Guilherme Coan Hobold

UMA ABORDAGEM AUTOMÁTICA PARA DESCOBERTA E COMPOSIÇÃO DE SERVIÇOS WEB SEMÂNTICOS

Dissertação submetida ao Programa de Pós Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de mestre em Ciência da Computação
Orientador: Prof. Dr. Frank A. Siqueira

Florianópolis
2012

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária
da UFSC.

Hobold, Guilherme Coan
Uma abordagem automática para descoberta e composição de serviços web semânticos [dissertação] / Guilherme Coan Hobold ; orientador, Frank Augusto Siqueira - Florianópolis, SC, 2012.
138 p. ; 21cm

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. Serviços Web Semânticos. 3. SAWSDL. 4. Descoberta. 5. Composição. I. Siqueira, Frank Augusto. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Guilherme Coan Hobold

UMA ABORDAGEM AUTOMÁTICA PARA DESCOBERTA E COMPOSIÇÃO DE SERVIÇOS WEB SEMÂNTICOS

Esta Dissertação foi julgada adequada para obtenção do Título de "mestre" e aprovada em sua forma final pelo Programa de Pós Graduação em Ciência da Computação

Florianópolis, 21 de Setembro de 2012.

_____
Prof. Ronaldo dos Santos Mello, Dr.
Coordenador do Curso

Banca Examinadora:

_____
Prof., Dr. Frank A. Siqueira
Orientador
Universidade Federal de Santa Catarina

_____
Prof., Dr. Cássio Vinicius Serafim Prazeres
Universidade Federal da Bahia

_____
Prof., Dr. Renato Fileto
Universidade Federal de Santa Catarina

_____
Prof., Dr. Ricardo José Rabelo
Universidade Federal de Santa Catarina

Aos meus familiares e amigos, que estiveram ao meu lado durante toda a minha jornada acadêmica.

AGRADECIMENTOS

Tornar-se um profissional de qualidade frente a tantas barreiras que precisam ser transpostas é, além de desafiador, um orgulho imensurável.

Agradeço a Deus por ter me dado pernas para caminhar ao encontro dos meus sonhos e por ter sido um companheiro, ainda que silencioso, durante toda minha jornada. Sou grato por ter olhado a mim e as pessoas que amo.

Pai, mãe e irmão, vocês são mais que fundamentais na minha vida. Com vocês eu tive os maiores e mais significativos aprendizados: ser um homem responsável, centrado, munido de valores admiráveis e seguro para fazer minhas escolhas. O meu amor por vocês é incondicional e supera qualquer vontade própria. Vê-los fortes e tê-los ao meu lado é o suficiente para minha felicidade. Este trabalho só pode ser concluído com o apoio de vocês, mesmo quando me mandavam dormir após horas a frente do computador. Não há nada que juntos não podemos superar. Somos mais do que quatro pessoas, formamos uma força única: uma verdadeira família.

Aos meus amigos, parceiros de todas as horas. Sem eles minha jornada teria sido menos alegre e divertida. Muito do meu aprendizado eu devo a eles, devido as nossas conversas e trocas de ideias. Pessoas que não ficarão para trás e que serão lembradas para sempre.

Aos mestres pela oportunidade de aprender com seus ensinamentos. Ao Frank, orientador e amigo, por ter passando seus conhecimentos durante nossas conversas e reuniões para que eu pudesse realizar a construção desse projeto.

Em especial aos meus amigos André Krummenauer, Gabriel Nunes Menezes, Leonardo Stahelin Coelho, Pedro Galoppini e Rodrigo Lamin. Amigos, sócios, parceiros e irmãos. Poucos têm o privilégio e a oportunidade de conviver com pessoas que compartilham dos mesmos ideais e sonhos. Juntos formamos um time de sucesso chamado Involves Tecnologia.

A persistência é o caminho do êxito.
(Charles Chaplin, 1965)

RESUMO

A adição de semântica à descrição de Serviços Web visa permitir a automação dos processos de descoberta e composição de serviços. Para que isso seja possível, é necessário que sejam desenvolvidas ferramentas que façam com que esses processos sejam realizados dinamicamente, sem a necessidade de intervenção humana. Diversos trabalhos na área de Web Semântica têm explorado a descoberta e a composição de Serviços Web através da descrição semântica das suas funcionalidades. Este trabalho apresenta uma abordagem para a descoberta e a composição de Serviços Web Semânticos de forma automática. Mediante o envio de uma requisição com a descrição das funcionalidades de um serviço, a abordagem proposta possibilita que composições sejam estabelecidas quando um único Serviço Web não é suficiente para atender as necessidades dos usuários. Por meio de anotações semânticas baseadas em SAWSDL (*Semantic Annotations for WSDL*), a abordagem proposta realiza o *matching* semântico das funcionalidades dos serviços web disponíveis em repositórios de serviços com as informações enviadas na requisição e constrói um grafo de composições. No final da construção do grafo, um algoritmo analisa se dois ou mais caminhos levam à mesma informação desejada pelo usuário. Caso isso aconteça, uma função é aplicada para cada caminho, a fim de selecionar aquele com o menor grau de divergência semântica. A proposta da abordagem apresentada é criar composições de serviços web em tempo de requisição de modo que sejam capazes de combinar serviços pré-existentes para criar um novo serviço, tirando proveito das funcionalidades oferecidas por cada um e combinando-as a fim de oferecer funcionalidades ainda mais complexas e até inexistentes. Dessa forma, a abordagem torna o processo de busca e composição de serviços web capaz de informar não apenas serviços individuais, mas também composições de serviços. Como contribuições deste trabalho citam-se: um algoritmo para a seleção de composições baseado na qualidade semântica das mesmas e um protótipo em conjunto com uma infraestrutura para a descoberta e a composição de serviços web.

Palavras-chave: Serviços Web Semânticos. SAWSDL. Composição.

ABSTRACT

The addition of semantic to web services description allows the automation of the discovery and composition process. However, tools must be developed to allow these processes be performed dynamically without human intervention. Several studies in the area of the Semantic Web have explored the discovery and composition of Web Services through semantic description of its functionality. This work presents an approach for automatic discovery and composition of semantic web services. By sending a request with the description of a service functionality, the proposed approach enables compositions to be established when a single web service is not sufficient to meet the user needs. Through semantic annotations based on SAWSDL (Semantic Annotations for WSDL), the proposed approach performs the semantic matching of Web services capabilities available in services repositories with the information sent in the request and builds a composition graph. At the end of the construction of the graph, an algorithm analyzes if two or more paths lead to the same information desired by the user. If this happens, a function is applied to each path in order to select the one with the lowest semantic mismatch degree. The purpose of the presented approach is to create compositions of web services at request time so that they are able to combine pre-existing services to create a new service, taking advantage of the functionality offered by each and combining them to provide functionality yet more complex and even non-existent. Thus, the approach makes the process of discovery and composition of web services return not only individual services, but also compositions of services. The main contributions of this work are: an algorithm for selection of compositions based on its semantics quality, a prototype and an infrastructure for discovery and composition of web services.

Keywords: Semantic Web Service. SAWSDL. Composition.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API: Application Programming Interface
BPEL: Business Process Execution Language
CPU: Central Processing Unit
DOM: Document Object Model
HTML: HyperText Markup Language
IEEE: Institute of Electrical and Electronic Engineers
IDE: Integrated Development Environment
IMA: Interface-Matching Automatic
JDK: Java Development Kit
JUNG: Java Universal Network/Graph Framework
OWL: Web Ontology Language
OWL-S: Web Ontology Language for Services
QoS: Quality of Service
RDF: Resource Description Framework
RDFS: Resource Description Framework Schema
REST: Representational State Transfer
SAWDL: Samantics Annotations for WSDL
SMD: Semantic Mismatch Degree
SMDI: Semantic Mismatch Degree of Inputs
SMDO: Semantic Mismatch Degree of Outputs
SMDP: Semantic Mismatch Degree of Operation
SOA: Service Oriented Architecture
SOAP: Simple Object Access Protocol
SWS: Semantic Web Service
UDDI: Universal Description, Discovery, and Integration
UML: Unified Modeling Language
URI: Uniform Resource Identifier
URL: Uniform Resource Locator
W3C: World Wide Web Consortium
WSDL: Web Services Description Language
WS-BPEL: Web Service Business Process Execution Language
WSML: Web Service Modeling Language
WSMO: Web Service Modeling Ontology
XML: Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A Web, desde a sua origem, evoluiu de um grande repositório de conteúdos, como textos e imagens, para se tornar uma provedora de serviços (MCILRAITH *et al*., 2001). Esses serviços permitem, dentre outras atividades, que pessoas comuniquem-se, criem e divulguem conteúdos na Internet, comprem e vendam produtos, etc. Sob a perspectiva de usuários, estes serviços podem ser consumidos tanto por pessoas, por meio de aplicações e sistemas disponíveis na Internet, como por sistemas de informação, através dos serviços web. A interação entre sistemas, por meio dos serviços web, tem por objetivo intercambiar informações a fim de prover os resultados esperados por quem irá utilizá-los (RODRIGUEZ-MIER *et al*., 2011).

Serviços web são desenvolvidos para serem utilizados por sistemas computacionais e não diretamente por pessoas (PRAZERES, 2009). Eles têm sido empregados com o objetivo de promover a integração entre sistemas distribuídos e possibilitar a construção de canais de informação entre eles (MACHADO *et al*., 2006). No entanto, este processo de integração é executado por desenvolvedores de sistemas de forma *ad hoc*, uma vez que somente os seres humanos são capazes de compreender o significado semântico dos dados e os procedimentos executados por cada serviço (KELLER *et al*., 2006). Em virtude disso, identificar serviços web (descoberta) a fim de utilizá-los em integrações de sistemas (composições) heterogêneos e sujeitos a mudanças constantes é uma tarefa complexa e que exige tempo e dedicação por parte de desenvolvedores (MESMOUDI *et al*., 2011).

