractere ASCII ou uma string de caracteres entre os sinais <(menor que) e (maior que)>, geralmente estes os comandos são denotados com palavras em inglês. Grande parte das *tags* do BPEL, requerem seu fechamento. Então, tudo que estiver compreendido entre <comando> e </comando> será encarado como o corpo de um comando. A seguir descreveremos a semântica do conjunto de atividades básicas do padrão WS-BPEL 2.0.

<invoke>: esta atividade é usada para chamar uma operação em um dado *Web Services* oferecidos por prestadores de serviços a uma composição de serviços. Esta atividade pode incluir outras atividades básicas aninhadas, como manipulador de compensação e manipuladores de falhas. A listagem de código a seguir demonstra a sintaxe da atividade <invoke> (ALVES *et al.*, 2006).

```
<invoke partnerLink="NCName"
 portType="QName"?
 operation="NCName"
 inputVariable="BPELVariableName"?
 outputVariable="BPELVariableName"?
 standard-attributes> standard-elements
 <correlations>?
  <correlation set="NCName" initiate="yes|join|no"?
  pattern="request|response|request-response"? />+
 </correlations>
 <catch faultName="QName"?
  faultVariable="BPELVariableName"?
  faultMessageType="QName"?
  faultElement="QName"?>*
  activity
 </catch>
 <catchAll>?
```

```
    activity
  </catchAll>
  <compensationHandler>?
    activity
  </compensationHandler>
</invoke>
```

Listing 4.5: Sintaxe da atividade invoke.

<receive>: O elemento *receive* permite que um serviço do processo permaneça em estado de espera, agindo como um prestador de serviço, enquanto o processo recebe um pedido de um parceiro de serviço externo. Esta atividade específica um *PartnerLink* que contenha um *myRole* usado para receber mensagens, o *portType* (opcional) é a operação que espera que o parceiro invoca que (ALVES *et al.*, 2006). Este elemento possui um conjunto de atributos que atribui valores a uma comunicação esperada, são eles:

- *partnerLink*: define o *PartnerLink* correspondente, ao parceiro de serviço que esta participando da troca de mensagens do processo.
- *portType*: define qual o parceiro de serviço que o processo espera receber uma mensagem de requisição.
- *operation*: define a operação de um serviço que vai receber uma mensagem de um outro processo.
- *variable*: define qual variável vai armazenar a mensagem de solicitação.
- *createInstance*: este atributo é definido com um valor booleano, caso seja definido como "SIM", uma nova instancia do processo de troca de mensagem sera criada, para que seja possível receber os dados da ou das mensagens trocadas entre os vários parceiros do processo, caso seja definido como não o processo não poderá receber mensagens pois não haverá uma instancia de uma atividade para receber os dados da ou das mensagens trocadas entre os parceiros (ALVES *et al.*, 2006). A listagem a seguir demonstra a sintaxe da atividade <receive>.

```
<receive partnerLink="NCName"
  portType="QName"?
  operation="NCName"
  variable="BPELVariableName"?
  createInstance="yes|no"?
  messageExchange="NCName"?
```

```
 standard-attributes>
 standard-elements
 <correlations>?
  <correlation set="NCName" initiate="yes|join|no"? />+
 </correlations>
 <fromParts>?
  <fromPart part="NCName" toVariable="BPELVariableName" />+
 </fromParts>
</receive>
```

Listing 4.6: Sintaxe da atividade receive.

Existem outras atividades básicas na linguagem WS-BPEL, estas são listadas a seguir:

- <reply>: Esta atividade é usada para enviar uma resposta a um pedido previamente aceito através de uma mensagem da atividade <receive>. Uma atividade <reply> pode especificar um atributo variável que faz referência a variável que contém os dados da mensagem a ser enviada (ALVES *et al.*, 2006) . A listagem a seguir demonstra a sintaxe da atividade <reply>.

- <wait>: Esta atividade é especificada quando a necessidade de utilizar um temporizador para realiza uma pausa por um período de tempo ou data especificado é atingido. Se o período de tempo especificado for zero ou nulo então a atividade <Wait> é completada imediatamente (ALVES *et al.*, 2006).

