UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE ELETRICIDADE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

Eduardo Devidson Costa Bezerra

COMPOSIÇÃO DINÂMICA DE SERVIÇOS WEB SEMÂNTICOS UTILIZANDO ABORDAGENS DA ENGENHARIA DIRIGIDA POR MODELOS

São Luís
2011

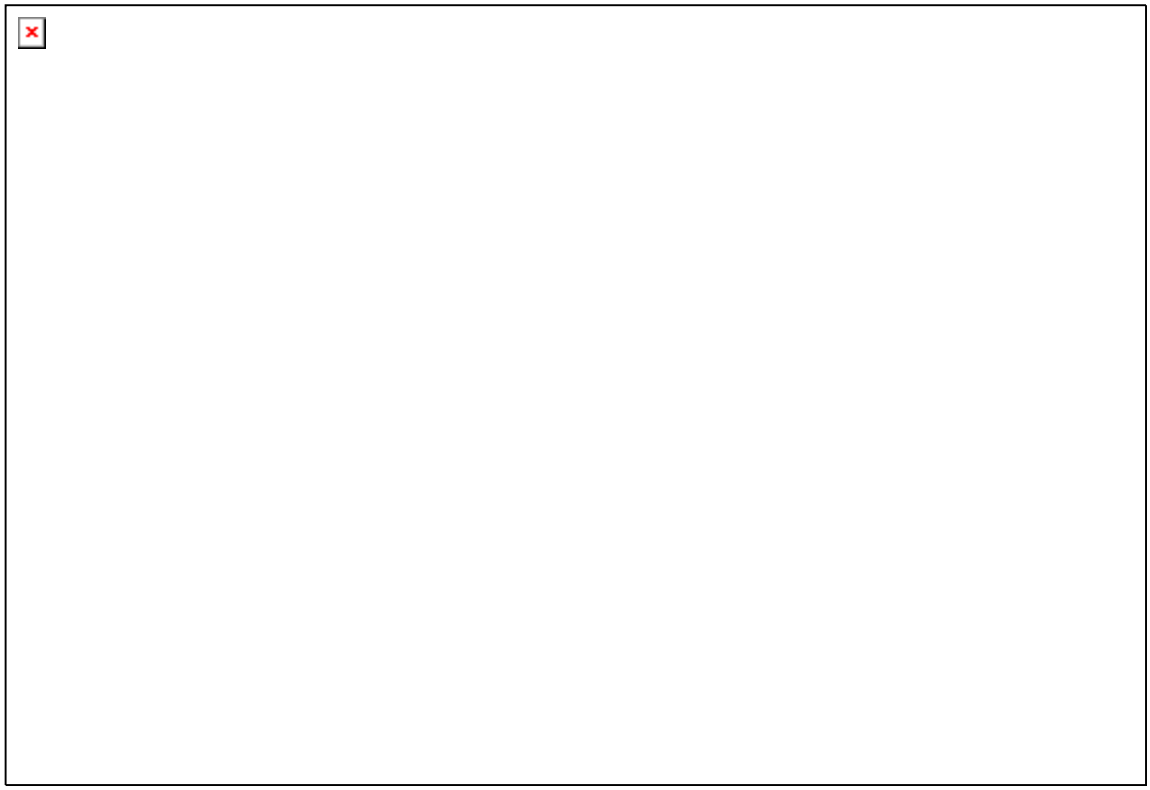
Eduardo Devidson Costa Bezerra

COMPOSIÇÃO DINÂMICA DE SERVIÇOS WEB SEMÂNTICOS UTILIZANDO ABORDAGENS DA ENGENHARIA DIRIGIDA POR MODELOS

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão para obtenção do título de Mestre em Engenharia de Eletricidade, área de Concentração Ciência da Computação.

Orientador: Ph.D. Zair Abdelouahab
Co-orientador: Dr. Denivaldo Cicero Pavão Lopes

São Luís
2011



COMPOSIÇÃO DINÂMICA DE SERVIÇOS WEB SEMÂNTICOS UTILIZANDO ABORDAGENS DA ENGENHARIA DIRIGIDA POR MODELOS

Eduardo Devidson Costa Bezerra

Dissertação aprovada em 29 de julho de 2011.

Prof. Zair Abdelouahab, Ph.D.
(Orientador)

Prof. Denivaldo Cicero Pavão Lopes, Dr.
(Co-orientador)

Profa. Daniela Barreiro Claro, Dra.
(Membro da Banca Examinadora)

Prof. Ginalber Luiz de Oliveira Serra, Dr.
(Membro da Banca Examinadora)

Prof. Samyr Beliche Vale, Dr.
(Membro da Banca Examinadora)

Primeiramente a Deus por iluminar cada passo do meu caminho. Aos meus pais pela educação, aos meus irmãos, a toda minha família e ao meu filho por todos os dias me presentear com um sorriso.

Agradecimentos

Em primeiro lugar, agradeço a Deus pela sua força e perseverança que me tem abençoado.

Aos meus pais, por todos os ensinamentos e educação de berço que carrego para o resto da vida.

Aos meus irmãos, pela amizade e apoio que sempre posso contar.

Ao meu filho Enzo pelo amor puro que me tem agraciado e a minha mulher Luciana pelo apoio.

A toda minha família.

Ao professor Denivaldo Lopes por sua orientação, paciência, competência e dedicação.

Ao professor Zair Abdelouhab por sua orientação, apoio, sabedoria e serenidade.

Agradeço e dedico ao professor Alexandre Vidal pelo exemplo de força, luta e superação.

A todos os meus amigos do LESERC em especial a Michael, Paulo, Dayvid, Ruy e Kleberson pela força.

A todos os meus amigos do LABSAC em especial a Lianna, Jéssica, Neto, Vladimir pela força.

Aos antigos amigos da graduação em especial a Victor Hugo e Jone pela força.

A todos os professores do Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão, que foram de fundamental importância para a minha formação.

Aos meus colegas e chefe de trabalho pela compreensão.

A todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

A CAPES pelo suporte financeiro durante o período do mestrado.

Aos professores da banca examinadora.

“É humanamente impossível ter ao seu lado a força de Deus e sentir-se fraco.”

Autor Desconhecido

RESUMO

A utilização da internet como forma de publicar novas aplicações e disponibilizar novas funcionalidades tem consolidado o uso da tecnologia de serviços web. Uma vez que essa tecnologia viabiliza um alto grau de interoperabilidade e autonomia, a tecnologia de serviços web fornece uma infra-estrutura básica para o desenvolvimento e a composição de novos serviços, o que tem beneficiado à gerência de processos de negócio oferecendo assim, a agilidade necessária e requerida pelos empreendimentos frente à necessidade de rápidas mudanças no ambiente de negócios. Recentemente, novos paradigmas vêm sendo desenvolvidos para fazerem face à complexidade cada vez mais crescente no desenvolvimento, manutenção e evolução de softwares. Dentre eles a Engenharia Dirigida por Modelos (MDE – *Model Driven Engineering*) e as Ontologias se destacam como os mais promissores. Neste trabalho, apresenta-se uma abordagem para realizar a composição dinâmica de um novo serviço web utilizando técnicas de *Matching* de modelos (ou metamodelos) que representem serviços. Tal representação deverá conter aspectos semânticos e estruturais do serviço web conseguidos através de abordagens de Ontologias. Sendo assim, vislumbra-se estabelecer correspondências e medir o grau de similaridade entre modelos investindo na pesquisa de MDE e Ontologias com o intuito de gerar uma ferramenta que possa auxiliar na Composição Dinâmica de um Serviço. Para validar a abordagem, um estudo de caso será apresentado.

Palavras-chave: Composição de Serviços Web, Metamodelagem, Ontologias, Engenharia Dirigida por Modelos, *Matching*.

ABSTRACT

The use of the Internet as a way to publish new applications and deliver new functionalities has consolidated the use of web services technology. Once web services technology enables a high degree of autonomy and interoperability, it provides a basic infrastructure for the development and composition of new services, which has benefited the business process management providing the agility required by enterprises to meet the need of rapidly changing business environment. Recently, new paradigms are being developed to deal with the increasing complexity in the development, maintenance and evolution of software systems. Among the new paradigms, Model Driven Engineering (MDE) and the Ontology stands out as the most promising for handling complex software systems. This work presents an approach to perform the dynamic composition of web services using techniques of match models (metamodels) that represent services. Models representing services must include semantic and structural aspects of the web service achieved through approaches to ontologies. Thus, we conjecture to establish matchings and measure the degree of similarity between models and investing in research about Ontologies and MDE in order to generate a tool that can assist in a Dynamic Composition of Web Services. A case study is presented to illustrate this approach.

Keywords: Web Service Composition, Metamodels, Ontologies, Model Driven Engineering, Matching.

LISTA DE FIGURAS

Figura 1 – Triângulo SOA 27
Figura 2 – Comunicação entre a composição BPEL e os Partner Links 30
Figura 3 – Nível Superior da Ontologia OWL-S 33
Figura 4 – EMF unifica Java, XML e UML 37
Figura 5 – Modelo Ecore e suas Fontes 37
Figura 6 – Atributos de QoS 41
Figura 7 – A Arquitetura Multicamada em WebTransact 42
Figura 8 – Código WSDL e extensão WSTL 43
Figura 9 – Ontologia do domínio de procura de preço e instâncias de serviços 46
Figura 10 – Visão Geral da Abordagem 48
Figura 11 – Arquitetura do Protótipo do Sistema 49
Figura 12 – Script XSLT para criação de descrições OWL-S 53
Figura 13 – Arquitetura de OWL-S Composer 2.0 55
Figura 14 – Arquitetura de Web Services 57
Figura 15 – Fluxo de composição baseado no modelo MVC 58
Figura 16 – Arquitetura de Web Services com suporte a composição dinâmica 59
Figura 17 – Níveis de Abstração dos Elementos da Abordagem 66
Figura 18 – Geração de Ferramentas e Artefatos 68
Figura 19 – Diagrama de Classes do Framework 69
Figura 20 – Metodologia do Framework 71
Figura 21 – Estrutura Geral da OWL-S Profile 73
Figura 22 – Metamodelo Semântico de Serviços 76
Figura 23 – Metamodelo de Matching 77
Figura 24 – Metamodelo de Mapeamento (sMatch) 78
Figura 25 – Probabilidades no Modelo Semântico 81
Figura 26 – Fluxo do Algoritmo de Matching 82
Figura 27 – Tratamento de Qualidades no Algoritmo de *Matching* 84
Figura 28 – Gráfico fuzzy para representar a qualidade do serviço 86
Figura 29 – Gráficos fuzzy para representar a disponibilidade Média e Alta do serviço 87
Figura 30 – Gráficos fuzzy para representar as relações entre entradas, regras e saída 88
Figura 31 – Regras de Inferência Fuzzy 89
Figura 32 – Gráfico com Funções de Saída para Qualidade 90
Figura 33 – Tratamento de Processos no Algoritmo de *Matching* 91
Figura 34 – Papéis das Ferramentas 94
Figura 35 – Ecore para genmodel 95
Figura 36 – Ferramenta para criação de Modelos de Serviços 96
Figura 37 – Modelo de Serviço em XML *Metadata Interchange* 97
Figura 38 – Arquitetura de MT4MDE e SAMT4MDE 98
Figura 39 – Arquitetura adaptada de [25] para o *Matching* de Modelos Semânticos 101
Figura 40 – Ferramenta para realizar *Matching* de Modelos de Serviços 102
Figura 41 – Estrutura do Estudo de Caso 107

Figura 42 – Composição de Agendamento de Vôo: Diagrama de Caso de Uso _____ 108
Figura 43 – Agendamento de Vôo: Diagrama de Classes _____ 108
Figura 44 – Agendamento de Vôo: Diagrama de Sequência _____ 109
Figura 45 – Modelo Semântico: FlightReservation_____ 110
Figura 46 – Modelo Semântico: FlightFast_____ 111
Figura 47 – Resultado do Matching: FlightReservation vs. FlytFast_____ 112
Figura 48 – Modelo Semântico: FlyNow_____ 113
Figura 49 – Resultado do Matching: FlightReservation vs. FlyNow _____ 114
Figura 50 – Modelo Semântico: FlightRight _____ 115
Figura 51 – Resultado do Matching: FlightReservation vs. FlightRight ____ 116
Figura 52 – BPEL do Estudo de Caso _____ 117
Figura 53 – Funcionalidade de Geração BPEL _____ 118
Figura 54 – Matches Reais e Matches Automáticos _____ 123
Figura 55 – Simulação dos Tipos e Níveis de Qualidade dos Serviços ____ 123
Figura 56 – Relação entre Confiabilidade, Disponibilidade e Qualidade ___ 124
Figura 57 – Relação entre Atraso, Disponibilidade e Qualidade _____ 124

LISTA DE TABELAS

Tabela 1 – Trabalhos Relacionados sobre Descrição de Serviços Web baseadas em MDA _____ 45
Tabela 2 – Tabela de relações de mapeamento entre Ontologias OWL e Diagramas de Classes _____ 51
Tabela 3 – Tabela de relações de mapeamento entre OWL-S e Diagramas de Sequência _____ 51
Tabela 4 – Tabela de relações de mapeamento entre UML, XMI e OWL-S__ 52
Tabela 5 – Tabela de Critérios segundo Análise de trabalhos Relacionados_ 62
Tabela 6 – Tabela de Conceitos e Propriedades de uma DSL_____ 75
Tabela 7 – Comparativo dos Valores de Matching_____ 119
Tabela 8 – Medidas de Qualidade do Algoritmo de Matching _____121
Tabela 9 – Critérios e Comparações com trabalhos relacionados _____ 128

LISTA DE CÓDIGOS

Listagem 1 – Parte do Algoritmo de *Matching* _____ 103
Listagem 2 – Parte do Algoritmo de *Matching* de Processos _____ 104
Listagem 3 – Parte do Algoritmo de *Matching* de Resultados _____ 104
Listagem 4 – Fragmento BPEL de Invocação FlightReservation _____ 118

LISTA DE SIGLAS

API	*Application Programming Interface*
APT	*Abstract Process Template*
ATL	*Atlas Transformation Languages*
BPEL	*Business Process Execution Language*
CORBA	*Common Object Request Broker Architecture*
DAML-S	*DARPA Agent Markup Language for Services*
DSL	*Domain Specific Language*
DWSC	*Dynamic Web Service Composition*
EMF	*Eclipse Modeling Framework*
ESP	*E-Services Platform*
GEF	*Graphical Editing Framework*
GMF	*Graphical Modeling Framework*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IBM	*International Business Machines*
IDE	*Integrated Development Environment*
JET	*Java Emitter Templates*
MDA	*Model Driven Architecture*
MDE	*Model Driven Engineering*
MDSD	*Model Driven Software Development*
MOF	*Meta Object Facility*
MT4MDE	*Mapping Tool for Model Driven Engineereing*
MVC	*Model-View-Control*
OMG	*Object Management Group*
OWL	*Web Ontology Language*
OWL-S	*Ontology Web Language for Services*
PIM	*Platform Independent Model*
PPC	*Process Partner Contract*
PSM	*Platform Specific Model*
SAMT4MDE	*Semi-Automatic Matching Tool for MDE*
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SSD	*Service Semantic Description*
UDDI	*Universal Description, Discovery and Integration*
UML	*Unified Modeling Language*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
USM	*UML Service Model*
W3C	*World Wide Web Consortium*
WS	*Web Service*
WS-BPEL	*Business Process Execution Language for Web Services*
WSDL	*Web Services Description Language*
WSTL	*Web Service Transaction Language*
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*
XSLT	*eXtensible Stylesheet Language Transformation*

SUMÁRIO

LISTA DE FIGURAS 10
LISTA DE TABELAS 12
LISTA DE CÓDIGOS 13
LISTA DE SIGLAS 14
1. Introdução 18
1.1. Contexto 18
1.2. Problemática 19
1.3. Motivação 20
1.4. Objetivos 21
1.4.1. Objetivos específicos 21
1.5. Metodologia 22
1.6. Apresentação da Dissertação 24
2. Fundamentação Teórica 26
2.1. SOA e Serviços Web 26
2.2. Linguagem de Execução de Processos de Negócios para Serviços Web (WS-BPEL) 28
2.3. Composição BPEL de Serviços Web 29
2.4. Representação do Conhecimento 30
2.4.1. Web Semântica 31
2.4.2. Ontologias 31
2.4.3. OWL-S 32
2.5. Engenharia Dirigida por Modelos 34
2.5.1. Arquitetura Dirigida por Modelos 35
2.5.2. Eclipse Modeling Framework 36
2.5.3. Domain-Specific Language 37
2.6. Síntese 38
3. Estado da Arte 39
3.1. Composição de Serviços Web Estática e Dinâmica 40
3.2. Evolução das Abordagens de Composição de Serviços Web 43
3.3. Trabalhos Relacionados 45
3.3.1. Composição de Serviços Web e Ontologias 45
3.3.2. Uma Abordagem Dirigida por Modelos para Composição Dinâmica de Serviços Web 47

3.3.3. Uma Abordagem Dirigida por Modelos para descrever Serviços Web Semânticos: De UML para OWL-S 50

3.3.4. Similaridade Semântica na Composição de Sistemas de Informação através dos Serviços Web 54

3.3.5. Composição Dinâmica de Serviços Web utilizando padrões de projeto. 56

3.4. Análise dos Trabalhos Relacionados 60

3.5. Síntese 64

4. Proposta de um *Framework* para suportar a Composição Dinâmica de Serviços Web 65

4.1. Abordagem 65

4.1.1. Primeira Etapa 65

4.1.2. Segunda Etapa 68

4.1.3. Modelagem do *Framework* 69

4.2. Metodologia para suportar a Composição Dinâmica de Serviços Web 71

4.3. Metamodelos Desenvolvidos 72

4.3.1. Metamodelo Semântico de Serviços 73

4.3.2. Metamodelo de Mapeamento entre Modelos Semânticos de Serviços Web 76

4.4. Algoritmo de Matching de Modelos 79

4.4.1. Qualidade de Serviço 83

4.4.2. Processos de Serviços 90

4.5. Síntese 93

5. Implementação do Protótipo do *Framework* DWSC 94

5.1. Ferramenta para criação de Modelos Semânticos de Serviços Web 95

5.2. Ferramenta de Correspondência para Modelos de Serviços 97

5.3. Implementação do Algoritmo de *Matching* 103

5.4. Síntese 105

6. Estudo de Caso, Testes e Execução 106

6.1. Apresentação do Estudo de Caso 106

6.2. Modelagem do Estudo de Caso 107

6.3. Utilização do Protótipo e Testes 109

6.4. Avaliação de Resultados 119

6.5. Síntese 125

7. Conclusões e Trabalhos Futuros 126

7.1. Objetivos Alcançados 126

7.2. Limitações ____ **130**
7.3. Trabalhos Futuros ____ **130**
7.4. Contribuições ____ **131**
7.4.1. Científicas ____ **131**
7.4.2. Tecnológicas ____ **132**
7.4.3. Sociais ____ **132**
8. Referências bibliográficas ____ **133**

1. Introdução

Este primeiro capítulo descreve o contexto em que o trabalho foi desenvolvido, trazendo suas principais contribuições, a problemática, a motivação, os objetivos, a descrição da metodologia empregada para a execução da pesquisa e a apresentação dos capítulos desta dissertação.

1.1. Contexto

A utilização da Web como forma de publicar novas aplicações e disponibilizar novas funcionalidades tem consolidado o uso da tecnologia de serviços Web. Essa tecnologia vem trazer consideráveis benefícios à gerência de processos de negócio, uma vez que propicia a agilidade necessária e requerida pelos empreendimentos frente às necessidades atuais das rápidas mudanças no ambiente de negócios [1][2]. Uma das principais vantagens em usar serviços Web está na sua interoperabilidade, obtida devido à adesão a protocolos padrões [3], amplamente difundidos na Internet, e à existência de uma arquitetura padrão, que define os mecanismos necessários para a sua utilização. Essa interoperabilidade possibilita a combinação de serviços dando origem a outros serviços Web, caracterizando assim uma composição de serviços Web (*Web Service Composition*).

De fato, serviços Web tornaram-se a base das novas iniciativas de computação distribuída e de negócios eletrônicos, pois permitem construir redes de aplicações colaborativas distribuídas, dentro e entre organizações, onde os serviços Web, na forma de módulos auto-contidos, são descritos, publicados, localizados e dinamicamente invocados.

Uma vez que a tecnologia de serviços Web viabiliza um alto grau de interoperabilidade e autonomia, ela fornece uma infra-estrutura básica para construção e composição de novos serviços. A composição automática de serviços apresenta desafios, pois a localização, seleção e composição devem satisfazer necessidades específicas do novo serviço que deve ser composto, e para isso mecanismos de descrição devem ser empregados.

O estudo e a aplicação da Engenharia Dirigida por Modelos (MDE) conjuntamente com Ontologias para suportar a Composição Dinâmica de Serviços Web, vislumbra ser uma das soluções mais promissoras para fornecer uma resposta viável ao gerenciamento da complexidade de desenvolvimento, manutenção e evolução de softwares baseados na Arquitetura Orientada a Serviços (SOA - *Service-Oriented Architecture*).

1.2. Problemática

O problema abordado nesta dissertação é a composição dinâmica de serviços web, que ainda possui vários pontos em aberto.

Primeiramente, dada a **falta**[1] de um dos serviços de uma composição, tem-se a necessidade de descobrir um novo serviço substituto, considerando suas informações. Surge então o primeiro desafio que é tratar as diversas características dos serviços que estão armazenados em um repositório. Muitas vezes, essas características são complexas ou pobremente descritas (por exemplo, usando WSDL - *Web Services Description Language*) e até erroneamente interpretadas.

Segundo, uma vez descoberto um serviço, este deve ser adequadamente reutilizado em um processo definido em uma linguagem de definição de processos de negócio. É necessário saber (ou inferir) os objetivos dos serviços, identificar as restrições sobre a sua aplicabilidade e os seus efeitos em um sistema. No momento em que vários serviços básicos são compostos, a verificação do comportamento conjunto torna-se uma questão fundamental, tanto na correção do comportamento de todos os serviços (macro comportamento) como na sua eficiência. Utilizar ontologias como forma de "descritores" mais precisos, conceitualmente mais ricos para os serviços Web, deve atender à necessidade dessa descoberta e adequação permitindo um melhor suporte para a composição, para tal tarefa alguns subproblemas precisam de soluções:

---

[1] **Falta** de um serviço web, no contexto deste trabalho, significa a queda de um servidor onde está publicado o serviço web ou o tempo de resposta para a requisição de um serviço web foi excedido (*timeout*) ou a execução de um serviço web resultou em uma exceção.

- Como extrair metadados ou informações sobre Serviços Web para fazer a composição dinâmica;
- Como utilizar Ontologias e Semântica Web para representar Serviços Web;
- Como construir modelos e metamodelos que representem os serviços;
- Como fazer o *Matching* para estabelecer as correspondências entre os modelos dos serviços, antes de fazer a composição de serviços;
- Como utilizar *Matching* de Modelos para aprimorar o que está sendo descoberto;
- Como resolver o problema da criação manual de modelos de correspondência que é uma tarefa enfadonha e propensa a gerar erros.

Os benefícios da aplicação da engenharia dirigida por modelos e ontologias na composição dinâmica de serviços web são pouco avaliados. Assim, depois de analisado o principal problema abordado e alguns subproblemas referentes a composição dinâmica de serviços web utilizando abordagens de MDE e Ontologias, pode-se concluir que o foco da pesquisa, trata da modelagem, geração, *matching* (casamento, correspondência) e *binding* (ligação) de serviços web.

1.3. Motivação

A Composição Dinâmica de Serviços Web utilizando *Matching* de modelos baseados em ontologias de serviços e na engenharia dirigida por modelos parece ser uma proposta promissora para resolver o problema da composição dinâmica de serviços web. Como Metamodelos e Ontologias estão fortemente relacionados, o uso de ambos para identificação de

correspondências entre modelos parece sugestivo. Diversas organizações (por exemplo, OMG e o Projeto Eclipse), instituições de pesquisa e empresas estimulam o desenvolvimento de pesquisa em MDE. Algumas pesquisas realizadas pelos orientadores [4][5][6][7][8], demonstram que MDE é um paradigma promissor para o desenvolvimento de software. MDE e Ontologias conjuntamente podem ser utilizados para suportar a composição dinâmica de serviços Web graças ao *Matching* de modelos semânticos de serviços.

1.4. Objetivos

Fornecer uma abordagem baseada em Engenharia Dirigida por Modelos (MDE) conjuntamente com Ontologias para a Composição Dinâmica de Serviços Web e propor um *framework* para realizar a composição de serviços. MDE deve fornecer uma resposta viável ao gerenciamento da complexidade de desenvolvimento, manutenção e evolução de serviços web dentro de um contexto de composição dinâmica de serviços. Desta forma, Ontologias devem atender a descoberta e adequação de serviços e permitir um melhor suporte para a composição.

1.4.1. Objetivos específicos

- Estudar e aplicar a engenharia dirigida por modelos e ontologias para o desenvolvimento de sistemas de softwares como serviços web compostos;
- Estudar e utilizar ontologias para semi-automatizar a geração de modelos de correspondência entre metamodelos e/ou modelos semânticos para suportar a composição dinâmica de serviços Web;
- Utilizar ontologias para obter informações semânticas sobre os serviços envolvidos;

- Utilizar *Matching* de Modelos (ou Metamodelos) semânticos baseados em Ontologias de Serviços.

1.5. Metodologia

Nesta seção, a metodologia da pesquisa científica será fornecida, partindo da pesquisa bibliográfica, com o objetivo de coletar informações de livros, teses e dissertações, periódicos, anais de congressos, especificações e páginas web para uma total contextualização da literatura especializada, ou seja, o estado da arte dos três principais espaços tecnológicos que envolvem a pesquisa:

a. Engenharia Dirigida por Modelos (MDE), mais especificamente aplicações da Arquitetura Dirigida por Modelos (MDA), incluindo linguagens de transformação, definição de transformação, especificação de correspondências, metamodelos e *Matching* de metamodelos;
b. Arquitetura Orientada a Serviços (SOA), Serviços Web, a linguagem WSDL (*Web Services Description Language*), a linguagem WS-BPEL (*Web Services Business Process Execution Language*), *Web Service Composition*;
c. Ontologias mais precisamente OWL-S (*Ontology Web Language for Services*), metamodelo de definição de Ontologias e semântica web.

Outras etapas de estudo e utilização de possíveis ferramentas e metodologias empregadas podem ser citadas a seguir:

- Propor uma abordagem baseada em MDE, Ontologias e na filosofia SOA para projetar sistemas de software implementando-os como uma composição de serviços;
- Propor um metamodelo de serviços baseado em aspectos de Semântica Web, Ontologias e UML;

- Propor um *framework* para realizar a composição dinâmica de serviços baseado em MDE e Ontologias;
- Estudar e construir uma Linguagem Específica de Domínio (DSL) para o domínio de serviços web semânticos;
- Utilizar algoritmos de *Matching* e Ontologias para suportar a criação de modelos de correspondência baseados em similaridades estruturais e semânticas entre modelos;
- Propor uma abordagem para criação de modelos de correspondência conforme as similaridades sintáticas e semânticas extraídas de ontologias;
- Utilização de UML (*Unified Modeling Language*) para a criação de modelos independentes de plataforma;
- Propor especificações de correspondências e definições de transformação entre UML com perfil e o metamodelo de serviços para possibilitar a geração semi-automática de código final;
- Utilização da linguagem de transformação ATL (*Atlas Transformation Language*) para escrever as definições de transformação;
- Avaliação dos modelos de correspondências entre modelos, segundo os critérios: *Precision*, *Recall*, *F-Measure* e *Overall* [7];
- Estudo do ambiente de programação Eclipse, bem como construção e uso de *plug-ins,* mais precisamente DSL Toolkit que é parte do projeto do *Framework* de Modelagem Eclipse (EMF) que foi utilizado durante o desenvolvimento do trabalho proposto;
- Estudo das ferramentas MT4MDE e SAMT4MDE para a geração semi-automática das especificações das correspondências entre os modelos semânticos de serviços;
- Levantamento de trabalhos relacionados;
- Avaliação de ferramentas de estudos científicos ou de mercado, como Oracle SOA Suite 11g e Eclipse SOA para montar composições de Serviços Web.

1.6. Apresentação da Dissertação

Este trabalho de dissertação está estruturado em 7 (sete) capítulos como seguem:

O primeiro capítulo introduz o leitor ao tema do trabalho através do contexto tecnológico no qual este trabalho está inserido, o cenário em que vai ser desenvolvido, exposição da problemática, motivação para propor determinadas ferramentas, abordagens e tecnologias na solução, ainda neste capítulo são tratados os objetivos geral e específicos, e finalmente a metodologia da pesquisa.

No segundo capítulo, a fundamentação teórica necessária para a compreensão e desenvolvimento do presente trabalho é apresentada. Os conceitos correlacionados e aplicados a solução, como arquitetura SOA e Serviços Web, linguagem para execução de processos de negócios estendida para serviços web (WS-BPEL) e tipos de composição de serviços web serão todos descritos. Ainda neste capítulo, trata-se formas de representar a informação através das Ontologias, OWL, OWL-S e Semântica Web. E, finalmente, abordaremos a Engenharia Dirigida por Modelos (MDE) e suas aplicações como a Arquitetura Dirigida por Modelos (MDA), *Framework* de Modelagem do Eclipse (EMF) e Linguagens Específicas de Domínio (DSL).

No terceiro capítulo, apresenta-se o estado da arte de pesquisas e trabalhos relacionados envolvendo a composição de serviços web conjuntamente com Ontologias, bem como a composição de serviços web conjuntamente com MDE. Para finalizar este capítulo, uma análise dos trabalhos relacionados é realizada.

No quarto capítulo, a proposta de um *Framework* para suportar a composição dinâmica de serviços web é apresentada, fazendo um *overview* da abordagem utilizada, exposição de metamodelos relacionados e diagramas para ilustrar o *framework*, metodologia de desenvolvimento, abordagens de *matching* de modelos.

No quinto capítulo, a implementação do protótipo através das ferramentas para suportar a composição dinâmica de serviços é apresentada. A ferramenta para criação de modelos semânticos de serviços e artefatos, ferramenta de correspondências para MDE (MT4MDE) e para geração semi-automática de correspondências para MDE (SAMT4MDE) e, finalmente, o gerador de definição de transformação são apresentados neste capítulo.

No sexto capítulo, um estudo de caso e sua execução são propostos, incluindo sua modelagem. Em seguida, a utilização do protótipo, a avaliação dos resultados e os testes são descritos.

Finalmente, no sétimo capítulo, as conclusões, as contribuições, as limitações, os trabalhos futuros e os resultados alcançados neste trabalho são apresentados.

2. Fundamentação Teórica

Este capítulo se limita à apresentação da fundamentação teórica, base teórica que fundamenta o trabalho, onde será dado um enfoque geral de modo a familiarizar o leitor com a teoria que será utilizada e necessária para o desenvolvimento deste trabalho. Os principais conceitos e as tecnologias exigidas para a compreensão das abordagens utilizadas com foco nos objetivos esperados são explorados.

Dentre os principais conceitos aplicados a solução, inicialmente, aborda-se a arquitetura SOA e Serviços Web, a linguagem para execução de processos de negócios estendida para serviços web (WS-BPEL) e os tipos de composição de serviços web. Ainda, neste capítulo, as formas de representar a informação através das Ontologias, OWL, OWL-S e Semântica Web serão apresentadas e discutidas. E finalmente, abordar-se a Engenharia Dirigida por Modelos (MDE) e as suas aplicações como a Arquitetura Dirigida por Modelos (MDA), o *Framework* de Modelagem do Eclipse (EMF) e as Linguagens Específicas de Domínio (DSL).

2.1. SOA e Serviços Web

Um dos principais problemas de TI (Tecnologia da Informação) até o surgimento de SOA (*Service-Oriented Architecture*) era a integração. Segundo Matjaz B. Juric e Marcel Krizevnik [11], qualquer projeto de integração impunha um enorme peso financeiro e de recursos sobre o departamento de TI. Em muitos casos, a integração era resolvida com soluções pontuais: uma conexão com o Banco de Dados, uma transferência de arquivos.

SOA permitiu às organizações enfrentar os desafios da integração, de modo que um serviço criado uma única vez, poderia ser reutilizado em toda parte. Reutilização possibilitou uma folga nos prazos apertados de

implementação e reduziu os custos de manutenção. Em meio aos diferentes métodos e soluções utilizados para gerenciar problemas relacionados as rápidas mudanças nos ambientes de negócio, SOA é a mais recente abordagem arquitetural relacionada com integração, desenvolvimento e manutenção de sistemas de informação empresariais complexos.

Alguns fatores importantes da filosofia SOA são necessários para fazê-la funcionar efetivamente. Os princípios básicos que sustentam SOA são ilustrados na Figura 1. Primeiramente, é necessário prever uma definição abstrata do serviço, incluindo detalhes que permitem a qualquer cliente em potencial usar o serviço fazendo a ligação (***bind***) de modo apropriado. Segundo, os provedores de serviços precisam publicar (***publish***) detalhes de seus serviços de modo que os interessados em usá-los possam compreender mais precisamente o que eles fazem e obter informações necessárias para conectá-los. Em terceiro, aqueles que necessitam dos serviços precisam de alguma maneira descobrir (***find***) quais serviços estão disponíveis que atendam a sua necessidade [10]. Finalmente, para o funcionamento da abordagem *bind/publish/find*, padrões devem ser estabelecidos para definir o quê e como publicar, como encontrar a informação e os detalhes específicos de como fazer a ligação com o serviço.

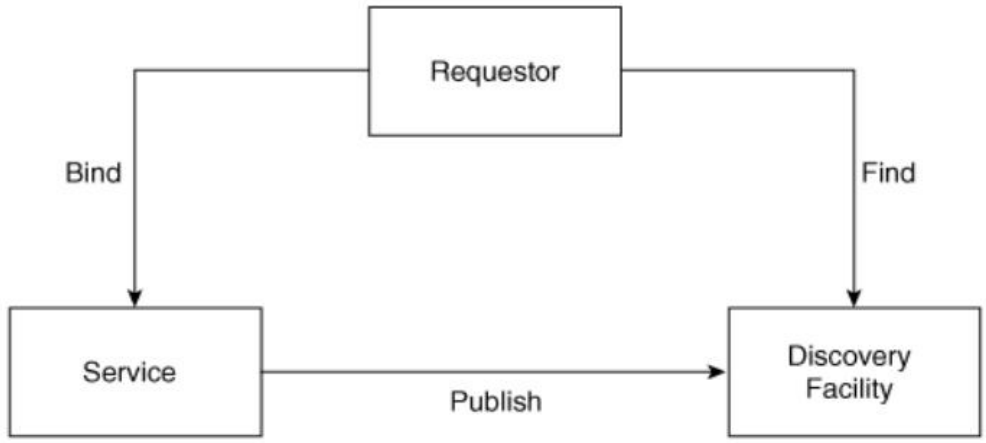


Figura 1 – Triângulo SOA. Fonte [10]

Frequentemente, a Arquitetura Orientada a Serviços (SOA) e Serviços Web são vistos ou concebidos por pessoas como uma única entidade, combinados, mas na realidade são distintos em importantes aspectos. SOA representa um conceito abstrato de arquitetura, uma abordagem para a construção de sistemas de softwares baseados em componentes fracamente acoplados (por exemplo, serviços web ou objetos de CORBA) que são descritos de maneira uniforme o que possibilita a descoberta e a composição dos mesmos.

Serviços Web representam uma estratégia importante para a realização de SOA. Uma definição geral descreve que Serviços Web são vistos como uma aplicação acessível por outras aplicações através da Internet [12]. Segundo o consórcio de empresas de tecnologia W3C (*World Wide Web Consortium*) que assegura e promove a evolução das especificações WSDL (*Web Services Description Language*) e SOAP (*Simple Object Access Protocol*), define Serviços Web como segue:

"Um sistema de software projetado para suportar interoperabilidade *machine-to-machine* (entre máquinas) através da rede. Possui uma interface descrita em um formato processável por máquina (especificamente WSDL). Outros sistemas interagem com o serviço web de maneira prescrita por sua descrição usando mensagens SOAP, RESTFUL, normalmente transmitida através do protocolo HTTP (*Hypertext Transfer Protocol*) com serialização XML (*eXtensible Markup Language*) em conjunto com outros padrões web relacionados" [3].

2.2. Linguagem de Execução de Processos de Negócios para Serviços Web (WS-BPEL)

Business Process Execution Language for Web Services (BPEL, WS-BPEL ou BPEL4WS) é um padrão comumente aceito para a definição de processos de negócio em uma composição de serviços, ou seja, fornece uma

linguagem para especificar processos de negócios e estados de processos e como eles se relacionam com os serviços web [10] [11]. Neste trabalho de dissertação BPEL é apoiada por uma maioria de fornecedores de softwares, incluindo Oracle, IBM, Microsoft entre outros.

Em sistemas de informação empresariais, BPEL pretende alcançar um importante papel, fornecendo um ambiente onde os processos de negócio possam ser desenvolvidos de uma maneira fácil e eficiente, diretamente executados, monitorados e rapidamente adaptados às necessidades das empresas [11]. Dentre alguns objetivos da sua especificação, inclui-se como um processo de negócio utiliza os serviços Web para atingir seu objetivo, e inclui a possibilidade de especificar serviços que fornecem processos de negócio. Dentre os principais conceitos de SOA está a composição de serviços em processos de negócio. Os serviços são compostos em uma determinada ordem e seguem um conjunto de regras para fornecer suporte para os processos de negócio.

2.3. Composição BPEL de Serviços Web

Como foi visto nas seções anteriores, com BPEL pode-se definir processos de negócio que fazem uso de serviços, e processos de negócio que externam (tornam públicas) suas funcionalidades como serviços web. A combinação de serviços isolados para realizar determinada tarefa caracteriza uma composição de serviços web.

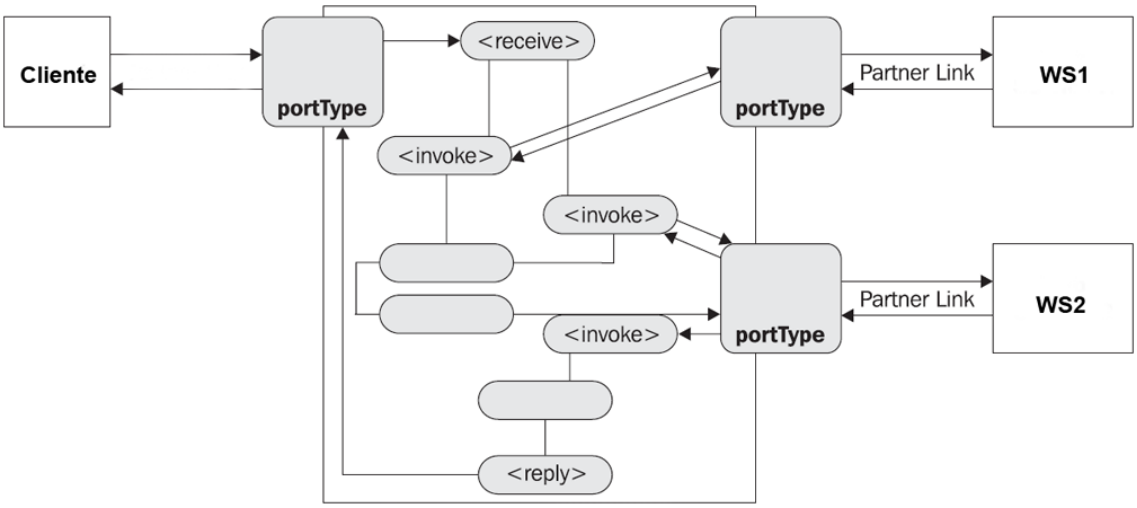


Figura 2 – Comunicação entre a composição BPEL e os Partner Links. Fonte [56]

Como ilustra a Figura 2, a comunicação entre uma composição BPEL e os *PartnerLinks* é de suma importância para entender como os serviços web estão relacionados e são invocados para realizar determinada tarefa. Os *PartnerLinks* possuem a localização dos servidores onde estão disponibilizados os serviços web parceiros (WS1, WS2). Cada elemento *PartnerLink* representa um serviço web chamado de parceiro. O *PortType* no contexto de serviços web é um único serviço web, o elemento *PortType* é o conjunto de todas as operações que um serviço web possui. *Invoke* representa a invocação de uma operação em outro serviço web. *Receive* aguarda um cliente invocar o processo de negócio através do envio de uma mensagem, ou seja, é o ponto de entrada da composição. *Reply* é usado para enviar a resposta de uma operação que foi chamada de forma síncrona [56].