Apesar da grande quantidade de serviços web disponíveis nos dias de hoje, é frequente o caso em que dois ou mais serviços precisam ser combinados de modo a satisfazer as necessidades de um sistema (LÉCUÉ *et al*., 2009). As tecnologias atuais empregadas para a descoberta de serviços, no entanto, são limitadas quanto à identificação de possíveis composições de serviços em virtude da forma como eles são descritos. Em outras palavras, as funcionalidades dos serviços web são descritas apenas sintaticamente, o que impossibilita que computadores processem e compreendam as informações inerentes a sua execução (KELLER *et al*., 2006). Consequentemente, as ferramentas de descoberta de serviços limitam-se a analisar apenas palavras-chave, em detrimento do conteúdo semântico dos serviços.

Os Serviços Web Semânticos, que fazem uso de ontologias para a descrição dos serviços, surgem como uma solução para a integração de

sistemas sem a intervenção humana, possibilitando que serviços sejam publicados, descobertos, compostos e executados automaticamente (PRAZERES, 2009). Com a adição de significado à descrição dos serviços web de forma estruturada e padronizada, eles tornam-se auto-descritivos no que tange à forma como devem ser executados e à semântica associada às suas funcionalidades. Logo, sistemas computacionais ganham autonomia para realizarem inferências sobre os dados providos pelos serviços e, consequentemente, automatizarem os processos associados à execução dos mesmos (BERNERS-LEE *et al*., 2001).

1.1 MOTIVAÇÃO

A quantidade de serviços web disponíveis na Internet torna-se cada vez maior à medida que empresas adotam essa tecnologia como forma de proverem seus serviços (ELGAZZAR *et al.*, 2010). Normalmente os serviços web realizam operações e disponibilizam funcionalidades que estão relacionadas à área de atuação dessas empresas. Depois de desenvolvidos, esses serviços são publicados e mantidos em algum dos diversos repositórios disponíveis na web.

Nesse contexto, encontrar um Serviço Web em meio a tantos que execute uma tarefa específica e possua determinadas características é um processo nada trivial (LU *et al.*, 2012). Muitas vezes se perde tempo à procura de serviços que sequer existem. Em outros casos até existem serviços que proveem tal tarefa, mas não possuem as características esperadas. O fato é que a busca por serviços web demanda tempo e esforço de desenvolvimento (AZMEH *et al.*, 2010). Certas situações exigem que serviços sejam combinados a fim de atenderem as necessidades de quem irá consumi-los. E nesse processo, mais tempo e mão de obra são despendidos.

Exemplificando em termos práticos, pode-se imaginar um sistema que permite organizar eventos. Um evento engloba uma série de atividades como a reserva do local, contratação de músicos, da empresa que faz a decoração e da empresa que prepara e organiza a comida que será servida. Se feitas manualmente, quem está organizando o evento deve encontrar quem provê cada um desses serviços e então solicitá-los separadamente. Se a busca for realizada de forma automática, uma ferramenta poderia encontrar e sugerir uma composição de serviços que executasse todas as etapas necessárias. Além disso, a ferramenta poderia

ser inteligente a ponto de identificar pré-condições antes da execução de cada etapa.

São comuns as situações onde composições de serviços web criadas manualmente deixam de funcionar em função de um dos serviços da composição se tornar inexistente (AZMEH *et al.*, 2010). Nesses casos, a composição deixa de ser executada e, consequentemente, de prover seu serviço. Para solucionar o problema, desenvolvedores precisam refazer a composição, de forma a substituir o serviço inexistente por outro adequado. Essa situação exige que sejam feitas alterações na composição, além de demandar tempo e envolver pessoas que tenham o conhecimento necessário para realizar a manutenção na construção da composição (AZMEH *et al.*, 2010).

Trabalhos reportados na literatura propõem abordagens com o objetivo de automatizar os processos de descoberta e composição de serviços web (PAOLUCCI *et al.*, 2002; PRAZERES, 2009; TRAN *et al.*, 2009; ZHANG *et al*., 2003). Além de servirem se base para a abordagem proposta, solucionar limitações apresentadas por alguns desses trabalhos e, consequentemente, proporcionar agilidade e facilidades frente à integração de serviços web serviu de motivação para o desenvolvimento deste trabalho. Além disso, propor uma abordagem que exija o mínimo de alterações nos serviços web existentes a fim de adaptá-los à infraestrutura apresentada e ainda seja passível de ser utilizada em qualquer ambiente de desenvolvimento, independentemente da linguagem de programação empregada, também fez parte da motivação deste trabalho.

1.2 PERGUNTA DE PESQUISA

Esta dissertação visa propor uma abordagem acerca da descoberta e composição de serviços web de modo que contribua para responder a seguinte pergunta:

Serviços web anotados semanticamente com SAWSDL podem ser descobertos e compostos em tempo de requisição sem a intervenção humana e com isso agilizar a execução desses processos?

Nesta pergunta de pesquisa, o SAWSDL é uma linguagem empregada para a descrição semântica de serviços web. Ela provê anotações que são adicionadas ao descritor dos serviços web sem implicar em alterações na estrutura do documento. Em contrapartida,

possui recursos limitados de expressividade no que diz respeito à descrição semântica.

Por agilizar os processos de descoberta e composição entende-se diminuir o tempo despendido por desenvolvedores para encontrar serviços web publicados em vastos repositórios de serviços e retirar a responsabilidade dos desenvolvedores de compor serviços web de forma manual. Pretende-se dessa forma, abstrair a construção da composição de quem irá utilizá-la e tornar esses processos mais transparentes ao usuário, abstendo-o de ter que se preocupar com questões técnicas, como por exemplo, a localização do serviço, suas operações e os tipos de dados trocados entre diferentes serviços.

1.3 OBJETIVOS

1.1.1 Objetivo Geral

Este trabalho visa realizar o estudo e a especificação de uma abordagem para a posterior implementação de um protótipo capaz de descobrir e compor Serviços Web Semânticos de forma automática e em tempo de requisição. O principal intuito do trabalho é retirar a responsabilidade do desenvolvedor de encontrar e compor serviços manualmente. Como resultado, o protótipo deve proporcionar a abstração do processo de descoberta e composição de serviços web, bem como tornar esses processos mais fáceis, sem exigir conhecimentos detalhados a respeito de Serviços Web Semânticos, e rápidos. Para alcançar esse objetivo, os serviços web devem estar descritos semanticamente segundo os padrões especificados para os Serviços Web Semânticos.

1.1.2 Objetivos Específicos

O objetivo geral do trabalho pode ser alcançado através da obtenção de quatro objetivos específicos:

1) Concepção de um modelo para a descoberta e a composição automática de serviços web em tempo de requisição;
2) Especificação de um modelo de requisição para a solicitação de serviços web com base em requisitos funcionais;
3) Definição de um algoritmo para a seleção da melhor composição baseado na qualidade semântica;

4) Implementação de um protótipo para comprovar a viabilidade da abordagem e realização de experimentos para comprovar os benefícios da abordagem proposta;

1.4 DELIMITAÇÃO DE ESCOPO

Não é intenção deste trabalho realizar a execução das composições descobertas. Além disso, o trabalho não contempla como desenvolver serviços web e tampouco como utilizar tecnologias para descrevê-los semanticamente. Requisitos de qualidade de serviços web (QoS) não são abordados neste trabalho e, portanto, não são considerados durante a realização dos processos de descoberta e composição.

1.5 HIPÓTESE DE PESQUISA

Com o intuito de orientar a pesquisa e elaboração deste trabalho, foram definidas quatro hipóteses de pesquisa:

- É possível especificar as funcionalidades requeridas de serviços web através de um modelo de requisição de modo que elas possam ser mapeadas para a descrição dos serviços web disponíveis;
- Através de graus de exatidão resultantes da comparação entre conceitos é possível selecionar a melhor composição de serviços web para atender uma requisição específica com base na qualidade semântica;
- É possível compor serviços web anotados com a linguagem SAWSDL sem intervenção humana;
- É possível descobrir e compor serviços web em tempo de requisição com um tempo de resposta aceitável.

1.6 JUSTIFICATIVA DAS HIPÓTESES

Tecnologias relacionadas à Web Semântica permitem relacionar os elementos que descrevem as funcionalidades dos serviços web com conceitos de ontologias. Sistemas computacionais são capazes de inferir classificações e relacionamentos entre conceitos de ontologias e com

isso interpretar a semântica associada às funcionalidades dos serviços. Caso os conceitos estejam associados às entradas, saídas e operações dos serviços web, é possível que sistemas computacionais estabeleçam a correspondência entre os requisitos funcionais dos serviços web e a requisição por meio do *matching* semântico. Como resultado, tem-se o grau de similaridade entre os conceitos e, a partir deles, torna-se possível compará-los a fim de identificar aquele que possui a menor divergência semântica.

Em função das tecnologias empregadas para descrever a semântica dos serviços web e do poder de inferência atribuído aos sistemas computacionais, a automatização dos processos de descoberta e composição torna-se viável. Contudo, o tempo de resposta para a descoberta ou a composição de serviços pode ser alto dado o fato de o processo ser realizado em tempo de requisição e devido à grande quantidade de serviços web disponíveis. Nesse sentido, pretende-se utilizar recursos, como repositório de composições encontradas e *cache* de ontologias, para minimizar o tempo de resposta.