- <assign>: A atividade <assign> é utilizada para copiar dados de uma variável para outra, assim como também, para a construção de novas expressões e inserir dados utilizando expressões. O uso de expressões é motivado pela necessidade de realizar cálculos simples (como incrementar os números de sequência) (ALVES *et al.*, 2006). Expressões podem operar sobre variáveis, propriedades, constantes e literais para produzir um novo valor. A listagem a seguir demonstra a sintaxe da atividade <assign> .

- <throw>: Esta atividade é utilizada quando um processo de negócio precisa sinalizar uma falha interna que ocorrera durante o seu fluxo de execução. A atividade <throw> fornece o nome para a falha, e pode, opcionalmente, fornecer dados com informações sobre a mesma. Um manipulador de falhas pode usar estes dados para lidar com a falha e para preencher quaisquer mensagens de falha que precisam ser enviados para outros serviços (ALVES *et al.*, 2006). A listagem a seguir demonstra a sintaxe da atividade <throw>.

- <terminationHandler>: A rescisão forçada de um escopo começa desativando manipuladores do escopo do evento e encerra a sua atividade principal e todas as instâncias de

manipulador de funcionamento do evento. Depois disso, o <terminationHandler> personalizado para o âmbito de aplicação, se estiver presente, é executado. Caso contrário, o manipulador de encerramento padrão é executado (ALVES *et al.*, 2006). A listagem a seguir demonstra a sintaxe da atividade <terminationHandler>.

- <compensate>: Esta atividade é usada para iniciar a compensação de falhas em todos os âmbitos internos que já tenham terminado a sua execução por completo com sucesso de modo padrão (ALVES *et al.*, 2006). A listagem a seguir demonstra a sintaxe da atividade <terminationHandler>.

Para construir atividades mais complexas ou estruturadas no padrão WS-BPEL 2.0 é preciso combinar as primitivas básicas com atividades de controle.

Atividades Estruturadas

As atividades estruturadas descrevem como um processo de negócio pode ser criado a partir da composição de atividades básicas. Compor atividades básicas permite que um processo expresse: fluxos de controle, tratamento de exceção, chamada a eventos externos ao processo, assim como também coordenar as trocas de mensagens entre as partes envolvidas em um processo de negócio (ALVES *et al.*, 2006).

É possível definir atividades estruturadas através de fluxos de controle. Um controle sequencial é fornecido pelas atividades <sequence>, <if>, <while>, <repeatUntil> e <forEach>. É possível também definir concorrência e sincronização entre atividades de um determinado processo, esta atividade estruturada é assegurada pela atividade <flow> e uma variação paralela da atividade <forEach>. A escolha de um serviço especifico por um evento interno ou externo ao processo em um fluxo também é possível através da atividade <pick>.

Em alguns casos é possível representar a semântica de uma atividade específica através da combinação de outras atividades, por exemplo, uma sequência pode ser descrita através da atividade <sequence>, entretanto a combinação da atividade de controle <flow> com atividades básicas como <receive> por exemplo definindo as ligações de um fluxo corretamente, é possível descrever sequências sem uso da atividade nativa <sequence>. Desta maneira é possível combinar atividades arbitrariamente para aumentar a expressividade da linguagem (ALVES *et al.*, 2006). A seguir descreveremos as atividades estruturadas do padrão WS-BEPL2.0.

<sequence>: A atividade <sequence> é responsável por expressar a sequencia de uma ou mais atividades em ordem léxica. Esta atividade só finaliza sua execução depois que a ultima

atividade de seu escopo é executada. O código abaixo demonstra a sintaxe da atividade de controle <sequence> (ALVES *et al.*, 2006). A listagem a seguir demonstra a sintaxe da atividade <sequence>.

```
<sequence standard-attributes>
 standard-elements
 activity+
</sequence>
```

Listing 4.7: Sintaxe da atividade sequence.