2.4. Representação do Conhecimento

A ideia de Ontologias aplicadas em Inteligência Artificial surgiu há algum tempo como um meio de compartilhar conhecimento [15]. Seguindo o desenvolvimento de Ontologias e tecnologias Web relacionadas (ex: HTML e XML), Tim Berners-Lee, Jim Hendler e Ora Lassila idealizaram a próxima geração da Web, chamada de Web Semântica [13].

2.4.1. Web Semântica

Uma das formas de representar a Web Semântica é através do uso de ontologias. A Web Semântica tem o potencial de representar, através de conceitos, relações e axiomas, as coisas na Web (ex: páginas web, aplicações e pessoas) bem como suas relações, portanto deve fornecer serviços representados com informações semânticas adicionais.

A Web Semântica é a nova geração da Web que tenta representar a informação para que ela possa ser usada por máquinas e não apenas para fins de exibição, mas também para automação, integração e reuso entre aplicações (Boley et al. 2001) [13]. Um dos objetivos da Web Semântica é permitir o raciocínio sobre as entidades de dados em diferentes páginas ou recursos da Web. A Web Semântica também pode ser vista como uma extensão da Web atual na qual a informação é dada com um significado bem definido, permitindo que computadores e pessoas trabalhem em cooperação [14].

2.4.2. Ontologias

Muitas definições do conceito de Ontologia existem em IA (Inteligência Artificial) e computação em geral. O conceito mais largamente citado é:

Ontologia é uma especificação de uma conceituação [15].

Esta definição é certamente a mais concisa e requer alguns esclarecimentos. Primeiro, “conceituação” significa uma abstração ou visão simplificada de mundo. E o significado de “mundo” se refere a algum fenômeno no mundo, ou para algum tópico (ou tópicos), ou alguma área sujeita.

Segundo, “especificação” significa uma representação formal e declarativa. Na estrutura de dados que representa a ontologia, o tipo de

conceito utilizado e as restrições sobre seu uso estão estabelecidos declarativamente, de forma explícita e usando uma linguagem formal [13].

Em IA (Inteligência Artificial), o termo “ontologia” tem grande possibilidade de vir a significar uma das duas definições afins (Chandrasekaran et al 1999.) [13]:

- Um vocabulário de representação, muitas vezes especializado para algum domínio ou assunto;
- Um corpo de conhecimento que descreve algum domínio particular, usando um vocabulário de representação.

Em ambos os casos, há sempre uma estrutura de dados associada que representa a ontologia. Para o contexto deste trabalho, uma ontologia pode ser vista como um **modelo de dados** que representa um conjunto de conceitos, seus relacionamentos e axiomas dentro de um domínio e pode ser utilizada para realizar inferências sobre os objetos do domínio em questão.

2.4.3. OWL-S

Serviços permitem um efeito de alguma ação ou mudança no mundo real, tais como venda de um produto ou controle de um dispositivo físico. Entre os recursos mais importantes da web estão aqueles que fornecem serviços. A Web semântica deve permitir aos usuários localizar, selecionar, contratar, compor e acompanhar os serviços baseados na web, automaticamente [24]. OWL-S é uma ontologia de serviços que faz essas funcionalidades se tornarem possíveis.

Para a web, a ontologia trata sobre a descrição exata de uma informação na web e as relações entre essas informações. Antes de entrar em OWL-S devemos entender alguns conceitos de OWL. Então, OWL foi projetada para oferecer uma maneira comum de processar o conteúdo de informações na Web ao invés de somente exibí-las.

OWL é uma parte da visão da Web Semântica, um futuro onde a informação terá significado exato na web possibilitando que essas informações venham a ser processadas por computadores a partir de objetivos em alto

nível. Por exemplo, um usuário poderá sentar-se a frente do computador e solicitar ao computador que agencie sua viagem a uma Copa do Mundo informando a condição que seu capital para esse fim é de R$ 10.000. Logo, a partir desse objetivo em alto nível, o PC em comunicação com outros na rede, poderá coordenar vários serviços combinados (agência de viagens, hotéis, linhas aéreas, aluguel de veículos, formas de pagamento, etc.) a fim de trazer todas as informações necessárias sobre esse objetivo e se será possível realizá-lo com o capital existente, isso através da integração de informações na web.

Uma ontologia OWL pode definir classes, propriedades de objetos e dados e axiomas. Já OWL-S é uma ontologia/linguagem OWL para (formalmente) descrever propriedades e capacidades (ou funcionalidades) de serviços web. Sua estrutura geral é composta por três sub-ontologias com suas respectivas capacidades como mostra a Figura 3.

Figura 3 – Nível Superior da Ontologia OWL-S.
Fonte[39]

De forma mais detalhada, as três subontologias são descritas a seguir [59]:

- *ServiceProfile* representa de forma adequada o que um serviço faz. Possui a capacidade de especificação, o que facilita a consulta do serviço através dos critérios de busca determinados. Possui características Gerais dos Serviços, limitações sobre sua

aplicabilidade, qualidade do serviço e requisitos satisfatórios para o uso do serviço;

- *ServiceModel* define como interagir com o serviço, detalhando precondições para uma requisição de serviço, condições sob as quais um resultado irá ocorrer, controle de fluxos de serviços. Ou seja, define como invocar um serviço e o que acontece quando ele é executado;
- *ServiceGround* define o que é preciso para realizar um serviço. Especifica protocolos de comunicação, formatos de mensagens, número de porta utilizada para chamar o serviço, tipo de serialização, etc. Realiza o mapeamento de entradas e saídas para WSDL.

2.5. Engenharia Dirigida por Modelos

Recentemente, novos paradigmas estão sendo desenvolvidos para fazer face à complexidade cada vez mais crescente do desenvolvimento, manutenção e evolução de software. Dentre estes novos paradigmas, a Engenharia Dirigida por Modelos (ou *Model Driven Engineering* - MDE) e as ontologias se destacam como os mais promissores [16][17][18][19][10]. Em MDE, o ciclo de vida de desenvolvimento de software não é muito diferente das abordagens tradicionais. A diferença fundamental é que os artefatos criados durante o processo de desenvolvimento são modelos “formais”, isto é, modelos que podem ser compreendidos e então manipulados por computadores. Sendo assim, MDE é uma proposta complementar aos processos tradicionais de desenvolvimento de softwares. MDE é uma linha de pesquisa promissora, podendo abranger: linguagens e motores de transformação [20], ferramentas para a especificação de correspondências entre modelos [6][7], definições de transformação [4][5], algoritmos de *metamodel matching* [7], metodologias [21], processos [8][22], e a aplicação de MDE em plataformas finais como Serviços Web [4][5][23].

MDE não pretende substituir os processos tradicionais de desenvolvimento de software, e sim contribuir para seu aprimoramento, porém, de uma maneira racional para mover informações de uma fase para outra fase

do desenvolvimento. Dessa forma, MDE fornece respostas rápidas e eficientes para atender as necessidades inesperadas de novos requisitos tanto funcionais quanto não-funcionais, apresentando uma melhor e rápida adaptação de novas tecnologias e integração de software desenvolvidos em diversas plataformas [25].

Modelo é “uma descrição de um (ou de uma parte de) sistema expresso em uma linguagem bem definida, isto é, respeitando um formato preciso (uma sintaxe) e um significado (uma semântica). Esta descrição deve ser conveniente para uma interpretação automatizada por computador” [26].

A criação de modelos é feita utilizando uma linguagem de modelagem bem definida que apresente uma sintaxe e uma semântica para regulamentar a criação dos elementos e suas relações. Uma linguagem de modelagem é uma especificação formal bem definida que contém os elementos de base para construir modelos. Além disto, uma linguagem de modelagem é concebida dentro de um domínio limitado e com objetivos específicos [25].

2.5.1. Arquitetura Dirigida por Modelos

A Arquitetura Dirigida por Modelos (ou Model-Driven Architecture - MDA) da *Object Management Group* (OMG) [27][28], *Eclipse Modeling Framework* (EMF) do Projeto Eclipse [29] e *Domain Specific Language* (DSL) do *Eclipse Modeling Project* [30] são exemplos de MDE. MDA e EMF apresentam similaridades conceituais, mas diferem quanto à tecnologia, por exemplo, MDA utiliza MOF [9], UML e profiles, enquanto que o EMF utiliza Ecore [29] e favorece a criação de linguagens específicas ao domínio (DSL) [30]. A proposta MDA da OMG é basicamente constituída pela criação de modelos independentes da plataforma (*Platform-Indepedent Model - PIMs*) e a transformação de PIMs em modelos dependentes (específicos) da plataforma (*Plaftorm-Specific Model - PSMs*). Assim, a complexidade de desenvolvimento de software é gerenciada por modelos, favorecendo o desenvolvimento, a manutenção e a evolução de softwares [18] [31].

MDA foi desenvolvida sobre as normas da OMG, incluindo: UML (Unified Modeling Language) e XMI (*XML Metadata Interchange*). Esta iniciativa apresenta alguns benefícios, tais como [26]:

- **Produtividade**: a transformação do PIM (Platform Independent Model) para o PSM (Platform Specific Model) precisa ser definida uma única vez e pode ser aplicada no desenvolvimento de diversos sistemas. Devido a este fato, tem-se uma redução no tempo de desenvolvimento;
- **Portabilidade**: um mesmo PIM pode ser automaticamente transformado em vários PSMs de diferentes plataformas, através de mapeamentos;
- **Interoperabilidade**: diferentes PSMs gerados a partir de um mesmo PIM podem ter relacionamentos entre si.

2.5.2. Eclipse Modeling Framework

EMF (*Eclipse Modeling Framework*) é um poderoso *framework* que facilita a geração de código para construção de aplicações Java baseadas em simples definições de modelos. Projetado para fazer a modelagem prática, ou seja, definir, editar, consultar e validar modelos. EMF é útil principalmente para o programador Java e unifica três importantes tecnologias: Java, XML e UML [29].

Como ilustra a Figura 4, EMF se encontra no centro das tecnologias envolvidas como um facilitador para definir um modelo em qualquer uma dessas formas, possibilitando também a geração de qualquer outra forma e classes de implementação correspondente.

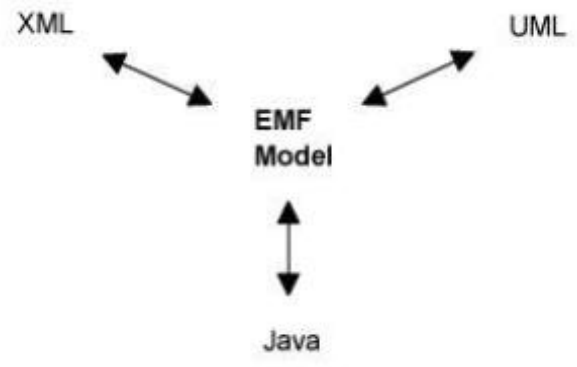


Figura 4 – EMF unifica Java, XML e UML.
Fonte [29]

O metamodelo usado para representar modelos em EMF é chamado Ecore. Interfaces Java anotadas e opcionalmente outras formas de modelos podem ser geradas a partir de um modelo Ecore, ou seja, sua serialização XMI (***X****ML* ***M****etadata* ***I****nterchange*) é o centro do universo EMF como ilustra a Figura 5.

XMI é a forma serializada de um modelo Ecore, pode ser visto como uma representação primária, ou padrão que possibilita a serialização concisa de metadados usando XML.

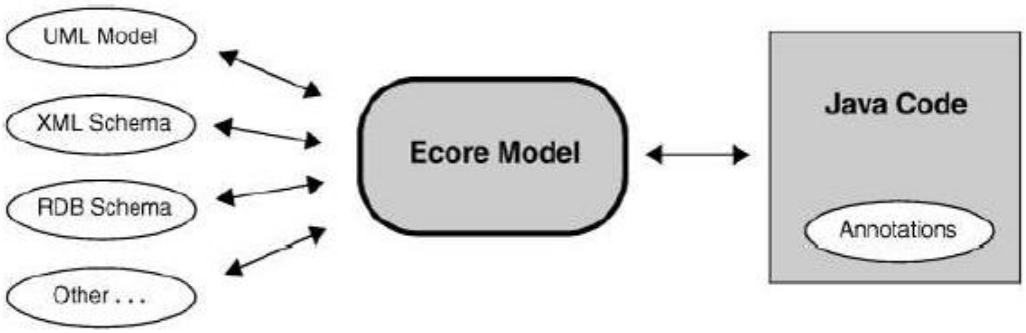


Figura 5 – Modelo Ecore e suas Fontes.
Fonte [29]

2.5.3. Domain-Specific Language

DSL (*Domain-Specific Language*) é uma linguagem projetada para ser utilizada em um conjunto específico de tarefas [30]. O termo “domínio específico” é determinado pelo **projetista da linguagem** que define os conceitos que vão compor a linguagem.

A estrutura de uma DSL é capturada no seu metamodelo, geralmente designado por sua sintaxe abstrata**.** Um **metamodelo** é apenas um modelo que fornece uma base para a construção de outro modelo. Embora ambos sejam modelos, um (modelo) é expressado em termos do outro (metamodelo), ou seja, um modelo é uma instância (*instance of*) ou está conforme (*conform to*) o outro [30].

Dentro do universo de DSL's existe um conjunto de ferramentas chamado DSL Toolkit que emergiu de uma coleção de projetos relacionados às Tecnologias de Modelagem e Desenvolvimento de Software Dirigido por Modelos (MDSD). DSL Toolkit foi criado para coordenar e orientar as capacidades de MDSD (*Model Driven Software Development*) dentro do Eclipse. Depois de projetada a linguagem, DSL Toolkit dá o suporte para a construção de outras instâncias customizadas do ambiente de desenvolvimento. Desse modo, um potencial **desenvolvedor de aplicações** utiliza a linguagem projetada com o ambiente construído, para criar aplicações pertencentes ao domínio em questão. Finalmente, um possível **usuário final** consome os programas elaborados pelo desenvolvedor de aplicações.

2.6. Síntese

Neste capítulo, uma revisão dos principais conceitos das tecnologias utilizadas para o desenvolvimento deste trabalho foi apresentada e tais conceitos foram separados em seções referentes aos espaços tecnológicos envolvidos. Sobre a arquitetura SOA foram abordados alguns problemas de TI até o surgimento da arquitetura, os benefícios de sua utilização, os princípios básicos que a sustentam, sua relação com Serviços Web e sua importância para a realização de SOA.

Algumas definições de Serviços Web e sua relação com WS-BPEL foram apresentados e como os processos de negócio definidos em BPEL, que fazem o uso de Serviços Web, podem realizar uma composição.

Conceitos importantes relacionados a representação do conhecimento foram abordados, tais como Web Semântica, OWL-S e Ontologias.

Finalmente, a Engenharia Dirigida por Modelos e suas aplicações foram descritas. Conceitos de MDA e seus benefícios como produtividade, portabilidade e interoperabilidade foram apresentados. Os objetivos do *Framework* de Modelagem Eclipse (EMF) e Linguagens Específicas do Domínio (DSL) e suas aplicações fecharam este capítulo.

3. Estado da Arte

Este capítulo visa descrever as principais abordagens para realizar a composição de serviços web utilizadas no meio acadêmico e industrial. Algumas vantagens e desvantagens das abordagens pesquisadas na literatura são relatadas. A diferença entre composição estática e dinâmica é descrita através da apresentação resumida de algumas ferramentas e abordagens. Uma evolução histórica das abordagens de composição de serviços desde 2002 até 2010 é apresentada. Trabalhos com relações mais próximas da solução proposta neste trabalho são analisados e comparados segundo alguns critérios estabelecidos. Os trabalhos selecionados serviram de base para o desenvolvimento desta dissertação.

3.1. Composição de Serviços Web Estática e Dinâmica

A composição estática de serviços web ocorre durante o tempo de projeto (*design-time*) quando a arquitetura e o projeto de software são planejados. Os componentes de software são escolhidos, ligados entre si, compilados e finalmente implantados. Este processo funciona bem enquanto os parceiros, ambiente de negócio e componentes de serviços, não são alterados, ou raramente são modificados. *Microsoft BizTalk Server* e *BEA WebLogic* são exemplos de mecanismos de composição estática [32].

Se uma empresa provedora de serviços publicar um novo serviço, ou se um serviço antigo for substituído por um novo serviço, inconsistências poderão acontecer. Neste caso, a mudança na arquitetura de software e ligação a outros serviços serão inevitáveis, ou na pior das hipóteses, até alterar a definição dos processos ou modificar o projeto do sistema. Uma composição estática nesses casos é restritiva e limitada. As ferramentas *e-flow* da HP (*Hewlett-Packard*) ou *StartWSCoP* (*Start Web Services Composition Platform*), ambas implementam um mecanismo de composição dinâmica [33].

O ambiente de serviços é altamente flexível e dinâmico, novos serviços são criados e disponibilizados diariamente, o número de provedores de serviços está em constante crescimento. Idealmente, processos de serviços devem se adaptar de forma transparente às mudanças no ambiente de negócios e requisitos de clientes devem ser obedecidos com mínima intervenção do usuário. A HP desenvolveu um sistema chamado *e-flow* para especificar, articular e monitorar composição de *e-services*. A HP define e-service como um meio em que as empresas oferecem seus produtos, serviços, recursos e *know-how* via Internet. *E-service* pode ser utilizado por pessoas (*business to consumer*), empresas (*business to business*) e dispositivos eletrônicos, e é portanto, um termo mais amplo do que *e-commerce* ou e-*business* [33].

E-flow é executado sobre Plataformas *E-Services* (ESPs), tais como HP *e-speak* ou Sun Jini, os serviços compostos são modelados por um grafo definindo o fluxo de invocações dos serviços, similar a um diagrama de

atividades UML. Sun et al. (2003) [32] introduz *StartWSCoP* com foco na composição dinâmica de serviços que consiste em quatro etapas:

- Provedores de serviços registram seus serviços em um registrador de serviços web;
- Um mecanismo de composição de serviços decompõe os requisitos do usuário em um serviço abstrato e envia uma requisição SOAP;
- O registrador de serviços web retorna um conjunto de serviços concretos;
- Um mecanismo de composição de serviços envia uma requisição SOAP para o serviço concreto e faz sua ligação (*bind*).

Para dar suporte a Qualidade de Serviço (QoS) na composição dinâmica de Serviços Web, o WSDL é estendido com propriedades de QoS, tais como tempo, custo ou confiabilidade:

```
<portType name="GetTotalFilmCost">
        <operation name="getTotalFilmCost"
                cost="20"
                time="50"
                reliabitity="90%">
        ...
        </operation>
</portType>
```

Figura 6 – Atributos de QoS.
Fonte [33]

Uma camada baseada em ontologia é adicionada ao UDDI para conseguir correspondência semântica de serviços web.

Pires et al. [34] introduz um *framework* chamado WebTransact que fornece uma infraestrutura necessária para construir composições de serviços web confiáveis.

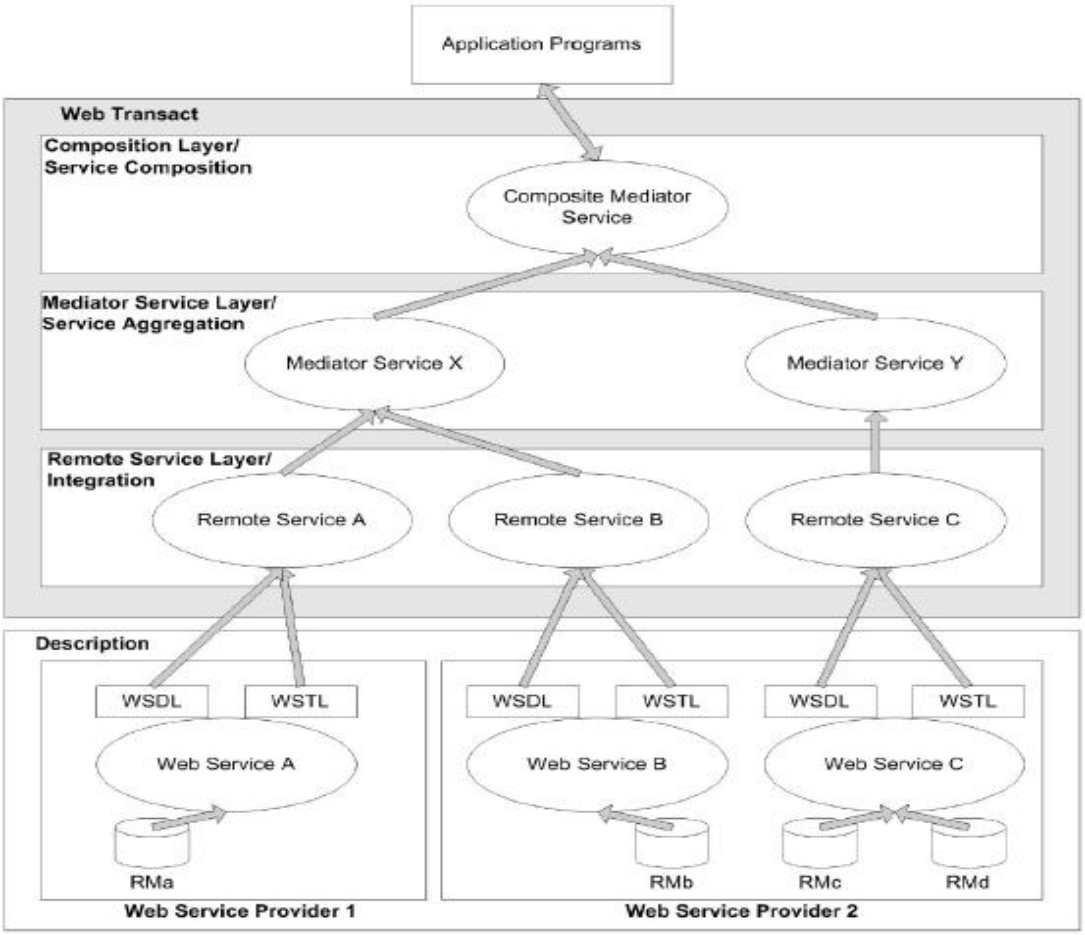


Figura 7 – A Arquitetura Multicamada em WebTransact. Fonte [34]

Em *WebTransact*, o *framework* é composto por uma estrutura de multicamadas contendo uma Camada de Composição de Serviços, uma Camada de Agregação de Serviços, uma Camada de Integração e uma Camada de Descrição como ilustra a Figura 7.

Iniciando na camada inferior de descrição (*Description Layer*), *WebTransact* usa WSDL para descrever as funcionalidades do serviço e adiciona uma Linguagem para Transação de Serviços Web – WSTL (*Web Service Transaction Language*) em cima de WSDL melhorando-o com funcionalidades que facilitam a composição de serviços web. Em suma, WSTL descreve o suporte para transação na web. O exemplo de código da Figura 8 ilustra um código WSDL e a extensão WSTL correspondente:

```
<operation name="getTotalFilmCost">
        <input message="tns:cancelSoapIn"/>
        <output message="tns:cancelSoapOut"/>
</oparation>

<wstl:transactionDefinitions>
<wstl:transactionBehavior operationName="getTotalFilmCost"
                          type="retriable"/>
</wstl:transactionDefinitions>
```

Figura 8 – Código WSDL e extensão WSTL.
Fonte [34]

A camada de integração define um serviço remoto como uma unidade lógica de trabalho que realiza um conjunto de operações em um determinado site. Cada operação tem um comportamento transacional bem definido.

A camada mediadora agrega serviços remotos semanticamente equivalentes provendo uma visão homogênea de serviços remotos heterogêneos. Serviços semanticamente equivalentes provêem diferentes descrições WSDL, mas com a mesma funcionalidade.

Para realizar a composição, um serviço mediador de composição descreve os padrões de interação de um conjunto de operações cooperantes necessárias para realizar determinadas tarefas.

3.2. Evolução das Abordagens de Composição de Serviços Web

Conforme tem aumentado o número de serviços web e diferentes desenvolvedores podem descrever o mesmo serviço de formas diferentes, processar automaticamente serviços em tempo de execução está se tornando cada vez mais necessário. Para tornar os serviços web mais acessíveis para processos automatizados, há um crescente interesse em associar semântica aos serviços web [38].

Para construir descrições de serviços web mais ricas, alguns métodos e abordagens desenvolvidos ao longo dos anos foram sumarizados na Tabela 1. Tais métodos foram divididos em dois tipos [38]:

- Construções de descrições sintáticas, tais como, WSDL [41] [48] ou BPEL [39] [40] [45] [46] [52];
- Geração de descrições semânticas baseadas em OWL [42] ou OWL-S [38] [43] [44] [47] [49] [50] [51] [54];

Autor	Ano	Característica	Saída
Sebastian et al. [39]	2002	Especificação de BPEL baseada em uma abordagem orientada a processos.	BPEL
Koch et al. [40]	2004	Transformação de diagramas de sequência usando uma linguagem para composição de serviços web.	BPEL
Skogan et al. [41]	2004	Uso de Diagramas de Atividades UML para projetar composição de serviços web.	WSDL
Djuric et al. [42]	2004	Uso de Ontologia UML Profile (OUP) para desenvolver ontologias.	OWL
Jaeger et al. [43]	2005	Especificação da descrição semântica de serviços web usando UML, XMI e XSLT.	OWL-S
Timm & Gannod [44]	2005	Especificação de processos atômicos OWL-S usando diagramas UML.	OWL-S
Gronmo & Oldevik [45]	2005	Introdução de uma ferramenta de modelo de transformação que transforma UML em BPEL.	WSDL, BPEL
Martinez et al. [46]	2005	Introdução de ZenFlow como uma ferramenta de escrita para composição de serviços web.	BPEL
Gronmo et al. [47]	2005	Construção de Ontologias OWL-S a partir de diagramas UML para descoberta, invocação e composição de serviços web.	OWL-S
Xiaofeng et al. [48]	2006	Transformação de modelos independentes de plataforma para interfaces de serviços web.	WSDL
Yang and Chung [49]	2006	Geração de OWL-S *Service Model* a partir de diagramas de classes e *state-chart* UML.	OWL-S
Timm & Gannod [50]	2007	Uso de diagramas de atividades para representar controle de fluxo para composição de processos.	OWL-S
Bensaber & Malki [51]	2008	Engenharia reversa em arquivos WSDL em modelos UML e geração de descrições OWL-S a partir de UML.	OWL-S
Il-Woong &	2009	Uso de MDA para transformar modelos UML em suas	OWL-S

Kyong-Ho [38]		respectivas representações OWL-S.	
Xin Li, Xinhuai, Zhaoteng and Xiaozhou [52]	2010	Combinação das abordagens de planejadores de Inteligência Artificial com Workflow para melhorar a manutenção de composições em BPEL.	BPEL
V. Sena, R. Amorim, D. Claro and D. Lopes [54]	2010	Propor um *plug-in* OWL-S Composer para IDE Eclipse facilitando a interoperabilidade de serviços Web semânticos e desenvolvimento de um algoritmo de similaridade semântica na descoberta de serviços.	OWL-S

Tabela 1 – Trabalhos Relacionados sobre Descrição de Serviços Web baseadas em MDA. **Estendida de Fonte [38]**

Alguns trabalhos anteriores com geração de descrição BPEL tratam processos complexos, mas não suportam semântica. A maioria dos trabalhos que geram descrição semântica de serviços web a partir de modelos UML não consideram composição de processos com vários controles de construção. Um sistema geralmente é modelado com diferentes tipos de diagramas e dentre eles diagramas complexos e diagramas relacionados. Abordagens anteriores não consideram relações entre a complexidade e constituição de diagramas.

3.3. Trabalhos Relacionados

Para o desenvolvimento desta dissertação foram destacados alguns trabalhos [35] [36] [37] [38] [54] [55] [56] que terão influência sobre a forma como a solução será construída e servirão de base para comparação com este trabalho.

3.3.1. Composição de Serviços Web e Ontologias

Alguns trabalhos relevantes tratam de propostas de composição dinâmica de serviços web utilizando ontologias ou semântica web. Dentre eles

a composição automática de serviços web semânticos, como em [35] e [36], propõe o uso de ontologias para fazer a descoberta automática dos serviços que satisfazem a necessidade do cliente. Já na abordagem "*WS Description for the Semantic Web*" [37], a diferença está na forma do uso da ontologia. Em vez de ser usada para identificar as **entradas do serviço** e seus efeitos sobre o sistema, é usada para identificar o **serviço** em si. E o algoritmo não apenas escolhe o serviço, mas constrói uma composição de serviços, com o menor tempo de execução e o melhor fluxo de dados, para a realização da necessidade do cliente. Para a especificação da ontologia do serviço, é construída uma estrutura hierárquica que parte do serviço mais genérico até o serviço mais especializado, como mostra a Figura 9. Cada nodo da hierarquia irá especificar um serviço abstrato, definindo o nome e o tipo dos parâmetros de entrada e saída.

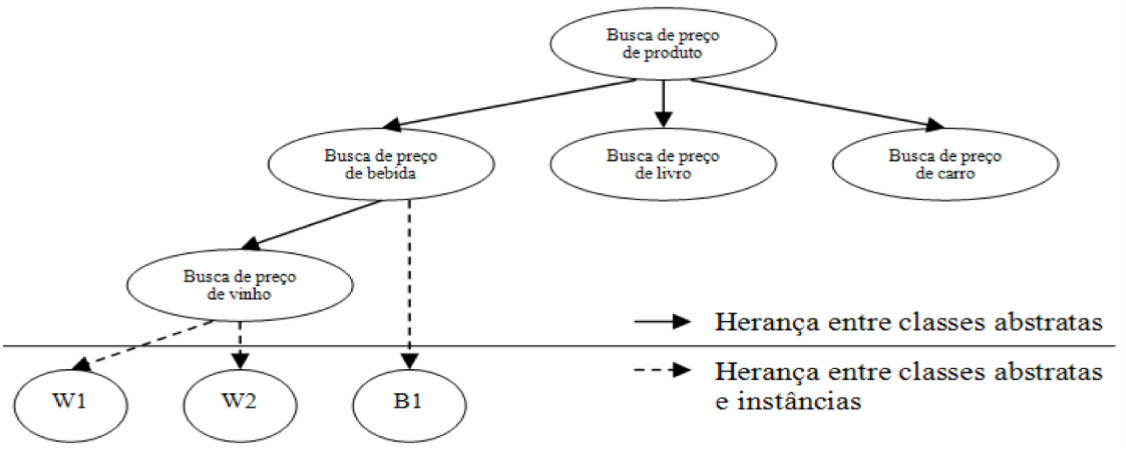


Figura 9 – Ontologia do domínio de procura de preço e instâncias de serviços. Fonte [37]

No exemplo da Figura 9, segundo o autor [37], observa-se que o serviço W1 e o serviço W2 pertencem a mesma ontologia. Assim, pode-se deduzir que eles têm a mesma interface e a mesma semântica. Já o serviço B1 tem uma semântica mais genérica, significando que os parâmetros de entrada e de saída recebem e retornam uma categoria mais genérica. Uma necessidade do cliente é expressa através de uma *composite service query* de uma maneira muito similar a uma descrição de serviço em DAML-S[2]. Esta *query* inclui a descrição e a interface do serviço composto, definindo as entradas, as saídas e as restrições da composição. O usuário pode especificar

[2] http://www.daml.org/

parcialmente como a composição do serviço deve trabalhar e o tipo dos serviços individuais esperados [37].

A desvantagem desta abordagem é que as regras do cliente e do serviço estão escritas baseadas na mesma ontologia, logo é preciso o cliente e o desenvolvedor do serviço ter o mesmo entendimento sobre os conceitos da ontologia.

3.3.2. Uma Abordagem Dirigida por Modelos para Composição Dinâmica de Serviços Web

Existem certos requisitos de negócio que sempre são difíceis de serem atendidos por um único serviço, logo é desejável compor um conjunto de serviços web utilizando processos de negócio para realizar esses tipos de tarefas. A definição de processos de negócio para realizar a composição de serviços web exige conhecimento de especialistas em negócios e recursos humanos. Dessa maneira, nesta abordagem o principal objetivo é permitir a rápida entrega e desenvolvimento de processos de negócio reutilizáveis [53].

Uma visão geral do funcionamento da abordagem começa com a projeção de dois modelos UML para definir aspectos básicos e semânticos de processos e serviços, como ilustra a Figura 10. O núcleo do paradigma de MDA é a transformação de Modelo Independente de Plataforma (PIM) para Modelo Específico de Plataforma (PSM). O primeiro Modelo de Serviços Independente de Plataforma descreve a semântica das funcionalidades dos serviços. O segundo Modelo de Processos Independente de Linguagem descreve aspectos relacionados aos requisitos de negócio.

Em seguida, implementações de serviços web e processos de negócio são geradas automaticamente, a partir dos dois modelos já citados anteriormente, seguindo padrões de mapeamentos e regras de transformação pré-definidos. O Modelo SSD (*Service Semantic Description*) contém a descrição semântica dos serviços disponíveis e o Modelo PPC *(Process Partner Contract*) que contém descrição semântica dos requisitos de contratos

entre os processos associados. As implementações dos serviços e o APT (*Abstract Process Template*) também são gerados.

Finalmente, um "*matchmaking*" ou correspondência entre SSD e PPC para selecionar dinamicamente o serviço qualificado é realizado. Depois de selecionado, ele é encaixado (composto) no APT para então ser produzidos os processos em BPEL.

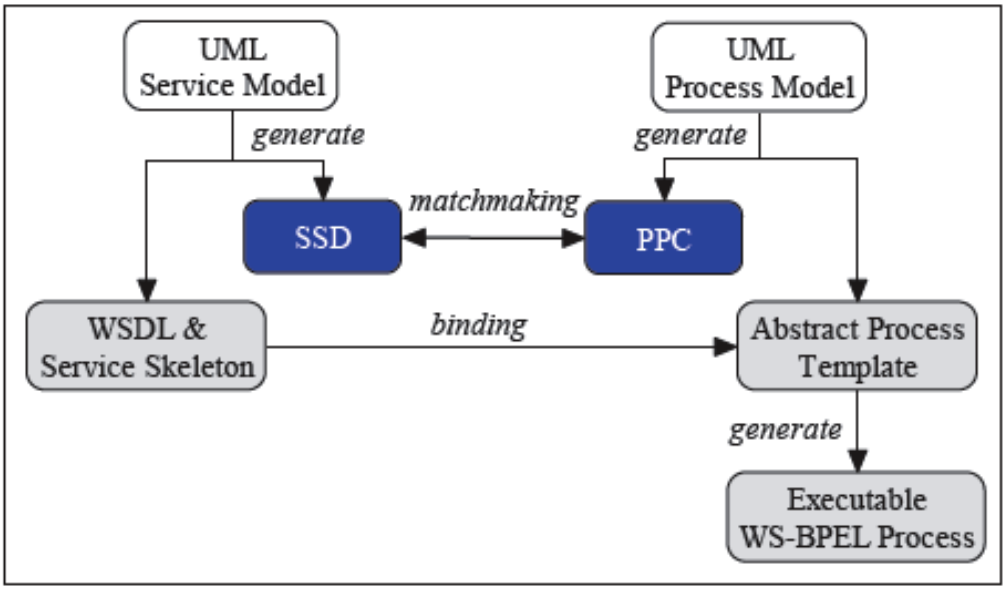


Figura 10 – Visão Geral da Abordagem.
Fonte [53]

A arquitetura do protótipo do sistema é ilustrada na Figura 11. Vale destacar alguns módulos e papéis dentro desta abordagem proposta, como o *designer* (projetista) de serviços web, que constrói modelos de serviços independentes de plataforma usando o USM (*UML Service Model*), em seguida transforma os modelos para os seus correspondentes em WSDL e SSD.

O módulo *Service Skeleton Generator* recebe como entrada o WSDL que é saída do *Web Service Modeler* e gera os *skeletons* que serve de base para os desenvolvedores, como guia para a implementação dos serviços.

O *designer* de processos permite a construção de modelos de negócio independente de linguagem utilizando o UPM (*UML Process Model*), e então transforma os modelos em APTs e PPCs.

O repositório semântico de serviços provê publicação e descoberta de capacidades de serviços, informações sobre semântica e interfaces dos serviços.

O APT *Binder* provê um editor para ligar (encaixar, acoplar) os serviços encontrados em um APT dinamicamente e transforma o APT correspondente em um processo de negócio executável, como ilustra a Figura 11.

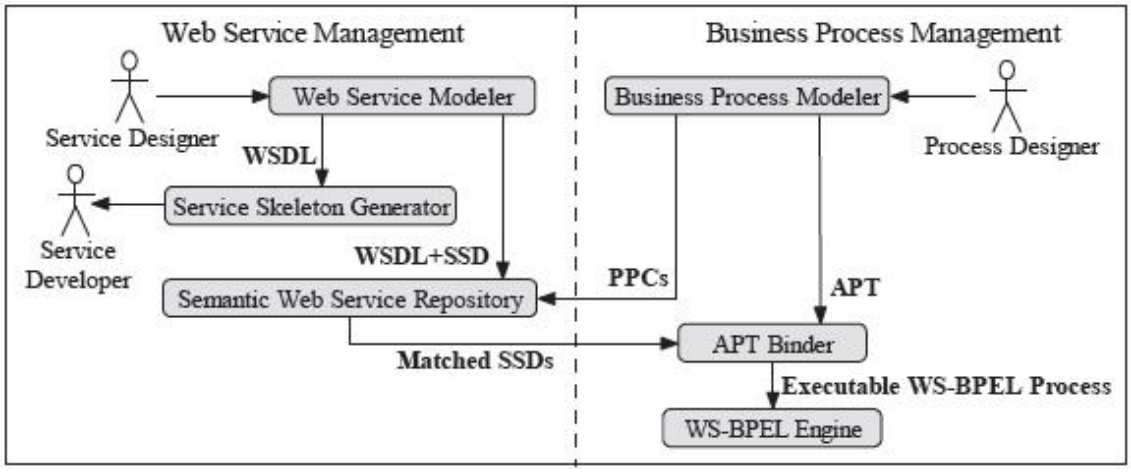


Figura 11 – Arquitetura do Protótipo do Sistema.
Fonte [53]

Conforme mostrado na Figura 11, o sistema pode ser dividido em duas partes, uma que realiza o gerenciamento de serviços web e a outra o gerenciamento de processos de negócios. A parte de gerenciamento de serviços web abrange todo o ciclo de vida dos serviços, incluindo o projeto, modelagem, desenvolvimento e publicação. Ao mesmo tempo, a parte de gerenciamento de negócios cobre quase todo o seu ciclo de vida, incluindo o projeto, modelagem, geração, *binding*, publicação e execução. No geral, o protótipo fornece um ambiente integrado de desenvolvimento de serviços web e composição. Como ponto negativo, a ausência de um metamodelo semântico de serviços baseado em ontologias, o que possibilitaria a extensão de características representativas de serviços.