1.7 ESTRUTURA DA DISSERTAÇÃO

Além do capítulo introdutório, este trabalho é composto de mais cinco capítulos que estão estruturados da seguinte forma:

- O capítulo 2 trata da fundamentação teórica para o entendimento deste trabalho. Ele aborda as principais características e conceitos relacionados à Web Semântica e Serviços Web Semânticos. Dentro desse contexto, são apresentadas e descritas as tecnologias existentes para descrever semanticamente os serviços web.
- O capítulo 3 realiza uma revisão bibliográfica elencando e descrevendo abordagens para a descoberta e composição de serviços web. Ao final, destaca os principais trabalhos relacionados a essas abordagens. Uma comparação entre esses trabalhos e a abordagem proposta neste trabalho é realizada e fim de elencar os pontos fortes e fracos de cada um.
- O capítulo 4 apresenta a abordagem proposta para a descoberta e composição automática e em tempo de requisição de Serviços Web Semânticos.
- O capítulo 5 apresenta o protótipo desenvolvido para avaliar a abordagem. Nessa seção é detalhada toda a arquitetura do

protótipo, bem como o seu funcionamento e as tecnologias envolvidas. Por fim, experimentos e testes realizados demonstram o desempenho e validam a abordagem proposta.

- O capítulo 6 finaliza este trabalho e apresenta as conclusões do trabalho bem como os direcionamentos futuros do projeto.

2 FUNDAMENTOS TECNOLÓGICOS

A web tornou-se o principal meio para a publicação e acesso à informação. Sua escalabilidade e a rapidez na divulgação e acesso às informações permitiram que fosse criado um vasto repositório de conteúdos (KELLER *et al*., 2006). O avanço exponencial na quantidade de recursos informacionais disponíveis na web faz dela uma poderosa fonte de informações em escala global. Distribuídos fisicamente em uma infraestrutura de redes e servidores, esses conteúdos são criados, divulgados, lidos e assimilados por pessoas. A web, na sua concepção, foi projetada pra uso exclusivo dos seres humanos, uma vez que somente eles são capazes de compreender o significado semântico dos dados (BERNERS-LEE *et al*., 2001).

Os computadores não são capazes de compreender a informação contida nos conteúdos disponíveis na web e em contrapartida não oferecem nenhum tipo de apoio no processamento semântico de informações. Apesar das constantes evoluções tecnológicas e do alto poder de processamento dos computadores, eles são incapazes de interpretar o significado dos dados armazenados e exibidos por eles. Duas tendências complementares estão prestes a transformar a web em uma rede "inteligente", que conecta computadores a fim de fornecer suporte para as interações humanas em um nível muito mais elevado do que o disponível atualmente (KELLER *et al*., 2006). Essas tendências surgem como uma proposta para solucionar o caos informacional decorrente da grande quantidade de conteúdos na web. Quando combinadas, permitirão automatizar processos que antes dependiam única e exclusivamente da intervenção humana. Essas duas tendências são a Web Semântica e os Serviços Web Semânticos.

2.1 WEB SEMÂNTICA

Em um futuro próximo, os computadores terão autonomia e poder para "entender" o significado dos dados que hoje são apenas exibidos (BERNERS-LEE *et al*., 2001). A Web Semântica surge nesse contexto com o objetivo de adicionar uma camada semântica à web e assim possibilitar que o significado de dados e informações sejam compreendidos também pelos computadores. A Web Semântica é uma extensão da web atual onde os conteúdos são relacionados a um significado bem definido, estruturado e padronizado a fim de possibilitar

que sejam manipulados por máquinas. Esse relacionamento é o primeiro passo para o desenvolvimento da Web Semântica (RAMALHO, 2006).

Organizar e estruturar a semântica dos conteúdos disponíveis na web é o princípio básico da Web Semântica. Assim, agentes de software serão capazes de realizar tarefas complexas e sofisticadas para os usuários sem que os mesmos precisem intervir no processo (BERNERS-LEE *et al*., 2001). Esse cenário, no entanto, é complexo e requer que os conteúdos publicados na Web Semântica sejam descritos semanticamente. A semântica desses conteúdos deve ser expressa de forma explícita, precisa e de modo que seja acessível. Nesse sentido, o sucesso da Web Semântica depende significativamente dos métodos e formalismos utilizados para a representação do conhecimento. São eles que definem como e de que forma a semântica deve ser expressa sem dar margem a imprecisão e a erros de interpretação. Somente assim, com uma estrutura padronizada os computadores tornam-se capazes de inferir e tomar decisões a partir da semântica dos conteúdos (PRAZERES, 2009).

Sob o ponto de vista da Web Semântica, não só a capacidade de armazenagem e processamento dos computadores é explorada como também a capacidade de análise semântica de dados. Os computadores, que atualmente disponibilizam e exibem dados para os humanos, passam a realizar inferências e estabelecer relações entre diferentes informações. Essas atividades tornam os computadores mais inteligentes e eficientes na realização de tarefas. Um exemplo do que se espera com isso é o processamento semântico de páginas HTML por parte dos navegadores (browsers). Ao invés de apenas exibir uma página HTML, o navegador será capaz de interpretar o significado do conteúdo da página e assim relacioná-lo com outras informações.

Diversos estudos e esforços têm ocorrido no sentido de desenvolver formalismos que possibilitem estruturar e descrever os aspectos semânticos dos recursos disponíveis na web (SHADBOLT *et al*., 2006). Nesse âmbito, tecnologias como a linguagem XML, o padrão RDF e as ontologias são estudadas a fim de suportar esses formalismos e assim possibilitar o desenvolvimento e avanço da Web Semântica.

2.1.1 XML

O XML[1] (*Extensible Markup Language*) é uma meta-linguagem de marcação utilizada para formatar, organizar e estruturar dados. Baseada em *tags*, a sintaxe da linguagem XML é simples e legível por humanos e capaz de ser processada por computadores. Sua flexibilidade permite que sejam criadas *tags* diferentes para representar estruturas variadas. Não existe nenhuma limitação quanto à quantidade ou ao número de *tags* que podem ser criadas. Assim, documentos baseados em XML podem ter suas próprias *tags* e adequá-las ao seu contexto. Fica a cargo de quem cria os documentos XML definir as *tags* que serão utilizadas e a estrutura delas.

No contexto da Web Semântica, a linguagem XML é adotada como padrão para a estruturação do conteúdo dos recursos e para o intercâmbio de informações. Isso se deve ao fato de ser uma linguagem de padrão aberto e de fácil processamento por parte dos computadores. Contudo, a linguagem XML permite estruturar dados apenas de forma sintática e é incapaz de expressar o significado deles (BERNERS-LEE *et al*., 2001).

2.1.2 RDF

Para suprir algumas limitações do XML, o padrão RDF[2] (*Resource Description Framework*) surge como um modelo, baseado na sintaxe de XML e padronizado pelo W3C (*World Wide Web Consortium*), para descrever os recursos disponíveis na web. Esses recursos podem ser desde abstrações de conceitos, como carro e pessoa, até conteúdos como documentos textuais, vídeos, imagens e áudios. Tudo o que está na web e pode ser identificado é considerado um recurso. Recursos na web são identificados por meio de URIs (*Uniform Resource Identifier*). Uma URI é um identificador único e formado por uma cadeia de caracteres que especifica o nome do recurso, a sua localização e como acessá-lo. Por meio delas, qualquer recurso pode ser identificado e localizado na web. A sintaxe http://example.org/exemplo/de/URI/resource.txt é o exemplo de uma URI.

[1] http://www.w3.org/XML/
[2] http://www.w3.org/RDF/

Sob o ponto de vista de utilização do RDF, os recursos não precisam ser alterados para que possam ser descritos. Cada recurso tem o seu descritor RDF que é um documento ou arquivo separado do recurso ao qual ele se refere. Isso caracteriza o próprio descritor RDF como um recurso e por isso podem existir descritores que descrevem descritores (RAMALHO, 2006). O descritor RDF possui informações ou metadados a respeito de um recurso. Ele é baseado em afirmações onde são definidas propriedades e a elas associados valores que descrevem o recurso. O RDF é semelhante a um diagrama de classes, do paradigma de orientação a objetos, onde as classes equivalem aos recursos, os atributos de classe às propriedades do recurso e o valor dos atributos ao valor das propriedades. A fim de exemplificar, uma página HTML pode ser considerada um recurso e ter como propriedade o criador da página, cujo valor é o nome do desenvolvedor. Outra propriedade de uma página HTML pode ser a data de publicação e o seu valor uma data qualquer.

As afirmações utilizadas para descrever um recurso formam uma tripla composta por um sujeito, um predicado e um objeto. O sujeito é o recurso a ser descrito, o predicado é uma propriedade ou característica desse recurso e o objeto é o valor do predicado (BERNERS-LEE *et al*., 2001). O sujeito e o predicado referenciam um recurso e uma propriedade, respectivamente, por meio de URIs. O objeto pode ou não ser uma URI e isso depende da propriedade em questão. Com base no exemplo da página HTML, o endereço da página é o sujeito, o predicado é criador e o valor o nome do criador. Uma propriedade pode ser reutilizada por diferentes recursos, uma vez que basta referenciá-la por meio da sua URI. Outro detalhe importante é que uma URI utilizada como objeto em um descritor pode ser utilizada em outro como sujeito (RAMALHO, 2006). Um exemplo disso é um email, que hora pode ser utilizado como valor de uma propriedade e hora como recurso a ser descrito. Um descritor RDF, portanto, pode ser composto por uma série de declarações a respeito do recurso que descreve.