<if>: A atividade de controle <if>, define um comportamento condicional para um fluxo a partir da avaliação de uma expressão booleana. É possível aninhar varias condições utilizando a atividade <if> ou o opcional <elseif>, seguidos por um elemento <else> opcional, que servirá como elemento default em caso de nenhuma das opções aninhadas serem avaliadas verdadeiramente. A partir da a avaliação verdadeira de uma expressão a atividade envolvida pelo escopo da atividade <if> é executada, caso contrario o elemento <else> ou <elseif> são executados em sequencia. Esta atividade de controle só termina quando as atividades executadas através das avaliações das expressões são encerradas, ou imediatamente quando uma <condição> não é avaliada como true e nenhum ramo <else> é especificado (ALVES *et al.*, 2006). A listagem a seguir demonstra a sintaxe da atividade <if>.

```
<if standard-attributes>
 standard-elements
 <condition expressionLanguage="anyURI"?>bool-expr</condition>
  activity
 <elseif>*
 <condition expressionLanguage="anyURI"?>bool-expr</condition>
  activity
 </elseif>
 <else>?
  activity
 </else>
</if>
```

Listing 4.8: Sintaxe da atividade if.

<while>: Assim como nas linguagens de programação tradicionais a atividade <while> no BPEL 2.0 representa a repetição de uma ou mais atividades em seu escopo, onde a execução de uma iteração é executada após uma expressão booleana ser avaliada verdadeira. (ALVES *et al.*, 2006). A listagem a seguir demonstra a sintaxe da atividade <while>.

```
<while standard-attributes>
 standard-elements
 <condition expressionLanguage="anyURI"?>bool-expr</condition>
  activity
</while>
```

Listing 4.9: sintaxe da atividade while.

<repeatUntil>: Assim como <while>, a atividade de controle <repeatUntil> também representa uma execução repetida de uma ou mais atividades em seu escopo, esta repetição é executada ate que a avaliação de uma dada expressão booleana deixe de ser verdadeira, a avaliação da expressão booleana é feita depois de cada iteração desta atividade de controle, esta atividade é amplamente utilizada em situações que é necessária uma estrutura de controle que execute uma repetição pelo menos uma vez (ALVES *et al.*, 2006). A listagem a seguir demonstra a sintaxe da atividade <repeatUntil>.

```
<repeatUntil standard-attributes>
 standard-elements
  activity
 <condition expressionLanguage="anyURI"?>bool-expr</condition>
</repeatUntil>
```

Listing 4.10: Sintaxe da atividade repeat until.

<forEach>: A atividade <forEach> também pode executar varias atividades em paralelo, controlada por duas expressões <startCounterValue> e <finalCounterValue>, ao iniciar, esta atividade avalia estas duas expressões. Uma vez que estas expressões sejam avaliadas verdadeiramente, a atividade <forEach> irá executar sua atividade <scope> contida exatamente N+1 vezes, onde n é igual ao <finalCounterValue> menos o <startCounterValue>. Se essas expressões não retornam valores válidos, o processo BPEL: lança uma *invalidExpressionValue*, e a atividade termina sua execução, ou caso o <startCounterValue> seja maior do que o <finalCounterValue>, a atividade <scope> não deve ser realizada e a atividade <forEach> está completa (ALVES *et al.*, 2006). A listagem a seguir demonstra a sintaxe da atividade <forEach>.

```
<forEach counterName="BPELVariableName" parallel="yes|no"
 standard-attributes>
 standard-elements
 <startCounterValue expressionLanguage="anyURI"?>
  unsigned-integer-expression
 </startCounterValue>
 <finalCounterValue expressionLanguage="anyURI"?>
  unsigned-integer-expression
```

```
9  </finalCounterValue>
10 <completionCondition>?
11  <branches expressionLanguage="anyURI"?
12   successfulBranchesOnly="yes|no"?>?
13   unsigned-integer-expression
14  branches>
15 </completionCondition>
16 <scope ...>...</scope>
17 </forEach>
```

Listing 4.11: Sintaxe da atividade forEach

Existem outras atividades estruturadas na linguagem WS-BPEL, estas são listadas a seguir:

- <pick>: Esta atividade de controle espera pela ocorrência de um evento específico em um conjunto de eventos, uma vez que este evento específico ocorre, imediatamente a atividade associada a este evento é executada. Uma vez que a atividade <pick> aceita um evento, ela não mais aceita eventos daquele conjunto de eventos. Esta atividade dispõe de um conjunto de ramos onde em cada um contendo uma associação evento-atividade (ALVES *et al.*, 2006). A atividade <pick> termina quando a atividade associada ao evento termina sua execução.