3.3.3. Uma Abordagem Dirigida por Modelos para descrever Serviços Web Semânticos: De UML para OWL-S

Atualmente, com a grande quantidade e tipos de serviços web, existe um crescente interesse em semântica de serviços web baseada em ontologias. OWL-S é um “padrão de fato” baseado em ontologias para descrever semântica de serviços web. Devido a complexidade da gramática de OWL-S, é difícil (ou inviável) construir descrições OWL-S manualmente. Logo, o principal objetivo desta abordagem é utilizar uma abordagem dirigida por modelos para gerar ontologias OWL-S a partir de modelos UML, que são amplamente usados para modelagem e desenvolvimento de softwares [38].

O método proposto é baseado em Perfis UML para representar características de OWL-S, uma ontologia de um domínio é transformada em diagramas de classe, e diagramas UML são estendidos para representar o comportamento dos processos de negócio. Finalmente, um arquivo XMI (*XML Metadata Interchange*), extraído a partir de diagramas UML, é transformado em uma representação OWL-S através de um script XSLT (*eXtensible Stylesheet Language Transformation*).

O método proposto é dividido em três etapas [38]:

- **Modelagem de Ontologia**: As relações hierárquicas entre os conceitos de ontologias e suas propriedades constituintes são representados por um diagrama de classe. As relações de mapeamento entre os elementos de um diagrama de classe UML e uma ontologia OWL são apresentados na Tabela 2;

OWL	Class diagram
owl:Class	Class Name
owl:DatatypeProperty	Attribute
owl:ObjectProperty	
rdfs:subClassOf	Generalization

Tabela 2 – Tabela de relações de mapeamento entre Ontologias OWL e Diagramas de Classes. Fonte [38]

- **Modelagem de Processos:** Enquanto diagramas de classes e objetos representam informações estáticas, diagramas de sequência e atividades representam o comportamento de processos de negócio. A Tabela 3 apresenta as relações de mapeamento entre os elementos de um diagrama de sequência e construções OWL-S. Construções de controle OWL-S são descritos através de fragmentos de interação;

<table>
<tr><th>Type</th><th>OWL-S</th><th>Sequence diagram</th></tr>
<tr><td>Process</td><td>atomic process</td><td>Operation</td></tr>
<tr><td rowspan="8">Control construct</td><td rowspan="2">Sequence</td><td>Synchronous</td></tr>
<tr><td>Asynchronous</td></tr>
<tr><td>If-Then-Else</td><td rowspan="2">interaction fragment(alt)</td></tr>
<tr><td>Choice</td></tr>
<tr><td>Repeat-While</td><td rowspan="2">interaction fragment(loop)</td></tr>
<tr><td>Repeat-Until</td></tr>
<tr><td>Split + Join</td><td rowspan="2">interaction fragment(par)</td></tr>
<tr><td>Any-Order</td></tr>
<tr><td rowspan="3">Parameter</td><td>Input</td><td>arguments of operation</td></tr>
<tr><td>Output</td><td>return of operation</td></tr>
<tr><td>parameterType</td><td>type of attributes</td></tr>
<tr><td rowspan="2">Category</td><td>categoryName</td><td rowspan="4">Tagged value</td></tr>
<tr><td>taxonomy</td></tr>
<tr><td rowspan="2">Profile</td><td>serviceName</td></tr>
<tr><td>textDescription</td></tr>
<tr><td rowspan="3">Condition</td><td>Precondition</td><td rowspan="3">Constraint</td></tr>
<tr><td>Result</td></tr>
<tr><td>Effect</td></tr>
</table>

Tabela 3 – Tabela de relações de mapeamento entre OWL-S e Diagramas de Sequência. Fonte [38]

- **Transformação XSL (XSLT):** Nesta etapa, um arquivo XMI é extraído a partir de diagramas UML. Este arquivo XMI é transformado em uma Ontologia OWL-S via *script* XSLT. A Tabela 4 lista o mapeamento entre UML, XMI e construções OWL-S.

UML	XMI	OWL-S
Action	<UML2:Activity>	Process
Pin	<UML2:InputPin>	Input
Pin	<UML2:OutputPin>	Output
Pin to Pin	<UML2:ActivityEdge>	{In|Out}put Binding
Tagged Value	<UML2:TaggedValue> <UML:Stereotype name='categoryName'>	categoryName
Tagged Value	<UML2:TaggedValue> <UML:Stereotype name='taxonomy'>	taxonomy
Tagged Value	<UML2:TaggedValue> <UML:Stereotype name='serviceName'>	serviceName
Tagged Value	<UML2:TaggedValue> <UML:Stereotype name='textDescription'>	textDescription
Constraint	<UML2:Constraint> <UML:Stereotype name='preCondition'>	Precondition
Constraint	<UML2:Constraint> <UML:Stereotype name='Result'>	Result
Constraint	<UML2:Constraint> <UML:Stereotype name='Effect'>	Effect
Sequence	<UML2:ActivityEdge>	Sequence
Fork	<UML2:ForkNode>	Split

Tabela 4 – Tabela de relações de mapeamento entre UML, XMI e OWL-S. Fonte [38]