O padrão RDF possibilita descrever recursos e o relacionamento entre eles, porém limita-se a descrevê-los individualmente e de forma sintática. Não é possível expressar o contexto de utilização desses recursos e nem associá-los a domínios de aplicação (PRAZERES, 2009). O RDF *Schema*[3], ou simplesmente RDFS, é uma extensão do RDF que dispõe de mecanismos para criar relações semânticas entre recursos. Ele adiciona características ao RDF

[3] http://www.w3.org/TR/rdf-schema/

que permitem relacionar recursos e propriedades a conceitos como o de classes. Isso quer dizer que é possível agrupá-los em categorias e relacioná-los a conceitos do domínio da aplicação.

O RDFS auxilia na criação de vocabulários e com isso possibilita que cada domínio de aplicação tenha seu próprio vocabulário. Aplicações, em consenso com a semântica desses vocabulários, são capazes de interpretar as afirmativas feitas acerca dos recursos de forma precisa. Assim é possível agregar certo grau de semântica aos recursos descritos com RDFS. Apesar de o RDFS permitir que ontologias simples sejam especificadas, sua expressividade é limitada. Isso porque não provê recursos como o de cardinalidade das propriedades, conectivos lógicos de negação, disjunção e conjunção (RAMALHO, 2006).

Uma ontologia é um modelo lógico e formal de dados que representa um domínio de interesse através de conceitos e relacionamentos entre esses conceitos. Em outras palavras, uma ontologia serve para representar um conhecimento específico sobre o mundo, ou uma parte dele, descrevendo-o através de classes, atributos e relacionamentos (VALLET *et al*., 2006). Por meio de ontologias é possível expressar a hierarquia ente conceitos, as relações entre eles, restrições de integridade e instâncias de conceitos.

2.1.3 Ontologias

Segundo a definição de (CORCHO *et al*., 2003), uma ontologia é composta por cinco elementos que são: conceito, relacionamento, função, axioma e instância. Os conceitos são representações de classes com suas respectivas propriedades. Pessoa é um exemplo de conceito e nome uma propriedade desse conceito. Em muitos casos um conceito está atrelado a outro, seja por meio de associação, composição, generalização ou especialização. Essa interação entre conceitos ocorre por meio do relacionamento entre eles. A função é um tipo específico de relacionamento que necessita de uma ou mais variáveis. Por meio de axiomas é possível fazer afirmações acerca de um conceito. Eles servem para explicitar uma informação e assim facilitar a dedução de novos conhecimentos. Por fim, as instâncias representam os indivíduos pertencentes a uma determinada classe ou conceito. Uma Ferrari é um exemplo de instância do conceito Carro.

Quando se descreve um recurso apenas sintaticamente por meio de palavras-chave, por exemplo, a ambiguidade torna-se um problema e impede que a semântica seja expressa de forma apropriada e com

precisão. Isso porque um mesmo termo pode sofrer variação de significado, dependendo do contexto no qual está inserido (BERNERS-LEE *et al*., 2001). Um exemplo disso é o termo "São Paulo", que pode ter o seu significado associado a um time de futebol, a um estado, cidade ou ainda um santo. Por meio de ontologias é possível solucionar o problema de ambiguidade, inerente às linguagens naturais, de forma lógica. Através delas é possível distinguir os diferentes significados por meio dos relacionamentos entre conceitos.

Segundo (CORCHO *et al*., 2003), as ontologias podem ser classificadas em ontologias de domínio e ontologias independentes de domínio. O primeiro tipo engloba as ontologias que tratam de assuntos específicos, com conceitos restritos a alguma área do conhecimento como, por exemplo, informática, medicina e biologia. O segundo tipo de ontologias engloba aquelas que discorrem de assuntos genéricos e aplicáveis em diversas áreas do conhecimento como, por exemplo, o tempo (PRAZERES, 2009). A Figura 1 apresenta um exemplo de ontologia de domínio sobre veículos.

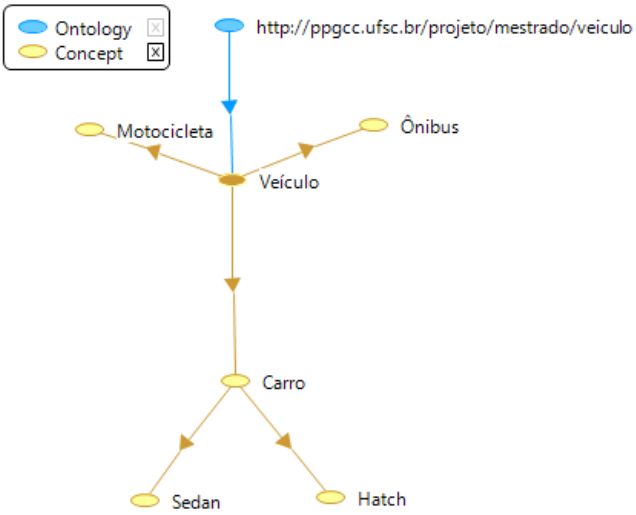

Figura 1 – Exemplo de ontologia de domínio de veículos adaptada de (PRAZERES, 2009)

Uma ontologia pode ainda ser classificada quanto ao seu formalismo em: informal, semi-informal, semi-formal ou formal (BALANCIERI, 2010). No primeiro tipo as ontologias são expressas em

linguagem natural, sem nenhuma preocupação com regras e estruturação. As semi-informais ainda utilizam linguagem natural, porém de forma estruturada. As semi-formais e formais já utilizam linguagens artificiais de forma estruturada, porém nas formais os termos são bem definidos com semântica formal, e teoremas e validações são utilizados.

No contexto da Web Semântica, uma ontologia é representada por um documento ou um arquivo estruturado composto por axiomas e relacionamentos entre eles (BERNERS-LEE *et al*., 2001). Esses arquivos são criados por meio de linguagens semi-formais e formais para criar ontologias complexas. Dentre as linguagens de ontologias existentes, a OWL (*Web Ontology Language*) tem sido a recomendada desde 2004 pelo W3C para a especificação de ontologias (RAMALHO, 2006).

A OWL surgiu de uma união entre duas iniciativas, DAML+ONT e OIL, para solucionar situações em que só o uso do RDFS não era suficiente. Logo foi padronizada e adotada pelo W3C para integrar o grupo de padrões e tecnologias que compõem a Web Semântica. A linguagem OWL não substitui os padrões RDF e RDFS e sim os complementa. Ela utiliza a base de definições fornecidas por esses padrões e adiciona novas primitivas que permitem aumentar o poder de expressividade das ontologias (PRAZERES, 2009). Se comparada ao RDFS, a OWL aumenta a capacidade de expressão bem como a capacidade de realizar inferências e raciocínio sobre as ontologias.

O formalismo da linguagem OWL permite que técnicas de inferência extraiam conhecimentos expressos de forma implícita. O poder de inferência é diretamente proporcional à expressividade da linguagem utilizada para especificar a ontologia. Por meio do processo de raciocínio, empregado para deduzir novos conhecimentos, é possível assegurar a qualidade de uma ontologia, visto que permite testar se existe contradição entre conceitos, se a estrutura hierárquica dos conceitos está consistente, etc. (BALANCIERI, 2010). Artefatos de software que oferecem mecanismos de raciocínio e inferência para ontologias são conhecidos como máquinas ou motores de inferência. Eles podem ser utilizados por outras aplicações que desejam manipular ontologias.

```
<owl:Class rdf:ID=“Veiculo"/>
<owl:Class rdf:ID=“Carro">
    <rdfs:label>Carro</rdfs:label>
    <rdfs:subClassOf rdf:resource=”#Veiculo” />
</owl:Class>
```

Figura 2 – Exemplo de declaração de classes e subclasses. Adaptado de (RAMALHO, 2006)

Por meio de ontologias OWL é possível criar tanto ontologias de domínio como ontologias independentes de domínio. Ele disponibiliza uma terminologia que permite criar representações de classes, propriedades e subclasses. Consequentemente é possível realizar operações de generalização, especialização, composição e associação de conceitos. Na Figura 2 é possível visualizar a declaração da classe Veículo e também da sua subclasse Carro com OWL. Apesar de a linguagem OWL fornecer a base para a construção de ontologias, não existe nenhuma metodologia ou abordagem que guie esse processo. Qualquer tipo de conhecimento pode ser representado e a sua modelagem depende de quem as constrói. O problema relacionado à criação de ontologias está no fato de que diferentes grupos de pessoas possuem diferentes visões acerca do mesmo domínio ou área do conhecimento. Isso faz com que surjam ontologias distintas para representar o mesmo conhecimento.

As ontologias criadas em OWL podem ser representadas em arquivos que utilizam a sintaxe XML e RDF. Elas são disponibilizadas na web como um recurso que pode ser identificado e localizado por meio de uma URI. Assim, descritores de conteúdos podem referenciar os conceitos de uma ontologia desde que sua URI seja conhecida.

2.1.2 Estrutura da Web Semântica

A combinação dos padrões e tecnologias apresentados nas seções anteriores, bem como a forma como eles estão distribuídos, constitui a infraestrutura da Web Semântica. Essa infraestrutura forma a base para que sejam desenvolvidas aplicações inteligentes e que explorem os recursos da Web Semântica. Ela foi proposta em 2000 pelo W3C e é dividida em camadas sobrepostas que visam especificar os

padrões adotados e a forma como eles se relacionam (KOIVUNEN *et al.*, 2001). Na Figura 3 é possível visualizar as cinco camadas que compõem a estrutura da Web Semântica.