- <flow>: A atividade <flow> é responsável por promover sincronização e concorrência entre duas ou mais atividade, de acordo com a necessidade do processo de negócio. Com esta atividade, serviços específicos em um processo de negócio podem ser executados paralelamente, ou simultaneamente (ALVES *et al.*, 2006). A listagem a seguir demonstra a sintaxe da atividade <flow>.

4.2 WS-BPEL no contexto π-SOD-M

Em construção !!!

4.3 Adaptação *π-Sercive Composition Model*

Em construção !!!

4.4 Transformação PIM-PSM (WS-BPEL)

Em construção !!!

4.4.1 Regras de Transformação (*Plugin Module*)

O processo de transformação entre modelos consiste em especificar uma entidade de um modelo fonte como entrada, e transforma-lo em uma entidade do modelo de destino tendo o resultado da transformação como saída (NETO, 2012). Tanto o modelo de entrada como o modelo de saída devem estar de acordo com as regras de modelagem de seus meta-modelos. Existem diversos tipos de transformação de modelos que diferem nas suas entradas e saídas e também no modo como são expressas (MILLER; MUKERJI, 2003).

Transformações entre modelos também é uma forma de garantir a consistência de um conjunto de modelos de uma plataforma em especifico, de acordo com a modelagem proposta por um engenheiro de software. O objetivo final do emprego de transformações entre modelos no desenvolvimento de aplicações é reduzir o esforço empregado na construção de um software, assim como também o número de erros. Este processo automatiza a construção, modificação e documentação de um software sempre que necessário (JOUAULT; KURTEV, 2006).

Em π-SOD-M, (Neto) define transformações verticais entre modelos em dois níveis: de modelos PIM's para modelos PSM's e de modelos PSM's para código de maquina executável. Três modelos de transformações são definidos em π-SOD-M: π-Use Case para π-*Service Process* (PIM para PIM); π-*Service Process* para π-*Service Composition* (PIM para PIM); e π-*Service Composition* para π-PEWS (PIM para PSM). Nós propomos aumentar o número de modelos de transformações, expandindo o leque de plataforma especifica do método π-SOD-M, adicionando um modelo de transformação no nível de PSM para a linguagem WS-BPEL 2.0, complementando o método com o modelo de transformação π-*Service Composition* para WS-BPEL.

Todas as regras de transformações entre modelos propostas na metodologia π-SOD-M são descritas em linguagem natural (Neto). Estas regras asseguram a corretude entre os conceitos especificados em cada nível de abstração proposto pela metodologia, assim como também, no refinamento e processamento das transformações em diferentes níveis. A Figura 4.1 ilustra o conjunto de tipos de regras que são definidos na metodologia π-SOD-M para cada tipo de transformação.

As entidades ilustradas ao lado esquerdo da figura representam os elementos do meta-

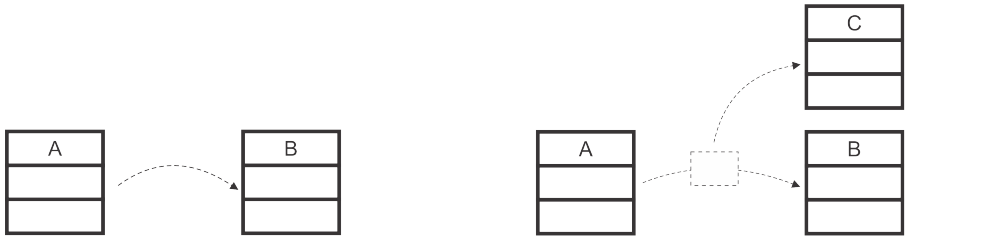

Figura 4.1: Modelo de regras de transformação entra entidades.

modelo fonte enquanto que as entidades ilustradas do lado direito representam os elementos do modelo de destino. A Figura 4.1a ilustra a regra de transformação em que uma única entidade de origem é transformada numa entidade do modelo alvo. As regras ilustradas na figura 4.1b são mais avançadas, onde um elemento de origem é transformado em dois ou mais elementos diferentes no modelo alvo, por exemplo um elemento "A", pode gerar um elemento "B"e um elemento "C"no modelo alvo.

Em construção!!!!