Nesta abordagem, o script XSLT é implementado com o JDOM (*Java Document Object Model*) para transformar arquivos XMI em suas versões OWL-S, parte deste script é ilustrado na Figura 12.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"xmlns:xsl=http://www.w3.org/1999/XSL/Transform...>
<xsl:output method="xml" indent="yes"/>
<xsl:template match="//model">
  <xsl:apply-templates select="package"/>
</xsl:template>
<xsl:template match="package">
<xsl:element name="rdf:RDF">
  <xsl:call-template name="main"/>
</xsl:element>
</xsl:template>
<xsl:template name="main">
  <xsl:if test="@stereoType='Service'">
   <xsl:element name="service:Service">
     <xsl:attribute                                    name="rdf:ID"><xsl:value-of
select="@name"/>Service</xsl:attribute>
     <xsl:element name="service:presents"><xsl:attribute name="rdf:resource">#
       <xsl:value-of select="@name"/>Profile</xsl:attribute>
     </xsl:element>
   </xsl:element>
  </xsl:if>
  <xsl:call-template name="ServiceProfile"/>
  <xsl:call-template name="ProcessModel"/>
</xsl:template>
<xsl:template name="ServiceProfile">
 <xsl:element name="profile:Profile">
  <xsl:for-each select="//UML2:CallAction[@name]">
   <xsl:if test="@stereoType='ServiceCategory'">
    <xsl:element name="profile:ServiceCategory">
     <xsl:attribute name="rdf:ID"><xsl:value-of select="@name"/></xsl:attribute>
    </xsl:element>
   </xsl:if>
  <xsl:for-each>
 <xsl:call-template name="ProfileInputs"/>
 </xsl:element>
</xsl:template>
```

Figura 12 – Script XSLT para criação de descrições OWL-S.
Fonte [38]

O método proposto pode representar todas as construções de controle OWL-S e suporta diagramas complexos misturados com diagramas de sequência. Uma característica essencial da proposta é o reuso de modelos UML existentes. Além disso, desenvolvedores de software podem especificar descrições semânticas de serviços web representadas por diagramas, sem precisar aprender todos os detalhes da gramática OWL-S.

3.3.4. Similaridade Semântica na Composição de Sistemas de Informação através dos Serviços Web

O crescente desenvolvimento de sistemas de informação requer a utilização de ferramentas amigáveis e funcionais para facilitar e agilizar o processo de construção destes sistemas. A utilização de ferramentas amigáveis e funcionais no processo de desenvolvimento de um serviço web já foi incorporado em diversos ambientes integrado de desenvolvimento (IDE - *Integrated Development Environment*) como o Eclipse IDE, o que aumenta a produtividade e diminui a curva de aprendizagem de um desenvolvedor Web [54].

Em D. Claro et al [54] é proposto o desenvolvimento e a incorporação de um algoritmo de similaridade semântica na descoberta de serviços web através de um *plugin,* chamado OWL-S *Composer* dentro de um ambiente integrado como o Eclipse. Assim, este trabalho adiciona características semânticas aos serviços web, garantindo um arquivo de saída sintaticamente e semanticamente correto, ampliando a utilização destes serviços em sistemas de informação.

Como ilustra a Figura 13, o OWL-S Composer 2.0 foi desenvolvido sob a versão 3.4.1 do Eclipse, utilizando os plugins EMF, GMF, GEF e JET. O GMF, EMF e GEF não sofreram modificações nestas novas versões. Especificamente, o GMF é responsável pela geração dos diagramas semanticamente similares, o JET gera automaticamente o código da composição em OWL. Dentre as ferramentas independentes, OWL-S API foi fundamental para interpretar serviços retornados pelo OWL-S Discovery e mapear suas funcionalidades no metamodelo [54].

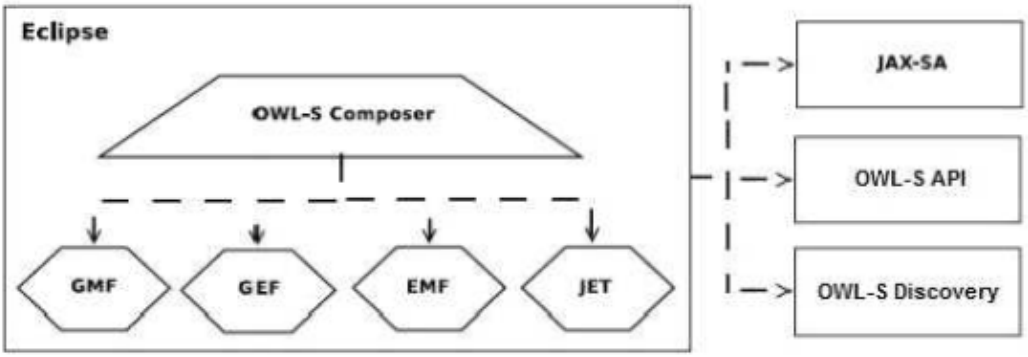


Figura 13 – Arquitetura de OWL-S Composer 2.0.
Fonte [54]

O OWL-S Discovery implementa um algoritmo de descoberta semântica oferecendo uma opção ao usuário de encontrar serviços que podem ser relevantes, tornando o OWL-S Discovery flexível com opções de apenas etapas semânticas funcionais ou descritivas. O OWL-S Composer 2.0 utiliza apenas a parte semântica funcional do OWL-S Discovery.

Na validação do *plugin* OWL-S *Composer 2.0*, os requisitos mínimos para uma ferramenta de composição semântica de serviços foram avaliados. Dentre os critérios de validação, vale destacar o “Grau de Similaridade”. Ou seja, a ferramenta deve oferecer opções ao usuário quanto ao grau de similaridade semântica que os serviços são encontrados, além de informar o grau correspondente ao apresentar o resultado. Os graus de similaridades disponíveis são *Exact, Plugin, Subsumes e Sibling* [54].

Os graus de similaridade também denominados de filtros semânticos trabalham com conceitos, relações e axiomas dentro de uma ontologia de um domínio. O grau mais alto de similaridade é denominado *Exact*, onde dois termos são idênticos ou filhos direto dentro da ontologia de um domínio [54].

Este trabalho propôs o desenvolvimento de um algoritmo híbrido (*OWL-S* Discovery) e a incorporação deste no OWL-S Composer 1.0. Desta maneira, o OWL-S Composer 2.0 oferece suporte a mais uma etapa no ciclo de vida dos serviços web, a fase da descoberta, além da composição visual dos serviços web.

3.3.5. Composição Dinâmica de Serviços Web utilizando padrões de projeto.

Em [55], as invocações a serviços disponíveis na Internet são construídas de forma estática, sempre referenciando o mesmo serviço web e o mesmo *web method*. Quando este serviço apresentar baixa disponibilidade o desempenho da aplicação será reduzido. Para evitar este problema, é necessário que a aplicação tenha a habilidade de identificar o melhor serviço disponibilizado e então possa invocá-lo. A inserção de novos protocolos e novas funcionalidades na arquitetura de serviços web pode permitir que as aplicações encontrem serviços disponíveis na Internet e, além disso, possam medir a qualidade do serviço disponível e assim direcionar sua chamada para o melhor serviço. Padrões de projeto são usados como um instrumento para uma melhor compreensão da arquitetura proposta [55].

O principal objetivo do trabalho em [55] é modificar a arquitetura de serviços web para que a aplicação possa suportar composição dinâmica com serviços web, permitindo que a composição seja feita em tempo de execução pela aplicação.

A arquitetura de um serviço web é formada por três participantes: o solicitante do serviço, o provedor de serviços e o registro de serviços que é o diretório como ilustra a Figura 14.

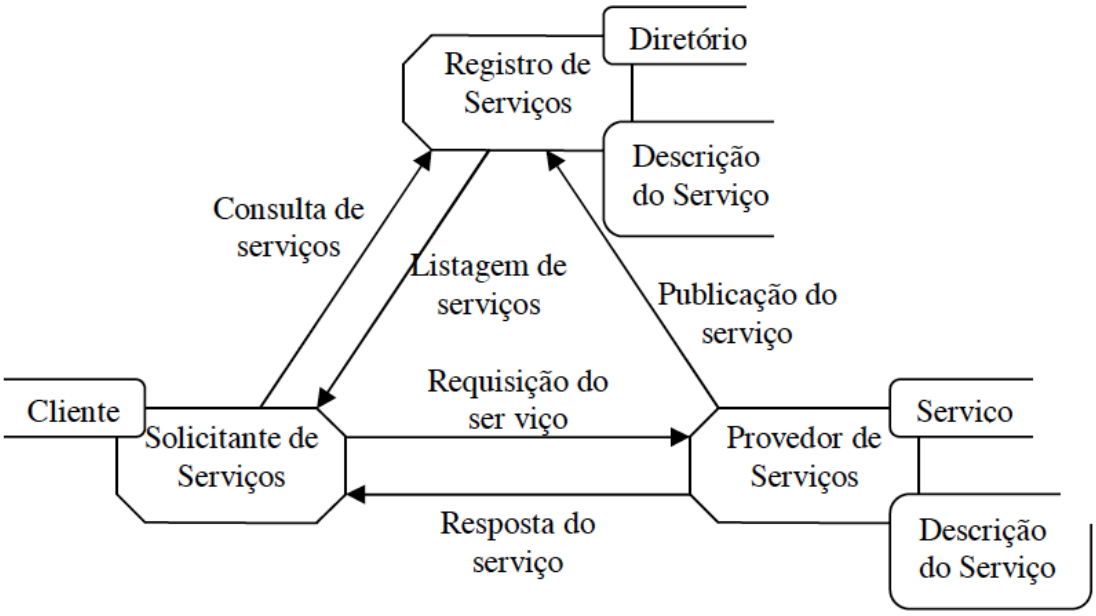


Figura 14 – Arquitetura de Web Services.
Fonte [55]

Algumas características da composição dinâmica de serviços podem ser capturadas por padrões de projeto. Como ilustra a Figura 15, a arquitetura proposta em [55] é baseada em padrões de projeto. A característica de receber notificações de novos serviços publicados no diretório de serviços é capturada pelo padrão *Observer*. A segunda característica que é a escolha de um entre vários serviços disponíveis para composição é capturada pelo padrão *Strategy*. Ambos os padrões são compreendidos pela arquitetura MVC (*Model, View, Controller*). Este último define uma dependência um-para-muitos entre objetos, de maneira que quando um objeto muda seu estado todos os seus dependentes são notificados e atualizados automaticamente; e também: pode variar o comando a ser executado dependendo do contexto do sistema. Usar padrões de projeto na modelagem da arquitetura de serviços web torna mais legível o diagrama e explicita o papel de cada componente dentro da estrutura [55].

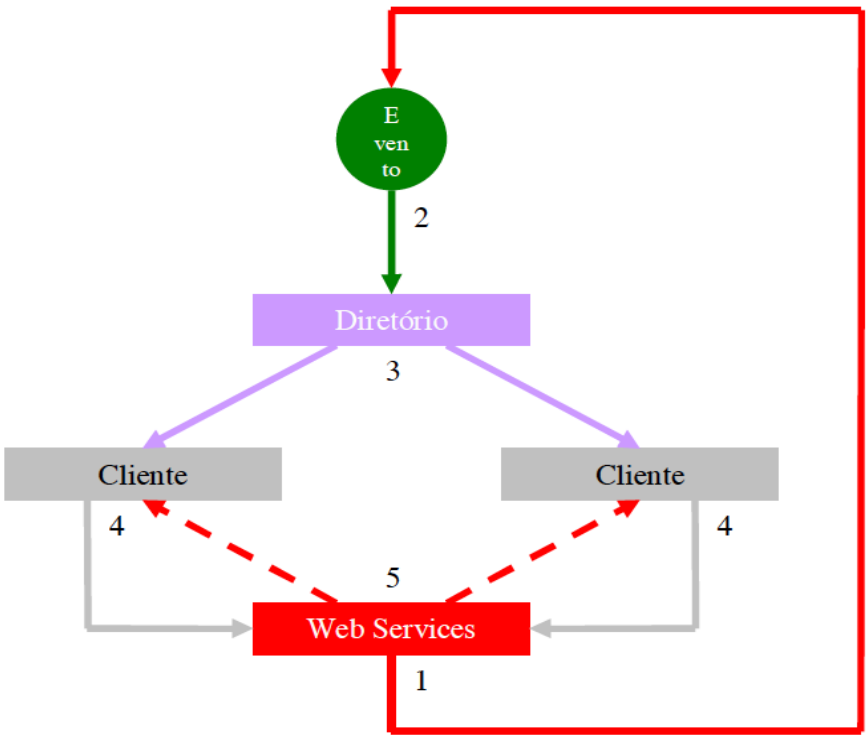


Figura 15 – Fluxo de composição baseado no modelo MVC.
Fonte [55]

Como ilustra a Figura 15, quando a publicação de um novo serviço ocorre, é gerado um evento (1). Esta publicação é capturada pelo diretório de serviços (2), por estar recebendo o registro de serviços. Assim, o diretório pode notificar os clientes (3) dizendo que um novo serviço está disponível. Com a notificação, os clientes atualizam a lista de serviços disponíveis e passam a levantar estatísticas sobre os novos serviços. Quando existir uma composição mais otimizada do que a composição atual, os clientes desfazem a composição e criam uma nova composição (4 e 5) [55].

A arquitetura real utilizando como o molde a pseudo-arquitetura baseada em padrões de projeto foi projetada como ilustra a Figura 16. Para que as colaborações entre os componentes sejam realizadas, os componentes precisam ser alterados e novos protocolos de comunicação precisam ser inseridos.

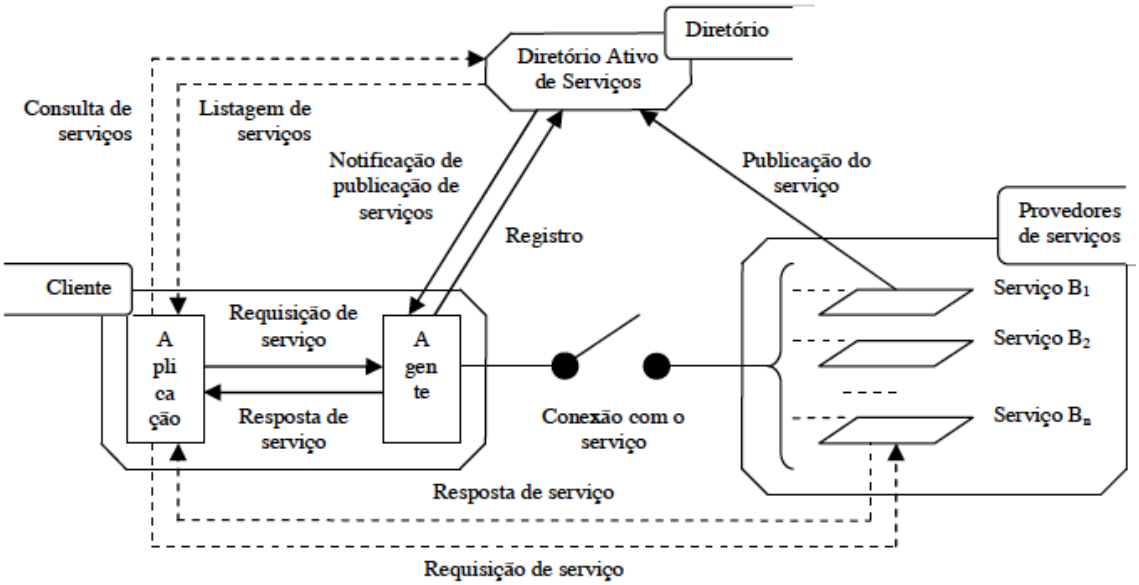


Figura 16 – Arquitetura de web services com suporte a composição dinâmica. Fonte [55]

O diretório de serviços passa a ser chamado diretório ativo de serviços por notificar a aplicação cada vez que um novo serviço é publicado. Os protocolos de consulta de serviços e listagem de serviços passam a ser desnecessários, sendo substituídos pelo registro e notificação de publicação de serviços. O registro do cliente no diretório ativo é uma chamada ao serviço web que passa sua localização (URL) e qual o tipo de serviço esperado (rótulo utilizando ontologias para marcação). Utilizando estes parâmetros, o diretório de serviços faz uma chamada a um serviço web para a aplicação, invocando a notificação de publicação de serviços, passando como parâmetros a localização, o nome do método e os parâmetros do serviço publicado [55].

No lado cliente, existe um agente responsável pela composição, ou seja, o cliente é dividido em duas partes, a aplicação propriamente dita e o agente. Dessa forma, a aplicação faz uma chamada de um método para o agente que redireciona a chamada para o serviço web com **maior disponibilidade**, ou seja, o agente funciona como um mecanismo de QoS que requisita o melhor serviço. Neste trabalho, a principal unidade de medida é o tempo de resposta.

Em resumo, esta abordagem tem o objetivo de modificar a arquitetura de serviços web. Para que o cliente garanta estar usando o melhor serviço, ele deve estar constantemente observando o estado atual da rede. É preciso uma visão atualizada dos serviços disponíveis e seus estados atuais

para a escolha do serviço com maior disponibilidade. Muitas vezes surge a necessidade de realizar uma sequência de invocações todas para o mesmo serviço, e isto não é tratado pela arquitetura e implementação deste trabalho [55].

3.4. Análise dos Trabalhos Relacionados

Nesta seção, alguns critérios destacados entre os trabalhos relacionados serão tratados. A análise destes trabalhos e a comparação com a abordagem proposta nesta dissertação são fundamentais para a definição dos critérios avaliativos que servirão de base para conclusões e considerações do presente trabalho .

Dentre os critérios, cita-se:

1. A geração automatizada de modelos semânticos de serviços com informações que permitem um *matching* mais preciso quando ocorrer a busca pelo serviços desejado;
2. Modelos podem estar representados em um formato XMI que é interpretável por várias ferramentas de metamodelagem;
3. Fácil extensão do modelo de serviços através do seu metamodelo semântico, essa característica implica em conhecimento especialista sobre metamodelagem, porém diminui eficientemente o custo com manutenção de modelos e reduz o tempo de desenvolvimento;
4. Os serviços podem estar organizados hierarquicamente, partindo dos serviços mais genéricos para os serviços mais específicos. Ou ainda, estruturados como metamodelos que descrevem modelos, incluindo características genéricas como classificação, categoria, produto relacionado com o serviço, etc;
5. Algumas abordagens devem permitir a rápida entrega e desenvolvimento de processos de negócio reutilizáveis;
6. Utilização de padrões “de fato”, ou padrões, arquiteturas e protocolos já consagrados na Internet é também um fator importante;

7. Utilização de Ontologia em Alto Nível (*Upper Ontology*). “Uma ontologia em alto nível (ou superior) é limitada aos conceitos que são meta, genéricos, abstratos e filosóficos, portanto são gerais o suficiente para lidar com uma ampla gama de áreas de domínio (em alto nível). Conceitos específicos de um dado domínio não são incluídos, no entanto ela fornece uma estrutura e um conjunto de conceitos gerais, sob os quais ontologias de um domínio específico (por exemplo, medicina, engenharia, financeiro, etc.) podem ser construídas” [14];
8. O uso de modelos UML convencionais expandidos para suportar semântica de serviços web e utilização de UML para a criação de modelos independentes de plataforma;
9. Desenvolvimento de uma Ontologia de domínio através de uma DSL.

Abordagens / **Critérios**	Ankolenkar, M. Burstein, J. R. Hobbs, O. Lassila [37], **WS Description for the semantic web**	Chenting Zhao, Zhenhua Duan, Man Zhang [53], **A Model-Driven Approach for Dynamic Web Service Composition**	Il-Woong Kim; Kyong-Ho Lee [38], **A Model-Driven Approach for Describing Semantic Web Services: From UML to OWL-S**	D. Lopes, D. Claro, V. Sena and R. Amorim [54], **Similaridade Semântica na Composição de Sistemas de Informação através dos Serviços Web**	Martins, Rogério Samuel de Moura [55], **Composição Dinâmica de Serviços Web utilizando padrões de projeto**
Geração automatizada de modelos semânticos de serviços.	Não	Sim	Sim. Representados por OWL-S.	Sim	Não
Geração e Exportação em XMI.	Não	Não	Sim	Sim	Não
Grau de Similaridade Semântica entre os Serviços Web	Não	Não	Não	Sim	Não
Construção de serviços e processos de negócio independente da tecnologia da plataforma ou linguagem de descrição de processos de negócio	Não	Sim	Sim	Sim	Não
Seleção Dinâmica de Serviço e capacidade de *binding*.	Não	Sim	Não	Não	Sim

Fácil extensão do modelo de serviços através do seu metamodelo semântico.	**Não** possui um metamodelo	**Não** possui um metamodelo	Não	Não	Não
Estrutura Hierárquica (do serviço mais genérico para o serviço mais específico).	Sim	Não	**Sim**. Representados por Diagramas de classe UML	Sim	Não
Estrutura por Classificação, Categoria, Produto, etc.	**Não**. Usa hierarquia	Não	Não	Não	Não
Reutilização em alto nível de processos de negócio em qualquer plataforma.	Não	**Sim.** WS-BPEL	Sim	Sim	Não
Utilização de padrões “de fato” ou consagrados.	Não	**Sim**. UML, WSDL	Sim. OWL-S, UML	Sim	Sim
Redução do custo, tempo e complexidade de desenvolvimento.	Não	Sim	Sim	Sim	Não
Utilização de Ontologia em Alto Nível (*Upper Ontology*).	Não	Não	Não	Não	Não
Rápida entrega no desenvolvimento de processos de negócio reutilizáveis.	Não	Sim	**Não**. Modificar Diagramas de Classe e Sequência para um novo processo	**Sim.** De forma gráfica com menus específicos a disposição.	Não
Uso de Modelos convencionais UML expandidos para suportar semântica de serviços web.	Não	Sim	Sim.	Não	**Não**. Padrões de Projeto
Desenvolvimento de uma Ontologia do Domínio através de uma DSL.	Não	Não	Não	Não	Não
Utilização de UML para a criação de modelos independentes de plataforma.	Não	Não	Sim	Não	Não

Tabela 5 – Tabela de Critérios segundo Análise de trabalhos Relacionados.

A Tabela 5 ilustra um comparativo, salientando os principais critérios apontados entre os trabalhos pesquisados. No decorrer da análise dos trabalhos, procurou-se selecionar características (ou critérios) com forte relação com o projeto, *framework*, objetivos e contribuições do trabalho proposto nesta

dissertação. Dentre os trabalhos relacionados, a abordagem “Similaridade Semântica na Composição de Sistemas de Informação através dos Serviços Web” por D. Claro, V. Sena, R. Amorim and D. Lopes [54] e “*A Model-Driven Approach for Describing Semantic Web Services: From UML to OWL-S*” por Il-Woong Kim; Kyong-Ho Lee [38] atenderam a maior parte dos critérios apontados. Dentre os critérios atendidos nos dois trabalhos, vale destacar a geração automatizada de modelos semânticos de serviços, reutilização e desenvolvimento de processos de negócio independentes de plataforma, estrutura hierárquica para representação dos serviços (genérico para específico), redução do custo, tempo e complexidade de desenvolvimento de serviços web. Dessa forma, as soluções empregadas por estes trabalhos têm maior semelhança com a solução apresentada nesta dissertação.

Os trabalhos “WS Description for the semantic web” por Ankolenkar, M. Burstein, J. R. Hobbs, O. Lassila [37] e “Composição Dinâmica de Serviços Web utilizando Padrões de Projeto” por Martins, Rogério Samuel de Moura [55] não têm muitos critérios atendidos, porém critérios como seleção dinâmica de serviços, capacidade de *Binding* e utilização de padrões consagrados são importantes, pois foram estudados para o desenvolvimento da solução proposta neste trabalho.

O trabalho “A Model-Driven Approach for Dynamic Web Service Composition” por Chenting Zhao, Zhenhua Duan, Man Zhang [53] apesar de utilizar uma abordagem dirigida por modelos para realizar a composição dinâmica de serviços web, não possui um metamodelo semântico de serviços baseado em ontologias, para facilitar a extensão de características representativas de serviços.

Na análise dos trabalhos relacionados, além de destacar os critérios atendidos e não atendidos de cada trabalho pesquisado, o principal objetivo é desenvolver neste trabalho de dissertação uma proposta que vise cobrir os melhores critérios de cada abordagem, bem como os critérios não atendidos por alguns trabalhos, todos relacionados na Tabela 5.

3.5. Síntese

Este capítulo de Estado da Arte, relata a base da pesquisa bibliográfica realizada neste trabalho, descrevendo as principais abordagens para fazer a composição de serviços web utilizadas no meio acadêmico e industrial. Dentro dessa pesquisa sobre composição dinâmica de serviços web, várias vertentes de soluções foram encontradas. Um grande esforço tem sido gasto pela indústria para descobrir soluções baseadas nas tecnologias WSDL + WS-BPEL. Já a comunidade de Web Semântica e Laboratórios de Pesquisa propõe soluções baseada em OWL-S + Ai *planning*. Uma melhor eficiência no desenvolvimento de processos de negócios tem sido alcançada, devido a abordagens de MDE fornecer uma resposta viável ao gerenciamento da complexidade de desenvolvimento, manutenção e evolução de softwares.

Composição de Serviços Web Estática e Dinâmica foram apresentadas segundo algumas abordagens e exemplificadas através de ferramentas de mercado, de empresas como Microsoft, HP, etc. Alguns trabalhos relacionados sobre Descrição de Serviços Web baseadas em MDA foram apresentados. Uma tabela composta pelos campos **Autor, Ano, Característica e Saída** foram preenchidos para cada abordagem, tal tabela foi estendida englobando os trabalhos recentes de 2010. Alguns trabalhos, fortemente relacionados com a dissertação, foram destacados como base de comparação e influência sobre o desenvolvimento da solução proposta. Finalmente uma análise destes trabalhos, ponderando os critérios organizados em uma tabela comparativa, foi realizada com o objetivo de enriquecer a pesquisa.

4. Proposta de um *Framework* para suportar a Composição Dinâmica de Serviços Web

Neste capítulo, apresenta-se um *framework* chamado *Dynamic Web Service Composition based on MDE and Ontologies* (DWSC) com o objetivo de realizar a recuperação de uma composição de serviços web, dada uma falta em um dos serviços compostos.

Uma metodologia é proposta para descrever os passos para suportar a composição dinâmica de serviços web. Em seguida, os metamodelos desenvolvidos que vão compor a solução proposta são apresentados.

4.1. Abordagem

A abordagem proposta para fazer a composição dinâmica de serviços web é dividida em duas etapas, a primeira referente a criação dos modelos semânticos de serviços e a segunda referente ao *Matching* (casamento, correspondência) das informações destes modelos semânticos de serviços.

4.1.1. Primeira Etapa

Na primeira etapa da abordagem, dar-se-á a criação de modelos semânticos de serviços para fins de comparação de suas características. Tais modelos são desenvolvidos conforme o metamodelo semântico de serviços proposto neste trabalho (ver seção 4.3), que possui aspectos semânticos e estruturais de serviços, empregados com Semântica Web, Ontologias e UML.

Esta primeira etapa foi organizada em três níveis, de acordo com os níveis de abstração dos elementos (metamodelo, modelos e artefatos), como ilustra a Figura 17.

Primeiro Nível: Metamodelos

Como ilustra a Figura 17, no nível de metamodelos, encontra-se a proposta do **metamodelo semântico de serviços web (OWL-S/UML *Metamodel*)**, que este está conforme o metamodelo OWL-S e o metamodelo UML (*Ontologies Metamodel* e *UML Metamodel*).

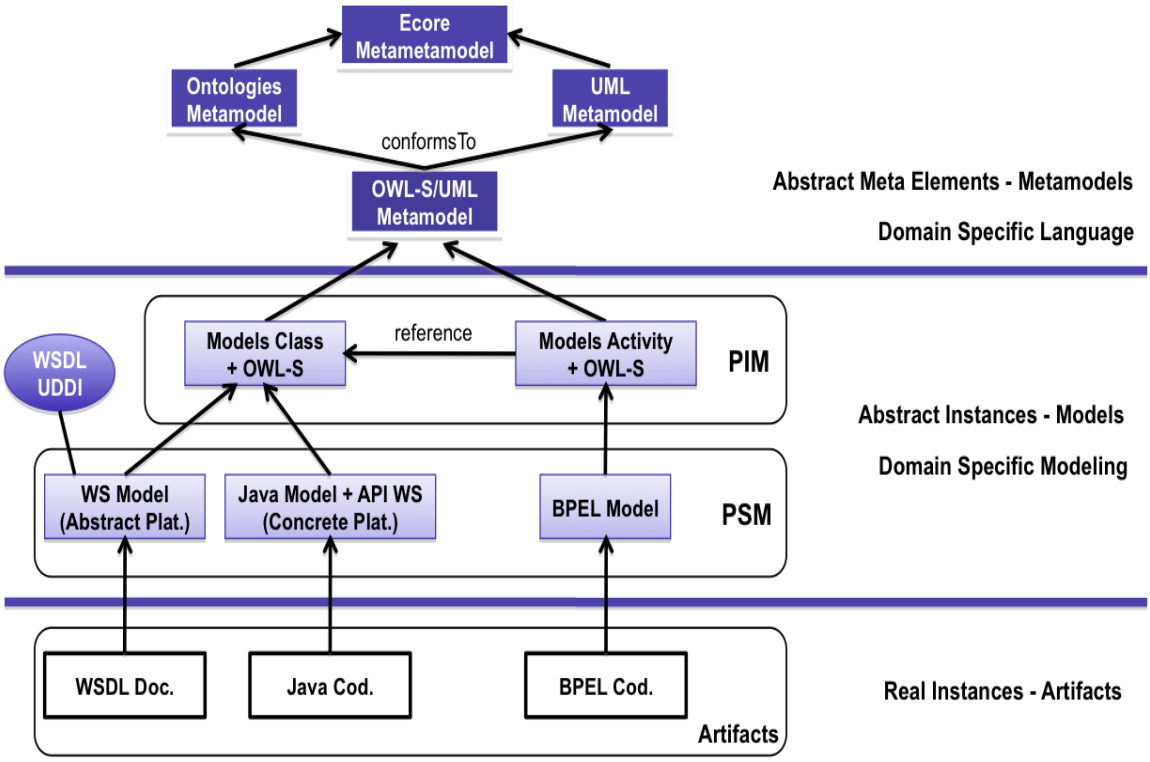


Figura 17 – Níveis de Abstração dos Elementos da Abordagem.

Através do metamodelo semântico proposto, que será apresentado neste capítulo na subseção 4.3.1, os aspectos semânticos e estruturais de um serviço web serão descritos. Este metamodelo tem em sua essência características do metamodelo de Ontologias (OWL-S) e do metamodelo UML como ilustra a Figura 17, por sua vez, estes dois metamodelos estão conforme o *Ecore Metametamodel*.

Segundo Nível: Modelos

Como ilustra a Figura 17, no nível intermediário de Modelos Específicos do Domínio (*Domain Specific Model*), todos os elementos são criados conforme o **Metamodelo Semântico de Serviços** (*OWL-S/UML Metamodel*). Neste nível, modelos de serviços representados por diagramas de classe UML (***Class Models***) definem os aspectos semânticos conforme OWL-S. Quando um modelo de serviço é criado, os processos, *workflow* ou aspectos estruturais dos serviços são modelados através dos diagramas de atividades (***Activity Models***). *Activity Models* fazem referência aos modelos semânticos de serviços (*Class Models*). Como ilustra a Figura 17, esses dois modelos compõem os **PIMs** (***Platform Independent Model***) em MDA e a partir deles são gerados modelos do tipo **PSM** (***Platform Specific Models***).

Como ilustra a Figura 17, ***WS Model*** é específico para a plataforma de serviços web, mas este não descreve a plataforma como um todo, ou seja, descreve apenas informações semânticas que não são consideradas pela plataforma. Portanto, ***WS Models*** cobre a parte abstrata da plataforma de serviços web, não descartando o WSDL que descreve a parte concreta de serviços, tais como *types, message, operation, port type, binding*.

WS Models mais o WSDL serão publicados, localizados e comparados em um repositório (“UDDI”). Por outro lado, o Modelo Java mais a API Web Service descrevem a plataforma, ou seja, a parte concreta da plataforma. O último modelo, neste segundo nível, é específico para BPEL (*Business Process Enterprise Language*) e está conforme com o *Activity Model*.

Terceiro Nível: Artefatos

No Nível de Artefatos da abordagem, ilustrado na Figura 17, instâncias reais baseadas nos modelos específicos de plataforma são apresentadas. Estes artefatos são documentos WSDL, documentos BPEL com interações entre os processos de negócio ou serviços e, finalmente, artefatos como código Java de acordo com a API de Serviços Web.

4.1.2. Segunda Etapa

Na segunda etapa da abordagem, dada uma falta em um serviço web da composição, a busca e composição dinâmica de um novo serviço web será realizada. Para tal tarefa, as informações do modelo semântico do serviço web que faltou serão utilizadas para realizar o *Matching* (correspondência) com as informações dos modelos semânticos de serviços disponíveis no repositório.

A Figura 18 ilustra como os modelos de serviços são usados para realizar o *Matching* de modelos.

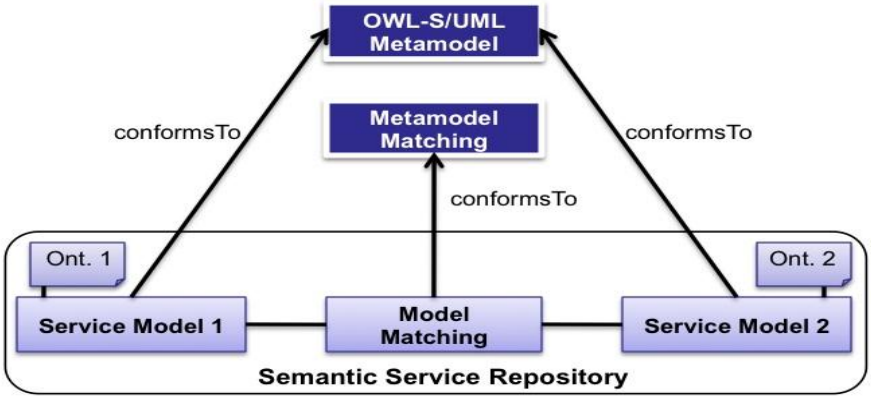


Figura 18 – Geração de Ferramentas e Artefatos.

Os modelos ***Service Model* 1** e ***Service Model* 2** são criados conforme o Metamodelo Semântico proposto (OWL-S/UML *Metamodel*). ***Ont*. 1** refere-se ao perfil do serviço baseado em Ontologias. Observa-se na Figura 18, que o **Modelo de Matching (*Model Matching*)** estabelece as correspondências entre os elementos de *Service Model 1 e Service Model 2*. Para isso, o *Model Matching* deve estar conforme ao **Metamodelo de Matching** proposto (***Metamodel Matching***), que foi projetado para o domínio de serviços e será apresentado mais a frente, juntamente com o **Metamodelo Semântico de Serviços Web**.

Os modelos de serviços (*Service Models*) conforme o metamodelo semântico proposto estarão contidos no repositório semântico de serviços (*Semantic Service Repository*).

4.1.3. Modelagem do *Framework*

O *framework* DWSC suporta o desenvolvimento de componentes para recuperar uma composição de serviços web. Para a modelagem do *framework*, um diagrama de classes foi criado, para representar as relações entre as classes de alguns componentes que serão desenvolvidos, como ilustra a Figura 19.

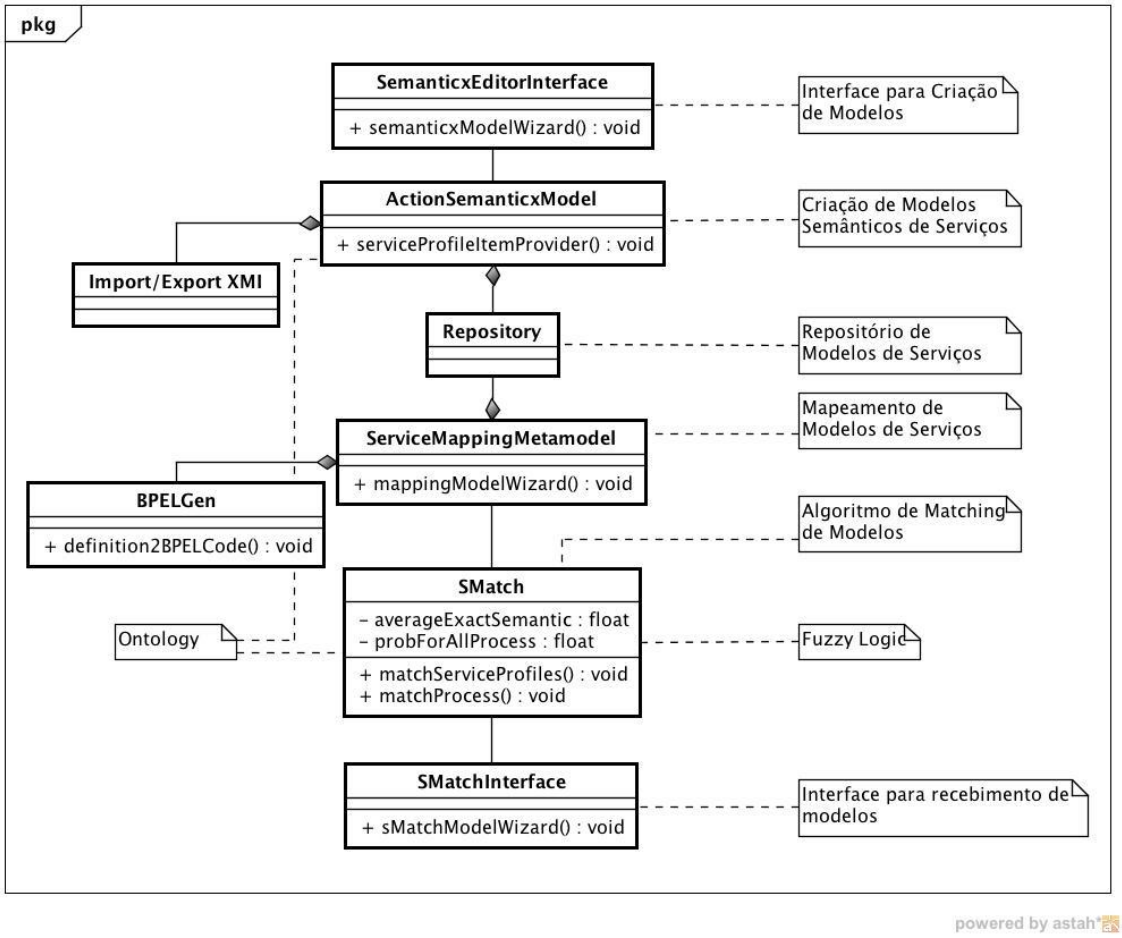


Figura 19 – **Diagrama de Classes do Framework.**

O *framework* DWSC é composto pelas seguinte classes:

- *SemanticxEditorInterface* – funcionalidade de interação com o usuário através de *Wizards* e *Interface*s permitindo manipular, criar e editar modelos semânticos de serviços web;
- *ActionSemanticxModel* – funcionalidades para criação e edição de modelos semânticos de serviços baseados em ontologias (*Ontology*).

Opções para salvar e excluir modelos do repositório e importar ou exportar modelos para o formato XMI;

- *Repository* – todos os modelos semânticos criados são armazenados no repositório de modelos de serviços;
- *Import/Export XMI* – permite importar ou exportar todas as informações de um modelo semântico de serviço para um arquivo XML no formato XMI (*XML Metadata Interchange*). Um modelo no formato XMI é reconhecido e manipulável por outras ferramentas de metamodelagem;
- *SMatchInterface* – funcionalidade de interação com o usuário através de *Wizards* e *Interface*s permitindo realizar o *Matching* de modelos semânticos de serviços web existentes no repositório;
- *SMatch* – responsável pela comparação de informações semânticas entre dois modelos de serviços. Permite navegar entre os modelos e calcula o grau de similaridade entre dois modelos cujo valor é conseguido comparando elementos de modelos, quanto maior for a quantidade de elementos iguais entre dois modelos maior será seu grau de similaridade (ver seção 4.4). A representação das características semânticas dos serviços e navegação entre os modelos é conseguida através de ontologias (*Ontology*);
- *BPELGen* – é responsável pela geração de código na linguagem de processos de negócio BPEL, através do método *definition2BPELCode()* que captura informações de correspondências entre elementos de modelos;
- *ServiceMappingMetamodel* – é responsável por criar correspondências entre os modelos já criados e armazenados no repositório.

4.2. Metodologia para suportar a Composição Dinâmica de Serviços Web

Esta seção trata da metodologia proposta para suportar a composição dinâmica de serviços web, seguindo a abordagem e o *framework* DWSC propostos.

Como ilustra a Figura 20, a metodologia usada no *framework* DWSC é apresentada na forma de diagrama de atividades, descrevendo cada passo a ser seguido para a utilização deste *framework*.

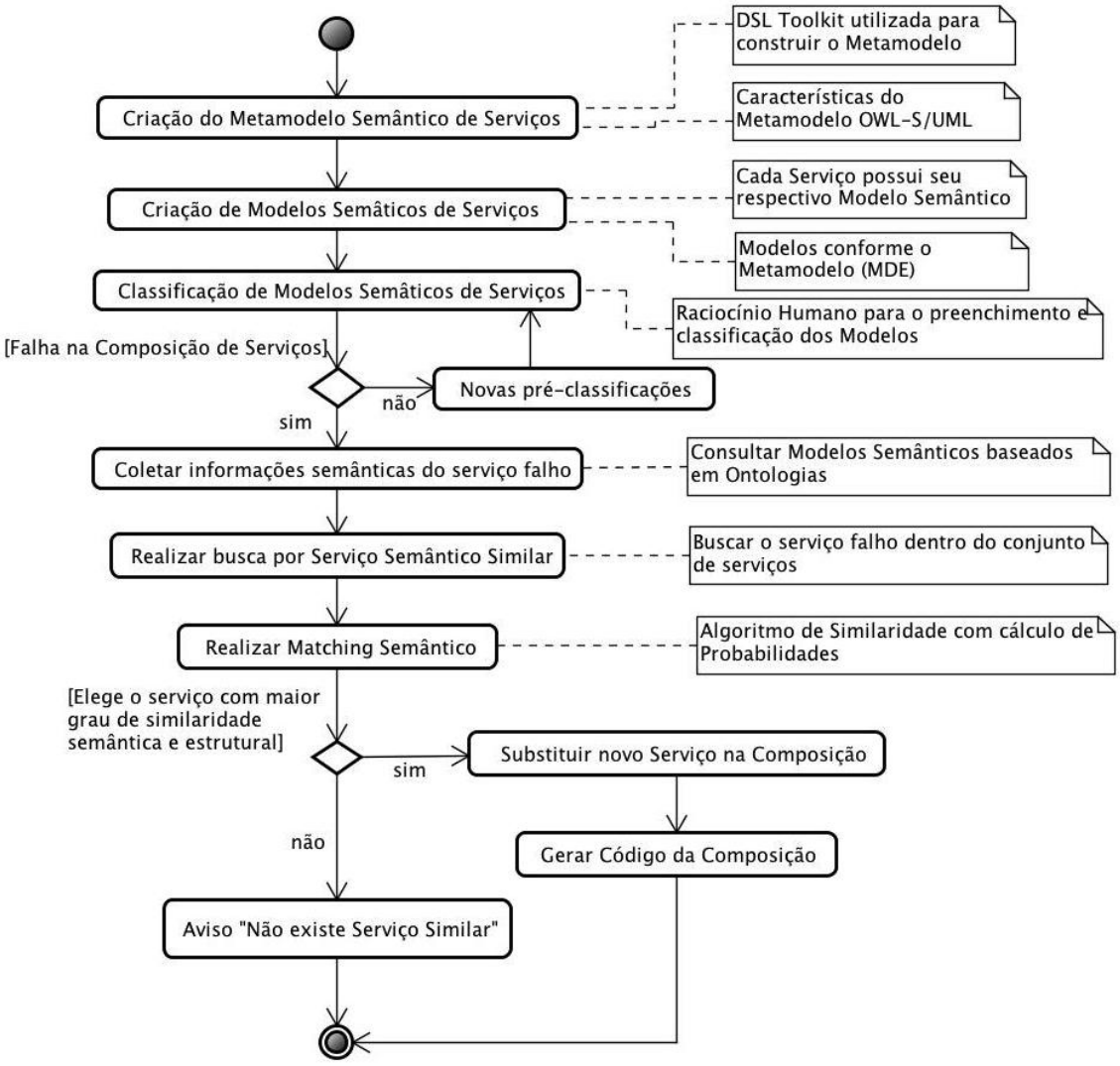


Figura 20 – Metodologia do Framework

Como ilustra a Figura 20, o primeiro passo da metodologia é criar o metamodelo semântico de serviços. A partir do metamodelo, tem-se uma ferramenta que foi desenvolvida com o objetivo de criar modelos com seus respectivos elementos dentro do domínio semântico de serviços. Tais modelos são preenchidos pelo desenvolvedor do serviço, seguindo as classificações estabelecidas pelo metamodelo. Essa classificação é significativa, pois exige raciocínio humano para enquadrar o serviço de modo coerente com o seu objetivo, tornando dessa maneira o *Matching* de informações de serviços classificadas por humanos mais preciso.

Caso um dos serviços que fazem parte da composição falte, suas informações semânticas serão utilizadas para realizar a busca por um serviço semântico similar. Caso não haja falta na composição, novos serviços poderão ser classificados.

Como ilustra a Figura 20, para realizar a busca por um serviço, será preciso coletar as informações semânticas do serviço que faltou, em seguida, comparar essas informações com as informações dos modelos existentes no repositório. Tal comparação é conseguida através do *Matching* de modelos, que utiliza um algoritmo de *matching* baseado em similaridade com cálculo de probabilidades. O algoritmo de *Matching* calcula as probabilidades para cada comparação entre modelos, em seguida elege o serviço com maior grau de similaridade semântica e estrutural. Esse algoritmo será detalhado na seção 4.4.

Quando eleito, o serviço com maior grau de similaridade substituirá na composição o serviço que faltou. Finalmente, o código da composição será gerado incluindo o novo serviço no lugar do serviço que faltou.

4.3. Metamodelos Desenvolvidos

Esta seção trata sobre o desenvolvimento do Metamodelo Semântico de Serviços e o Metamodelo de Mapeamento propostos neste trabalho.

4.3.1. Metamodelo Semântico de Serviços

Para fazer a composição dinâmica de serviços web é preciso ter informações ou metadados sobre os serviços envolvidos. Essas informações podem ser conseguidas utilizando ontologias ou semântica web. Uma ontologia em ciência da computação é um modelo de dados que representa um conjunto de conceitos dentro de um domínio e os relacionamentos entre estes [13]. Uma ontologia também pode ser vista como um vocabulário que descreve um determinado domínio e este vocabulário é computacionalmente manipulável [57]. Dentre as ontologias, destaca-se a OWL-S que é uma ontologia que permite uma fácil migração de descrições sintáticas e semânticas de serviços Web. Uma visão geral das três sub-ontologias OWL-S foi apresentado na Figura 3 do Capítulo 2. Dentre as três sub-ontologias, a denominada *Profile* tem o objetivo de especificação e descrição do que é realizado por um serviço, ou seja, traça o perfil de um serviço.

Figura 21 – Estrutura Geral da OWL-S Profile.
Fonte[24]

A sub-ontologia *Profile*, em uma visão mais detalhada como ilustra a Figura 21, define e possui características descritivas através das propriedades não funcionais na parte superior da Figura 21, como segue [24]:

- ***Service Profile*:** usado para especificar o tipo de serviço, provê uma **descrição do serviço**, o que o serviço oferece, requisitos para o serviço trabalhar;
- ***Actor*:** possui todas as informações sobre o provedor do serviço;
- ***ServiceParameter*:** aponta para o valor dentro de alguma ontologia OWL;
- ***serviceParameterName*:** nome do parâmetro atual que pode ser um literal ou URI do parâmetro de um processo;
- ***ServiceCategory*:** refere-se a classificação do serviço, possui um nome da categoria ***categoryName***;
- ***code*:** para cada tipo de serviço armazena um código associado para uma taxonomia;
- ***taxonomy*:** armazena uma referência para um esquema de taxonomia;
- ***qualityRating:*** garantia de qualidade providas pelo serviço.

A parte inferior da Figura 21 possui a descrição das funcionalidades de um serviço que descreve parâmetros de entrada, saída, pré-condições e efeitos.

Pesquisas sobre Ontologias serviram para definir aspectos semânticos do metamodelo de serviços. Especificamente, OWL-S *Profile* como ilustra a Figura 21, é a sub-ontologia que foi tomada como base para agregar conceitos e propriedades ao domínio de serviços web semânticos.

A base para criação do metamodelo semântico de serviços foi a concepção de uma DSL (*Domain-Specific Language*) que é uma linguagem projetada para ser utilizada em um conjunto específico de tarefas [30]. O termo “domínio específico” é determinado pelo projetista da linguagem que define os conceitos que vão compor a DSL.

Uma forma de organizar a construção de uma DSL é especificar seus conceitos, propriedades, tipos de propriedade e um exemplo do mundo real. No domínio de semântica de serviços web, cada propriedade de um conceito deverá ser preenchida, como ilustra a Tabela 6. Finalmente, o suporte a linguagem é conseguido com a utilização da DSL projetada.

Conceito	Nome da propriedade	Tipo da propriedade	Exemplo
ServiceProfile	nameService	String	TravelAgency
	textDescription	String	Reserva de pacotes de viagens
	serviceProduct	Enumeration	Viagens
	serviceClassification	Enumeration	Reserva
SActor	name	String	FlightRight
	webURL	String	www.flightright.com.br
	email	String	flightright@flightright.com
SCategory	categoryName	Enumeration	Turismo
	value	Enumeration	Muito utilizada
SQualitiyRating	ratingName	Enumeration	Confiabilidade
	valueRating	Enumeration	Alta
Sprocess	processName	String	reservarVoo
Sparameter	parameterName	String	codigoVoo
	locationParameter	Enumeration	entrada
	atributeType	Enumeration	String
SResult	smResult	String	SaidaString
SPrecondition	expression	String	Entrada1String

Tabela 6 – Tabela de Conceitos e Propriedades de uma DSL.

A coluna **"Exemplo"** na Tabela 6 foi preenchida propositalmente com informações de serviços web relacionados com o domínio de pacotes de viagens, reserva de vôos, agência de viagens, etc., ou seja, o domínio para onde o estudo de caso proposto neste trabalho está direcionado.

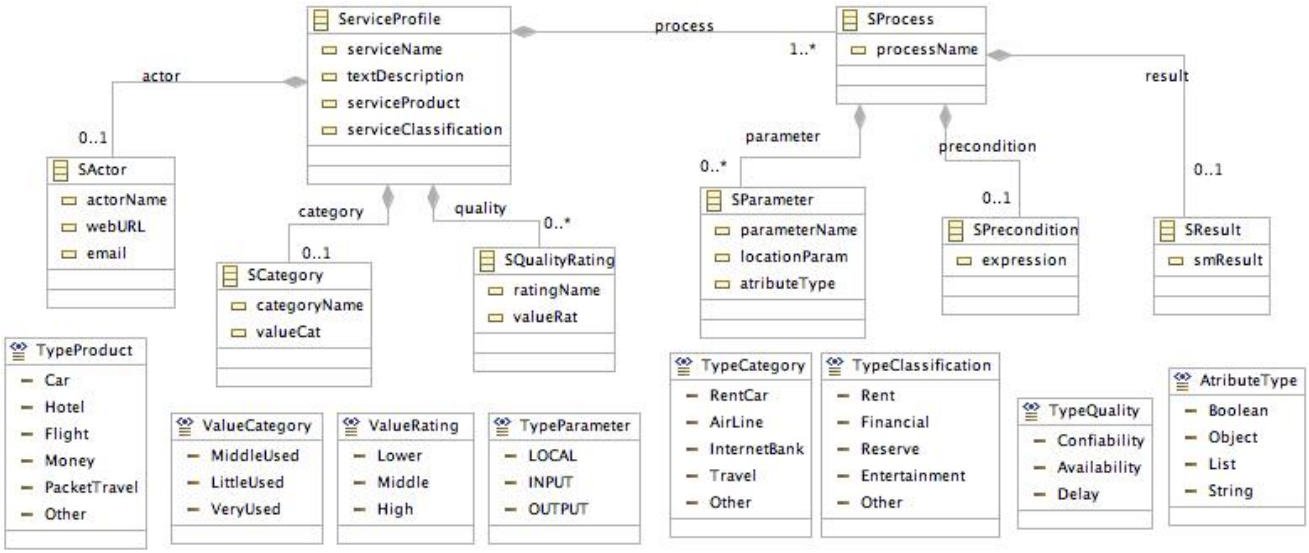


Figura 22 – Metamodelo Semântico de Serviços.

Na DSL, o **Metamodelo Semântico de Serviços** proposto, como ilustra a Figura 22, os aspectos semânticos são capturados a partir dos conceitos e propriedades estabelecidos na Tabela 6. Dentre as características semânticas, estão o perfil do serviço (*ServiceProfile*) que possui, dentre outros, os atributos produto (*serviceProduct*) e classificação (*serviceClassification*). *ServiceProfile* é composto pelas classes categoria (*SCategory*), qualidade (*SQualityRating*) e o provedor do serviço (*SActor*), ou seja, *ServiceProfile* define o que é realizado por um serviço e propriedades não funcionais.

Sobre aspectos estruturais, como ilustra o metamodelo da Figura 22, um perfil caracteriza um ou mais processos de um serviço. Um processo é definido por seus parâmetros (*SParameter*) de entrada, saída e possivelmente parâmetros locais (*locationParam*), e ainda condições (*SPrecondition*) para que o serviço seja realizado e o resultado (*SResult*) da execução.

Observa-se que o metamodelo proposto pode ser estendido para qualquer domínio de serviços web. Por exemplo, a classe *ServiceProfile* tem propriedades como nome do serviço, descrição do serviço e dois importantes parâmetros, o produto que o serviço está relacionado e sua classificação. No caso do atributo *serviceProduct*, um *Enumeration* denominado *TypeProduct* que contém a lista de produtos do domínio de viagens (*Hotel, Flight, Money, PacketTravel, etc.*) foi criado, mas nada proíbe estender esta lista inserindo produtos de vários outros domínios de serviços web.

Similarmente, pode-se estender a lista com os tipos de classificação dos serviços (*TypeClassification*), definir novos tipos de qualidade de serviços (*TypeQuality*) e seus respectivos valores (*ValueRating*), criar novas categorias (*TypeCategory*) e muito mais.

4.3.2. Metamodelo de Mapeamento entre Modelos Semânticos de Serviços Web

Em pesquisas realizadas sobre os trabalhos de D. Lopes e sua equipe de pesquisa [6][7][16], apresenta-se um algoritmo para realizar o

Matching de metamodelos, permitindo gerar especificações de correspondências entre elementos de metamodelos. Para tal tarefa, um Metamodelo de *Matching* tem a característica de especificações de correspondências de qualidade, para enfim, gerar definições de transformações a fim de se obter um código-fonte final.

Outra característica desse metamodelo é evitar a injeção de erros no código-fonte durante a criação de especificações de correspondências manuais. A Figura 23 apresenta o Metamodelo de *Matching* de D. Lopes, onde cada elemento demonstrado tem um papel bem definido na modelagem das correspondências entre os metamodelos. A descrição precisa destes elementos se encontra em [16].

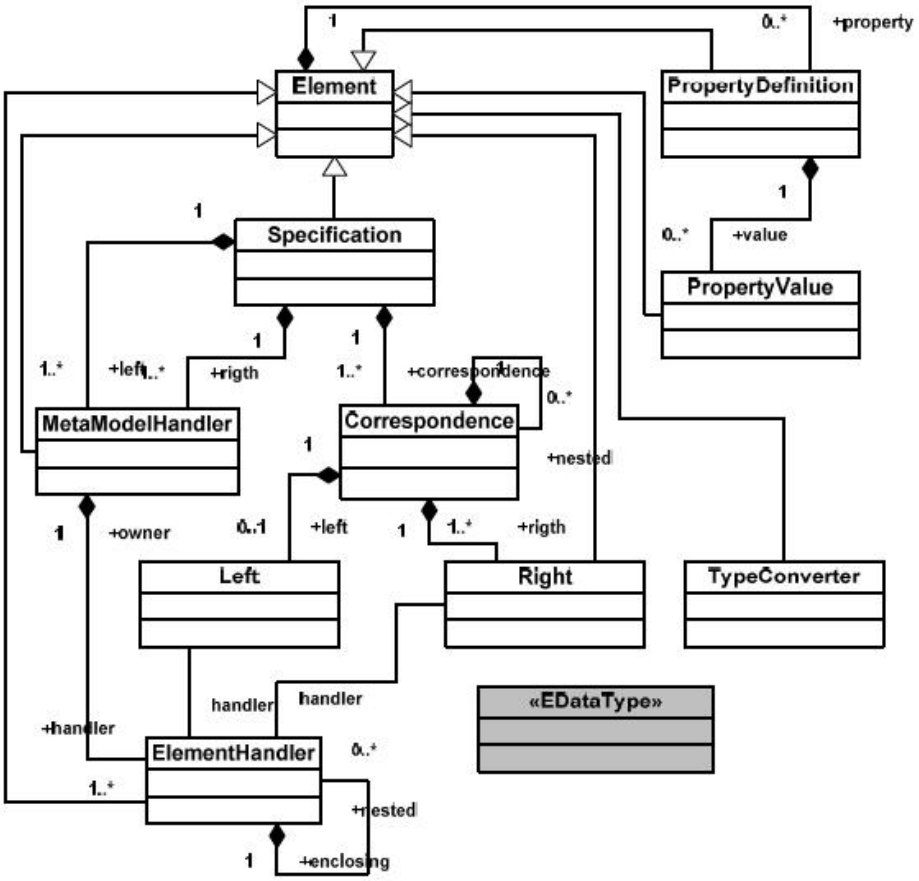


Figura 23 – Metamodelo de Matching.
Fonte: D. Lopes, 2006 [16]

O metamodelo de *Matching* [16] utiliza as similaridades para medir a distância semântica e estrutural entre dois metamodelos. Seguindo a idéia e o objetivo conseguido pelo metamodelo proposto por D. Lopes, a abordagem de

correspondência nesta dissertação propõe um Metamodelo de *Mapeamento* direcionado para o domínio de semântica de serviços, denominado *Semantic Match* abreviado para ***sMatch*** como ilustra a Figura 24.

O **Metamodelo de Mapeamento** (sMatch) proposto neste trabalho é a base para estabelecer as correspondências entre os elementos dos modelos de serviços.

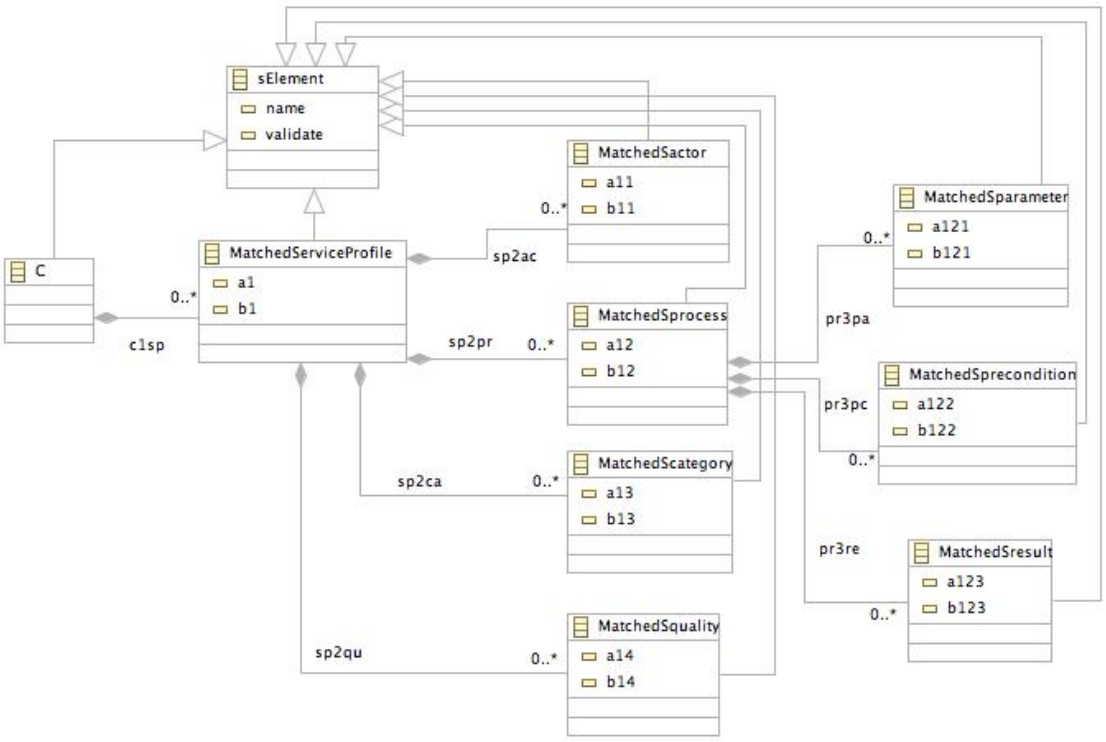


Figura 24 – Metamodelo de Mapeamento (sMatch).

Vale observar que o Metamodelo de *Matching* proposto por D. Lopes [16] e o *sMatch* proposto nesta dissertação trabalham em níveis de abstração diferentes. O primeiro está em um nível mais alto de abstração (nível de metamodelos), pois relaciona e faz correspondência entre **elementos de metamodelos**. O segundo, metamodelo *sMatch* está em um nível de abstração mais baixo (nível de modelos), pois o mesmo foi projetado para relacionar e fazer correspondências entre **elementos de modelos**.

Analisando a Figura 24 (***sMatch***), a classe *sElement* é uma generalização para todos os elementos do modelo, cada elemento herda um atributo nome (*name*) para efetuar a comparação e um *boolean validate* para confirmar as comparações positivas.

Observando a Figura 24 da esquerda para a direita, a classe C (Correspondência), como um primeiro nível, ou nó raiz, define as relações entre dois elementos de modelos, neste caso o *MatchedServiceProfile ou Profile*. Os elementos (perfis) são separados pelos atributos "*a*" referente ao lado esquerdo, elemento fonte ou procurado, relacionado com o perfil do lado "*b*", lado direito ou elemento alvo.

Seguindo o raciocínio, depois que todos os atributos dos perfis de serviços "a" e "b" são comparados, abaixo deste nível estão compostos os elementos *Actor, Quality, Category e Process*, todos também comparados e validados. Em seguida, no último nível, estão compostos a cada processo os elementos *Parameter, Condition e Result,* todos comparados e validados.

4.4. Algoritmo de *Matching* de Modelos

O algoritmo proposto para realizar o *Matching* é baseado na Teoria da Probabilidade, que é um ramo importante da Matemática pura, com campo de aplicação que se estende praticamente sobre todos os ramos da ciência natural, técnica e social. Suas raízes se encontram numa teoria matemática elementar, a teoria dos jogos de azar, estabelecida há três séculos [64]. Nesta época, à medida que novos jogos surgiam e cada vez mais complexos, como cartas, dados, etc., sentiu-se a necessidade de métodos racionais para calcular os riscos dos jogadores em vários jogos. De Mére (1607-1684), jogador profissional, teve a idéia de consultar o famoso matemático e filósofo Blaise Pascal (1654), em Paris, sobre algumas questões relacionadas com certos jogos de azar, iniciando uma correspondência entre Pascal e alguns de seus amigos matemáticos, sobretudo Pierre Fermat, em Toulouse [64].

Observações iniciais sobre a natureza dos jogos chegaram finalmente a uma conclusão e os matemáticos formularam o quociente **p = n/N**, onde **n** era o caso de números favoráveis e o **N** os casos possíveis nos vários jogos reais. Esta sequência de ideias conduziu a famosa definição clássica de probabilidade que diz assim: "*A probabilidade da ocorrência de um determinado evento é igual ao quociente de um número de casos possíveis, desde que todos esses casos sejam mutuamente simétricos*" [64].

De acordo com essa definição, como exemplo, podemos dizer que a probabilidade de obter uma face do dado em um lançamento é de 1/6 ou aproximadamente 0,1666 (16,66% de chance para cada face).

Definição axiomática (Kolmogorov (1933))[65]**:**

Uma probabilidade é uma função $P(\cdot)$ a valores reais definida em uma classe f de eventos de um espaço amostral Ω, que satisfaz as seguintes condições:

- (A1) $0 \leq P(A) \leq 1$ para todo $A \in f$,
- (A2) $P(\Omega) = 1$,
- (A3) Aditividade enumerável: para qualquer sequência $A_1, A_2, \ldots \in f$ de eventos dois a dois disjuntos [65],

$$P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i).$$

A tripla (Ω, f, P) é chamada um *espaço de probabilidade.*

Escrevendo o espaço amostral Ω como Ω = $\{\omega_1, \omega_2, \ldots\}$, para cada evento ω de i = 1, 2,..., ω_i então $p(\omega_i)$ é a probabilidade de um evento simples $\{\omega_i\}$ acontecer. A probabilidade de um evento $A \subset \Omega$ é definida por [65],

$$P(A) = \sum_{i:\, \omega_i \in A} p(\omega_i).$$

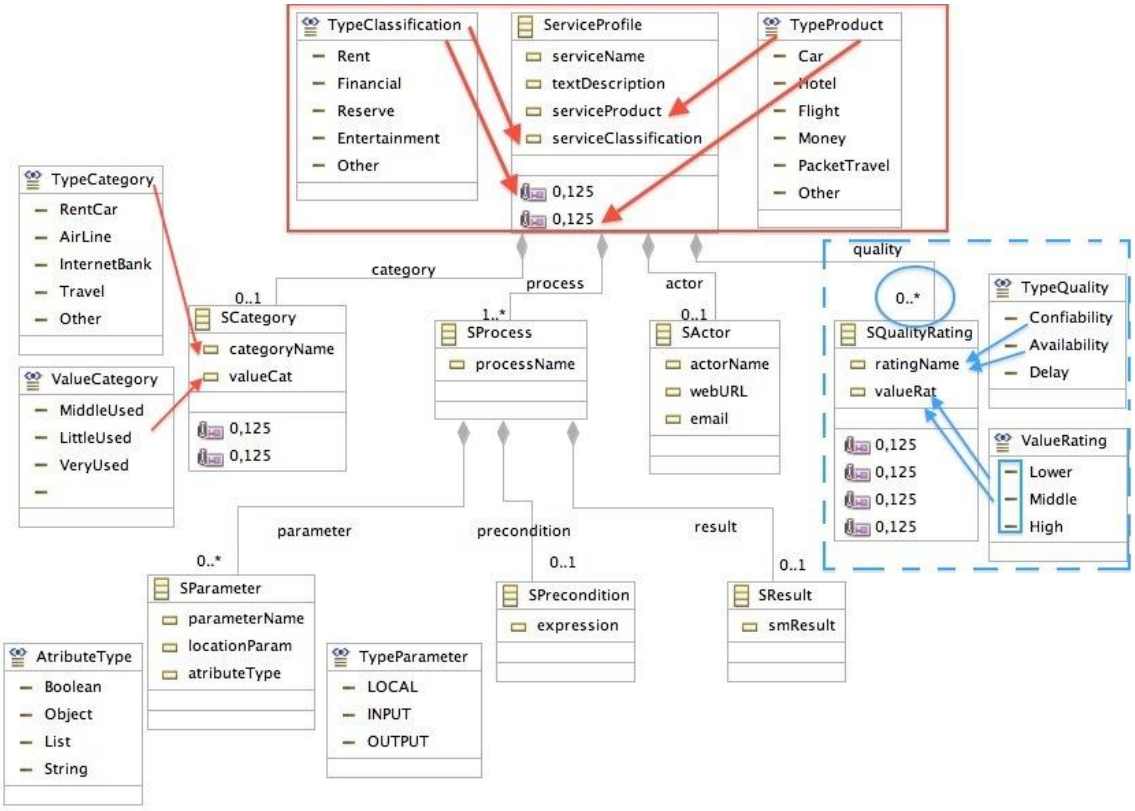


Figura 25 – Probabilidades no Modelo Semântico.

Nos modelos semânticos de serviços, dois tipos de parâmetros serão encontrados entre as classes que são *String* e *Enumeration* (*Enumeration* aqui denominados de *Types*). Por exemplo, como ilustra a Figura 26, para o parâmetro *serviceProduct* da classe *ServiceProfile*, criou-se um *Enumeration* (*TypeProduct*) referente ao tipo de produto que o serviço está relacionado. *TypeProduct* contém uma lista de produtos que um serviço poderá tratar. Similarmente, no quadro vermelho da Figura 25, tem-se o tipo de classificação dos serviços (*TypeClassification*). Os parâmetros *"Type"* nos fornece um resultado exato, pois se um produto de um serviço **F** (Serviço que **Faltou**) é um Hotel deve-se procurar somente por serviços que tenham o mesmo tipo de produto Hotel, ou seja, os serviços que trabalham com outro produtos como, Carro, Dinheiro, Vôo, Pacote de Viagens, etc., serão automaticamente descartados.

O Fluxo do Algoritmo de *Matching,* como ilustra a Figura 26, inicialmente, destaca-se a importância da pré-classificação dos serviços feita pelo desenvolvedor do serviço e também a escolha dos parâmetros *"Types"* que possibilita uma resposta precisa na comparação. Nesse sentido, o

algoritmo de *Matching* faz uma busca nos modelos realizando uma contagem de *Types* existentes.

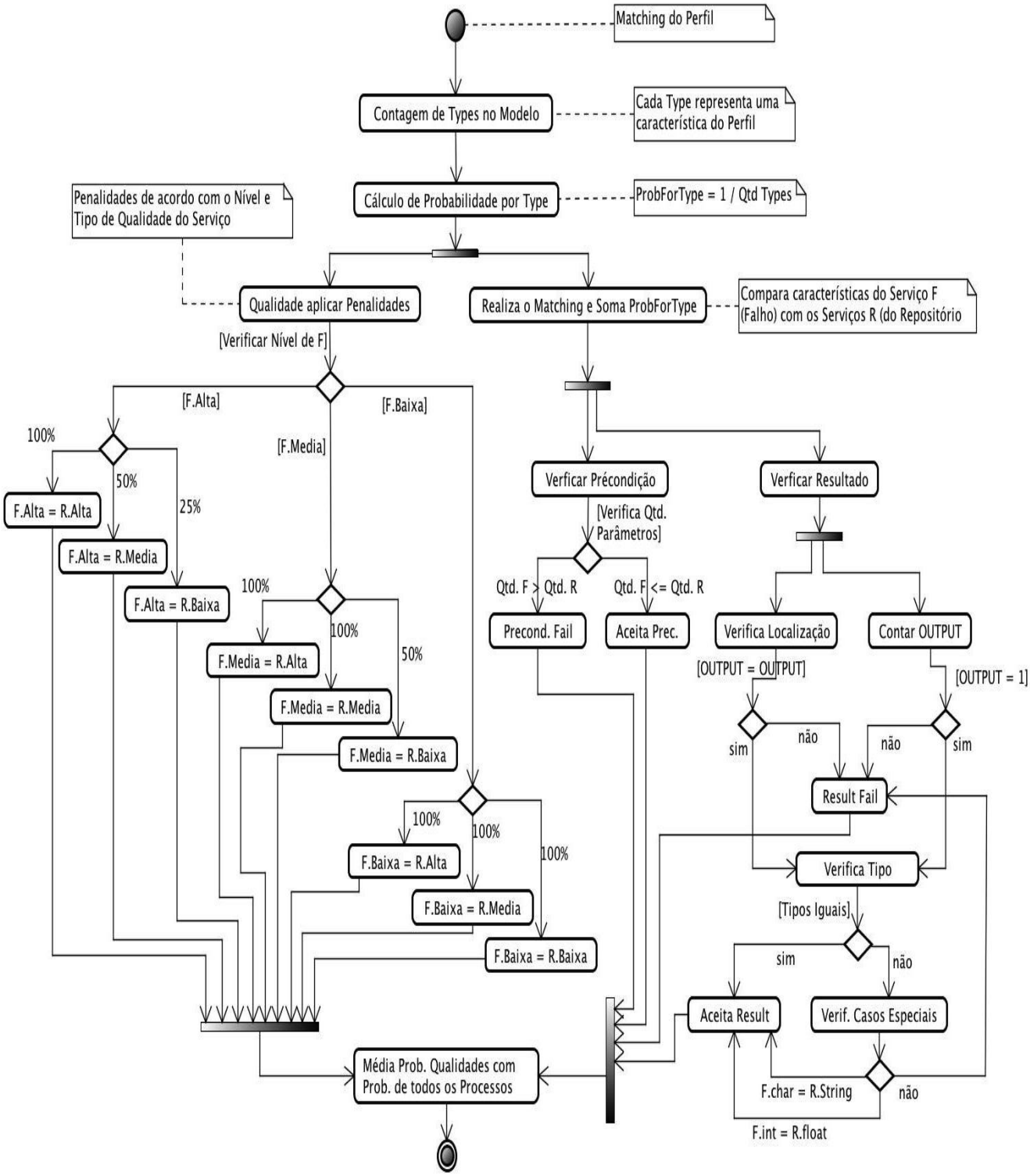


Figura 26 – Diagrama de Atividades do Algoritmo de Matching.

Como ilustra a Figura 25 no círculo azul dentro do quadro com linhas pontilhadas, a classe *ServiceProfile* pode não ter qualidade (*SQualityRating*) ou

pode ter várias qualidades (0..*). Essa quantidade de *Types* é variável devido as relações de cardinalidade entre as classes (0..* ou 1..*), o que implica em um cálculo de probabilidades locais que será tratado a seguir. No exemplo da Figura 25, conta-se 8 *Types* para classificar o perfil de um serviço **F**. São eles: *TypeProduct, TypeClassification, TypeCategory, ValueCategory, TypeQuality, ValueRating, TypeParameter e AtributeType.* Com o total de 8 *Types*, na comparação entre o modelo **F** (que **Faltou**) com o modelo **R** (modelo do **Repositório**), a **probabilidade local** para cada *Type* (ou característica) encontrada entre os modelos F e R é de 1/8 (0,125 ou 12,50%). A medida que o algoritmo de *Matching* navega nos modelos e realiza as comparações, para cada característica casada dos dois lados, ele soma 12,5%. Ou seja, uma característica encontrada, os modelos são 12,5% iguais, duas características encontradas e eles são 25% iguais, caso entre F e R o algoritmo encontre 6 características iguais dentre as 8 existentes, então os modelos F e R tem os perfis 75% semelhantes ou similares.

O algoritmo de *Matching* além de comparar parâmetros *Types* e somar **probabilidades locais**, também utiliza um mecanismo de penalidades. Caso os modelos empatem em várias características do perfil, o fator **Qualidade** do serviço (classe *SQualityRating*) poderá ser vista como decisiva para estabelecer o melhor grau de similaridade.

4.4.1. Qualidade de Serviço

Primeiramente, como um serviço pode ter n (muitas) qualidades (exemplos: *Confiability, Availability, Delay*), quanto mais qualidades melhor será a efetividade do cálculo de *Matching.* Considerando essas variações de cada **tipo** (confiabilidade, disponibilidade, atraso,) e seus respectivos **níveis** (exemplos: *Low, Middle, High*), para cada par de propriedades (**tipo**, **nível**) de qualidade, penalidades de 25%, 50% e 100% serão dadas, dependendo dos pares de propriedades encontrados nas comparações. Dessa maneira, a qualidade do serviço pode sofrer penalidades, isso de acordo com a comparação da qualidade do serviço F (que Faltou) com a qualidade do serviço R (do Repositório).

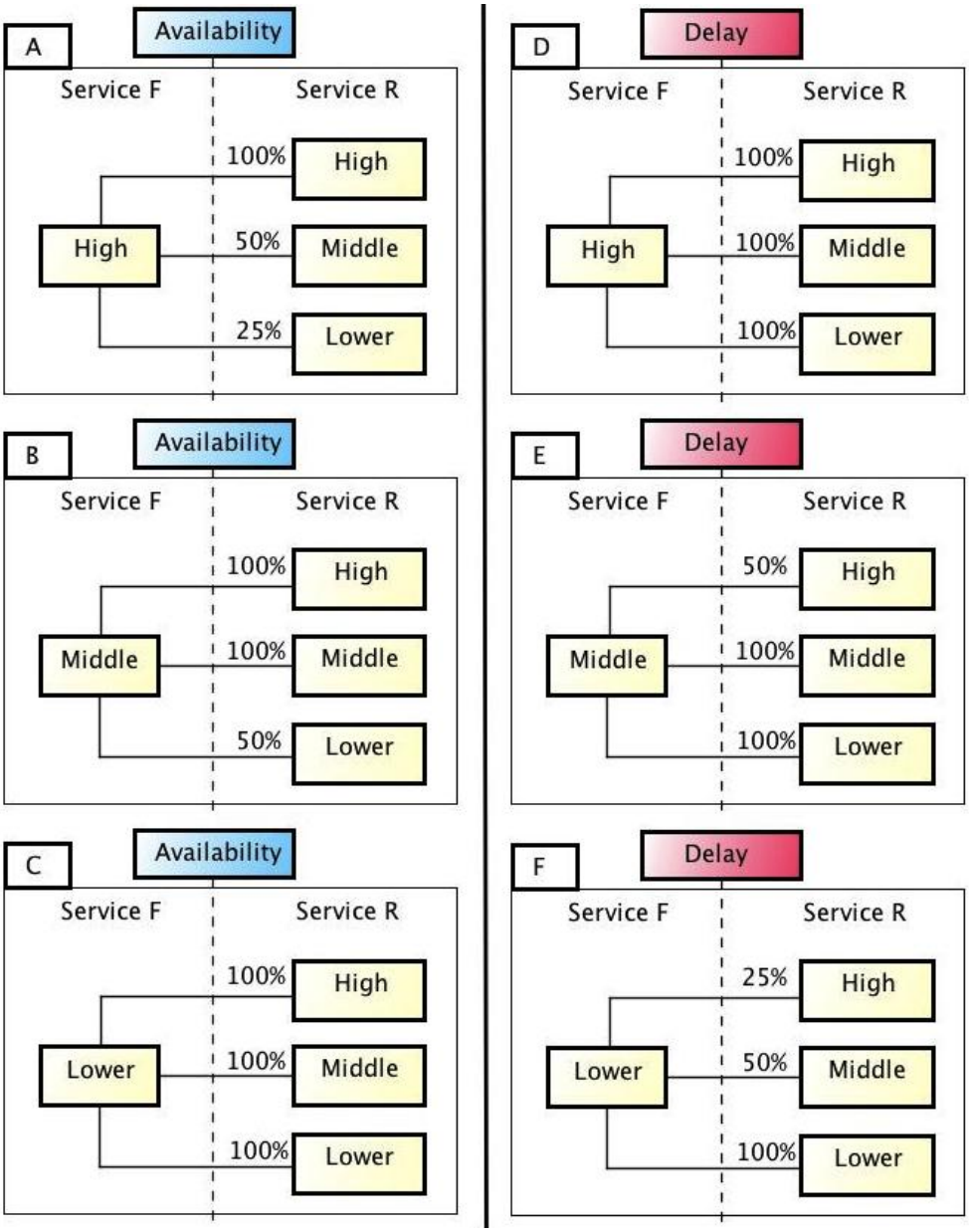


Figura 27 – Tratamento de tipos de Qualidades no Algoritmo de *Matching*.

Como ilustra a Figura 27, diferentes cenários (um total de nove) para Disponibilidade (*Availability*) são representados nos 3 quadros (**A**, **B**, **C**) no lado esquerdo da figura. Os três quadros do lado direito (**D**, **E**, **F**) na Figura 27, representam diferentes cenários para o Atraso (*Delay*) do serviço.

No quadro **A**, ao lado esquerdo superior, analisando somente a Disponibilidade (*Availability*) de um Serviço F (que Faltou), tem-se três cenários. Ou seja, se a disponibilidade do Serviço F é **Alta** (*High*) e a disponibilidade do serviço R também é **Alta** (*High* – *High*: primeiro cenário), então soma-se **100%** do valor da **probabilidade local** calculada anteriormente

para cada *Type* de *SQualityRating* (Figura 25, quadro azul), ou seja, 100% de 12,5%.

Se a disponibilidade do Serviço F era **Alta** e a disponibilidade do Serviço R é **Média** (*High – Middle*: segundo cenário) então o valor somado ao total que deveria ser de 12,5% sofrerá uma penalização de **50%**, logo só será somado a metade (50%) de 12,5% que será o valor de (0,125 * 0,5) ou 0,625 (6,25%) somados.

Para o último caso do quadro **A**, se Disponibilidade do Serviço F (que Faltou) é **Alta** e o Serviço R (Serviço procurado no Repositório) é **Baixa** (*High – Lower*: terceiro cenário), então será somado ao valor total de similaridade somente **25%** do merecido, ou seja, 25% de 12,5% (0,25 * 0,125) que é 0,03125 (ou 3,125%).

Como ilustra a Figura 27, à medida que o nível da disponibilidade do Serviço F vai diminuindo de *High* (quadro **A**) para *Middle* (quadro **B**) e para *Lower* (quadro **C**), as porcentagens das penalidades vão se alterando e a penalidade diminuindo. Ao ponto que, se uma disponibilidade de um Serviço que faltou F era é *Lower* (quadro **C**), qualquer nível de disponibilidade encontrada “do outro lado” dentre os serviços R (do Repositório) vai satisfazer 100% da qualidade necessária.

Analisando os tipos de qualidade sobre o algoritmo de *Matching*, na Disponibilidade (*Availability*) o mesmo comportamento pode ser atribuído para Confiabilidade (*Confiability*). Ou seja, quanto maior for a **disponibilidade** do serviço encontrado melhor será o nível do *Matching*, quanto maior for a **confiabilidade** do serviço melhor também será o nível do *Matching*. Mas para o caso do Atraso (*Delay*) do serviço, o cenário e os valores são totalmente invertidos (quadros **D**, **E**, **F**), pois quanto maior é o **atraso** de um serviço (tempo de resposta alto, mais demorado), pior será o nível final do *Matching*, como ilustra a Figura 27 na coluna direita.

A Lógica Fuzzy

Fuzzy em inglês, significa incerto, duvidoso, nebuloso. Expressa os valores com que lida, permitindo representar graus de certeza, de associação

ou valores de pertinência, intermediários entre os valores extremos de verdadeiro e falso (1 ou 0) [66].

Analisando o algoritmo de *Matching* proposto neste trabalho, o comportamento das suas devidas condições e penalizações podem ser representados pela Lógica *Fuzzy*, através do mapeamento das variáveis **tipos** e **níveis** de qualidade, que não tem equivalência matemática definida.

A estrutura básica de representação que define o comportamento de uma ou mais variáveis *fuzzy* em relação às outras, ou ainda o conjunto de relações condicionais, são denominadas de regras. As regras podem ser providas por especialistas ou podem ser extraídas de dados numéricos e, normalmente, são condicionais do tipo **se... então** (*if... then*) [66].

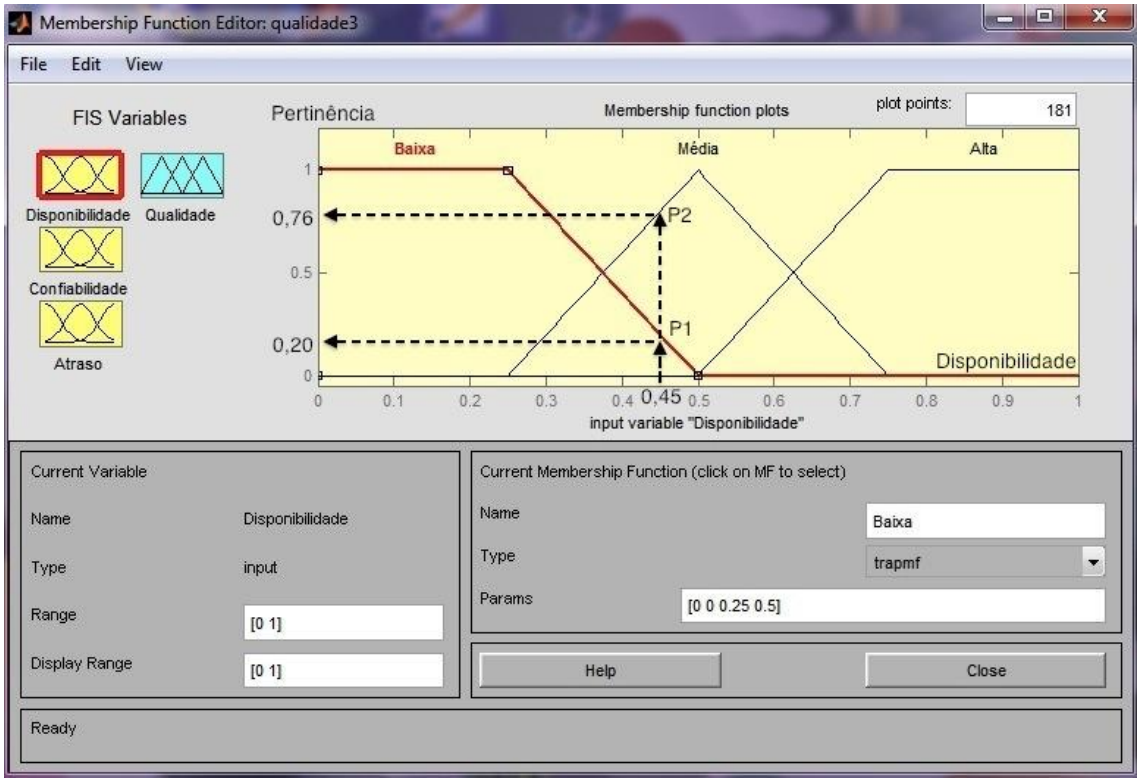


Figura 28 – Gráfico fuzzy para representar a qualidade do serviço.

Além das regras, para cada termo lingüístico um grau de pertinência é definido pelas funções de pertinência que fazem o papel das curvas de possibilidades na teoria clássica da lógica fuzzy. As funções de pertinência são normalizadas, ou seja, seu máximo é sempre “1” (100% de pertinência) e o seu mínimo é “0” (valor não pertence ao grupo). Segundo autor em [66], optar-se pelo uso de funções padrão apresenta vantagens em relação ao uso de outras

curvas. Em primeiro, elas são simples porém suficientes para o uso em lógica fuzzy. Além disso, são muito eficientes na maioria das plataformas, em termos computacionais e são de fácil interpretação [66].

Na Figura 28, pode-se visualizar uma representação *fuzzy* na ferramenta *MATLAB*[3] para a disponibilidade de um serviço. Por exemplo, a probabilidade de 45% (0,45) para a disponibilidade encontrada não é totalmente baixa nem totalmente média. Pode-se dizer, contudo, que 45% possui uma **Pertinência P1** de 0,20 **Baixa** e, ao mesmo tempo, **P2** de 0,76 **Média**. Assim, a disponibilidade em 45% está mais para Média do que para Baixa.

A Figura 29 ilustra os gráficos *fuzzy* para representar a disponibilidade do serviço nas duas situações restantes (Média e Alta).

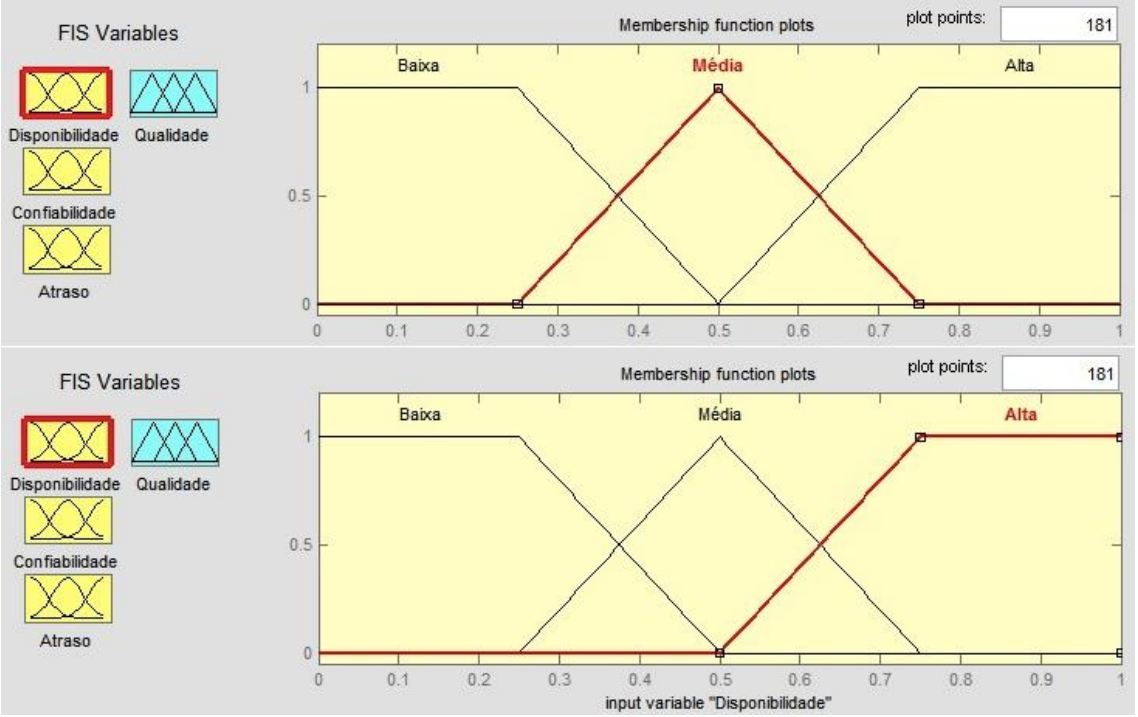


Figura 29 – Gráficos fuzzy para representar a disponibilidade Média e Alta do serviço.

A Figura 30 ilustra as relações entre os tipos de qualidades de serviços, que são as entradas para as regras definidas pelo especialista do

[3] MATLAB é um "software" interativo de alta performance voltado para o cálculo numérico. O MATLAB integra análise numérica, cálculo com matrizes, processamento de sinais e construção de gráficos em ambiente fácil de usar onde problemas e soluções são expressos somente como eles são escritos matematicamente, ao contrário da programação tradicional [69].

domínio. Tais regras, através de inferências, resultam em uma saída com o nível de qualidade estabelecido.

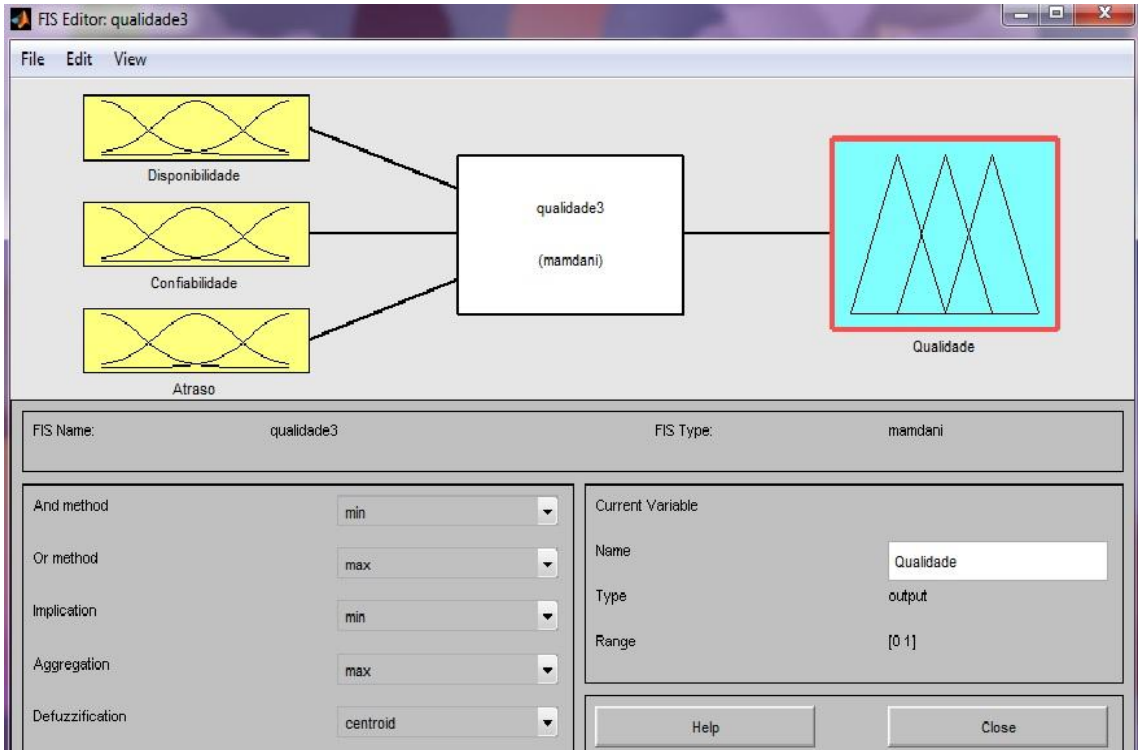


Figura 30 – Gráficos fuzzy para representar as relações entre entradas, regras e saída.

Dessa maneira, todas as condições para **disponibilidade**, **atraso**, inclusive a **confiabilidade**, ilustradas na Figura 27, foram mapeadas para Lógica *Fuzzy* através de vinte e sete Regras como ilustra a Figura 31.

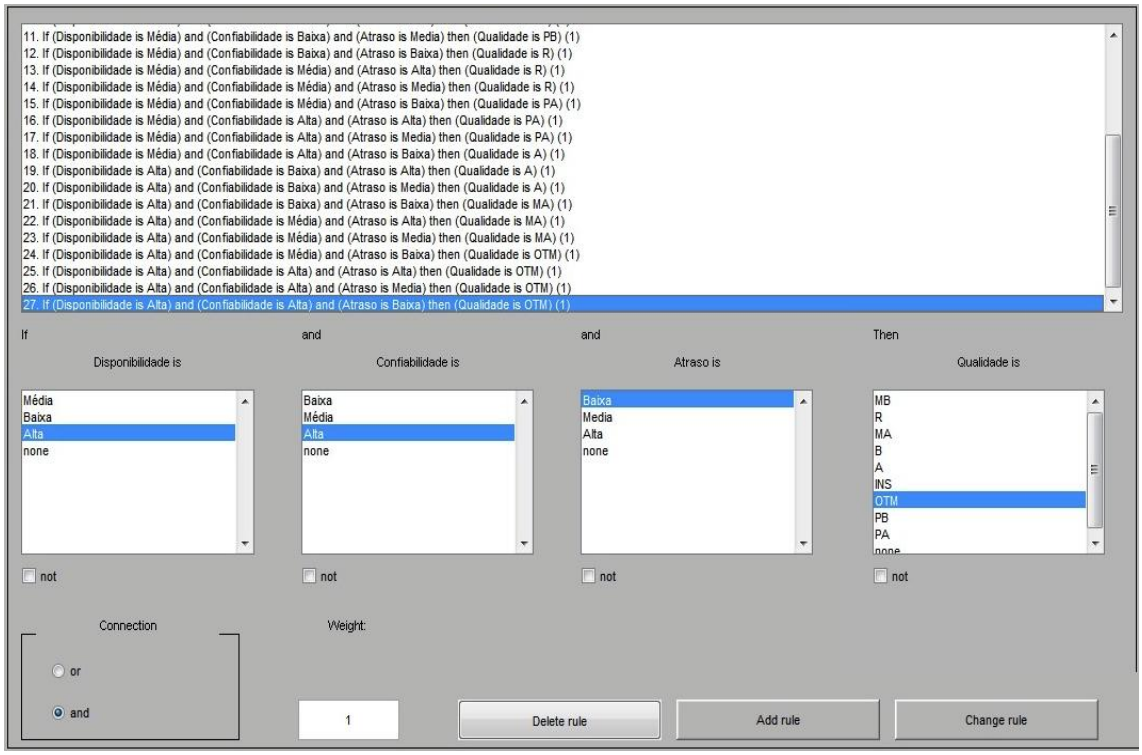


Figura 31 – Regras de Inferência Fuzzy.

Como ilustra a Figura 32, depois de definidas todas as regras de inferência, os possíveis resultados de saída se classificam em:

- INS: Insuficiente;
- MB: Muito Baixo;
- B: Baixo;
- PB: Pouco Baixo;
- R: Regular;
- PA: Pouco Alto;
- A: Alto;
- MA: Muito Alto;
- OTM: Ótimo.

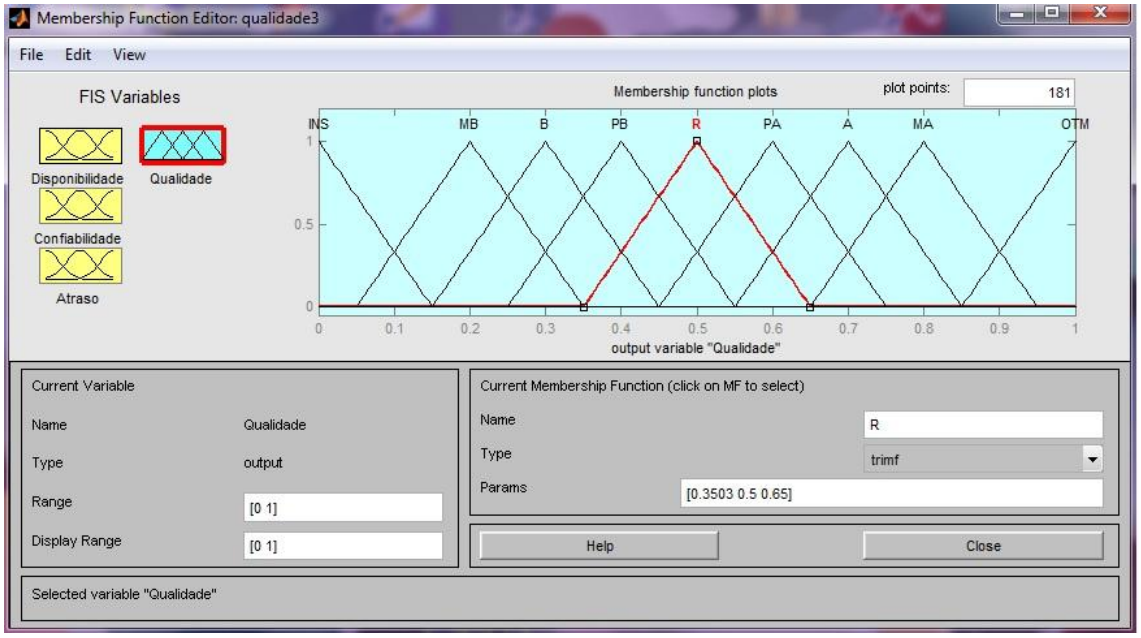


Figura 32 – Gráfico com Funções de Saída para Qualidade.

O objetivo do mapeamento da lógica do algoritmo de *Matching* para a lógica *fuzzy* é apresentar, nos capítulos de implementação, estudo de caso e testes, o comportamento do algoritmo de *Matching* proposto para o tratamento de Qualidade de Serviços, através de simulações e gráficos em 3D.

4.4.2. Processos de Serviços

Processos de serviços possuem características estruturais, tais como, pré-condições, resultados, parâmetros, etc. Logo, o cálculo do nível de similaridade para processos é diferente em relação ao cálculo realizado para comparar aspectos semânticos dos modelos de serviços. Ou seja, não basta comparar se os valores *Types* são iguais, mas deve ser feita uma análise da estrutura do processo, quantidade de processos, tipos, localização, relações e quantidade de parâmetros existentes, condições de entrada e saída.

Primeiramente, um serviço deve ter obrigatoriamente um processo ou poderá ter n (muitos) processos, logo o primeiro passo do algoritmo de *Matching* é calcular a **probabilidade local** para cada processo. Algumas condições deverão ser satisfeitas, pois além de comparados todos os parâmetros por localização (*output, input, local*) e tipos (*String, int, float*, etc...)

de processos, uma **pré-condição** deve ser satisfeita, como ilustra a Figura 33 coluna esquerda.

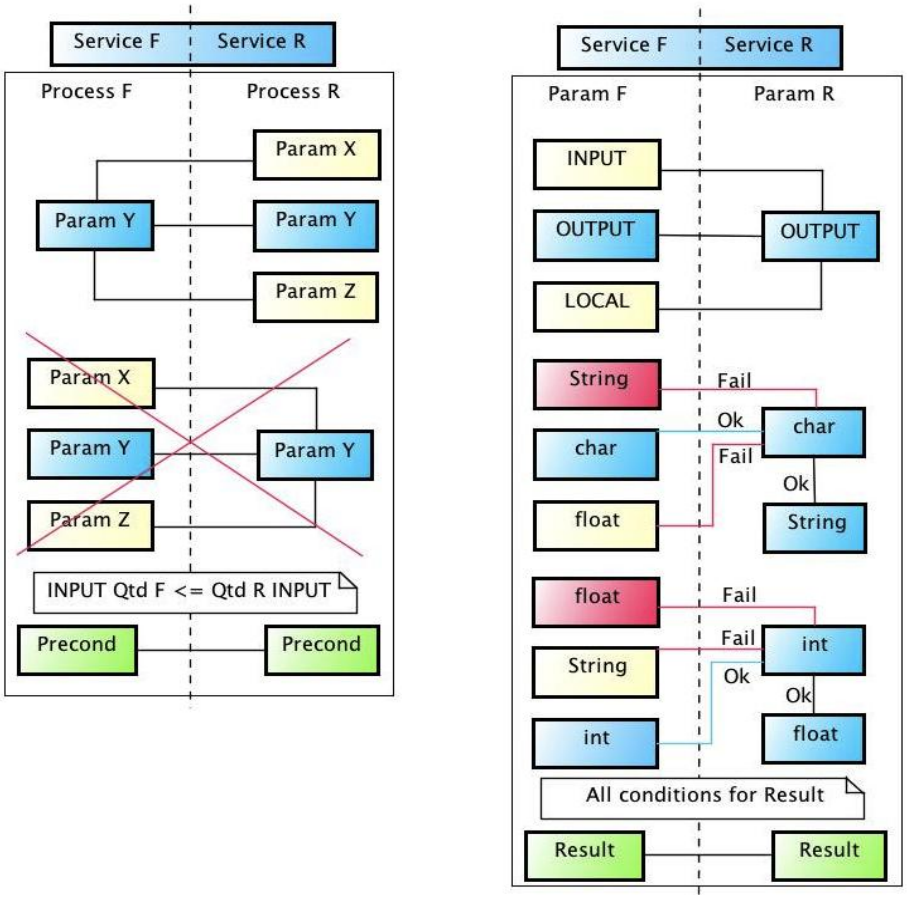


Figura 33 – Tratamento de Processos no Algoritmo de *Matching*.

Como ilustra a parte superior da Figura 33, se na assinatura de um processo F (que faltou) possuir mais parâmetros (*Param X, Y e Z*) do que um processo R (repositório) (*Param Y*) então este processo R deverá ser descartado, como ilustra o "X" (vermelho) na Figura 33. No cenário inverso, os parâmetros do processo F (em menor quantidade) poderão ser buscados dentro do conjunto de parâmetros do processo R (maior quantidade de parâmetros), lembrando que serão comparados para cada parâmetro, o tipo e a localização do mesmo.

Na coluna direita da Figura 33, tem-se condições de saída que devem ser satisfeitas, ou condições de resultado (*Result*). O primeiro teste para condição de saída, Figura 33, será que o processo R deverá ter somente um parâmetro com localização OUTPUT e o mesmo deverá ser comparado somente com outro OUTPUT no lado do processo F. A segunda condição para

o resultado do processo é que além da localização, serão comparados os tipos dos parâmetros iguais (ex: *String* ⇔ *String*). Nesta comparação, os parâmetros deveriam ser iguais, mas algumas particularidades para parâmetros diferentes podem ser aceitas, como ilustra a Figura 33. Por exemplo, se do lado A o retorno é do tipo *char*, do lado B poderá ser um *char* ou uma *String* na qual essa *String* pode ser representada por um *char*. Vai ocorrer um erro de comparação (desclassificação do serviço procurado) se do lado A o retorno for uma *String* e do lado B um *char,* pois não será possível representar uma *String* com vários caracteres, através de um *char*. O mesmo acontece entre *int* e *float*, pode-se representar um *int* através de um *float*, mas não será possível representar um *float* através de um *int*.

Finalmente, as probabilidades calculadas para cada processo são somadas ao valor de probabilidade total de todos os processos de um serviço.

4.5. Síntese

Este capítulo apresentou a proposta de um *framework* DWSC para suportar a Composição Dinâmica de Serviços Web. Uma abordagem foi apresentada, desenvolvida utilizando Ontologias e Semântica Web para representação de conceitos, com a finalidade de definir um perfil de um serviço web. No *framework* apresentado, os níveis de abstração dos elementos da abordagem foram separados e detalhados.

A abordagem foi dividida em duas etapas. A primeira etapa primeira referente a criação dos modelos semânticos de serviços conforme o metamodelo semântico proposto nesta dissertação e a segunda etapa referente ao *Matching* de modelos semânticos de serviços conforme o metamodelo de *matching* (*sMatch*) proposto. A modelagem do *Framework* proposto foi detalhada neste capítulo, bem como a proposta de uma metodologia a ser seguida para atingir o objetivo proposto neste trabalho. Os metamodelos desenvolvidos, que fazem parte da solução, foram detalhados.

Finalizando este capítulo, uma abordagem para o algoritmo de *Matching* de Modelos Semânticos de Serviços foi proposta baseado na Teoria da Probabilidade. Uma descrição e um diagrama de atividades foi projetado para ilustrar passo a passo o funcionamento do algoritmo através de fluxos detalhados.

5. Implementação do Protótipo do *Framework* DWSC

O *framework* DWSC foi implementado com o EMF (*Eclipse Modeling Framework*) que é um *framework* que facilita a geração de código para construção de aplicações Java baseadas em simples definições de modelos. O *framework* DWSC foi desenvolvido na forma de um *plug-in* para o IDE Eclipse, cujo objetivo geral é realizar a recuperação de uma composição de serviços web, dada uma falta em dos serviços compostos.

A Figura 34 ilustra um esquema geral da implementação das ferramentas para suportar a composição dinâmica, como tais ferramentas se relacionam e seus respectivos papéis.

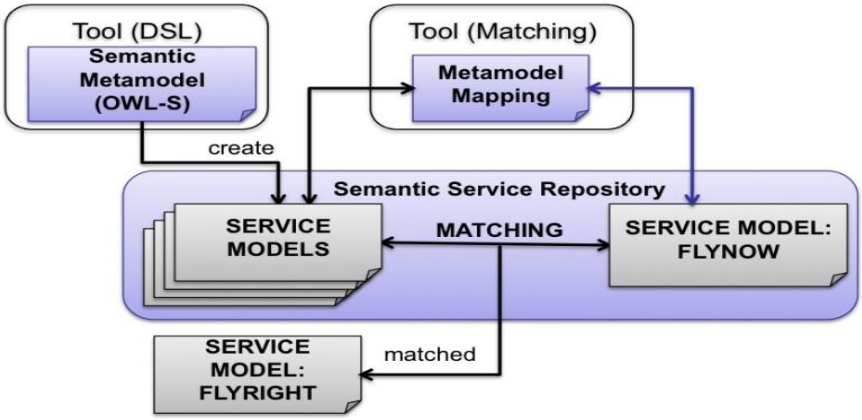


Figura 34 – Papéis das Ferramentas.

Como ilustra a Figura 35, a ferramenta *Tool (DSL)* foi criada inicialmente para suportar a criação de modelos semânticos de serviços (*Service Models*) e salvá-los no repositório (*Semantic Service Repository*).

Depois de criados, os modelos dos serviços com informações suficientes para realizar um *Matching* mais preciso, a ferramenta *Tool (Matching)* compara informações semânticas pré-classificadas e categorizadas pelos desenvolvedores de serviços web. Como ilustra a Figura 34, a ferramenta de *Matching* buscou o modelo do serviço que faltou (*SERVICE MODEL: FLYNOW*) e realizou comparações com todos os *Service Models*. Finalmente, foi escolhido o modelo do serviço (*SERVICE MODEL: FLYRIGHT*) que é mais similar ao modelo do serviço *FLYNOW.*

5.1. Ferramenta para criação de Modelos Semânticos de Serviços Web

O metamodelo semântico de serviço proposto no formato Ecore (linguagem de metamodelagem do EMF) é criado com a ferramenta Eclipse DSL Toolkit. A partir do metamodelo semântico **semanticx.ecore**, criou-se um modelo gerador (**semanticx.genmodel**) que é utilizado para o *Tooling* (geração de ferramenta). Ou seja, a partir do Metamodelo Semântico de Serviço são geradas classes que podem ser modificadas para criar a ferramenta ou *plug-in* correspondente, como ilustra a Figura 35.

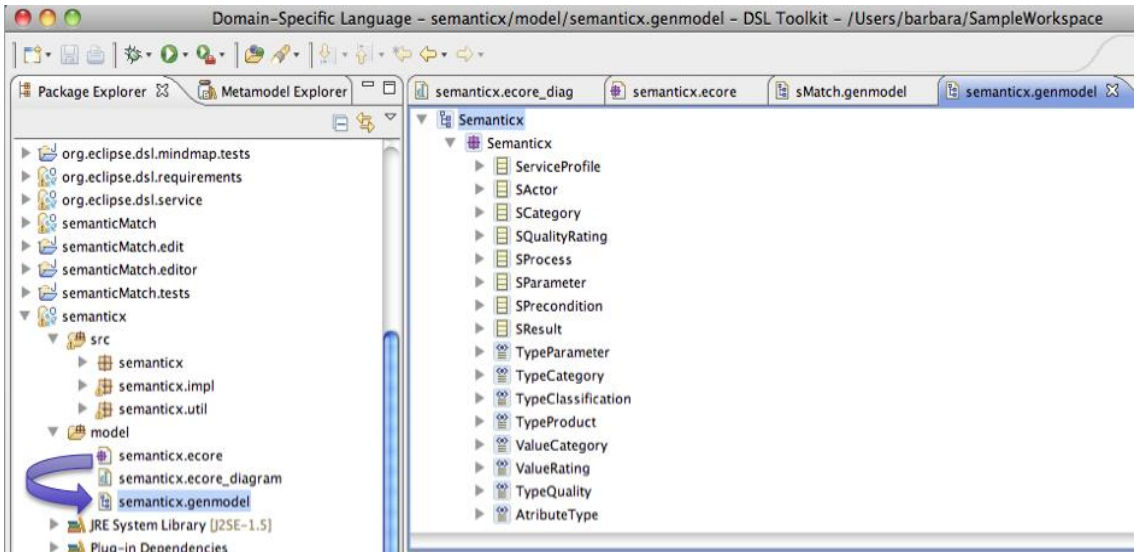


Figura 35 – Ecore para genmodel.

A ferramenta para criação de modelos de serviços, permite dar nome (*serviceName*), descrição (*textDescription*), criar características, parâmetros (*SParameter*), propriedades ao serviço. Além de classificar o serviço (*SClassification*), quanto ao produto (*serviceProduct*), categoria (*SCategory*), qualidade (*SQualityRating*), provedor (*SActor*), permite também definir, criar e descrever informações sobre processos (*SProcess*) envolvidos no serviço, como pré-condições (*SPrecondition*) e resultados (*SResult*) esperados para a execução, tipos (*AtributeType*) e localização (*TypeParameter*) de parâmetros, como ilustra a Figura 35.

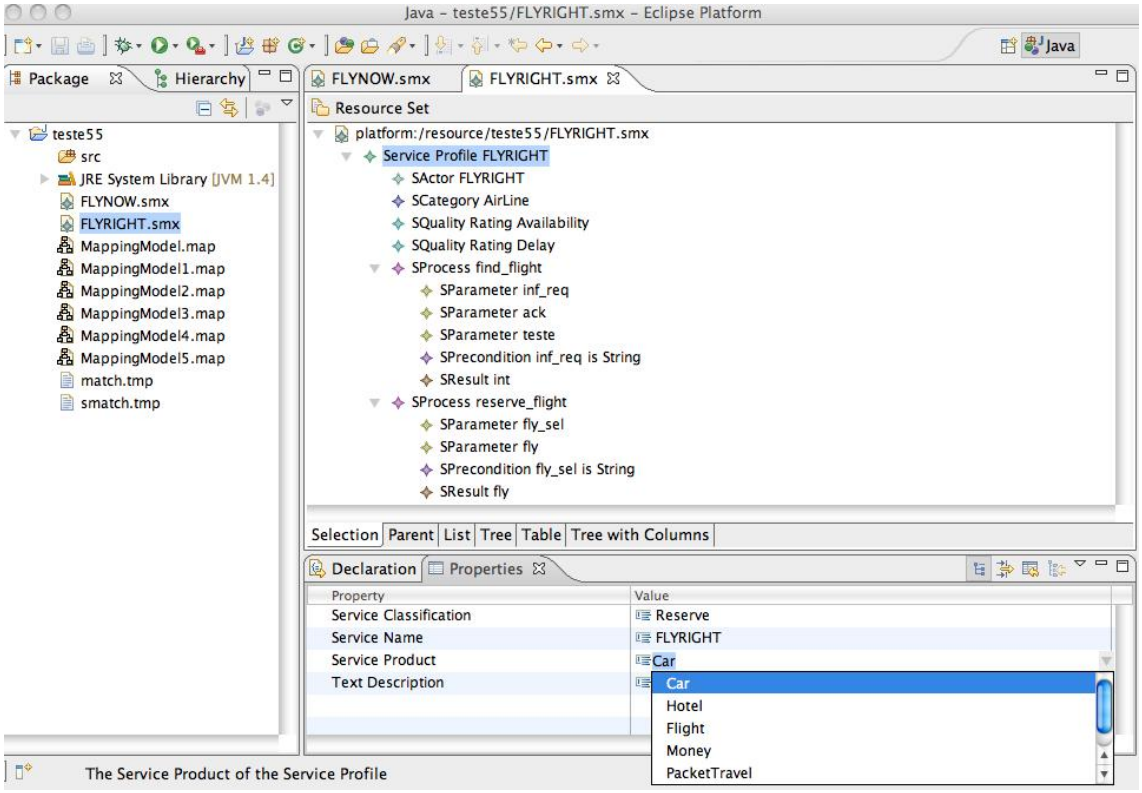


Figura 36 – Ferramenta para criação de Modelos de Serviços

No Modelo Semântico de Serviço ilustrado de forma hierárquica na Figura 36, observa-se que cada entidade e cada objeto têm propriedades e campos que devem ser preenchidos, incorporando uma representação conceitualmente mais rica para o serviço, em função das informações semânticas associadas a eles. Um importante benefício desta abordagem se dá ao fato das informações estarem pré-classificadas, o que facilita e possibilita uma abordagem de *Matching* semântico mais preciso, com uma busca direcionada e limitada ao universo de cada classificação.

Como ilustra a Figura 37, a ferramenta cria modelos semânticos, em formatos de arquivos *.XMI (*XML Metadata Interchange*) que habilita a troca de metadados entre diferentes ferramentas de modelagem.

FLYRIGHT.semantic_service