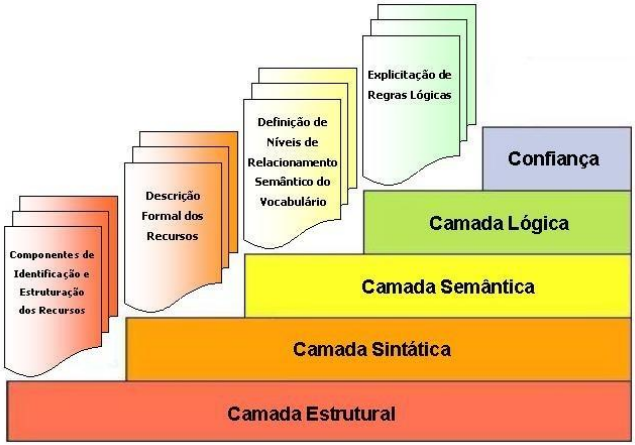

Figura 3 – Estrutura da Web Semântica (RAMALHO, 2006).

Todas as camadas possuem funções específicas que possibilitam o desenvolvimento da Web Semântica. Uma camada complementa a outra com a adição de novos padrões e tecnologias. Isso significa que as camadas são compatíveis e uma depende das outra.

A Camada Estrutural constitui a base da infraestrutura e é o alicerce das demais. Sua função é permitir que os recursos da web sejam identificados de maneira única e assegurar a confidencialidade e representação de dados. Para isso, essa camada utiliza o padrão de codificação Unicode para manipular e representar dados e as URIs para identificar os recursos.

A Camada Sintática fornece mecanismos para validar e definir regras sintáticas de maneira formal. Essas regras asseguram a integridade dos recursos por meio da utilização de XML, XML *Schema* e *Namespaces*[4]. Esses padrões permitem descrever e restringir o conteúdo dos recursos.

A Camada Semântica dispõe das funcionalidades necessárias para descrever os aspectos semânticos dos recursos. Nessa camada encontram-se as ontologias empregadas para representar conhecimentos específicos acerca de uma determinada área. Elas permitem que o

[4] http://www.w3.org/TR/REC-xml-names/

conhecimento seja explicitado por meio dos padrões RDF/RDFS e OWL.

A Camada Lógica permite que princípios lógicos possam ser entendidos por computadores. Por meio do relacionamento entre regras e ontologias, o poder de inferência dos computadores aumenta e com isso cresce a expressividade da semântica dos recursos.

A Camada de Confiança foi projetada para avaliar a confiabilidade das informações providas na Web Semântica. Ela é quem verifica a proveniência das informações e se estão descritas corretamente. Tecnologias associadas à assinatura digital são utilizadas nessa camada passa assegurar a credibilidade e validade dos recursos.

As camadas que constituem a estrutura da Web Semântica utilizam uma série e padrões determinados pelo W3C e vistos na seção 2.1.1. Esses padrões juntamente com a infraestrutura da Web Semântica possibilitam que agentes de software explorem a semântica inerente aos recursos disponíveis na web e com isso realizem processos de forma automática. O real poder da Web Semântica só será sentido quando uma gama de agentes de software inteligentes explorar o conteúdo semântico dos recursos, processar essas informações e compartilhar os resultados com outros agentes (BERNERS-LEE, 2001). Um exemplo de agente de software inteligente são os Serviços Web Semânticos.

2.2 SERVIÇOS WEB SEMÂNTICOS

Antes de entrar em detalhes sobre os Serviços Web Semânticos, faz-se necessário apresentar uma distinção entre serviços e serviços web a fim de definir os pressupostos subjacentes.

A noção de serviço, segundo (PREIST, 2004), como um provedor de valor em determinado domínio é facilmente confundida com a concepção de Serviço Web. Um serviço será tratado neste trabalho como uma tarefa responsável por prover algum valor, seja um número, uma confirmação ou qualquer outro dado, em resposta a sua execução. De acordo com o tipo de resposta e atividade realizada, o serviço é associado a um domínio específico, correspondente ao grupo de serviços com características e funcionalidades correlatas (KELLER *et al*., 2006). Serviços para obtenção de informações sobre o clima ou ainda serviços para consulta de produtos em uma loja são exemplos que se encaixam na definição de serviço.

No processo de execução de um serviço existe o provedor do serviço, responsável por executá-lo, e o solicitante do serviço,

interessado na obtenção da resposta ou resultado. Para executar o serviço, o provedor necessita determinadas informações sobre o solicitante. Essas informações, chamadas de entradas ou *inputs*, são necessárias para que o serviço seja executado adequadamente. Por exemplo, uma compra online necessita do número de cartão de crédito do comprador como entrada.

O significado de serviço pode ser ampliado e depende do contexto em que se encontra. Se for um serviço prestado por meio de um artefato de software na web, ele pode ser chamado de Serviço Web (KELLER *et al*., 2006). A definição de (CURBERA *et al*., 2003), descreve um Serviço Web como uma aplicação modular, auto-descritiva e auto-contida que pode ser executado pela web. Os serviços web disponibilizam uma interface que permite que clientes acessem e executem o serviço. Essa execução só é possível porque na interface do Serviço Web estão descritas as funcionalidades oferecidas por ele, bem como as entradas necessárias à execução de cada funcionalidade e o resultado delas.

A interface dos serviços web descreve, portanto, o que eles esperam do ambiente. Em um Serviço Web de divisão, por exemplo, espera-se que a entrada X seja um número real e a entrada Y um real diferente de zero. Essa condição chama-se suposição de entrada. O resultado provido pelo serviço em função das entradas será também um número real. Essa afirmativa provê a garantia da saída. A interface não trata de como a divisão ou o processo é realizado, apenas assegura que se as condições de entrada forem obedecidas o serviço produzirá o resultado esperado (ALFARO e HENZINGER, 2001).

Padronizado pelo W3C, a linguagem WSDL (*Web Service Description Language*) é atualmente recomendada e a mais utilizada para descrever a interface dos serviços web (CHINNICI *et al*., 2007). O WSDL é uma linguagem de descrição baseada em XML. Por meio de um conjunto de *tags* pertencentes ao vocabulário do WSDL, é possível descrever de forma sintática quais funcionalidades o Serviço Web oferece e como invocá-las. Essa descrição é exposta em um documento WSDL.

Os serviços web são definidos e descritos por meio de seis elementos disponíveis no documento WSDL: *types*, *interface*, *binding*, *endpoint* e *service*. No elemento *types* do WSDL estão representadas as estruturas de dados utilizadas na troca de mensagens. O elemento *interface* define as operações oferecidas pelo Serviço Web. Cada operação pode estar associada a uma entrada e uma saída que por sua vez estão associadas às estruturas de dados. O elemento *binding*

especifica o protocolo e o formato de dados necessários para a execução das operações. O elemento *endpoint* define o endereço de conexão do Serviço Web e por fim, o elemento *service* especifica um ou mais endereços de acesso ao Serviço Web.

A troca de mensagens entre o Serviço Web e o cliente, que invoca o serviço, pode ocorrer via protocolos como REST (*Representational State Transfer*) (PAUTASSO *et al.*, 2008) e SOAP[5] (*Simple Object Access Protocol*). O protocolo SOAP bem como a linguagem WSDL são baseadas em XML, o que torna os serviços web independentes de plataforma e linguagem de programação. Já Serviços Web baseados em REST utilizam requisições HTTP para a troca de mensagens, sendo que cliente e servidores HTTP estão disponíveis em boa parte das linguagens e sistemas operacionais (PAUTASSO *et al.*, 2008).

Diferentemente da maioria das aplicações, os serviços web são desenvolvidos para serem utilizados por outras aplicações e não por usuários finais. Eles têm sido, devido a sua estrutura e autonomia, amplamente empregados na integração entre sistemas distribuídos (MACHADO *et al*., 2006). O intercâmbio de dados é feito entre esses sistemas via canais de comunicação criados pelos serviços web, permitindo a troca de mensagens entre sistemas. Esse relacionamento entre sistemas distribuídos por meio de serviços web é sustentado por uma arquitetura orientada a serviços, ou simplesmente SOA (*Service-Oriented Architecture*). Esse modelo de arquitetura proporciona que sistemas distribuídos, pertencentes a qualquer domínio de negócio, se comuniquem por meio dos serviços web (LI *et al.*, 2009).

O Serviço Web e a aplicação que o invoca não precisam ser desenvolvidos com a mesma linguagem ou plataforma de desenvolvimento. Basta que o cliente tenha conhecimento da localização e acesso à interface com a descrição do serviço para que ele possa ser executado. Em outras palavras, o cliente precisa ter acesso ao documento WSDL. Apesar de o WSDL ser suficiente para descrever as funcionalidades dos serviços web, ele não permite expressar a semântica dos dados e informações inerentes à execução do serviço. Esta limitação de expressividade levou a estudos e pesquisas na área de Serviços Web Semânticos (SWS).

Os Serviços Web Semânticos são Serviços Web enriquecidos através de descrições semânticas (MARTIN *et al*., 2007). Eles, que fazem uso de ontologias para a descrição dos serviços, surgem como uma solução para a limitação de expressividade do WSDL. A proposta

[5] http://www.w3.org/TR/soap/

dos Serviços Web Semânticos é adicionar semântica explícita ao descritor dos serviços web por meio de tecnologias da web semântica. Automatizar processos como o de descoberta, composição e execução de serviços é uma das motivações por trás do uso dos Serviços Web Semânticos. Os computadores passam a ser capazes de realizar esses processos de forma automática, uma vez que os Serviços Web Semânticos têm suas propriedades e interface descritas sem ambiguidade empregando uma representação tratável computacionalmente (FONSECA, 2008).