Mapeando uma Action no π-SCM para Role no BPELexecutable

Mapeando um Service Activity no π-SCM para um PatnerLinkType no BPELexecutable

Mapeando um Control Nodes no π-SCM para um Structured Activity no BPELexecutable

- Caso uma sequencia de operações seja especificada um Control Node será mapeado para uma atividade estruturada sequence;
- Caso um Control Node seja especificado como uma execução de várias operações em paralelo ou a junção do resultado de várias operações em paralelo, sera mapeado para um flow;
- Caso um Control Node seja especificado como como uma escolha (Choice) entre duas operações será mapeado para uma atividade estruturada Pick.

Mapeando um Business Colaborator no π-SCM para um PatnerLink no BPELexecutable

Um Business Colaborator no modelo fonte é transformado para um PatnerLink no modelo alvo, ambas as estruturas são colaboradores de negócios internos e externos ao processo.

Mapeando um Rule:Condition no π-SCM para um Sequence Invoke PortTypes (operações) no BPELexecutable

Um Rule:Condition no modelo fonte é um conjunto de operações com um nome especifico, por exemplo a Rule:Condition buyMusic é a composição das operações pagamento e verificação de pagamento, então mapeamos para o modelo alvo como uma sequencia de operações especificas de parceiro de negocio.

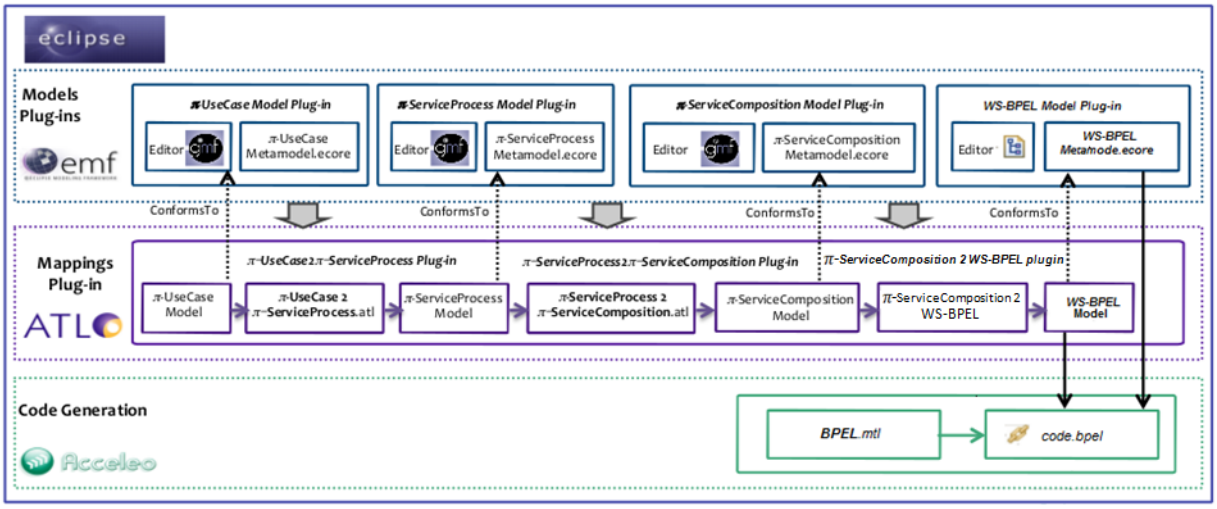

Figura 4.2: Ambiente de desenvolvimento π-SOD-M mais o complemento para a plataforma BPEL. Adaptado de (NETO, 2012)

A Figura 4.1 ilustra os três componentes que formam o ambiente de desenvolvimento do π-SOD-M. Como pode ser observado, todo o ambiente é executado na IDE Eclipse. O primeiro componente é o plugin de modelagem (*Models Plug-in*), este compreende os componentes que descrevem os meta-modelos π-SOD-M e como estes modelos devem ser criados. Existem quatro componentes de meta-modelagem, onde todos os componentes do módulo do plug-in de mapeamento (*Mappings Plugins*) dependem das definições feitas nesta camada (NETO, 2012).

Quando uma transformação de modelo é executada, os modelos criados devem estar em conformidade com seus respectivos meta-modelos. Cada vez que uma transformação é executada, é realizada uma verificação de consistência do modelo fonte e do modelo alvo, para garantir a coerência entre as transformações dos modelos. Uma vez que todas as transformações de modelos forem completadas o modelo PSM estará pronto, e poderá ser traduzido em código de uma plataforma especifica em particular.