```
<?xml version="1.0" encoding="UTF-8"?>
<semantic_service:ServiceProfile xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:semant
  <actor actorName="FLYRIGHT" webURL="http://www.flyright.com.br" email="contact@flyright.com.br"/>
  <category categoryName="AirLine"/>
  <quality ratingName="Availability" valueRat="Lower"/>
  <quality/>
  <process processName="find_flight">
    <parameter parameterName="inf_req" atributeType="String"/>
    <parameter parameterName="fly_list" locationParam="OUTPUT"/>
    <precondition expression="inf_req is not null"/>
    <result hasResult="true"/>
  </process>
  <process processName="reserve_flight">
    <parameter parameterName="fly_sel" atributeType="String"/>
    <parameter parameterName="fly" locationParam="OUTPUT" atributeType="Object"/>
  </process>
  <process processName="pay_flight">
    <parameter parameterName="pay_inf" atributeType="String"/>
    <parameter parameterName="ack_pay" locationParam="OUTPUT" atributeType="Boolean"/>
    <precondition expression="pay_inf is not null"/>
    <result hasResult="true"/>
  </process>
</semantic_service:ServiceProfile>
```

Figura 37 – Modelo de Serviço em XML *Metadata Interchange*.

Uma vez que os modelos de serviços são publicados no repositório de serviços semânticos, a ferramenta de *Matching,* aqui denominada de *Tool(Matching)*, como ilustra a Figura 34 no início deste capítulo, realiza a tarefa de criar correspondências semânticas e estruturais entre os elementos dos modelos de serviços.

5.2. Ferramenta de Correspondência para Modelos de Serviços

A ferramenta de correspondência (*Matching*) possui o metamodelo de mapeamento proposto (*MetamodelMapping*) como ilustrou a Figura 34, tal metamodelo é a base para estabelecer as correspondências entre os elementos dos modelos de serviços.

A Ferramenta de Correspondência MT4MDE

MT4MDE (*Mapping Tool for Model Driven Engineereing*) desenvolvida por D. Lopes [16] possibilita a identificação e validação de especificações de correspondências, bem como a geração de definições de transformação em ATL a partir de um modelo de correspondência.

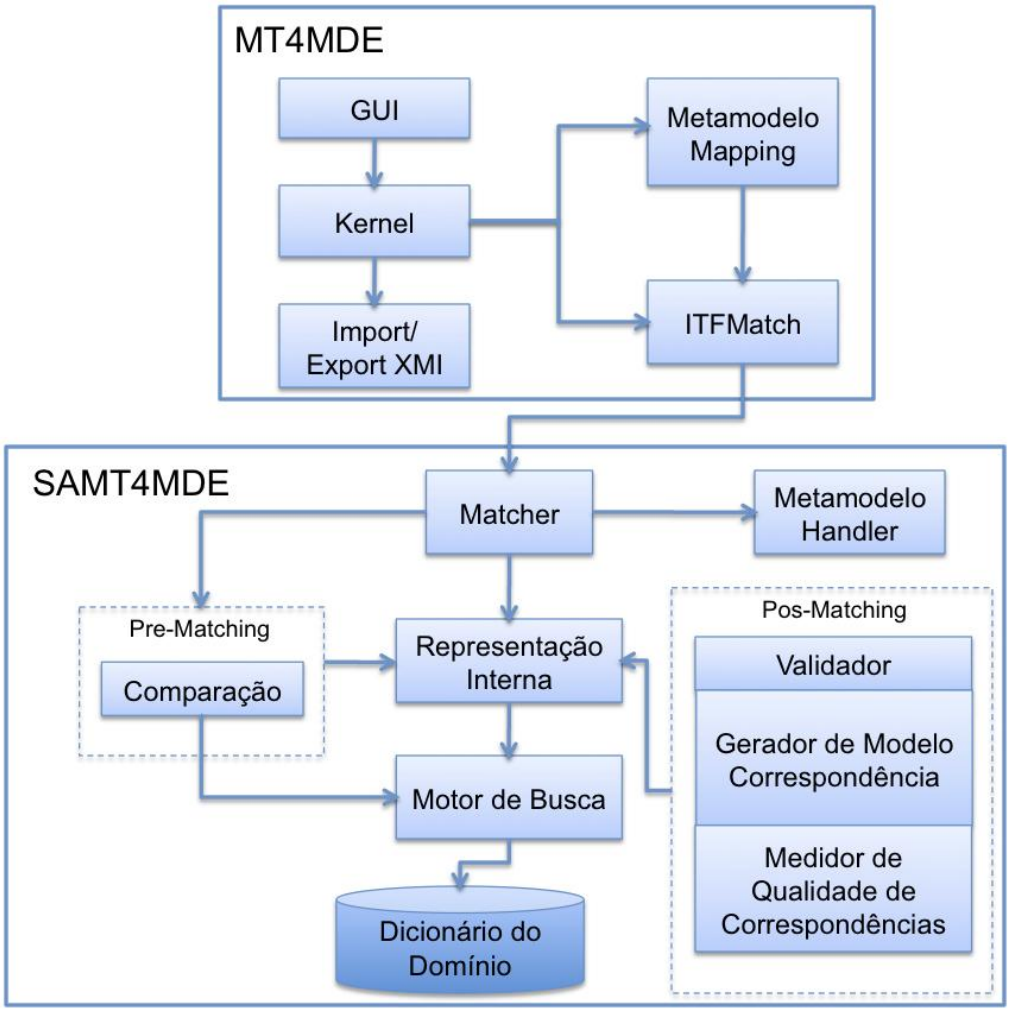


Figura 38 – Arquitetura de MT4MDE e SAMT4MDE.
Fonte [25]

Ela é composta pelos elementos conforme apresentado na Figura 38 e descritos a seguir [25]:

- GUI – é a interface gráfica;
- Kernel – executa todas as funcionalidades básicas de MT4MDE. É o núcleo da ferramenta;
- Importa/Exporta XMI – é o módulo que traduz o metamodelo no formato XMI para o formato Ecore. Ele permite também traduzir um metamodelo conforme o metamodelo Ecore no formato XMI;
- Matamodelo de *Mapping* – é um metamodelo usado para criar modelos de correspondência;
- ITFMatch – é a interface para manipular *Matcher*.

O MT4MDE foi implementado objetivando uma fácil extensão. Ferramentas que eventualmente precisem de mecanismos como geração, correspondência ou definição de transformação podem reutilizá-las [25].

A ferramenta de *Matching* proposta reutilizará parte da estrutura de MT4MDE, modificando aspectos como o algoritmo de *Matching*, descendo o Nível de Abstração da ferramenta, reescrevendo manipuladores de metamodelos para modelos, escrevendo novas classes e estruturas conforme o Metamodelo Semântico de Serviços e o Metamodelo de Correspondência (*sMatch)* propostos.

Geração Semi-automática de Correspondências SAMT4MDE

O SAMT4MDE (*Semi-Automatic Matching Tool for MDE*) teve como base a implementação do algoritmo de busca por similaridades estruturais e em *cross-relationship* [61]. A ferramenta SAMT4MDE desenvolvida por D. Lopes [16] [7] foi implementada utilizando o *framework* EMF. Tal ferramenta é uma extensão da ferramenta MT4MDE, como ilustra a Figura 38. SAMT4MDE é composta por [16] [7]:

- *Matcher* – módulo que implementa a interface *ITFMatch* para executar as funcionalidades de MT4MDE e coordena a criação de correspondências entre dois metamodelos;
- Representação Interna – representação mais adaptada para criação de correspondências como sendo um conjunto de elementos inter-relacionados;
- Metamodelo *Handler* – módulo que permite a navegação entre os metamodelos;
- Motor de Busca – realiza a busca de nomes sinônimos, ou seja, busca similaridades entre os elementos de metamodelos;
- Dicionário de Domínio – base de dados que armazena dicionário de domínios;
- Validador – recebe elementos correspondentes e interage com o usuário a fim de validá-los;

- Modelo Gerador de Correspondência – cria um modelo de correspondência a partir dos modelos validados pelo usuário;
- Medidor de Qualidade de Correspondências – avalia os resultados das medidas de qualidade das correspondências e fornece medidas de qualidade de correspondência.

SAMT4MDE complementa MT4MDE, esta junção tem a funcionalidade de determinar as correspondências entre dois metamodelos de forma semi-automática. A SAMT4MDE permite criação de um modelo de correspondência baseado nas correspondências [7].

Arquiteturas SAMT4MDE e MT4MDE adaptadas para o Matching de Modelos Semânticos

Depois de analisadas as ferramentas SAMT4MDE e MT4MDE, sua arquitetura foi adaptada para o problema do *Matching* de modelos semânticos de serviços como ilustra a Figura 39.

A ferramenta de *Matching* de modelos semânticos proposta neste trabalho, reutiliza boa parte da arquitetura original proposta por D.Lopes [6][7][16], readaptando módulos, componentes e classes, criando novos metamodelos projetados para o domínio deste trabalho.

Observa-se na Figura 39, nos círculos verdes, especificamente em MT4MDE, novos componentes e metamodelos foram projetados com novos objetivos:

- *Service Mapping Metamodel* – um novo metamodelo, como ilustrou a Figura 24 no Capítulo 4, usado para criar modelos de correspondência entre elementos de modelos semânticos de serviços;
- ITFSMatch – é a interface para manipular *sMatcher* agora tratando, não mais com pacotes (*packages*) de metamodelos, e sim com perfis de serviços (*ServiceProfiles*).

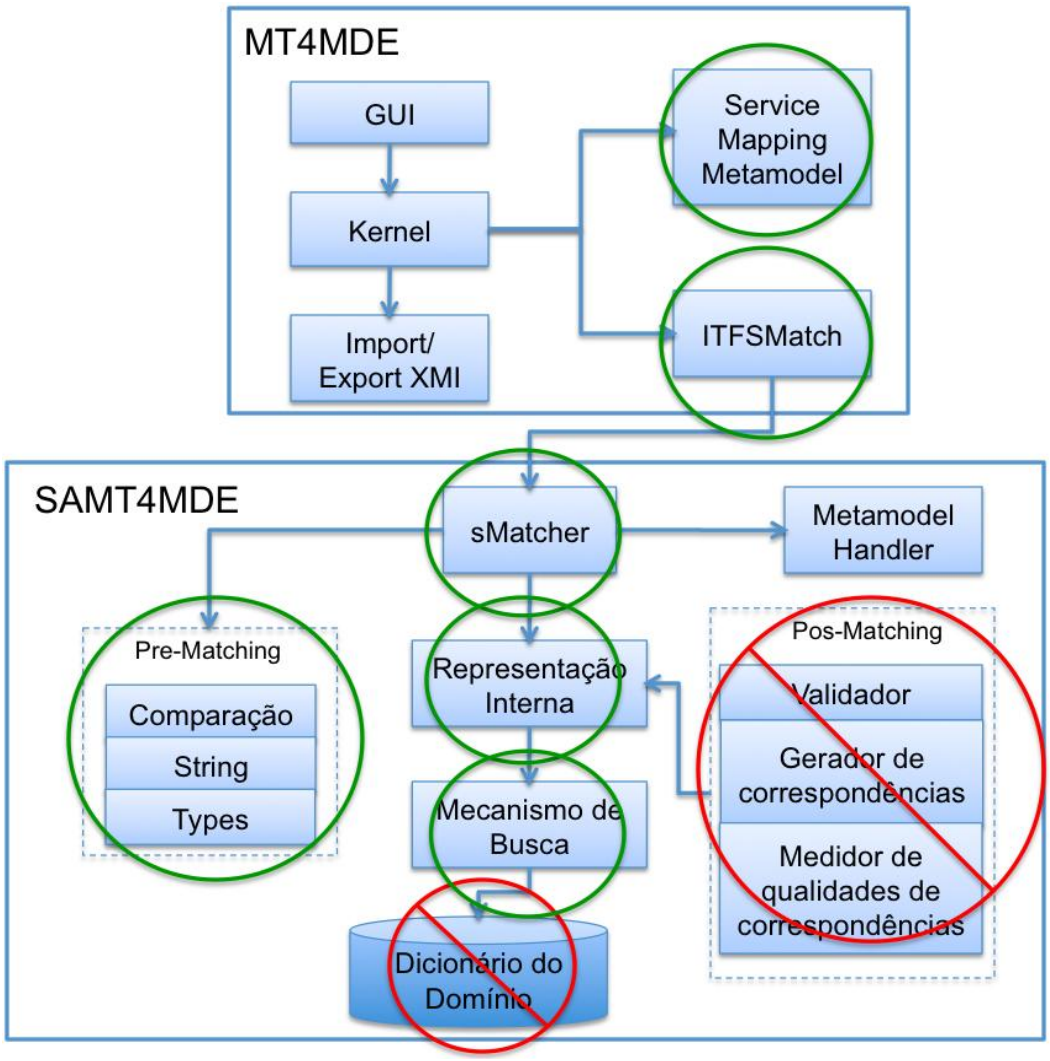


Figura 39 – Arquitetura adaptada de [25] para o *Matching* de Modelos Semânticos.

Ainda sobre a Figura 39, alguns componentes desnecessários foram retirados, identificados com um círculo e faixa vermelhos. Nos círculos verdes em SAMT4MDE, novas classes, metodologias de comparação e componentes foram reprojetados:

- *sMatcher* – módulo que implementa a interface *ITFSMatch* para executar as funcionalidades de MT4MDE e agora coordena a criação de correspondências entre dois modelos semânticos;
- Representação Interna – representação adaptada para criação de correspondências entre elementos dos nossos modelos semântico, e ainda sim tratado como um conjunto de elementos inter-relacionados;
- *Metamodel Handler* – módulo que permite a navegação agora entre modelos;

- Mecanismo de Busca – realiza a busca melhorada de nomes similares, através da divisão e comparação de palavras, e ainda buscas semanticamente exatas através de propriedades classificadas (*String e Types*). A busca por sinônimos foi praticamente cancelada, pois um Dicionário de Domínio não suportaria a dimensão de sinônimos para qualquer nome de serviço, podendo o mesmo dicionário ser substituído pelo WordNet [62] [63];
- Dicionário de Domínio – não mais utilizada a base de dados que armazena dicionário do domínio de metamodelos;
- Validador – recebe elementos correspondentes e interage com o usuário a fim de validá-los. O validador poderá ser descartado considerando uma seleção e composição de serviços automatizada.

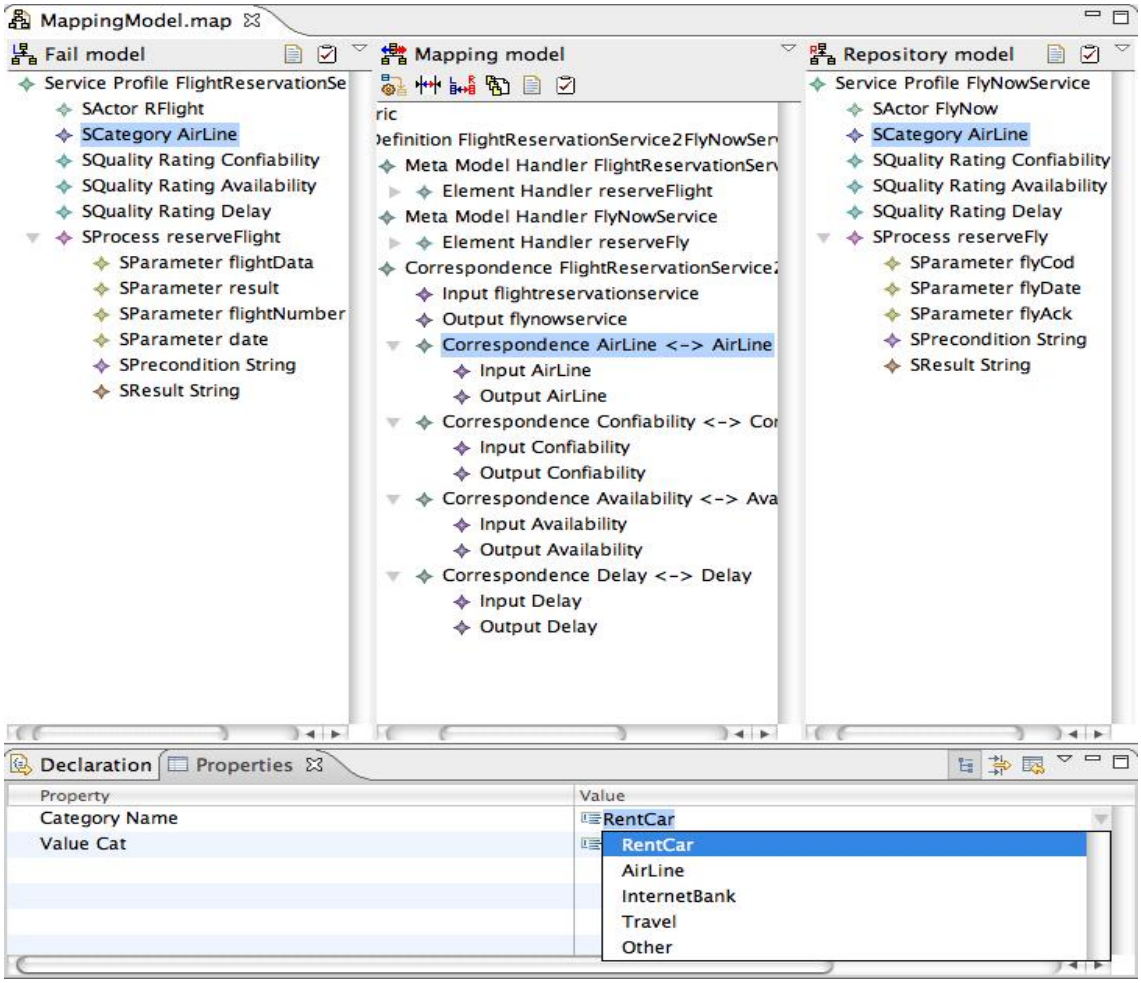


Figura 40 – Ferramenta para realizar *Matching* de Modelos de Serviços.

Como ilustra a Figura 40, a ferramenta realiza o *Matching* de dois modelos semânticos de serviços. Relaciona elementos de modelos da direita com os da esquerda e na sequência retorna o mapeamento.

5.3. Implementação do Algoritmo de *Matching*

O algoritmo para realizar o *Matching* é baseado na Teoria da Probabilidade como citado anteriormente na seção 4.4. Nesta seção, serão tratadas pequenas listagens de código referentes a implementação do algoritmo de *Matching* proposto.

Como ilustra a Listagem 1, um trecho do algoritmo de *Matching* em Java realiza comparações de Produto (Linha 282) e Classificação do Serviço (Linha 286). Em seguida, caso sejam iguais, o algoritmo realiza a soma da probabilidade para cada *Type* (+ *probForType*) nas Linhas 283 e 287.