Os Serviços Web Semânticos aliam características de diferentes tecnologias disponíveis na web. Eles agregam a expressividade na representação dos serviços por meio de tecnologias da Web Semântica com a dinamicidade proveniente dos serviços web. A Figura 4 demonstra as tecnologias e tendências que cooperam para o desenvolvimento dos Serviços Web Semânticos.

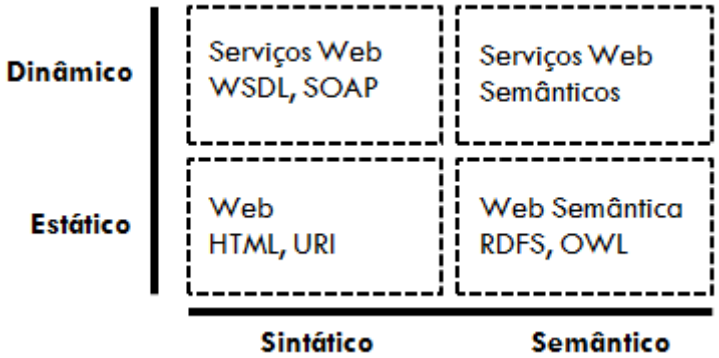

Figura 4 – Características dos Serviços Web Semânticos

O nível de abstração de um Serviço Web indica o grau de detalhamento empregado na sua descrição (KELLER *et al*., 2006). Conforme ilustrado na Figura 5, quanto mais detalhadas forem as funcionalidades dos serviços, maior a sua expressividade e possibilidade de automatização de processos. Utilizando apenas WSDL, é possível descrever as capacidades do serviço somente de forma sintática. Descendo um nível em detalhamento, já é possível descrever a semântica dos serviços em termos de propriedades, entradas e resultados. Assim os serviços são descritos como unidades isoladas que possuem entradas e saídas anotadas semanticamente. No último e mais detalhado nível, é possível descrever as pré-condições para a execução

do serviço bem como os efeitos decorrentes pela sua execução. As pré-condições representam as condições que devem ser satisfeitas para a execução do serviço e os efeitos as condições que serão satisfeitas com a execução. Isso possibilita determinar quando um serviço deve ser executado durante uma sequencia de execução de serviços.

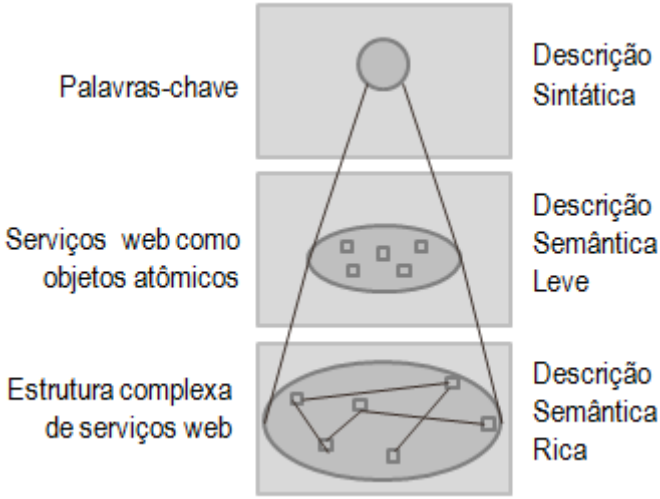

Figura 5 – Nível de abstração dos serviços web. Adaptado de (KELLER *et al.*, 2006)

Os Serviços Web Semânticos são capazes de oferecer um novo tipo de automação e integração de sistemas, onde uma cooperação totalmente flexível, aberta, em tempo real e com o mínimo de programação pode ser atingida. Para isso, padrões como OWL-S (*Web Ontology Language for Web Services*) (ANKOLEKAR, 2003), WSMO (*Web Service Modeling Ontology*) (ROMAN *et al.*, 2005) e SAWSDL (*Semantic Annotations for WSDL*) permitem que os componentes dos serviços web sejam relacionados a conceitos de ontologias. Neste trabalho foi adotado o SAWSDL, que por isso será detalhado na próxima seção.

2.2.1 SAWSDL

O SAWSDL é uma extensão do WSDL e a mais recente recomendação da W3C para descrição semântica de serviços web (KOPECKÝ *et al.*, 2007). Através dele é possível expressar a semântica

dos serviços web por meio de anotações. Essas anotações são adicionadas aos elementos que compõem o documento WSDL (operações, entradas, saídas, etc.) de modo a relacioná-los com conceitos de ontologias de domínio. A Figura 6 demonstra a relação entre os elementos e as ontologias de domínio. Assim, o SAWSDL propõe um formato padrão para a descrição de Serviços Web Semânticos. Os elementos do WSDL anotado não precisam estar relacionados com conceitos de uma mesma ontologia. Em outras palavras, diferentes ontologias podem ser utilizadas para anotar elementos de um WSDL (KOPECKÝ *et al*., 2007).

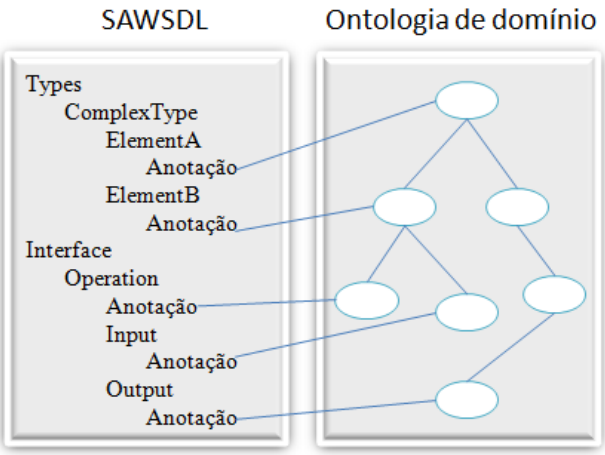

Figura 6 – Relação entre o SAWSDL e ontologias de domínio. Adaptado de (KOPECKÝ *et al*., 2007)

Como padrão, o SAWSDL fornece uma estrutura que suporta as diferentes tecnologias relacionadas aos Serviços Web Semânticos, como WSMO e OWL-S. Como exemplo, RDF e OWL podem ser combinados com SAWSDL para descreverem a semântica de serviços web. Por ser uma extensão do WSDL, a utilização do SAWSDL não implica em alterações na estrutura do documento WSDL que descreve os serviços web. Apenas são adicionadas anotações com o intuito de compor uma camada semântica. Assim, o documento WSDL descreve as funcionalidades de um Serviço Web tanto sintática quanto semanticamente.

Uma anotação corresponde a um atributo que pode ser adicionado a qualquer elemento do documento WSDL. O SAWSDL define três tipos de atributos: *modelReference, liftingSchemaMapping e*

loweringSchemaMapping. O atributo *modelReference* é utilizado com o objetivo de apontar para um ou mais conceitos semânticos. Essa correspondência entre o elemento do WSDL e o conceito ontológico é realizada através de uma ou mais URIs. Isso implica dizer que é possível relacionar um elemento a mais de um conceito. A Figura 7 mostra um exemplo da utilização do atributo *modelReference* em um elemento do tipo *simpleType*.

```
<xs:simpleType name="confirmation"
      sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/on
      tology/purchaseorder#OrderConfirmation">
      <xs:restriction base="xs:string">
            <xs:enumeration value="Confirmed" />
            <xs:enumeration value="Pending" />
            <xs:enumeration value="Rejected" />
      </xs:restriction>
</xs:simpleType>
```

Figura 7 – Exemplo de anotação baseada no modelo de referência.

Os atributos *liftingSchemaMapping* e *loweringSchemaMapping* são utilizados para realizar ajustes semânticos nas mensagens trocadas com o Serviço Web (OLIVEIRA *et al*., 2009). Tais ajustes servem para transformar a estrutura de dados especificada no WSDL para o formato exigido pelo modelo semântico associado e vice-versa. O *lifiting* realiza o mapeamento entre os tipos definidos no WSDL e os conceitos semânticos, enquanto o *lowering* faz o mapeamento no sentido contrário.

O documento WSDL anotado não contém o significado propriamente dito dos seus elementos, apenas ponteiros para ontologias que se encarregam de fazer isso. O SAWSDL não restringe uma linguagem para a representação semântica dos elementos. Isso torna o SAWSDL independente da linguagem utilizada para especificar as ontologias de domínio.

Quando comparado com outros descritores semânticos, como OWL-S e WSMO, o SAWSDL é mais prático e fácil de ser utilizado por ser uma extensão do WSDL. Isso significa que ele exige menor esforço para descrever os serviços semanticamente, uma vez que requer apenas a adição de anotações semânticas ao documento WSDL dos serviços (OLIVEIRA *et al*., 2009). No SAWSDL o mapeamento entre os elementos do WSDL e os conceitos semânticos é feito no próprio

documento WSDL, diferenciando-se das outras abordagens, nas quais isso é realizado em um documento à parte (PRAZERES, 2009). Essas vantagens frente às outras abordagens são contrapostas pelas limitações de expressividade na descrição dos serviços web. O SAWSDL não oferece mecanismos para descrever com precisão as pré-condições e os efeitos da execução de um serviço como existem nas outras abordagens. Por outro lado, permite que restrições comportamentais sejam especificadas em ontologias e referenciadas por entradas e saídas através de anotações, a fim de associar pré e pós-condições com as operações fornecidas por um Serviço Web. Além disso, ainda é possível combinar outras tecnologias com o SAWSDL a fim de atingir o máximo de expressividade.