Originalmente π-SOD-M, leva a transformação de modelo para código PEWS. Com nosso complemento, o π-SOD-M aumentará o leque de opções tendo como possibilidade de transformação de plataforma específica à linguagem de orquestração de serviços WS-BPEL 2.0. A

transformação do modelo PSM em código é a última fase do π-SOD-M. O componente de geração de código *Code Generation* depende do PSM gerado pelo último componente de transformação.

4.5 Comparação da geração de código WS-BPEL e π-PEWS

Em construção !!!

5 *Avaliação (Provas de Conceito)*

Em construção!!!

5.1 Estudo de Caso

Em construção!!!

5.1.1 To Publish Music

Refazendo!!!

5.1.2 Crime Map

Em construção!!!

5.1.3 GesIMED

Em construção!!!

5.2 Resultados

Atualmente empresas disponibilizam seu serviços pela internet para facilitar a troca de informações entre empresas e seus clientes . Partindo desta afirmação aplicações são construídas através da composição de serviços distintos de acordo com as necessidades (requisitos) da empresa. Para que estas aplicações garantam algum tipo de qualidade, as necessidades ou requisitos especificados pela empresa devem ser perfeitamente implementados para que haja uma satisfação subjetiva da empresa que requiriu o sistema. Atingir certo ponto de coerência entre requisitos e implementação não é uma atividade trivial para desenvolvedores. Além disso, os

desenvolvedores contam com um grande número de linguagens que orquestram serviços para construir tais aplicações, como por exemplo WS-BPEL 2.0.

Este trabalho tem como objetivo principal complementar a metodologia π-SOD-M, para que o mesmo forneça um meio de especificar aplicações baseadas em serviços na linguagem de orquestração de serviços WS-BPEL 2.0. O trabalho libera os desenvolvedores de detalhes específicos da plataforma WS-BPEL promovendo a especificação de aplicações através da modelagem, permitindo que requisitos funcionais sejam modelados através de composição de serviços. Estes requisitos são parte de uma especificação de processos de negócios, ao longo das fases da metodologia, as regras de negócio e assim como as restrições são modeladas utilizando conceitos mais concretos que conduzam a sua implementação em código executável.

Os benefícios de se trabalhar a partir destes resultados propõe um meio de fornecer uma metodologia orientada software que podem ser incluídas em diferentes fases de desenvolvimento, como na especificação das propriedades funcionais, na sua modelagem e implementação. Desta maneira, este trabalho complementa π-SOD-M proporcionando um ambiente de modelagem de aplicações composta por serviços para a plataforma WS-BPEL 2.0, este mesmo ambiente é capaz de transformar modelos criados através da modelagem em BPEL e transforma-los em código de aplicações executáveis.

5.3 Principais contribuições

As principais contribuições desse trabalho são as:

- A integração das atividades de modelagem de composição de serviços em WS-BPEL 2.0, arquitetura orientada a serviço através de transformações baseadas em modelos
- A geração de um meta-modelos WS-BPEL 2.0.
- Especificação de regras de transformação entre π-ServiceCompositionModel e WS-BPEL 2.0, procurando expressar, em todas as atividades, as informações contidas na atividade predecessora.
- Implementação das regras de transformação em ATL permitindo que ocorra um processo automatizado de transformações entre os modelos da atividade supracitada. O processo automatizado permite que sejam mantidas as informações
- O complemento de um ambiente integrado, π-SOD-M, que permite a criação de modelos com base em metamodelos bem definidos e regras de transformação entre seus elementos

agora para a plataforma WS-BPEL2.0. Tal ambiente facilita o acesso às informações aos modelos gerados pela metodologia e originados durante o processo de desenvolvimento, permitindo que requisitos possam ser adicionados e propagados facilmente para a construção de aplicações em WS-BPEL 2.0. Além disso, oferece suporte a rastreabilidade entre os modelos.

- A validação das regras de transformação usando o estudos de caso, o To Publish Music, usado para avaliar se os requisitos estão de acordo com as estratégias de composições geradas orientadas a serviços.