```
281
282             if(serviceProfileA.getServiceProduct().equals(serviceProfileB.getServiceProduct())){
283                 averageExactSemantic = (float) (averageExactSemantic + probForType); //0.125
284             }
285
286             if(serviceProfileA.getServiceClassification().equals(serviceProfileB.getServiceClassification())){
287                 averageExactSemantic = (float) (averageExactSemantic + probForType); //0.125
288             }
289
```

Listagem 1 – Parte do Algoritmo de *Matching*.

Para processos de serviços que possuem características estruturais, deve ser feita uma análise da estrutura do processo, quantidade de processos, tipos, localização, relação e quantidade de parâmetros. O primeiro passo do algoritmo de *Matching* é calcular a probabilidade local para cada processo como ilustra a Listagem 2 entre as Linhas 369 e 373. Ou seja, no momento em que o *loop* buscar o primeiro processo do lado A, a média para este processo é zerada (*averageThisProcess*). Em seguida, um valor de probabilidade local é calculado para este processo, ou seja, no perfil do serviço eram contados os *Types* e calculado um valor para cada *Type* (ex: 0,125 ou 12,5%). Já no cálculo para processos, cada probabilidade local poderá ter um valor diferente, como ilustra a Linha 372 (*probForProcess*).

```
362
363    private void matchProcess(ServiceProfile serviceProfileA, ServiceProfile serviceProfileB, MatchedServiceProfile mServiceProfile) {
364
365        int contProcessA=0;
366
367        for (Iterator iterProcessA = serviceProfileA.getProcess().iterator(); iterProcessA.hasNext();) {
368            SProcess processA = (SProcess) iterProcessA.next();
369            // Para cada processo do Serviço A (ou F) ele cácula uma Similaridade Local com todos os Processos do Serviço B (ou R)
370            averageThisProcess=0;
371            contProcessA++;
372            probForProcess = (float) 1/((processA.getParameter().size()) + 2); //0.25
373            //(qtd. Parameter + 2 =>> (1 Result + 1 Precondition))
374
375            for (Iterator iterProcessB = serviceProfileB.getProcess().iterator(); iterProcessB.hasNext();) {
376                SProcess processB = (SProcess) iterProcessB.next();
377
378                int phi = phiProcess(processA, processB);
379                if (phi >= 0) {
380
381                    MatchedSprocess mProcess = (MatchedSprocess) smatchFactory
382                            .create((EClass) smatchPackage
383                                    .getEClassifier("MatchedSprocess"));
384                    mProcess.setA12(processA.getProcessName());
385                    mProcess.setB12(processB.getProcessName());
386                    mProcess.setName(processA.getProcessName() + " <-> " + processB.getProcessName());
387                    //
388                    matchPrecondition(processA, processB, mProcess);
389
390                    matchResult(processA, processB, mProcess);
391
392                    matchParameter(processA, processB, mProcess);
393
394                    mServiceProfile.getSp2pr().add(mProcess);
395
396                }
397
398            }
399            System.out.println("Probabilidade por Processo ["+contProcessA+"]: "+ averageThisProcess);
400            probForAllProcess = probForAllProcess + averageThisProcess;
401
402
403        }
404    }
```

Listagem 2 – Parte do Algoritmo de Matching de Processos.

A Listagem 3 ilustra as relações entre *String* e *char* já citadas na seção 4.4. E também relações entre *int* e *float*, ou seja, pode-se representar um *int* através de um *float*, mas não conseguimos representar um *float* através de um *int*.