O sucesso na integração entre sistemas distribuídos baseados em Serviços Web depende da realização de processos como descoberta, composição e execução de serviços web. A adição de *links* para conceitos semânticos no documento WSDL visa permitir que esses processos sejam realizados de forma automática. A linguagem SAWSDL é, portanto, uma alternativa simples e ao mesmo tempo poderosa para descrever a semântica associada aos dados dos serviços web. Essa descrição é o ponto de partida para que sistemas computacionais sejam capazes de descobrir e compor serviços web sem a intervenção humana.

3 DESCOBERTA E COMPOSIÇÃO DE SERVIÇOS WEB

Para compreender o propósito da abordagem proposta neste trabalho se faz necessário entender os processos de descoberta e composição de Serviços Web apresentados neste capítulo.

3.1 DESCOBERTA DE SERVIÇOS WEB

O processo de descoberta de Serviços Web consiste na procura por serviços que desempenham determinada função. De modo geral, serviços são publicados com suas funcionalidades expostas e requisições consultam e buscam aqueles com funcionalidades específicas (MARTIN *et al*., 2007). Provedores de serviços desenvolvem e publicam os Serviços Web em repositórios e catálogos de serviços a fim de permitirem que usuários/desenvolvedores os encontrem e usem. Contudo, a busca por um Serviço Web em meio a tantos que execute uma tarefa específica é um processo nada trivial e que despende tempo (MESMOUDI *et al*., 2011). Além disso, a busca e a posterior seleção do serviço dependem intimamente da compreensão semântica do que ele realiza. Com o objetivo de reduzir a intervenção humana nesse contexto, tornam-se importantes as ferramentas que automatizam esse processo, minimizando o tempo de busca.

Inicialmente, portais web de serviços, como o Web-ServiceList (www.webservicelist.com), Remote-Methods (www.remotemethods.com), WSIndex (www.wsindex.org) e o XMethods (www.xmethods.net), foram criados para ajudar na divulgação de Serviços Web. Contudo, por não seguirem padrões definidos para Serviços Web, muitos deixaram de existir ou não foram considerados fontes confiáveis para a busca de serviços (AL-MASRI e MAHMOUND, 2008). Surgiram então máquinas de busca como as do Google capazes de encontrar Serviços Web através de *crawlers* que coletam informações acerca de documentos WSDL disponíveis em servidores web. Apesar disso, *crawlers* web foram desenvolvidos para coletarem informações de páginas web e não de Serviços Web (AL-MASRI e MAHMOUND, 2008). Nesse contexto surgem as ferramentas ou máquinas de busca desenvolvidas exclusivamente para lidar com Serviços Web.

Em se tratando de ferramentas para descoberta de Serviços Web, existem aquelas que realizam o processo de forma automática ou de modo semi-automático. Em ambos os casos, elas exigem que os usuários especifiquem as funcionalidades que desejam, no momento da

requisição, de maneira que possam ser mapeadas para a descrição dos serviços presentes nos repositórios. No caso das ferramentas semi-automáticas, o usuário especifica parâmetros que são utilizados para filtrar e classificar os Serviços Web (*ranking*). Posteriormente, o usuário seleciona manualmente aqueles que atendem as restrições especificadas. Essas ferramentas não selecionam os Serviços Web, pois não são capazes de compreender o significado das funcionalidades oferecidas por eles. Isso as difere das ferramentas que automatizam completamente o processo, visto que são capazes de interpretar a semântica associada às funcionalidades e por isso são autônomas.

3.1.1 Descrição de Serviços Web: Sintática e Semântica

Existem diferentes algoritmos para a realização do processo de descoberta de Serviços Web e cada qual apresenta diferentes níveis de complexidade, precisão e eficiência. O UDDI (*Universal Description, Discovery and Integration*) é uma especificação técnica para a construção de repositórios de Serviços Web nos quais provedores publicam os serviços que oferecem (KOURTESIS *et al.*, 2008). Essa publicação consiste no envio do WSDL com a descrição do Serviço Web para um repositório com acesso público. Assim usuários podem utilizar os repositórios UDDI para procurar os serviços que desejam e posteriormente obter acesso ao documento WSDL com as informações de como utilizar o serviço.

As informações disponibilizadas pelo UDDI podem ser classificadas em três categorias: Páginas Brancas, Páginas Amarelas e Páginas Verdes (OVERHAGE e THOMAS, 2003). Nas páginas brancas estão as informações sobre o provedor do serviço, como nome, telefone e descrição do negócio. Nas páginas amarelas estão disponíveis as informações sobre a classificação do serviço e do negócio. Por fim, as páginas verdes apresentam informações técnicas do Serviço Web. Essas informações são utilizadas no momento em que se busca um serviço específico. Contudo, esse processo é limitado à análise de palavras-chave e, portanto, sintático (TAMILARASE e RAMAKRISHNAN, 2012).

A análise sintática de palavras-chave é uma abordagem recorrente e talvez uma das mais utilizadas pelas ferramentas atualmente. Ela limita-se a analisar palavras-chave em detrimento do conteúdo semântico dos serviços. O usuário envia uma requisição com uma ou mais palavras que são comparadas à descrição sintática das funcionalidades dos serviços (documento WSDL) e também às

informações fornecidas pelos provedores, como categoria e classificação do serviço. A ambiguidade inerente às linguagens naturais prejudica a automatização desse tipo de busca e faz com que sejam retornados serviços que não estejam de acordo com o solicitado (KELLER *et al*., 2005).

As ferramentas para descoberta de Serviços Web tiveram uma grande evolução no sentido de automatizar esse processo com o surgimento dos Serviços Web Semânticos. Nessa abordagem, as ferramentas passam a ser capazes de realizar inferências baseadas nas ontologias que descrevem os serviços e consequentemente interpretar o significado das funcionalidades providas por eles. Isso faz com que elas ganhem autonomia para comparar semanticamente as funcionalidades expostas dos serviços existentes com as requeridas pelo usuário e selecionar aquelas consideradas similares.

Não só os atributos funcionais dos Serviços Web podem ser comparados durante a busca e seleção, mas também os atributos não funcionais e comportamentais dos mesmos. Isso significa que as ferramentas de descoberta podem analisar uma ou mais classes de atributos a fim de filtrar e ranquear os serviços mais adequados (KOURTESIS *et al.,* 2008). Esses atributos são descritos semanticamente através do seu relacionamento com conceitos de ontologias de domínio. Neste trabalho apenas os atributos funcionais (entradas, saídas e operações) são considerados no processo de descoberta. Não serão levados em conta requisitos não funcionais, como tempo de execução do serviço e disponibilidade.

A descoberta de Serviços Web Semânticos envolve a análise semântica das suas funcionalidades com o objetivo de verificar se realizam a atividade desejada. Essa análise passa essencialmente pela comparação das entradas e saída de cada operação com as entradas e saídas requeridas, respectivamente. Nesse caso, a comparação é realizada com os conceitos ontológicos associados a elas. A esse processo dá-se o nome de *matching* semântico (PRAZERES, 2009). Algoritmos de descoberta que fazem uso do *matching* semântico para comparar os atributos funcionais dos Serviços Web são conhecidos por realizarem *capability matching* (GAO *et al*., 2002).

Segundo (KELLER *et al*., 2006), o *matching* semântico é o processo de inferir possíveis explicações, diagnósticos e classificações a partir do modelo semântico. A partir das inferências, o *matching* visa estabelecer uma correspondência entre os objetivos especificados na requisição e os Serviços Web disponíveis, de modo a selecionar aqueles com potencial para realizar o serviço desejado.

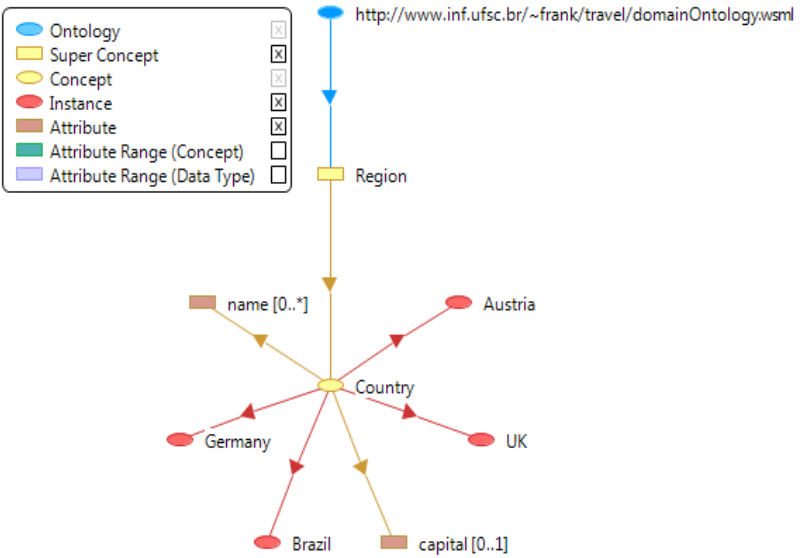

Figura 8 – Parte extraída de uma ontologia para viagens.

Para definir quão similares são dois conceitos ontológicos e se uma funcionalidade é capaz de retornar o que foi solicitado partindo de um conjunto conhecido de entradas, podem-se realizar inferências baseadas em relações de *subsumption*. Essa forma de inferência analisa o relacionamento e a hierarquia entre conceitos, propriedades e outros aspectos das ontologias. Ela permite deduzir se um conceito é supertipo/subtipo de outro, se possui determinada propriedade, se está associado a outro e assim por diante. No fragmento de uma ontologia de viagens, representado na Figura 8, é possível inferir que Brasil é uma instância de País e que Região é um super conceito de País, por exemplo.