5.4 Trabalhos futuros

À pesquisa desenvolvida nessa dissertação, tem alguns trabalhos futuros que podem ser relacionados, tais com:

- A extensão do meta-modelo WS-BPEL 2.0 para que o mesmo dê suporte a politicas de serviços web;
- Implementar alterações em uma das *engines* de código livre existente para suportar a execução de composições com politicas de serviço.
- Utilização de métricas para avaliar os resultados das transformações criadas, e a conformidade entre os modelos gerados. As possíveis métricas a serem utilizadas como base, são as fornecidas pelo documento *Model Driving Development* da *Engineering metrics Baseline*.
- Especificação e construção de um ambiente de configuração e reconfiguração dinâmica que providencie suporte a transformações orientadas a serviços entre as fases do ciclo de desenvolvimento de software.

Referências Bibliográficas

ALVES, Alexandre; ARKIN, Assaf; ASKARY, Sid; BLOCH, Ben; CURBERA, Francisco; GOLAND, Yaron; KARTHA, Neelakantan; STERLING; KöNIG, Dieter; MEHTA, Vinkesh; THATTE, Satish; RIJN, Danny van der; YENDLURI, Prasad; YIU, Alex. *Web Services Business Process Execution Language Version 2.0*. maio 2006. OASIS Committee Draft.

ARSANJANI, Ali; GHOSH, Shuvanker; ALLAM, Abdul; ABDOLLAH, Tina; GANAPATHY, Sella; HOLLEY, Kerrie. Soma: A method for developing service-oriented solutions. *IBM systems Journal*, IBM, v. 47, n. 3, p. 377–396, 2008.

BA, C.; FERRARI, M.H.; MUSICANTE, M. Building web services interfaces using predicate path expressions. *Proceedings of SBLP*, p. 147–160, 2005.

BA, Cheikh; FERRARI, Mirian Halfeld; MUSICANTE, Martin A. Composing web services with pews: A trace-theoretical approach. In: *Web Services, 2006. ECOWS '06. 4th European Conference on*. [S.l.: s.n.], 2006. p. 65 –74.

BOOTH, David; HAAS, Hugo; MCCABE, Francis; NEWCOMER, Eric; CHAMPION, Michael; FERRIS, Chris; ORCHARD, David. *Web Services Architecture*. [S.l.], fev. 2004. Disponível em: <http://www.w3.org/TR/2003/WD-ws-arch-20030808>.

BOOTH, David; MCCABE, Francis; CHAMPION, Michael. *Web Services Architecture - W3C Working Group Note 11 February 2004*. [S.l.]. Disponível em: <http://www.w3.org/TR/2004-/NOTE-ws-arch-20040211/>.

BROWN, Alan; JOHNSTON, Simon K; LARSEN, Grant; PALISTRANT, Jim. Soa development using the ibm rational software development platform: a practical guide. *Rational Software*, 2005.

CASTRO, Valeria De; MARCOS, Esperanza; WIERINGA, Roel. Towards a service-oriented mda-based approach to the alignment of business processes with it systems: from the business model to a web service composition model. *International Journal of Cooperative Information Systems*, World Scientific, v. 18, n. 02, p. 225–260, 2009.

Introduction to special issue on service oriented computing (soc). *ACM Trans. Web*, ACM, New York, NY, USA, v. 2, n. 2, p. 10:1–10:2, maio 2008. ISSN 1559-1131. Disponível em: <http://doi.acm.org/10.1145/1346337.1346338>.

FEIG, Ephraim; ZHANG, Liang-Jie (LJ); ARSANJANI, Ali; XU, Zhiwei. Services computing in action: Services architectures. In: *Web Services, 2007. ICWS 2007. IEEE International Conference on*. [S.l.: s.n.], 2007. p. xxxiv.

JOUAULT, Frédéric; KURTEV, Ivan. Transforming models with atl. In: *Proceedings of the 2005 international conference on Satellite Events at the MoDELS*. Berlin, Heidelberg:

Springer-Verlag, 2006. (MoDELS'05), p. 128–138. ISBN 3-540-31780-5, 978-3-540-31780-7. Disponível em: <http://dx.doi.org/10.1007/11663430_14>.

KREGER, H. *Web Services Conceptual Architecture (WSCA 1.0)*. [S.l.], maio 2001. Disponível em: <http://www.csd.uoc.gr/~hy565/newpage/docs/pdfs/papers/wsca.pdf>.