```
551        if(parameterA.getLocationParam().getName().equals("OUTPUT") && parameterB.getLocationParam().getName().equals("OUTPUT")){ // if OUT with OUT
552
553            if(parameterA.getAtributeType().getName().equals(parameterB.getAtributeType().getName())){ // if the same name (String, char, int, float,...)
554
555                averageThisProcess = averageThisProcess + probForProcess;
556
557            }else if(parameterA.getAtributeType().getName().equals("String") && parameterB.getAtributeType().getName().equals("char")){
558
559                averageThisProcess = averageThisProcess + probForProcess;
560
561            }else if(parameterA.getAtributeType().getName().equals("int") && parameterB.getAtributeType().getName().equals("float")){
562
563                averageThisProcess = averageThisProcess + probForProcess;
564            }
```

Listagem 3 – Parte do Algoritmo de *Matching* de Resultados.

5.4. Síntese

Este capítulo apresentou a implementação do protótipo de um *Framework* para suportar a Composição Dinâmica de Serviços Web. Um esquema geral detalhou a implementação das ferramentas. Em seguida, um esquema geral para detalhar o desenvolvimento das ferramentas para suportar a composição dinâmica foi apresentado. O relacionamento entre as ferramentas e seus respectivos papéis foram tratados nas seções subsequentes.

A ferramenta para a criação de modelos semânticos de serviços foi descrita, bem como a ferramenta para realizar a composição através do *Matching* (correspondência) de informações de modelos semânticos de serviços.

Neste capítulo, também foi apresentado uma visão geral das ferramentas MT4MDE e SAMT4MDE com suas arquiteturas relacionadas. E a proposta de extensão e adaptação dessas arquiteturas, para fins e objetivos diferentes do original. A implementação de novos módulos e desenvolvimento de novos modelos e como tudo se relaciona foi ilustrado.

Finalizando este capítulo, alguns trechos de código referentes a implementação do algoritmo de *Matching* de Modelos Semânticos de Serviços foram apresentados.

6. Estudo de Caso, Testes e Execução

Este capítulo apresenta um estudo de caso no domínio de serviços web de linhas aéreas, pagamentos e agendamento de vôo. Neste estudo de caso, pretende-se avaliar a funcionalidade do framework proposto, o *matching* de modelos semânticos de serviços descritos, bem como as ferramentas propostas no capítulo anterior.

O objetivo é criar uma composição de serviços web, em seguida criar os respectivos modelos semânticos de serviços web com a ferramenta baseada na DSL. Simular uma falta em um dos serviços web da composição e utilizar o modelo semântico do serviço que faltou como fonte para busca de um novo serviço similar no repositório. Para isso, a ferramenta de *Matching* de modelos semânticos de serviços será utilizada, para medir o grau de similaridade entre as comparações e eleger o modelo com maior grau de similaridade semântica e estrutural. Finalmente, um novo arquivo da composição de serviços web com as informações do novo serviço escolhido será gerado.

6.1. Apresentação do Estudo de Caso

Com o objetivo de ilustrar as funcionalidades do *framework* DWSC, um estudo de caso que consiste no desenvolvimento de uma composição de serviços web foi proposto, para realizar o agendamento de vôos como ilustra a Figura 41.

Esta composição permite de forma transparente para os clientes, realizarem a reserva de vôo e o pagamento do mesmo, através da requisição de um único serviço web de agendamento de vôo.

O termo orquestração é utilizado para descrever processos executáveis que interagem com serviços internos e externos, determinam regras de negócio e a ordem de execução das tarefas de processos geralmente longos, transacionais e de múltiplos passos [67].

Neste estudo de caso, a orquestração de serviços web que define a sequência e seleção dos serviços web participantes é realizada pela composição de serviços de Agendamento de Vôo, como ilustra a Figura 41.

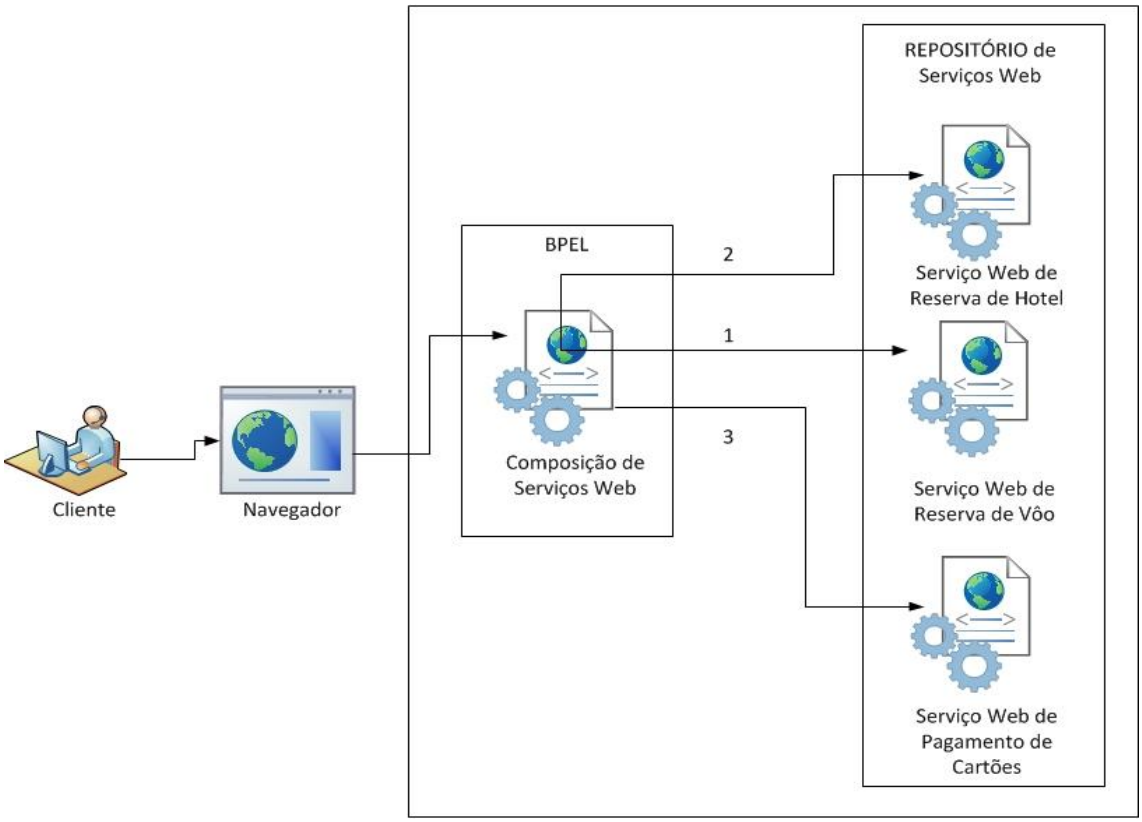


Figura 41 – Estrutura do Estudo de Caso.

6.2. Modelagem do Estudo de Caso

Na modelagem do Estudo de Caso de Agendamento de Vôo, os diagramas UML de Caso de Uso para delimitar o escopo do estudo de caso, diagrama de classes e diagrama de sequência foram desenvolvidos.

O diagrama de caso de uso organiza o comportamento de um sistema através de uma perspectiva do usuário. Como ilustra o diagrama de caso de uso da Figura 42, o serviço *FlightBooking* (agendamento de vôo) é uma composição de serviços web que realiza a orquestração de chamadas dos serviços web denominados *FlightReservation* (reserva de vôo) e *PaymentProcessing* (pagamento), a implementação destes três serviços web faz parte do projeto Eclipse SOA [68].

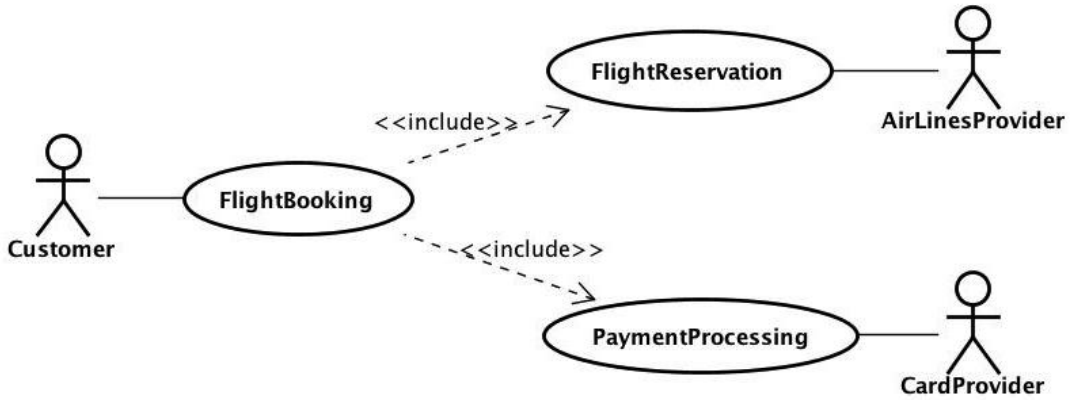


Figura 42 – Composição de Agendamento de Vôo: Diagrama de Caso de Uso.

Os diagramas de classes são utilizados para ilustrar uma visão estática do processo de um sistema, através de um conjunto de classes, interfaces, colaborações e seus relacionamentos [70].

Como ilustra o diagrama de classes da Figura 43, as três classes referentes a implementação dos serviços web (*FlightBookingImpl, PaymentReservationImpl, FlightReservationImpl*) herdam de *ServiceProfile* as características semânticas que um serviço poderá ter. Na classe *FlightBookingImpl* a operação *bookFlight* foi implementada realizando chamadas a operação *reserveFlight* da classe *FlightReservationImpl* e a operação *processPayment* da classe *PaymentProcessingImpl.*

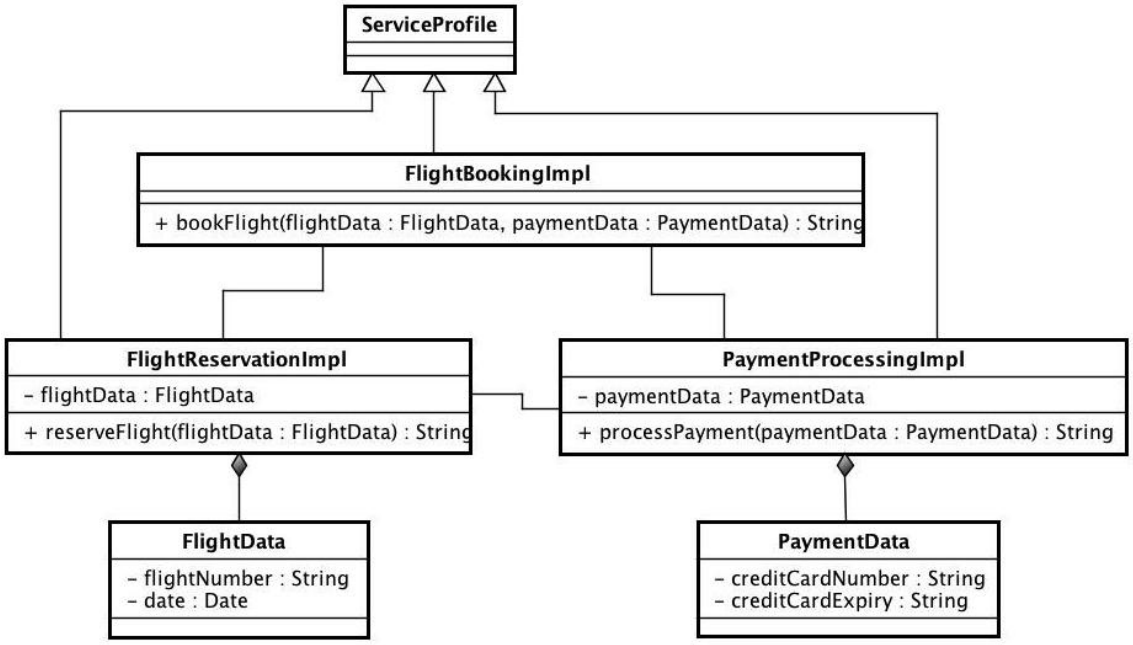


Figura 43 – Agendamento de Vôo: Diagrama de Classes.

O diagrama de sequência apresenta uma visão dinâmica de um sistema, dando ênfase à ordenação temporal das mensagens ilustradas em uma interação [70].

A Figura 44 apresenta o diagrama de sequência que ilustra a composição de serviços para realizar o agendamento de vôo por parte de um cliente. A aplicação cliente faz uma chamada ao serviço web *FlightBooking* através do método *bookFlight,* passando como dados de entrada as informações do vôo e do pagamento. Na sequência o serviço *FlightBooking* realiza duas chamadas para dois serviços distintos. O primeiro é uma chamada ao processo *reserveFlight* do serviço *FlightReservation* passando os dados do vôo como parâmetro. O segundo serviço é chamado através do processo *processPayment* passando os dados do cartão de crédito para efetivar o pagamento. Finalmente, se os dois serviços tiverem os resultados positivos, o serviço web *FlightBooking* retorna o sucesso das operações para o cliente.

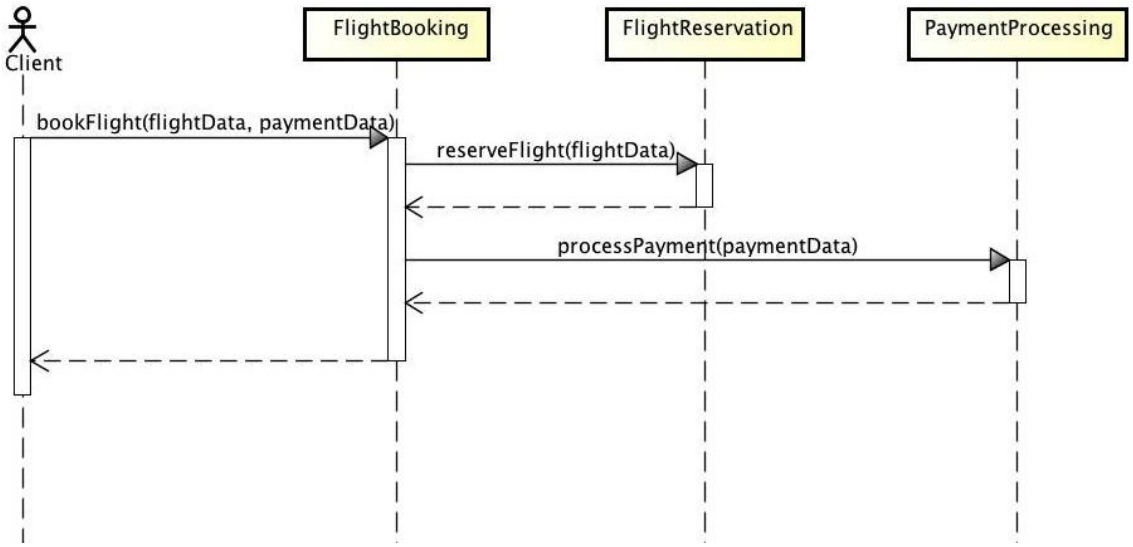


Figura 44 – Agendamento de Vôo: Diagrama de Sequência.

6.3. Utilização do Protótipo e Testes

Depois da etapa de modelagem do estudo de caso de agendamento de vôo e criação da composição de serviços web, o objetivo a seguir é criar os respectivos modelos semânticos de serviços web com a primeira ferramenta proposta do *framework* DWSC baseada na DSL.

O primeiro modelo semântico apresentado é o do serviço web *FlightReservation*, como ilustra a Figura 45. Todas as características semânticas do modelo (perfil) são preenchidas pelo desenvolvedor do serviço. Observa-se as informações pré-classificadas *SQuality Confiability = Lower*, *SQuality Availability = Lower e SQuality Delay = High*, ou seja, um serviço com uma qualidade de modo geral ruim. Este serviço se enquadra na categoria de linhas aéreas (*SCategory = AirLine*).

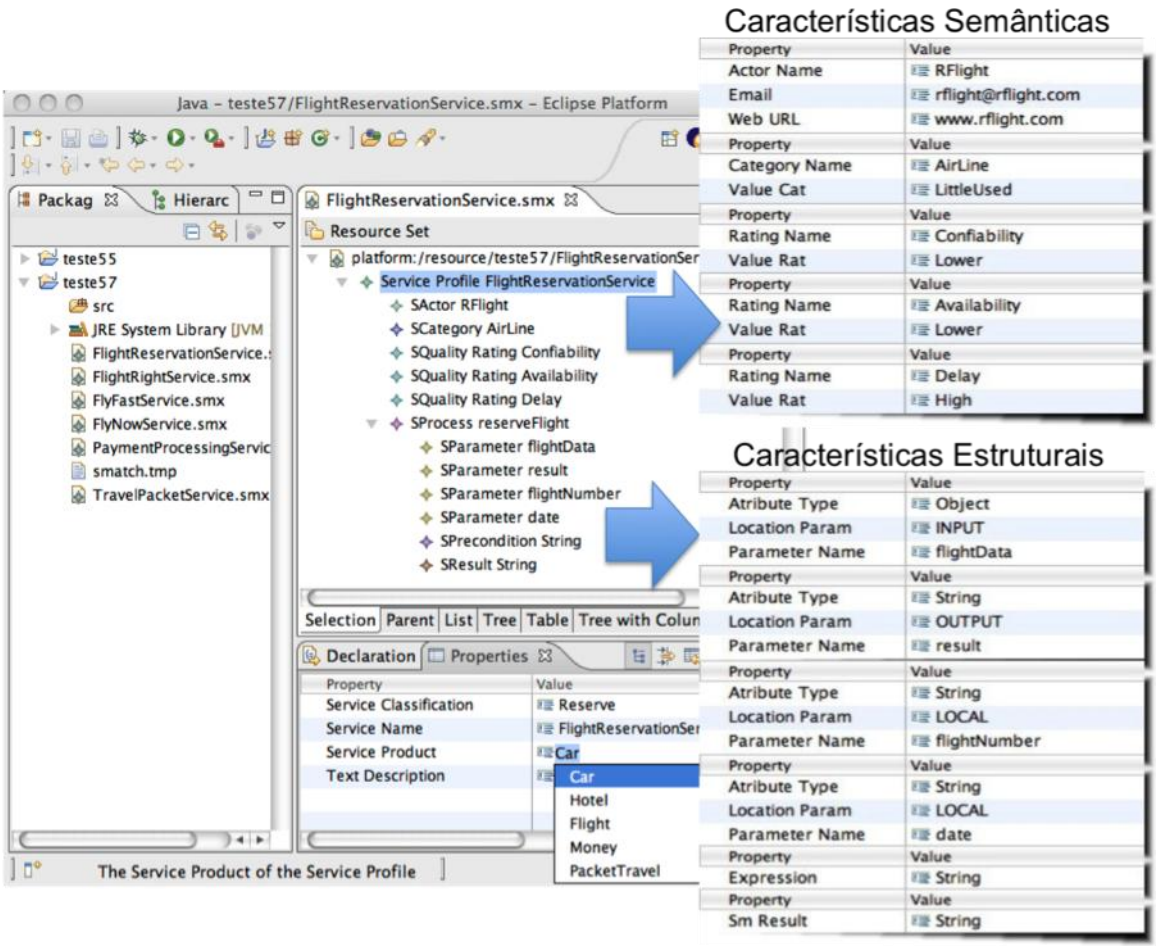


Figura 45 – Modelo Semântico: FlightReservation

A nível de processos, o serviço *FlightReservation* possui um processo *SProcess reserveFlight* que recebe um parâmetro *SParameter = flightData* de entrada (*Location Paramenter = INPUT*), este parâmetro é do tipo complexo (*Atribute Type = Object*). No mesmo processo, o parâmetro de retorno é denominado resultado (*SParameter = result*). Os dois últimos parâmetros do processo são locais, ou seja, utilizados somente dentro da

lógica de negócio do serviço web, como ilustrou o diagrama de classes da Figura 43.

Todos os modelos de serviços contidos no repositório foram pré-classificados, mas neste trabalho os modelos que serão comparados com o modelo do serviço *FlightReservation*, no qual será simulado uma falta, serão ilustrados. Os modelos eventualmente comparados estão ilustrados na Figura 46, Figura 48 e Figura 50.

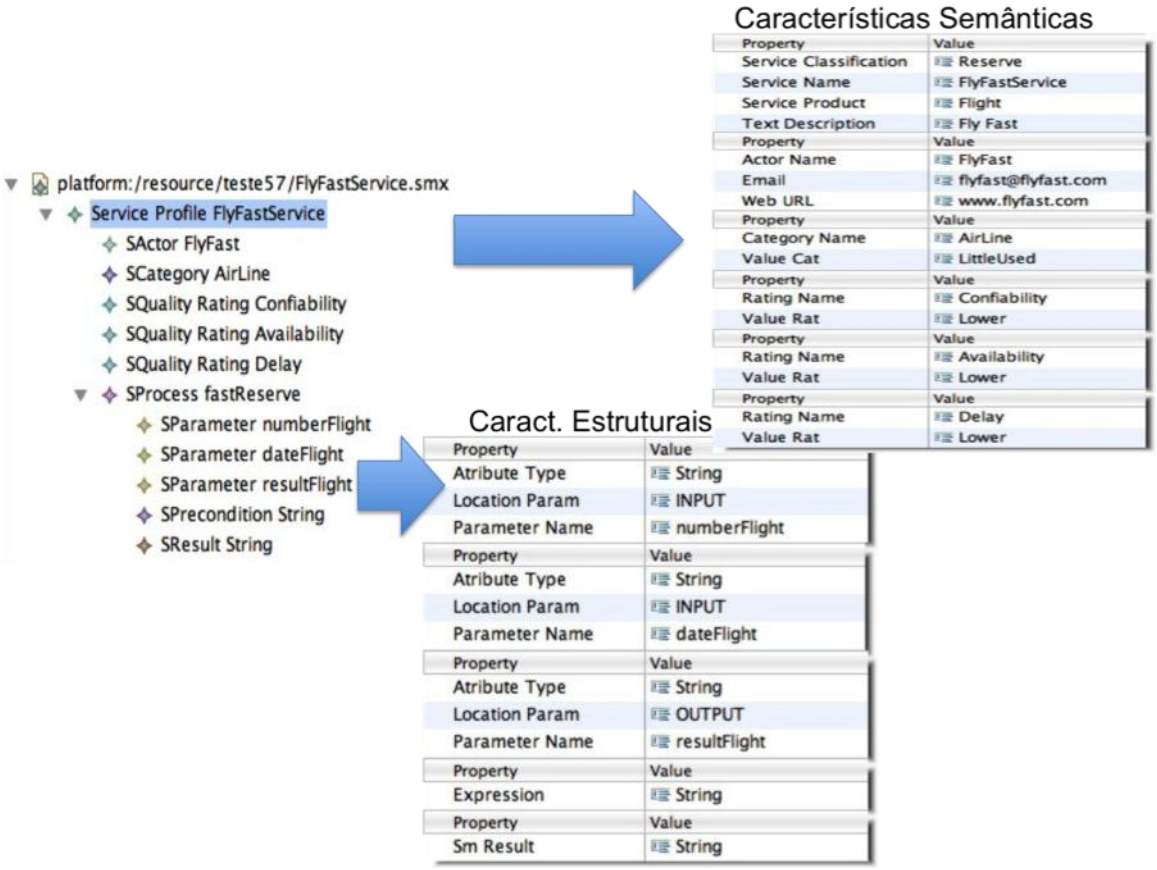


Figura 46 – Modelo Semântico: FlightFast

O resultado da comparação do modelo semântico do repositório *FlyFast* com o modelo *FlightReservation* que faltou é ilustrado com a ferramenta de *Matching* proposta, como ilustra a Figura 47.

A tela de ***Match*** gerada ao centro desta figura, ilustra o resultado das correspondências encontradas entre os dois modelos. Selecionando todas as correspondências geradas, tem-se como resultado o **Modelo de Mapeamento** que relaciona elementos dos dois modelos semânticos. Tal modelo de mapeamento é a base para a geração do fragmento de código BPEL responsável pela invocação do novo serviço.

Os valores das probabilidades resultantes das comparações são ilustrados na tela de **Console** na parte inferior da Figura 47. Observa-se na coluna esquerda (*Fail model*), que o processo *SProcess reserveFlight* possui quatro parâmetros (*SParameter*), uma precondição (*SPrecondition*) e uma condição de saída (*SResult*), totalizando seis elementos dentro deste processo. Nota-se na tela de **Console** que a probabilidade calcula para este único processo do serviço (Probabilidade por Processo [1] = 0.5) foi 50%. Este valor é justificável analisando a tela central de ***Match,*** que detectou três correspondências dentre as seis possíveis, ou seja, *SPrecondition*, *SResult* e um parâmetro *SParameter* que foram aceitos.

Caso o serviço tenha mais de um processo, as probabilidades calculadas para cada processo (Probabilidade por Processo [X]) são somadas e o resultado dividido pelo total de processos, ou seja, a média. O valor desta média é então atribuído a variável *ProbForAllProcess* como ilustra o **Console** da Figura 47.

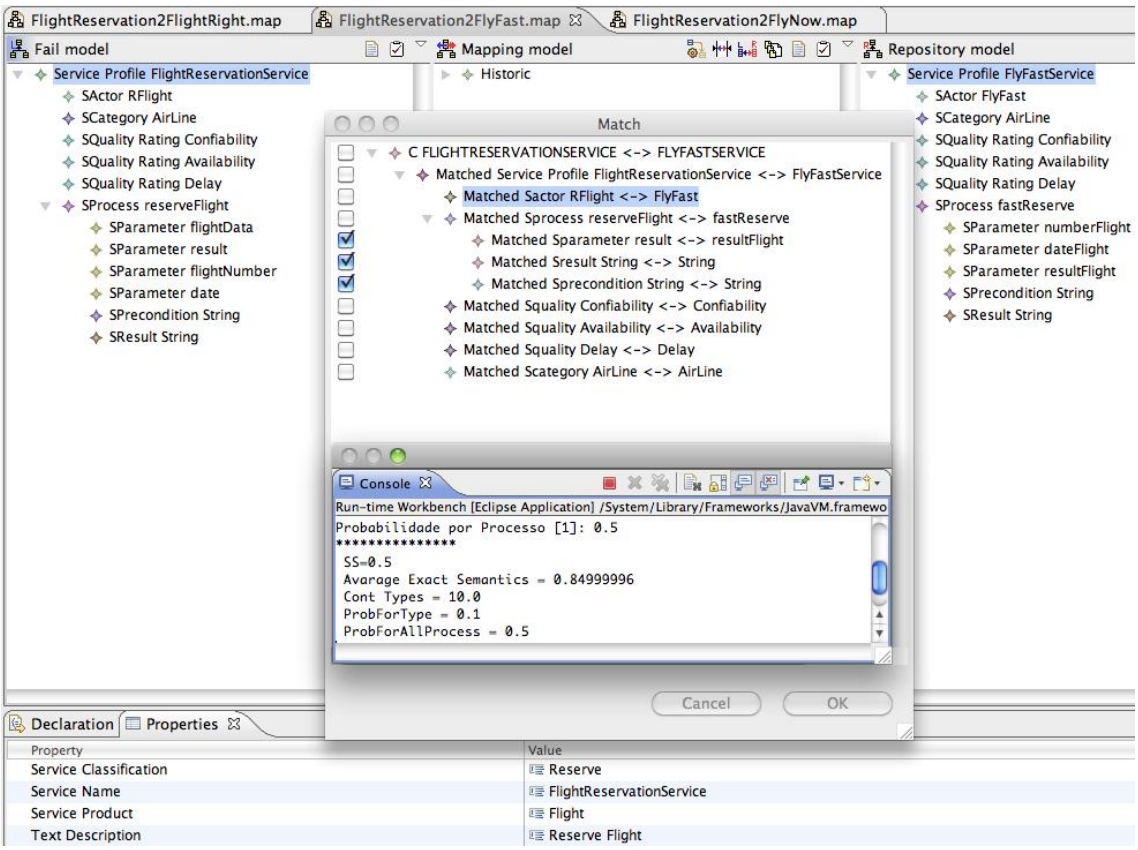


Figura 47 – Resultado do Matching: FlightReservation vs. FlytFast

Considerando o tratamento para o perfil de serviço, como explicado nas seções anteriores, o primeiro passo do algoritmo é contar todos *Types* existentes no modelo que faltou, *Cont. Types* = 10 no **Console**. Em seguida, calcular a probabilidade local que será somada a cada sucesso nas comparações entre as características semânticas do perfil do serviço, ou seja, *ProbForType* = 0.1 no **Console**.

Depois de obtidos esses valores, o algoritmo de *Matching* percorre todos os elementos semânticos dos modelos existentes no repositório. A medida que as características semânticas são comparadas, as devidas penalidades são aplicadas resultando em um valor de probabilidade. Na comparação da Figura 47, este valor é denominado de média de elementos semânticos exatos, ou *Avarage Exact Semantics* = 0.849 (84,9%) como ilustra no **Console** para esta comparação. Se somado as probabilidades das características semânticas 0.849 com 0.5 das estruturais, tem-se o valor total de **1.349**. Este valor servirá de parâmetro para a comparação seguinte.

Figura 48 – Modelo Semântico: FlyNow.

A Figura 48 ilustra o segundo modelo *FlyNow*, que será comparado com o modelo *FlightReservation* que faltou. Vale observar o parâmetro de saída do processo deste modelo de serviço que é um *Boolean flyAck* (*OUTPUT*). Em função deste parâmetro, a condição de saída não será satisfeita, como ilustra tela de ***Match*** da Figura 49.

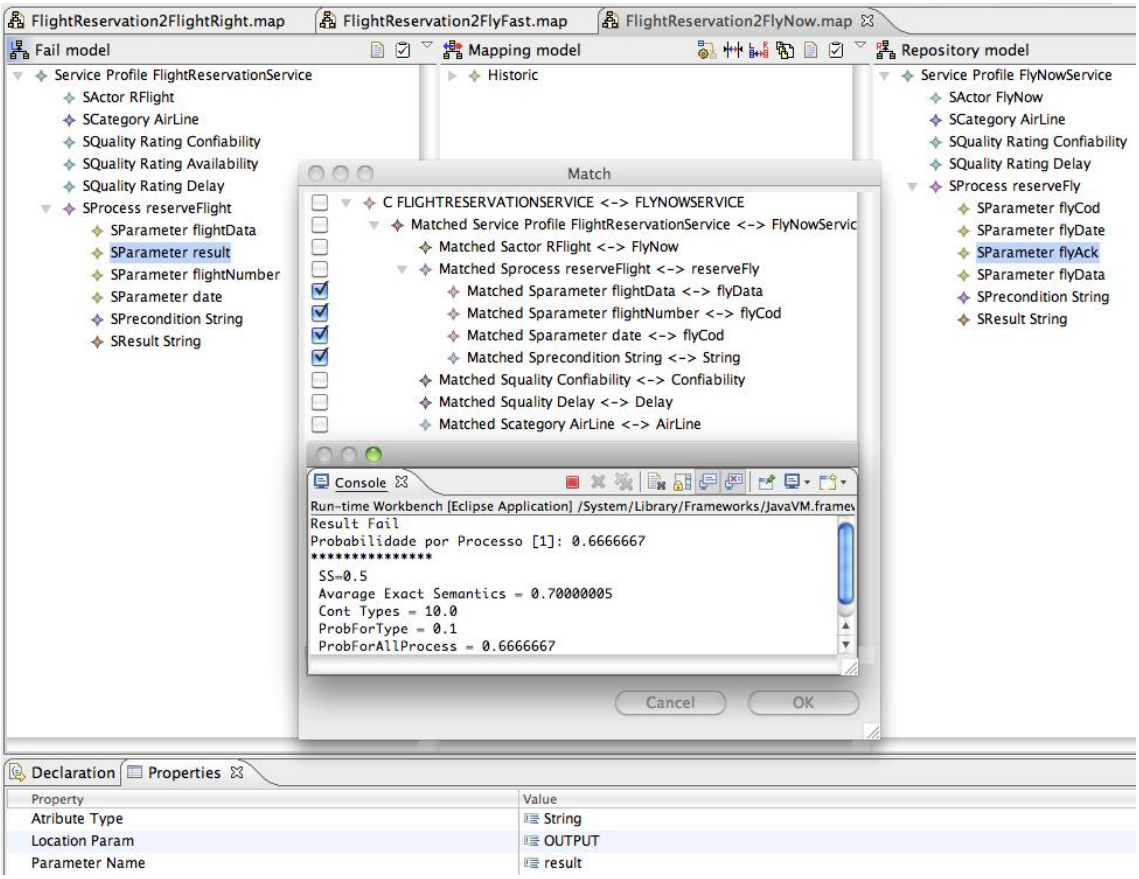


Figura 49 – Resultado do Matching: FlightReservation vs. FlyNow.

No resultado da comparação dos modelos *FlightReservation* com *FlyNow*, na tela **Console** da Figura 49, a primeira linha indica uma mensagem *Result Fail* que significa que a condição de saída não foi satisfeita. Embora na tela de ***Match*** indique que três parâmetros e a pré-condição foram satisfeitos, o que implica em uma probabilidade para este processo de 66,6%, que é superior ao valor de 50% da primeira comparação. Como o algoritmo obedece a análise estrutural do modelo, essa restrição de falta detectada impede a seleção de *FlyNow*, independente dos seus valores de probabilidades superiores.

Com relação as características semânticas de *FlyNow*, ele possui os níveis de qualidades médios (*Middle*) para Confiabilidade (*Confiability*) e Atraso (*Delay*), como ilustrou a Figura 48. Ou seja, existe uma balanceamento, pois uma confiabilidade de nível médio é **melhor** do que o nível baixo de *FlyFast* e respectivamente o atraso com o nível médio que é **pior** do que o nível baixo de *FlyFast*. Contudo, o fato do *FlightReservation* ter uma característica semântica a mais, denotada Disponibilidade (*Availability*) que não existe no modelo *FlyFast*, contribuiu para uma probabilidade da parte semântica baixa (70%) devido a falta de especificidade por parte do modelo *FlyFast*, como ilustra a Figura 49. Somando-se as probabilidades das características semânticas 0.70 com 0.66 das estruturais, tem-se o valor total para *FlyNow* **1.366** é superior a soma de **1.349** de *FlyFast*, porém *FlyNow* não poderia ser escolhido devido a quebra da condição de saída (SResult) estabelecida pelo algoritmo de *Matching*.

O modelo denominado *FlightRight* da Figura 50, foi totalmente preenchido e especificado permitindo uma melhor precisão nas comparações.

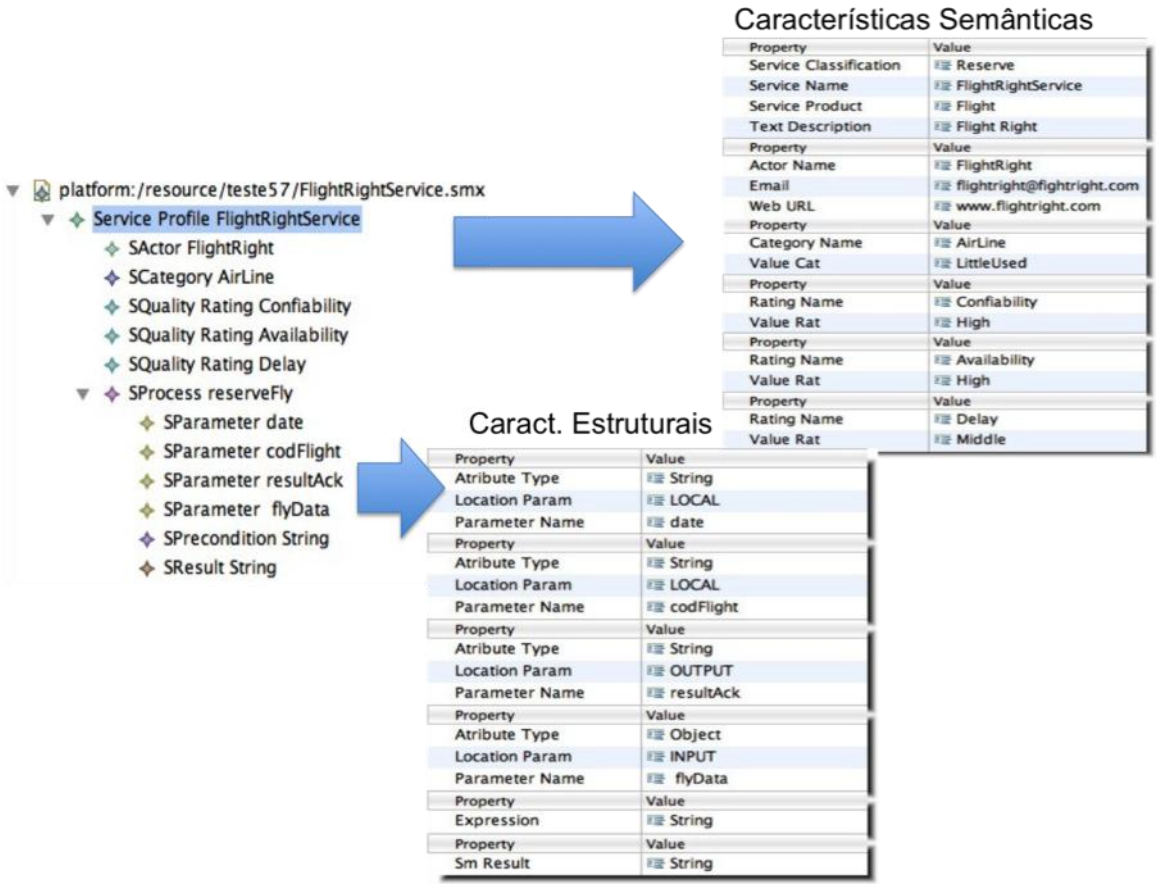


Figura 50 – Modelo Semântico: FlightRight.

O resultado do *Matching* entre os modelos *FlightRight* com *FlightReservation* é ilustrado na Figura 51.

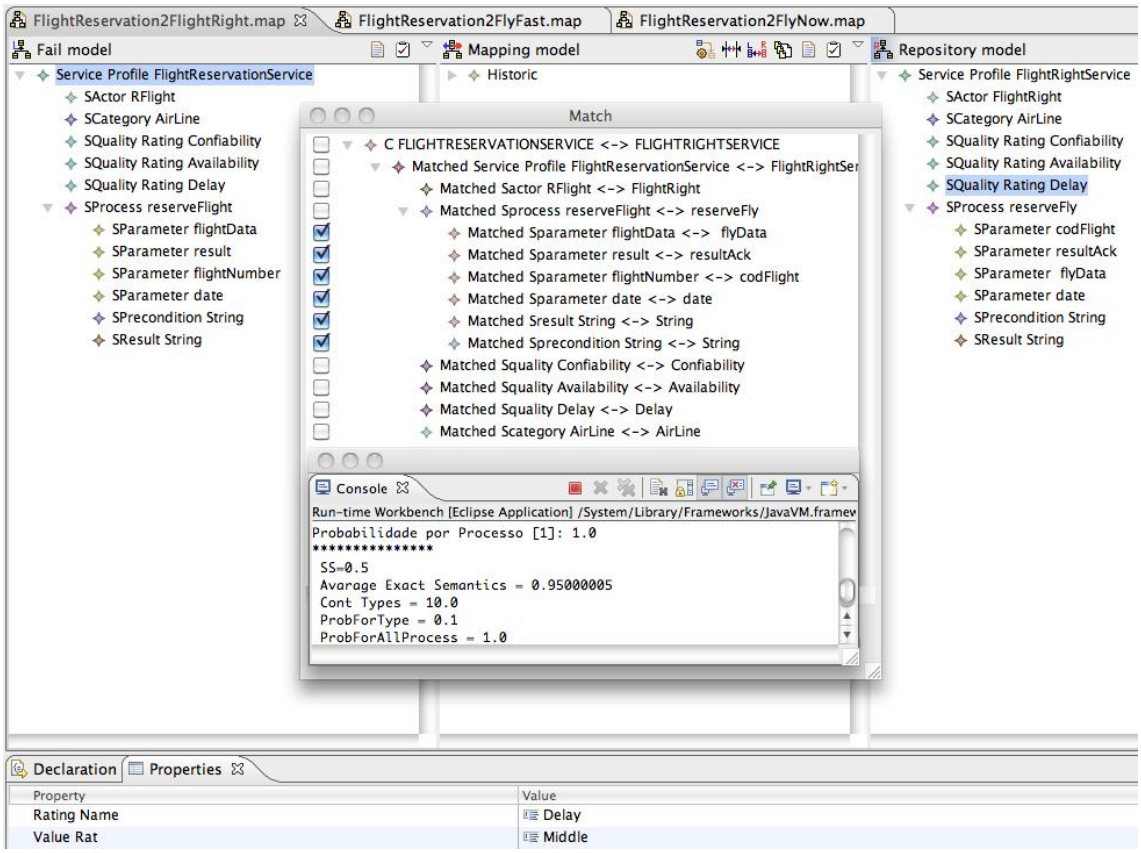


Figura 51 – Resultado do Matching: FlightReservation vs. FlightRight.

Nota-se na tela de ***Match*** desta figura, relacionado as características estruturais, todos os parâmetros, pré-condição e condição de saída foram encontrados, ou seja, probabilidade para processos com o valor de 100%.

No *Matching* das características semânticas, obteve-se uma probabilidade de 95%, como ilustra o **Console** da Figura 51. Este valor sofreu uma pequena penalização, pois o valor do atraso (*Delay*) do serviço que faltou *FlightReservation* é Baixo (*Lower*) e o valor do atraso do *FlightRight* é Médio (*Middle*), lembrando que quanto maior é o atraso pior é o serviço.

Geração de Fragmento BPEL

Depois de encolhido o modelo *FlightRight*, com o maior grau de similaridade, o modelo de mapeado (*Mapping Model*) gerado pelas

correspondências, deve permitir uma navegação entre os elementos de *FlightRight* facilitando a geração do fragmento de código BPEL. Como ilustra a Figura 52, o BPEL da composição dos três serviços envolvidos foi modelado com o objetivo de capturar o fragmento de código BPEL responsável pela invocação do novo serviço.

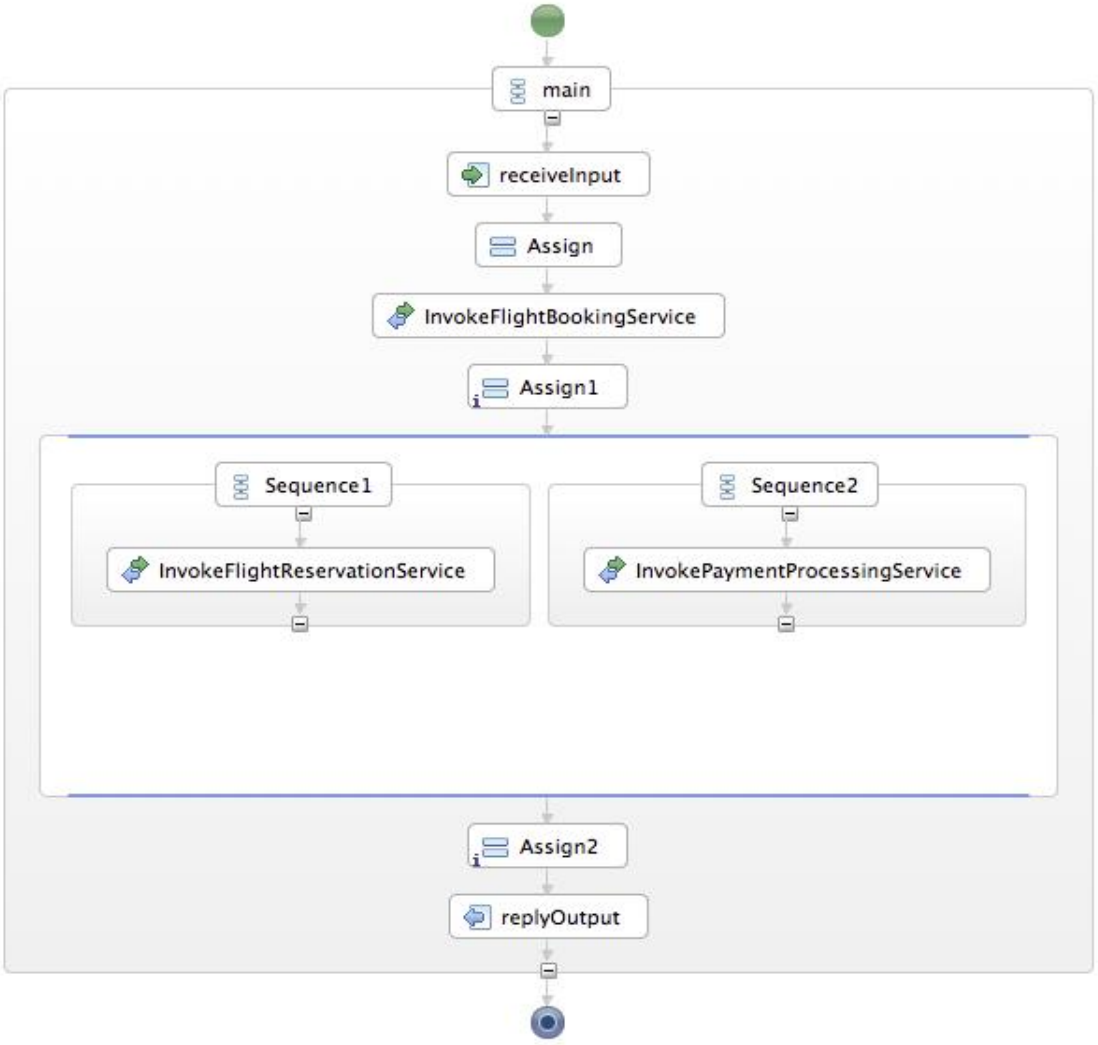


Figura 52 – BPEL do Estudo de Caso

Nota-se na Figura 52, que depois na primeira sequência de execução (*Sequence1*) o processo BPEL define uma invocação do serviço *FlightReservation* (*InvokeFlightReservationService*), em seguida, o serviço de pagamento é invocado (*InvokePaymentProcessingService*).

Considerando a falta do serviço *FlightReservation*, o objetivo então é capturar no código BPEL a sintaxe de invocação, para enfim, gerar o fragmento de código para realizar a chamada do novo serviço *FlightRight*.