LEYMANN, Frank. *Web Services Flow Language (WSFL 1.0)*. [S.l.], maio 2001.

MILLER, J.; MUKERJI, J. *MDA Guide Version 1.0.1*. [S.l.], 2003.

NETO, Plácido Antônio de Souza. *A Methodology for Building Reliable Service-Based Applications*. Tese (Doutorado) — Universidade Federal do Rio Grande do Norte, Departamento de Informática e Matemática Aplicada, Natal-RN, 2012.

PAPAZOGLOU, M.P.; TRAVERSO, P.; DUSTDAR, S.; LEYMANN, F. Service-oriented computing: State of the art and research challenges. *Computer*, v. 40, n. 11, p. 38 –45, nov. 2007. ISSN 0018-9162.

PAPAZOGLOU, M. P.; GEORGAKOPOULOS, D. Introduction: Service-oriented computing. *Commun. ACM*, ACM, New York, NY, USA, v. 46, n. 10, p. 24–28, out. 2003. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/944217.944233>.

PAPAZOGLOU, Mike P.; HEUVEL, Willem-Jan. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 16, n. 3, p. 389–415, jul. 2007. ISSN 1066-8888. Disponível em: <http://dx.doi.org/10.1007/s00778-007-0044-3>.

PAPAZOGLOU, Michael P; HEUVEL, Willem-Jan Van Den. Service-oriented design and development methodology. *International Journal of Web Engineering and Technology*, Inderscience, v. 2, n. 4, p. 412–442, 2006.

PAPAZOGLOU, Michael P.; HEUVEL, Willem-Jan Van Den. Service-oriented design and development methodology. *International Journal of Web Engineering and Technology*, Inderscience Publishers, Geneva, Switzerland, v. 2, n. 4, p. 412–442, jul. 2006. ISSN 1476-1289. Disponível em: <http://www.inderscience.com/offer.php?id=10423>.

PAPAZOGLOU, Michael P.; HEUVEL, Willem-Jan Van Den. Service-oriented design and development methodology. *Int. J. Web Eng. Technol.*, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, v. 2, n. 4, p. 412–442, jul. 2006. ISSN 1476-1289. Disponível em: <http://dx.doi.org/10.1504/IJWET.2006.010423>.

PAPAZOGLOU, Michael P.; TRAVERSO, Paolo; DUSTDAR, Schahram; LEYMANN, Frank; KRÄMER, Bernd J. 05462 service-oriented computing: A research roadmap. In: CUBERA, Francisco; KRÄMER, Bernd J.; PAPAZOGLOU, Michael P. (Ed.). *Service Oriented Computing (SOC)*. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. (Dagstuhl Seminar Proceedings, 05462). ISSN 1862-4405. Disponível em: <http://drops.dagstuhl.de/opus/volltexte/2006-/524>.

PAPAZOGLOU, Michael P.; TRAVERSO, Paolo; DUSTDAR, Schahram; LEYMANN, Frank. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 40, n. 11, p. 38–45, nov. 2007. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/mc.2007.400>.

PELTZ, C. Web services orchestration and choreography. *Computer*, v. 36, n. 10, p. 46 – 52, oct. 2003. ISSN 0018-9162.

TSALGATIDOU, Aphrodite; PILIOURA, Thomi. An overview of standards and related technology in web services. *Distrib. Parallel Databases*, Kluwer Academic Publishers, Hingham, MA, USA, v. 12, n. 2-3, p. 135–162, set. 2002. ISSN 0926-8782. Disponível em: <http://dx.doi.org/10.1023/A:1016599017660>.

WATSON, Andrew. A brief history of mda. *Upgrade, the European Journal for the Informatics Professional*, v. 9, n. 2, p. 7–11, 2008.

WESLEY, Ajamu. *WSFL in action, Part 1:When Web services cooperate.* 2002. Excerpted from 5th edition of the APA Publication Manual. Disponível em: <http://www.ibm.com-/developerworks/webservices/library/ws-wsfl1/>.

ZHANG, Li; JIANG, Wei. Transforming business requirements into bpel: A mda-based approach to web application development. In: *Semantic Computing and Systems, 2008. WSCS '08. IEEE International Workshop on*. [S.l.: s.n.], 2008. p. 61 –66.