```
<bpel:flow name="Flow">
    <bpel:sequence name="Sequence1">
        <bpel:invoke name="InvokeFlightReservationService" partnerLink="FlightReservationPL"
            operation="reserveFlight" portType="ns0:FlightReservation"
            inputVariable="FlightReservationPLRequest"
            outputVariable="FlightReservationPLResponse">
        </bpel:invoke>
    </bpel:sequence>
    <bpel:sequence name="Sequence2">
        <bpel:invoke name="InvokePaymentProcessingService" partnerLink="PaymentProcessingPL"
            operation="processPayment" portType="ns1:PaymentProcessing"
            inputVariable="PaymentProcessingPLRequest"
            outputVariable="PaymentProcessingPLResponse">
        </bpel:invoke>
    </bpel:sequence>
</bpel:flow>
```

Listagem 4 – Fragmento BPEL de Invocação FlightReservation

Como ilustra a Listagem 4, o novo fragmento de invocação deverá substituir o trecho de código entre as *Tags* de *Sequence1* e obedecer a Sintaxe especificada no fragmento. A funcionalidade de geração de código BPEL pode ser observada na ferramenta de *Matching*, como ilustra a Figura 53.

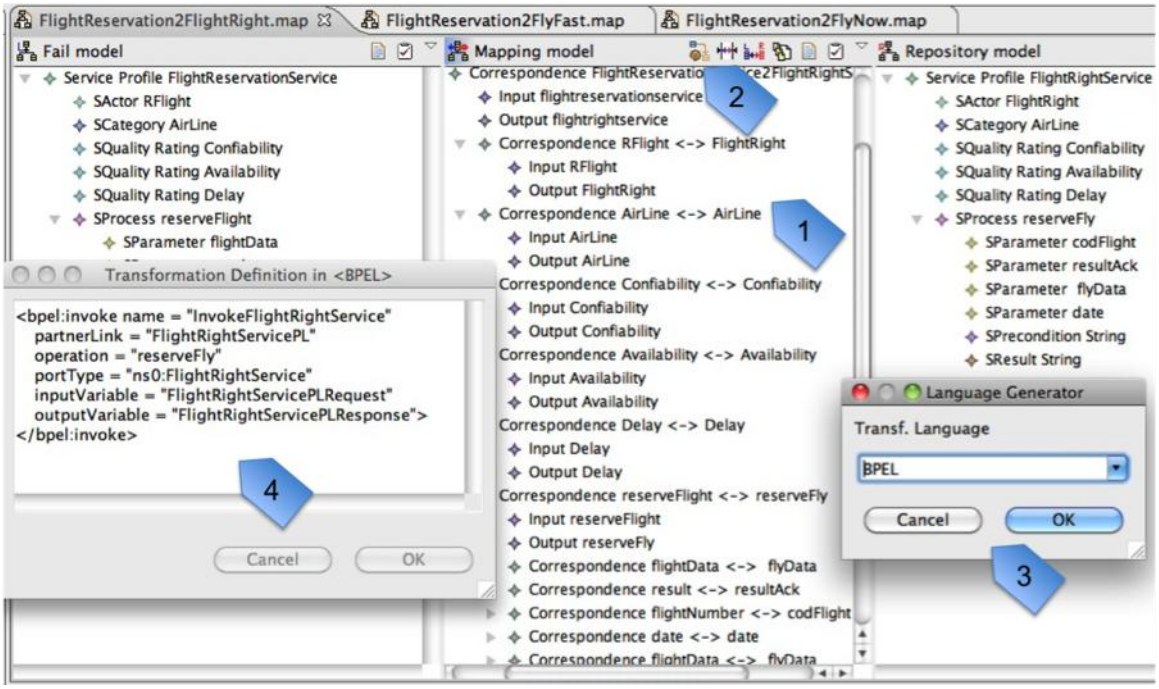


Figura 53 – Funcionalidade de Geração BPEL.

A área de *Mapping Model* apontado pela seta 1 azul, possui o modelo de mapeamento com as devidas correspondências. A partir delas é possível criar uma geração de código, clicando no botão apontado pela seta 2. A seta 3 aponta para a janela que se abre, com a possibilidade de escolher a linguagem resultante, ou seja, BPEL. Finalmente, o fragmento de código BPEL

é gerado capturando os elementos do modelo escolhido *FlightRight*, como aponta a seta 4 da Figura 53.

6.4. Avaliação de Resultados

Uma vez que as etapas de utilização da ferramenta proposta foram apresentadas, incluindo a criação dos modelos, *Matching* de características semânticas e estruturais e geração de fragmento BPEL, então mostra-se na Tabela 7 os valores de cada *Matching*, para fins de comparação.

Modelo Semântico de Serviço Web	*Matching* de Características Semânticas	*Matching* de Características Estruturais	Média das Probabilidades	Pré-condição e Resultado
FlightRight	95,0%	100,0%	0,975 = 97,5%	Classificado
FlyNow	70,0%	66,6%	0,683 = 68,3%	Desclassificado
FlyFast	84,9%	50,0%	0,674 = 67,4%	Classificado

Tabela 7 – Comparativo dos Valores de Matching

Na Tabela 7, vale observar que embora *FlyNow* tenha a média das suas probabilidades igual a 68,3% que é um valor superior a 67,4% de *FlyFast*, o *FlyNow* não pode ser escolhido para substituir o serviço que faltou, devido a sua desclassificação causada pelas regras de pré-condição e critério de saída.

A avaliação do algoritmo de *matching* proposto neste trabalho de dissertação pôde ser realizada através de medidas de qualidade de correspondências (*Match Quality Measures*) [71]. Como ilustra a Figura 54 adaptada de [71], existem inicialmente dois tipos de conjuntos no processo de *matching* (correspondências). O primeiro conjunto possui elementos (ou correspondências necessárias) que pretende-se encontrar, ou seja, correspondências criadas manualmente, todas verdadeiras (**Matches Reais**). Já o segundo conjunto possui correspondências criadas automaticamente pelo algoritmo, ou seja, pode conter correspondências verdadeiras ou falsas (**Matches Automáticos**).

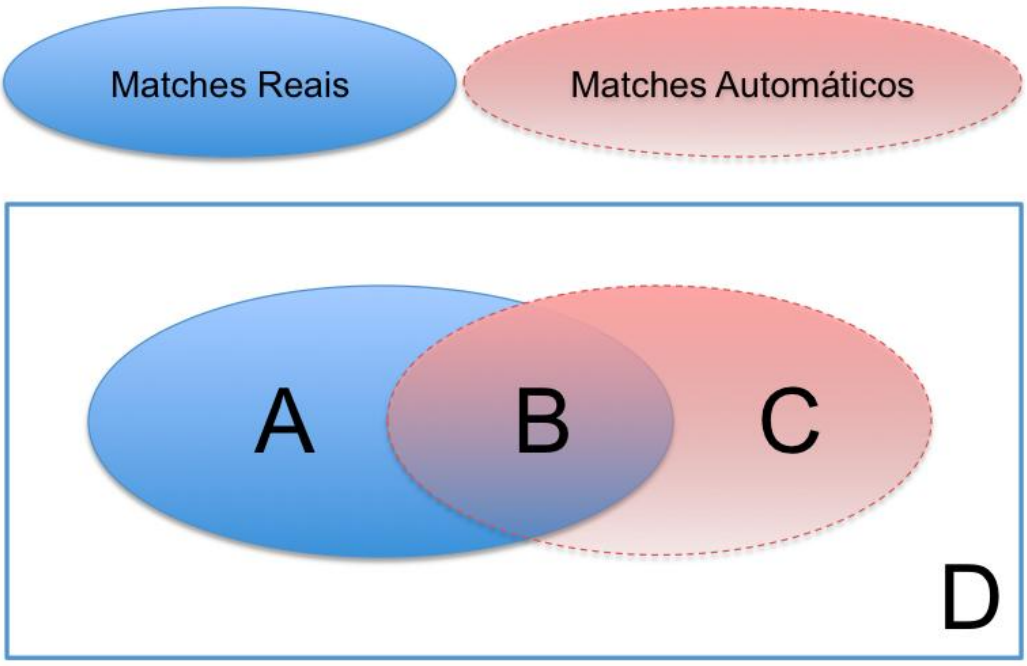


Figura 54 – Matches Reais e Matches Automáticos. Adaptada de [71].

Na hipótese dos **Matches Reais** e dos **Matches Automáticos**, surgem então os conceitos de *Precision, Recall, F-Measure* e *Overall,* que por sua vez são inerentes aos subconjuntos retornados ou não em cada comparação do algoritmo de *matching*. Seguem as definições destes subconjuntos [71]:

- Subconjunto **"A"** (falsos negativos – *false negatives*) são correspondências (*Matches*) necessárias, mas não automaticamente identificadas pelo algoritmo de *matching*;
- Subconjunto **"B"** (verdadeiros positivos – *true positives*) são correspondências que são necessárias e também foram automaticamente e corretamente encontradas pelo algoritmo de *matching*;
- Subconjunto **"C"** (falsos positivos – *falses positives*) são correspondências falsamente propostas como verdadeiras por um algoritmo de *matching* de forma automática;
- Subconjunto **"D"** (verdadeiros negativos – *true negatives*) são correspondências falsas que foram corretamente e automaticamente descartadas pelo algoritmo de *matching*.

Estes subconjuntos são usados para criar medidas de qualidade de *matching* como seguem [71]:

- $\mathrm{Precision} = \frac{|B|}{|B|+|C|}$, reflete a parte de correspondências reais dentre todas as correspondências retornadas;
- $\mathrm{Recall} = \frac{|B|}{|A|+|B|}$, especifica a parte real de correspondências que foram retornadas;
- $F\text{-}Measure = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$, representa a média harmônica entre *Precision* e *Recall*;
- $Overall = \mathrm{Recall} \cdot \left(2 - \frac{1}{\mathrm{Precision}}\right)$, quantifica o esforço necessário após o *matching* para adicionar falsos negativos e remover falsos positivos.

Um valor limite denominado limiar (valor de corte) foi estabelecido para identificar os elementos falsos positivos (conjunto **C**). Os elementos retornados pelo algoritmo que tem o grau de similaridade inferior ao limiar é considerado falso positivo, pois não servem para a composição de serviços. Já os elementos retornados que tem o grau de similaridade superior ao limiar são verdadeiros positivos (conjunto **B**). O valor ideal que define o limiar foi calculado retirando-se a média aritmética dos valores de similaridades de 10 (dez) comparações, resultando em um valor próximo de 65%.

Logo, considerando o limiar nas comparações, tem-se os seguintes resultados, como ilustra a Tabela 8: verdadeiros positivos (conjunto **B**) igual a 58,89%, falsos negativos (conjunto **A**) igual a 34,78% e falsos positivos (conjunto **B**) igual a 6,33%.

Detectados os valores dos subconjuntos **A**, **B** e **C,** tem-se então a base para o cálculo das medidas supra citadas (*precision, recall, f-measure e overall*). A precisão do algoritmo de *matching* que representa a relação entre a parte real (conjunto **B**) dentre todas as correspondências retornadas (**B+C**), teve seu valor ***Precision*** igual a **90,24%**. O *Recall* que representa a relação entre a parte real (conjunto **B**) de correspondências que foram retornadas pelo

algoritmo com todas as correspondências reais existentes (**A+B**), teve seu valor ***Recall*** igual a **62,86%**.

A média harmônica entre os valores de *Precision* e *Recall* que é denominada ***F-Measure*** teve seu valor igual a **74,12%**. Finalmente o *Overall* que representa o esforço para remover falsos positivos (conjunto **C**) e adicionar falsos negativos (conjunto **A**), teve seu valor ***Overall*** de **56,11%**.

A	B	C
2,5	97,5	0
31,7	68,3	0
32,6	67,4	0
76,6	0	23,4
12,8	87,2	0
78,8	0	21,2
13,9	86,1	0
81,3	0	18,7
9,7	90,3	0
7,9	92,1	0

Conjuntos	A	B	C	Média (B+C)
Médias	34,78	58,89	6,33	65,22

Medidas	Precision	Recall	F-Measure	Overall
Valores	90,29%	62,86%	74,12%	56,11%

Tabela 8 – Medidas de Qualidade do Algoritmo de *Matching*.

A lógica do algoritmo de *Matching* implementado em Java, que trata das características semânticas, pôde ser simulado através da lógica *fuzzy*. O objetivo do mapeamento da lógica do algoritmo de *Matching* para a lógica *fuzzy* é apresentar o comportamento do algoritmo de *Matching* que faz o tratamento de Qualidade de Serviços.

Como ilustra a Figura 55, o *Matlab FuzzyLogic ToolBox* possui um visualizador de regras que permite simular valores de entradas para cada variável e obter o resultado de saída.

No exemplo da Figura 55, para as similaridades das variáveis Disponibilidade com valor de 0,614 (61,4%), Confiabilidade com 0,604 (60,4%) e Atraso com 0,396 (39,6%), tem-se uma Qualidade de Serviço 68,2% similar.

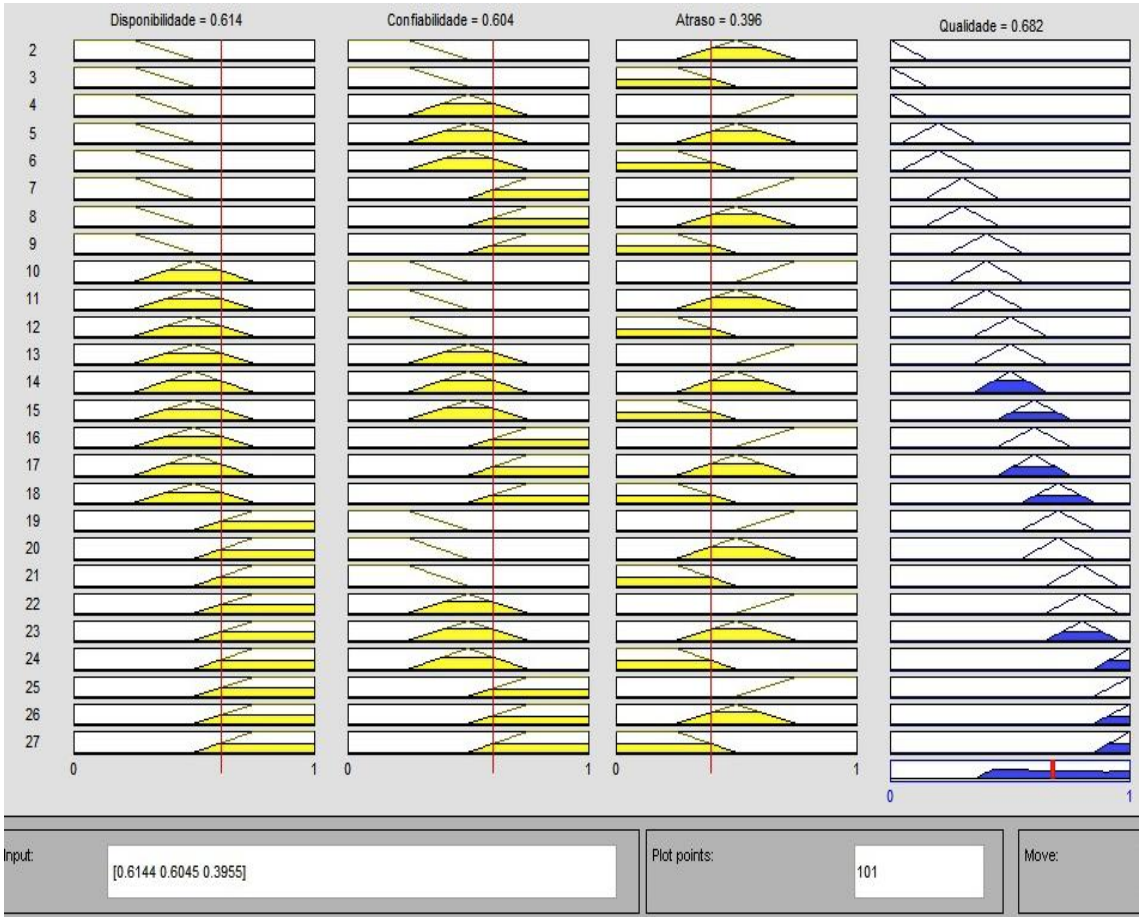


Figura 55 – Simulação dos Tipos e Níveis de Qualidade dos Serviços.

O resultado com todos os possíveis valores atribuídos as variáveis para efetuar simulações do algoritmo, pôde ser representado e visualizado em um gráfico 3D gerado pelo *Matlab Surface Viewer*, como ilustra a Figura 56.

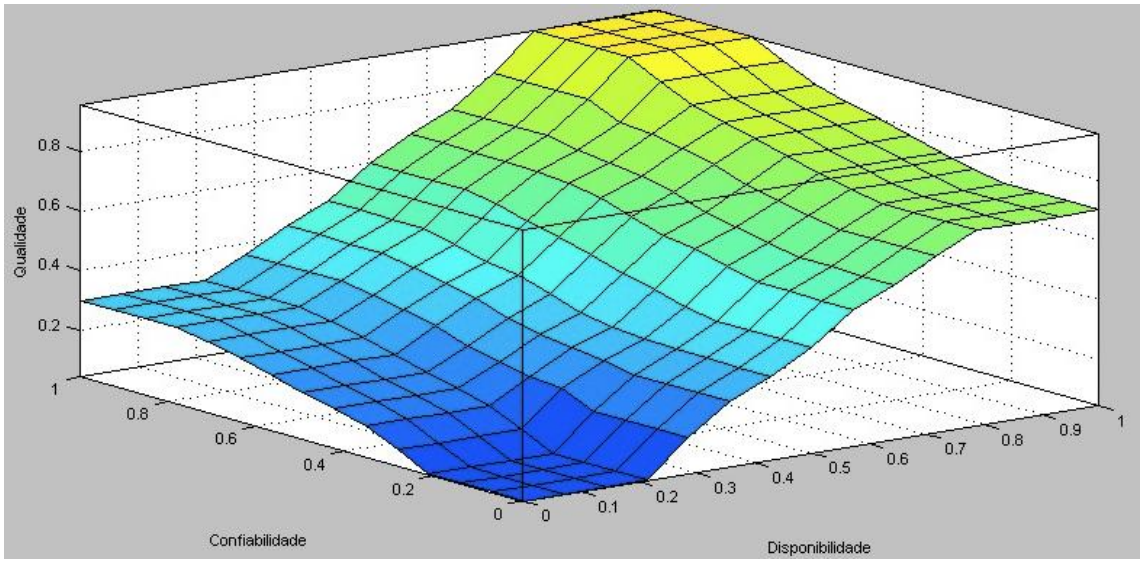


Figura 56 – Relação entre Confiabilidade, Disponibilidade e Qualidade.

O gráfico da Figura 56, representa o comportamento do algoritmo de *Matching*, quando são submetidos diferentes valores para as variáveis Disponibilidade, Confiabilidade e o impacto desses valores no nível de Qualidade. Nota-se que quanto mais próximo de zero são os valores de Confiabilidade e Disponibilidade, mais baixo é o nível de Qualidade de acordo com o eixo vertical do gráfico. A medida que os valores aumentam nos eixos de Confiabilidade e Disponibilidade, observa-se uma progressão no eixo vertical do nível de qualidade.

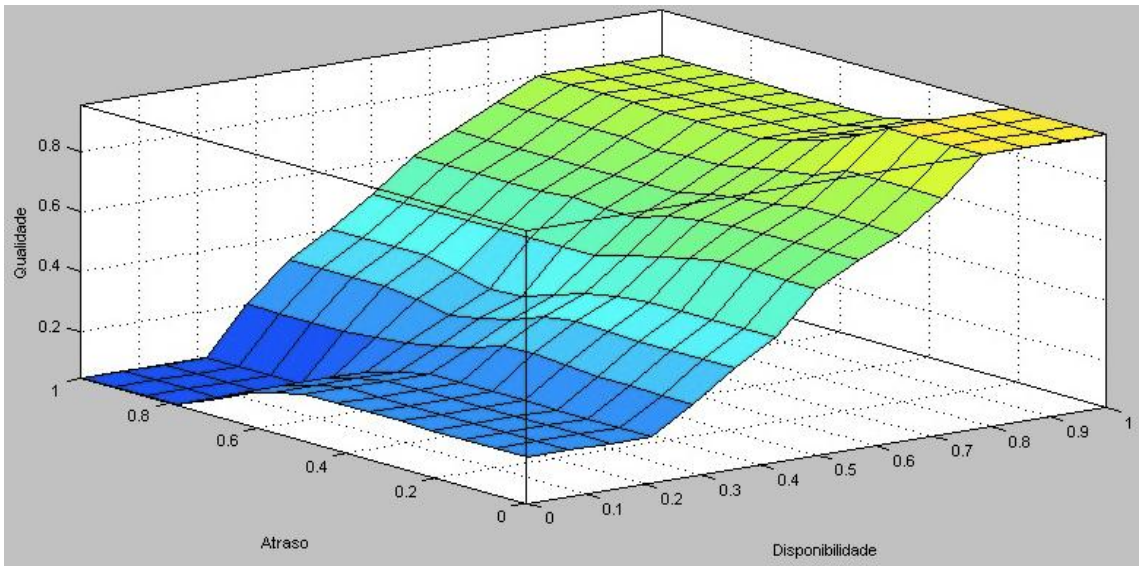


Figura 57 – Relação entre Atraso, Disponibilidade e Qualidade.

O gráfico da Figura 57, representa o comportamento do algoritmo de *Matching*, considerando que o Atraso é inversamente proporcional a Disponibilidade. Ou seja, o nível mais baixo de Qualidade, lado esquerdo da figura, é conseguido quando o valor da Disponibilidade está próximo de zero (0 – significa pouca disponibilidade) e o valor do Atraso está próximo de um (1 – significa tempo de resposta maior). Observa-se que na proporção que os valores no eixo Atraso vai diminuindo o nível de Qualidade aumenta. A medida que os valores no eixo Disponibilidade aumenta, o eixo vertical com nível de Qualidade também aumenta. Dessa forma, quando o Atraso tiver seus valores próximos de zero e a Disponibilidade tiver valores próximos de um, os níveis de Qualidade serão os maiores, como ilustra a Figura 57.

6.5. Síntese

Neste capítulo de Estudo de Caso e Testes, avaliou-se a funcionalidade do framework proposto, através de um estudo de caso inicialmente apresentado e modelado.

A apresentação das ferramentas de Desenvolvimento e *Matching* de modelos semânticos de serviços foi realizada em várias etapas até a geração de um artefato BPEL.

Tais etapas foram seguidas, primeiramente com a criação dos modelos semânticos de serviços web, a partir do protótipo da ferramenta de desenvolvimento de modelos. Depois de pré-classificadas as características semânticas e estruturais, realizou-se os testes de comparações semânticas e estruturais com o protótipo da ferramenta de *Matching*. Quando escolhido o modelo semântico de serviço com maior grau de similaridade, gerou-se o fragmento de código BPEL responsável pela invocação do novo serviço web eleito para integrar a composição.

Finalmente, os resultados das comparações entre os modelos semânticos de serviços foram avaliados. O algoritmo de *Matching*, implementado inicialmente em Java, teve sua lógica mapeada para lógica *fuzzy*, o que permitiu simulações de resultados e geração de gráficos 3D ilustrando seu comportamento.

7. Conclusões e Trabalhos Futuros

Esse capítulo apresenta os objetivos alcançados neste trabalho e suas limitações. Em seguida, sugestões para trabalhos futuros e suas devidas contribuições científicas, tecnológicas e sociais são apresentadas.

7.1. Objetivos Alcançados

Como principal objetivo superado neste trabalho, destaca-se o desenvolvimento de uma abordagem e um *framework* baseados em Engenharia Dirigida por Modelos (MDE) conjuntamente com Ontologias para suportar a Composição Dinâmica de Serviços Web.

Para tal tarefa, alguns resultados ou objetivos específicos foram alcançados, como foi sugerido no início deste trabalho de dissertação:

- Aplicação da Engenharia Dirigida por Modelos e Ontologias para o desenvolvimento e representação semântica de sistemas de softwares baseados em serviços web compostos;
- Criação e utilização de metamodelos baseados em Ontologias para semi-automatizar a geração de modelos de correspondência e criação de modelos semânticos, para suportar a composição dinâmica de serviços;
- Utilização de Ontologias como forma de “descritores” mais precisos, conceitualmente mais ricos para obter informações semânticas e estruturais sobre os serviços envolvidos em uma composição;
- Desenvolvimento de um Algoritmo para a realização do *Matching* (correspondência) de modelos semânticos baseados em Ontologias de Serviços.

Além de atendidos os objetivos especificados no início deste trabalho, alguns critérios destacados entre os trabalhos relacionados são retomados para a análise e comparação com a proposta desta dissertação. Como ilustra a Tabela 9, uma nova coluna referente a este trabalho (fonte e fundo verdes) foi

inserida para fins de comparação com os trabalhos relacionados.

Critérios / Abordagens	Composição Dinâmica de Serviços Web utilizando Abordagens de Engenharia Dirigida por Modelos e Ontologias	Ankolenkar, M. Burstein, J. R. Hobbs, O. Lassila [37], **WS Description for the semantic web**	Chenting Zhao, Zhenhua Duan, Man Zhang [53], **A Model-Driven Approach for Dynamic Web Service Composition**	Il-Woong Kim; Kyong-Ho Lee [38], **A Model-Driven Approach for Describing Semantic Web Services: From UML to OWL-S**	D. Lopes, D. Claro, V. Sena and R. Amorim [54], **Similaridade Semântica na Composição de Sistemas de Informação através dos Serviços Web**	Martins, Rogério Samuel de Moura [55], **Composição Dinâmica de Serviços Web utilizando padrões de projeto**
Geração automatizada de modelos semânticos de serviços.	**Sim**	Não	Sim	Sim. Representados por OWL-S.	Sim	Não
Geração e Exportação em XMI.	**Sim**	Não	Não	Sim	Sim	Não
Grau de Similaridade Semântica entre os Serviços Web	**Sim**	Não	Não	Não	Sim	Não
Construção de serviços e processos de negócio independente da tecnologia da plataforma ou linguagem de descrição de processos de negócio	**Sim**	Não	Sim	Sim	Sim	Não
Seleção Dinâmica de Serviço e capacidade de *binding*.	**Sim**	Não	Sim	Não	Não	Sim
Fácil extensão do modelo de serviços através do seu metamodelo semântico.	**Sim**	**Não** possui um metamodelo	**Não** possui um metamodelo	Não	Não	Não
Estrutura Hierárquica (do serviço mais genérico para o serviço mais específico).	**Não**	Sim	Não	**Sim**. Representados por Diagramas de classe UML	Sim	Não
Estrutura por Classificação, Categoria, Produto, etc.	**Sim**	**Não**. Usa hierarquia	Não	Não	Não	Não
Reutilização em alto nível de processos de negócio em qualquer plataforma.	**Sim**	Não	**Sim.** WS-BPEL	Sim	Sim	Não
Utilização de padrões “de fato” ou consagrados.	**Sim**	Não	**Sim**. UML, WSDL	Sim. OWL-S, UML	Sim	Sim
Redução do custo,	**Sim**	Não	Sim	Sim	Sim	Não

tempo e complexidade de desenvolvimento.						
Utilização de Ontologia em Alto Nível (*Upper Ontology*).	**Sim**	Não	Não	Não	Não	Não
Rápida entrega no desenvolvimento de processos de negócio reutilizáveis.	**Sim**	Não	Sim	**Não**. Modificar Diagramas de Classe e Sequência para um novo processo	**Sim.** De forma gráfica com menus específicos a disposição.	Não
Uso de Modelos convencionais UML expandidos para suportar semântica de serviços web.	**Sim**	Não	Sim	Sim	Não	**Não**. Padrões de Projeto
Desenvolvimento de uma Ontologia do Domínio através de uma DSL.	**Sim**	Não	Não	Não	Não	Não
Utilização de UML para a criação de modelos independentes de plataforma.	**Sim**	Não	Não	Sim	Não	Não

Tabela 9 – Critérios e Comparações com trabalhos relacionados.

Dentre os critérios atendidos fortemente relacionados com a abordagem, o *framework*, os objetivos e as contribuições deste trabalho, destacam-se:

1. A geração automatizada de modelos semânticos de serviços foi conseguida através da ferramenta de criação de modelos semânticos. O desenvolvedor de serviços já tem a sua disposição um modelo semântico já estruturado e apresentado em forma de árvore com as relações entre os seus elementos pré-estabelecidas;
2. Ambos os modelos semânticos e os modelos de correspondências tem suas representações em um formato XMI interpretável por varias ferramentas de metamodelagem;
3. Grau de Similaridade Semântica e Estrutural entre Serviços Web foi conseguido através do algoritmo e ferramenta de *Matching* propostos;
4. Desenvolvimento de serviços e processos de negócio independente da tecnologia da plataforma ou linguagem de descrição de processos de

negócio. Ou seja, tem-se modelos de serviços com informações semânticas e estruturais, que podem ser especificadas para outras linguagens, como por exemplo BPEL;

5. Seleção Dinâmica de Serviço em função do melhor grau de similaridade e a capacidade de *binding* devido a geração do fragmento BPEL responsável pela invocação do novo serviço;
6. Fácil extensão do modelo de serviços através do seu metamodelo semântico. Essa característica implica em conhecimento especialista sobre metamodelagem, porém diminui eficientemente o custo com manutenção de modelos e reduz o tempo de desenvolvimento. Nenhum dos trabalhos relacionados possui um metamodelo semântico de serviços;
7. Os serviços estruturados hierarquicamente, partindo dos serviços mais genéricos para os serviços mais específicos não é atendido por esta abordagem. Porém, é conseguida uma estruturação partindo dos metamodelos que descrevem modelos, incluindo pré-classificação de características genéricas (ex: categoria, produto, etc.);
8. Com a geração de BPEL, tem-se reutilização em alto nível de processos de negócio em qualquer plataforma. E uma rápida entrega no desenvolvimento de processos de negócio reutilizáveis;
9. Um fator importante foi a utilização de padrões “de fato” na internet, por exemplo, padrão BPEL, arquitetura SOA, formato XML, protocolo SOAP, etc.
10. Utilização de Ontologia em Alto Nível (*Upper Ontology*). “Uma ontologia em alto nível (ou superior) é limitada aos conceitos que são meta, genéricos, abstratos e filosóficos, portanto são gerais o suficiente para lidar com uma ampla gama de áreas de domínio (em alto nível). Conceitos específicos de um dado domínio não são incluídos, no entanto ela fornece uma estrutura e um conjunto de conceitos gerais, sob os quais ontologias de um domínio específico (por exemplo, medicina, engenharia, financeiro, etc.) podem ser construídas” [14]. Utilizou-se neste trabalho os conceitos de *OWL-S* para criar um metamodelo semântico de serviços composto em uma DSL;

11. Desenvolveu-se uma Ontologia do Domínio de Serviços através de conceitos e propriedades modelados em uma DSL;
12. Utilizou-se modelos UML convencionais expandidos para suportar semântica de serviços web e utilizou-se UML para a criação de modelos independentes de plataforma.

7.2. Limitações

Alguns pontos neste trabalho apresentam limitações, embora os principais objetivos tenham sido alcançados. Seguem as limitações:

- O *Matching* entre dois modelos semânticos de serviços é realizado em várias interações, através de *wizards*;
- Falta de um serviço de monitoramento e detecção de falta em uma composição de serviços web;
- Geração de código BPEL limitado a um fragmento BPEL;
- Falta de tratamento para todas as especificidades da sintaxe BPEL;
- Integração das ferramentas de criação de modelos e *matching* com um serviço de *deploy* (publicação) automático;
- Todo o processo de busca, *matching*, análise de similaridades, escolha do melhor serviço e geração de BPEL poderiam ser realizados através de um única interação com as ferramentas.

7.3. Trabalhos Futuros

Como sugestões de trabalhos futuros, vale observar:

- Extensão do algoritmo de *Matching* para análise semântica de nomes de serviços ou variáveis, através da busca por

sinônimos em dicionários ou integração com base de dados léxicas, por exemplo *WordNet*.

- Implementação de um serviço de monitoramento e detecção de falta em uma composição de serviços web em tempo de execução. Por exemplo, através do monitoramento dos logs de erros, tags <flow> <faultHandlers>, ou combinados com ferramentas Oracle BAM (*Business Activity Monitoring*) ou Oracle B2B (*Business-to-Business*).
- Extensão do metamodelo semântico de serviços para atender especificações de sintaxe das linguagens BPEL ou WSDL.
- Extensão do metamodelo de *matching* agregando semântica para obtenção de especificações de correspondências precisas e marcação de elementos e sub-elementos;
- Na ferramenta de *Matching*, implementar uma única interação para realizar comparações semânticas e estruturais, seleção do modelo semântico de serviço com maior grau de similaridade e geração do fragmento de código BPEL;
- Implementação da geração de código em outras linguagens de processos de negócio.

7.4. Contribuições

As principais contribuições neste trabalho de pesquisa, além dos objetivos alcançados, puderam ser divididas em três categorias designadas em contribuições científicas, tecnológicas e sociais.

7.4.1. Científicas

As principais contribuições científicas são:

- Proposta de uma abordagem baseada em MDE, Ontologias e na filosofia SOA para projetar sistemas de software robustos implementando-os como uma composição de serviços;
- Proposta de um *framework* baseado em MDE e Ontologias para determinar, utilizando-se um algoritmo de *matching*, similaridades

semânticas e estruturais entre modelos de serviços web, objetivando uma composição dinâmica de serviços;

- Estudo e Avaliação da aplicação da Engenharia Dirigida por Modelos e Ontologias na Composição Dinâmica de Serviços Web.
- Publicação de artigo *Dynamic Web Service Composition with MDE approaches and Ontologies* sobre o presente trabalho, em conferência internacional CISSE 2010 (*International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering*).

7.4.2. Tecnológicas

- Ferramenta de *matching* implementada como plug-in do Eclipse para executar a proposta do algoritmo de *matching* de modelos e ontologias para determinar similaridades sintáticas e semânticas;
- Criação do metamodelo de *matching,* algoritmo de *matching* e geração de modelos de correspondência;
- Criação de metamodelo semântico de serviços e criação dos modelos semânticos de serviços web, a partir do protótipo da ferramenta de desenvolvimento de modelos;
- Ferramenta implementada como plug-in do Eclipse que permite a criação de modelos de serviços com características semânticas e estruturais de serviços web;
- Geração de código em linguagem de processos de negócios BPEL, referente a uma composição de serviços web.

7.4.3. Sociais

- A formação de recursos humanos habilitados na área de Engenharia de Software, especificamente em engenharia dirigida por modelos.

8. Referências bibliográficas

[1] Casati, F., Ilnicki, S., Jin L.; Krishnamoorthy V., Shan, M., **Adaptive and Dynamic Service Composition in eFlow**, Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CaiSE 2000), Stockholm, Sweden, 2000. pp. 13- 31.

[2] Heuvel, W., Yang, J., Papazoglou, M.P., **Service Representation, Discovery, and Composition for E-marketplaces**, Proceedings of the 9th International Conference Cooperative Information Systems (CoopIS 2001), Trento, Italy, 2001, pp. 270-284.

[3] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.J., Nielsen, H. F., "**SOAP Version 1.2 Part 1: Messaging Framework-W3C Recommendation**", 24 June 2003. Available: http://www.w3.org/TR/SOAP/.

[4] Lopes, D., **Étude et applications de l'Approche MDA pour des plateformes de Services Web**, Thèse de Doctorat, Université de Nantes, juillet 2005.

[5] Lopes, D., Hammoudi, S., Bézivin, J. and Jouault, F., ***Generating Transformation Definition from Mapping Specification: Application to Web Service Platform***, The 17th Conference on Advanced Information Systems Engineering (CAiSE'05), LNCS 3520:309–325, June 2005.

[6] Lopes, D., Hammoudi, S., Bézivin, J. and Jouault, F., **Mapping Specification in MDA: From Theory to Practice**. In: Konstantas, D.; Bourrières, J.-P.; Léonard, M.; Boudjlida, N.. (Org.). Interoperability of Enterprise Software and Applications - INTEROP-ESA. : Springer, v. XVI, 2006, pp. 253-264.

[7] Lopes, D., Hammoudi, S., Souza, J. and Bontempo, A., ***Metamodel matching: Experiments and Comparison***, in Proc. of IEEE International Conference on Software Engineering Advances (ICSEA), IEEE Computer Society Press, 2006.

[8] Hammoudi, S., Lopes, D., Vale, S., ***First Steps Towards a Semi-Automatic Transformation Process in MDA***, In: 4èmes Journées sur l´Ingénierie Dirigée par les Modèles, Mulhouse, 2008.

[9] OMG, **Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification - Final Adopted Specification**, Available at http://www.omg.org/docs/ptc/07-07-07.pdf, 2007.

[10] Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Donald F., **Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More**, Publisher: Prentice Hall, Pub. Date: March 22, 2005.

[11] Juric, M., Marcel, K., **WS-BPEL 2.0 for SOA Composite Applications with Oracle SOA Suite 11g,** Publisher: Packt Publishing Ltd., Pub. Date: September 2010.

[12] Alonso, G., Casati, F., Kuno, H., Machiraju, V., **Web Service Concepts, Architecture and Applications,** Publisher: Springer, Pub. Date: 2004.

[13] Gasevic, D., Djuric, D., Devedzic, V., **Model Driven Engineering and Ontology Development,** Publisher: Springer, Pub. Date: 2009.

[14] Cardoso, J., **Semantic Web Services: Theory, Tools, and Applications,** Publisher: Information Science Reference, Pub. Date: 2007

[15] Gruber, T., **A translation approach to portable ontology specifications.** *Knowledge Acquisition*, 5:199-220, 1993.

[16] Lopes, D., Hammoudi, S., Abdelouahab, Z., **Schema Matching in the context of Model Driven Engineering: From Theory to Practice**. In: Sobh, Tarek; Elleithy, Khaled. (Org.). Advances in Systems, Computing Sciences and Software Engineering. : Springer, 2006, pp. 219-227.

[17] Schmidt, D., ***Model-Driven Engineering***, IEEE Computer, February 2006.

[18] Favre, J., ***Towards a Basic Theory to Model Driven Engineering***, UML 2004 – Workshop in Software Model Engineering (WISME 2004), 2004.

[19] Kent, S., ***Model Driven Engineering***, Proceedings of the Third International Conference on Integrated Formal Methods, Lecture Notes In Computer Science, Springer-Verlag, Vol. 2335,pp 286–298, May 2002.

[20] Atlas group and LINA and INRIA, **ATL: Atlas Transformation Language – ATL**, User Manual version 0.7, 2006.

[21] Gavras, A., Belaunde, M., Pires, L. and Almeida, J. P., ***Towards an MDA-based development methodology***, First European Workshop on Software Architecture (EWSA 2004), May 2004.

[22] Hammoudi, S., Alouini, W., Lopes, D., ***Towards a Semi-Automatic Transformation Process in MDA: Architecture and Methodology***. In: 10th International Conference on Enterprise Information Systems - ICEIS, 2008, Barcelona. 10th International Conference on Enterprise Information Systems - ICEIS. Sétubal : ICEIS Press, 2008.

[23] Aiello, M., ***The Role of Web Services at Home***, Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW'06), IEEE Computer press, pp 164-1610, 2006.

[24] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N. and Sycara, K., **OWL-S: Semantic Markup for Web Services.** W3C Member Submission. 22 November 2004. Data de acesso: 15/03/2011, Disponível em: http://www.w3.org/Submission/OWL-S.

[25] Lopes, D., **Introdução a Engenharia Dirigida por Modelos**. I Escola Regional de Computação Ceará Maranhão Piauí (ERCEMAPI), 2007.

[26] Kleppe, A., Warmer, J., Bast, W., **MDA Explained: The Model Driven Architecture: Practice and Promise**, Editora Addison- Wesley, 1º Edição, 2003.

[27] OMG, **Model Driven Architecture (MDA)**, document number ormsc/2001-07-01, July, 2001.

[28] OMG, **MDA Guide,** version 1.0.1, Document Number: omg/2003-06-01. OMG, June 2003.

[29] Budinsky, F., Steinberg, D., Merks, E., Paternostro, M., **Eclipse Modeling Framework**, Second Edition, Addison-Wesley Professional, 2009.

[30] GronBack, R., **Eclipse Modeling Project: A Domain Specific Language (DSL) Toolkit**, Addison-Wesley Professional, 2009.

[31] Bézivin, J., Hammoudi, S., Lopes, D. and Jouault, F., **B2B Applications, BPEL4WS, Web Services and .NET in the Context of MDA**, In: P.Bernus, M.Fox and J.B.M.Goossenaerts. (Org.). Knowledge Sharing in the Integrated

Enterprise - Interoperability Strategies for the Enterprise Architect. Boston: Springer/Kluwer, v. 183, pp. 225-236, 2005.

[32] Sun, H., Wang, X., Zhou, B. and Zou, P. (2003) **Research and Implementation of Dynamic Web Services Composition**, APPT 2003, LNCS 2834, Springer-Verlag Berlin Heidelberg, pp.457–466.

[33] Dustdar, S., Schreiner, W.**, A survey on web services composition,** Int. J. Web and Grid Services, Vol. 1, No. 1, 2005.

[34] Pires, P. F., Benevides, M. R. F. and Mattoso, M. (2003) **Building Reliable Web Services Compositions**, Web Databases and Web Services 2002, LNCS 2593, Springer-Verlag Berlin Heidelberg, pp.59–72.

[35] Hausmann, J. H., Heckel, R. and Lohmann, M. (2003), **Towards automatic selection of web services using graph transformation rules**, in: R. Tolksdorf and R. Eckstein, editors, Berliner XML Tage.

[36] Zhang, R., Arpinar, I. B., Meza, B. A. (2007). **Automatic Composition of Semantic Web Services using Process and Data Mediation**. In: Technical Report, LSDIS lab, University of Georgia, Februa ry 28.

[37] Ankolenkar, M. B., Hobbs, J. R., Lassila, O., et. Al. **WS Description for the semantic web**. The First Intl Semantic Web Conference, 2002.

[38] Kim, Il-W., Lee, K.-H., "**A Model-Driven Approach for Describing Semantic Web Services: From UML to OWL-S**", *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on , vol.39, no.6, pp.637-646, Nov. 2009*

[39] Thone, S., Depke, R. and Engels, G., "**Process-oriented, flexible composition of web services with UML**," in Proc. 3rd Int. Joint Workshop Conceptual Model. Approaches E-Bus.: A Web Service Perspective, 2002, pp. 390–401.

[40] Koch, N., Fraternali, P. and Wirsing, M., "**MDA applied: From sequence diagrams to web service choreography**" in Proc. 4th Int. Conf. Web Eng., 2004, pp. 132–136.

[41] Skogan, D., Grønmo, R. and Solheim, I., "**Web service composition in UML**," in Proc. 8th IEEE Int. Conf. Enterprise Distrib. Object Comput., 2004, pp. 47–57.

[42] Djuric, D., Gasevic, D., Devedzic, V. and Damjanovic, V., "**A UML profile for OWL ontologies**," in Proc. Model Driven Archit.: Found. Appl., 2004, pp. 138–152.

[43] Jaeger, M. C., Engel, L. and Geihs, K., "**A methodology for developing OWL-S descriptions**," in Proc. 1st Int. Conf., Interoperability Enterprise Softw. Appl., 2005, pp. 153–166.

[44] Timmand, J. T. E., Gannod, G. C., "**A model-driven approach for specifying semantic web services**," in Proc. 3rd IEEE Int. Conf. Web Services, 2005, pp. 313–320.

[45] Grønmo, R. and Oldevik, J., "**An empirical study of the UML model transformation tool (UMT)**," presented at the 1st Int. Conf. Interoperability Enterprise Software and Applications, Geneva, Switzerland, 2005.

[46] Martinez, A., Patino-Martinez, M., Jimenez-Peris, R. and Perez-Sorrosal, F., "**ZenFlow:Avisual web service composition tool forBPEL4WS**," in Proc. IEEE Symp. Vis. Languages Human-Centric Comput., 2005, pp. 181–188.

[47] Grønmo, R., Jaeger, M. C. and Hoff, H., "**Transformations between UML and OWL-S**," in Proc. 1st Eur. Conf. Found. Appl., 2005, pp. 269–283.

[48] Xiaofeng, Y., Jim, H., Yan, Z., Tian, Z., Linzhang, W., Jianhua, Z. and Xuandong, L., "**A model driven development framework for enterprise web services**," in Proc. 10th Int. Enterprise Distrib.Object Comput. Conf., 2006, pp. 75–84.

[49] Yang, J. H. and Chung, I. J., "**Automatic generation of service ontology from UML diagrams for semantic web services**," in Proc. 1st Asian Semantic Web Conf., 2006, pp. 523–529.

[50] Timm, J. T. E. and Gannod, G. C., "**Specifying semantic web service compositions using UML and OCL**," in Proc. 5th IEEE Int. Conf. Web Services, 2007, pp. 521–528.

[51] Bensaber, D. A. and Malki, M., "**Development of semantic web services: Model driven approach**," in Proc. 8th Int. Conf. New Technol. Distrib. Syst., 2008, pp. 307–317.

[52] Li, X. , Tang, X., Song, Z., Yuan, X. and Chen, D., "**AFlow: An Automated Web Services Composition System Based on the AI Planning and Workflow**", in: Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on 10-12 Dec. 2010, pp. 1067 – 1071.

[53] Zhao, C., Duan, Z., Zhang, M., "**A Model-Driven Approach for Dynamic Web Service Composition**", wcse, vol. 4, pp. 273-277, World Congress on Software Engineering, 2009.

[54] Sena, V. A., Claro, D. B., Amorim, R. and Lopes, D., "**Similaridade Semântica na Composição de Sistemas de Informação através dos Serviços Web**", In VI Simpósio Brasileiro de Sistemas de Informação (SBSI 2010) Marabá/PA, June 2010.

[55] Martins, R. S. M., "**Composição dinâmica de Web Services**", Dissertação (mestrado), 77p - Universidade do Vale do Rio dos Sinos, Programas de Pós-Graduação em Computação Aplicada, 2007.

[56] Rabelo, R. J., "**BPEL – Business Process Execution Language"**. Disponível em: http://www.das.ufsc.br/~rabelo/Ensino/DAS5316/MaterialDAS5316/PARTE2/BPM/BPEL%20%E2%80%93%20Business%20Process%20Execution%20Language%202009.pdf. Acesso em: 21 set. 2009.

[57] Dietz, J. L. G., Enterprise Ontology: Theory and Methodology, 1 edition, Springer, 2006.

[58] Bordbar, B. and Howells, G. and Evans, M. and Staikopoulos, **Model Transformation from OWL-S to BPEL via SiTra.** Third European Conference on Model Driven Architecture: Foundations and Applications (ECMDA), Haifa, Israel. Published: 2007 June.

[59] OWL-S: OWL Services Coalition (2004), OWL-S: Semantic Markup for Web Services. (2004), Data de acesso: 20/02/2011, Disponível em: http://www.daml.org/services/owl-s/1.1

[60] OMG Unified Modeling Language (UML), version 1.4 (formal/03-03-01), 2002, Data de acesso: 01/04/2011, Disponível em: http://www.omg.org/technology/documents/formal/uml.htm

[61] Pottinger, R. A. and Bernstein, P. A., **Merging Models Based on Given Correspondences**. Proceedings of the 29th VLDB Conference, pp. 826-873, 2003.

[62] Miller, G. A., WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11: pp. 39-41, 1995.

[63] Fellbaum, C., WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press, 1998.

[64] Gomes, S. C. and Monteiro, M. de N. de O., **A Evolução da Teoria da Probabilidade.** Disponível em: http://www.unama.br/Colunas/ServletVerArquivo?idColuna=303, Acesso em: 24 Abril 2011.

[65] Lebensztayn, E. e Coletti, C., **Probabilidade: Teoria e Exercícios**. Programa de Pós-Graduação em Estatística, Departamento de Estatística, Universidade de São Paulo. Disponível em: http://www.ime.usp.br/~seccpg/mae

[66] Weber, L. and Klein T. A. P., **Aplicações da Lógica Fuzzy em Software e Hardware**, Canoas (RS): Ed. Ulbra, 2003.

[67] Barros, A., Dumas, M., Oaks, P., **A Critical Overview of the Web Services Choreography Description Language (WS-CDL).** BPTrends, 2005.

[68] Wolf, O., **First Steps with the Eclipse SOA Platform for Java and SOA Developers.** Eclipse SOA Webcast. Disponível em: http://www.eclipse.org/eclipsesoa/gettingstarted.php

[69] Grupo PET - Engenharia Elétrica. **Curso de MATLAB**. Departamento de Engenharia Elétrica, Universidade Federal de Mato Grosso do Sul. Data de acesso: 28/05/2011, Disponível em: http://www.del.ufms.br/tutoriais/matlab/apresentacao.htm

[70] Booch, G., Rumbaugh, J., Jacobson, I., **UML: Guia do Usuário**, Segunda Edição, Rio de Janeiro - Elsevier, 2005.

[71] Do, H.-H., Melnik, S. and Rahm, E., "**Comparison of Schema Matching Evaluations**," Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems, pp. 221–237, 2003.