A inferência sobre os conceitos representados nas ontologias ocorre por meio das chamadas máquinas de inferência. Máquinas de inferência são responsáveis por extrair conhecimento até então implícito nas ontologias. A partir de uma série de regras aplicadas sobre um conjunto de axiomas e baseado nas relações de *subsumption*, uma máquina de inferência é capaz de gerar hipóteses e novos conhecimentos a partir de uma base de conhecimento, como é o caso das ontologias. Algoritmos de descoberta que empregam máquinas de inferências

durante a realização do *matching* tornam-se mais precisos e menos sujeitos a inconsistências.

O algoritmo proposto por (PAOLUCCI *et al*., 2002) e adaptado para este trabalho realiza *capability matching* para encontrar serviços com base nas funcionalidades providas por eles. Baseada nas relações de *subsumption* ele compara as funcionalidades dos serviços anunciados com a funcionalidade requerida e seleciona os serviços que serão úteis de alguma forma para o requerente. O algoritmo atribui um dos seguintes graus de exatidão ao comparar o conceito requerido com um conceito anunciado na descrição semântica de um serviço:

- ***Exact*** **–** Quando o conceito requerido e o anunciado são idênticos, ou seja, correspondem ao mesmo conceito da ontologia. Ou ainda quando o conceito anunciado é superclasse direta do conceito requerido (PRAZERES, 2009). Ex.: Veículo é uma superclasse direta de Carro.
- ***Plugin*** **–** Quanto o conceito anunciado é uma superclasse indireta do conceito requerido (SENA, 2009). Ex.: Veículo é uma superclasse direta de Carro e Carro uma superclasse direta de Carro Esportivo. Logo Veículo é uma superclasse indireta de Carro Esportivo.
- ***Subsume*** **–** Quando o conceito requerido é uma superclasse do conceito anunciado. Ou seja, o oposto do *plugin*.
- ***Fail*** **–** Quando não é possível estabelecer nenhum relacionamento entre o conceito anunciado e o requerido.

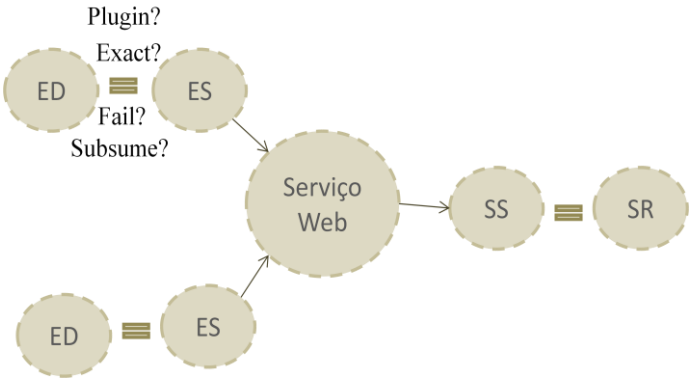

Figura 9 – *Matching* semântico

A Figura 9 apresenta um exemplo de *matching* semântico realizado durante o processo de descoberta de Serviços Web. Para cada serviço existente no repositório, é realizada a comparação dos conceitos associados às entradas disponíveis pelo usuário (ED) com as entradas do serviço (ES) e entre a saída requerida (SR) com a saída do serviço (SS). A forma como os graus de similaridade são utilizados pelas ferramentas de descoberta pode variar de uma abordagem para outra. Enquanto algumas desqualificam o serviço caso o *matching* de uma das entradas não seja exato, outras consideram os graus *plugin* e *subsume* aceitáveis.

Quando conceitos são comparados um a um, a análise de similaridade permite identificar de que forma eles se relacionam entre si. Um Serviço Web pode oferecer diversas operações e cada uma delas exigir um conjunto de entradas. Uma operação só será selecionada se todas as entradas estiverem disponíveis pelo usuário. Logo, é necessário comparar todas as entradas requeridas com as anunciadas para posteriormente analisar o resultado do conjunto como um todo. Isso também se aplica às saídas caso um único serviço deva fornecer todas as saídas solicitadas. Nesse caso, todas as saídas solicitadas devem ser comparadas com as saídas fornecidas pelo serviço a fim de identificar se o serviço será ou não selecionado. Isso não se repete na comparação das saídas durante o processo de composição, visto que basta uma operação retornar uma das saídas requeridas pelo usuário para que ela seja selecionada. Assim é possível determinar se o serviço satisfaz ou não as restrições impostas na requisição.

As ferramentas que fazem uso dessa abordagem conseguem automatizar o processo de descoberta bem como os processos subsequentes de composição e execução. A descoberta é só o primeiro passo para a execução de um serviço. Em muitos casos não é possível encontrar o serviço procurado, pois é possível que ele sequer exista. Quando a descoberta falha e não é possível encontrar um Serviço Web que por si só realize a tarefa desejada, uma alternativa é recorrer à composição de serviços.

3.2 COMPOSIÇÃO DE SERVIÇOS WEB

A composição de Serviços Web é a combinação de serviços pré-existentes para criar um novo serviço, tirando proveito das funcionalidades oferecidas por cada um e combinando-as a fim de oferecer funcionalidades ainda mais complexas e até inexistentes. Quando um único serviço não pode desempenhar uma funcionalidade

solicitada, a composição surge como uma alternativa para a realização do que se deseja (MARTIN *et al*., 2007). A construção de composições visa, dessa forma, reduzir o esforço e o tempo de desenvolvimento de aplicações por meio do reuso dos Serviços Web disponíveis.

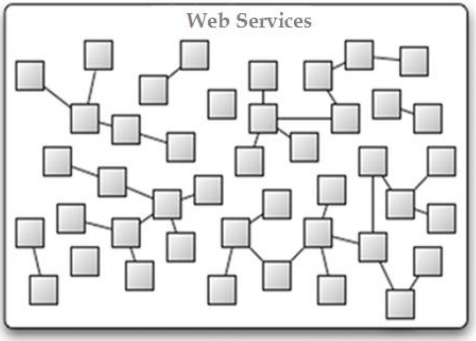

Figura 10 – Serviços Web disponíveis para composição.

Muitas vezes a execução de uma tarefa, como programar uma viagem de turismo, exige que serviços de diferentes empresas sejam executados concomitantemente ou sequencialmente, de modo que um dependa do sucesso do outro. No exemplo da viagem, estão envolvidas empresas da área de aviação, hoteleira e financeira, que oferecem os serviços de compra de passagens, reserva de hotel e consulta ao saldo do cartão de crédito, respectivamente. Para o sucesso da operação, o saldo do cartão de crédito deve ser consultado para que seja possível comprar a passagem e, uma vez comprada e confirmada a data da viagem, reservar o hotel. Em resumo, empresas buscam oferecer serviços integrados (composição de serviços), conforme ilustrado na Figura 10, a fim de partilharem seus recursos, competências e custos.

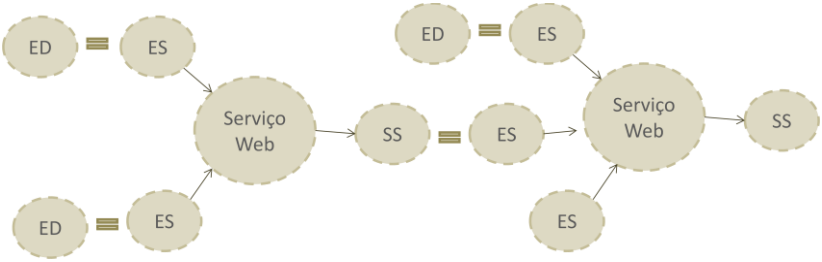

Figura 11 – Exemplo de composição de serviços.

A construção de composições de Serviços Web, conforme visto na Figura 11, é uma tarefa extremamente complexa e precisa entre outras coisas, coordenar a comunicação entre os serviços envolvidos, estabelecer a troca de dados entre eles, ajustar e adaptar os dados trocados e prover tratamento de erros. Isso se torna ainda pior se considerado que os Serviços Web publicados nos repositórios distribuídos ao longo da Internet podem sofrer modificações/alterações que afetam composições já estabelecidas. Para lidar com esses desafios, a construção de composições pode ser feita de forma automática, semi-automática ou manual, e resultar em uma composição dinâmica ou estática (DUSTDAR *et al*., 2005).

Uma composição é dita manual quando todo o processo é realizado pelo desenvolvedor, desde a seleção individual dos Serviços Web até a definição do fluxo de execução dos mesmos. A construção manual de composições é um processo complexo, propenso a erros e que despende tempo. Em contrapartida, uma composição é automática quando é criada por ferramentas ou aplicações sem o auxílio do desenvolvedor (PRAZERES, 2009). Nesse caso, o desenvolvedor apenas especifica o que deseja e a ferramenta busca estabelecer uma composição que o atenda. Não há a necessidade de o desenvolvedor encontrar os Serviços Web para montar a composição e nem estabelecer a sequencia de execução deles. As composições semi-automáticas são um modelo híbrido, onde as ferramentas apresentam uma lista de opções de Serviços Web que são passíveis de serem incluídos na composição. O desenvolvedor seleciona um Serviço Web dessa lista e a ferramenta apresenta novas opções de serviços com possibilidade de se integrarem a ele. Assim, a ferramenta auxilia e guia o desenvolvedor na construção da composição.

A composição estática ocorre em tempo de desenvolvimento, quando a arquitetura e a estrutura do sistema estão sendo codificadas (DUSTDAR *et al*., 2005). Os Serviços Web são descobertos, selecionados e conectados entre si de modo a estabelecer o fluxo de execução dos mesmos. Posteriormente o sistema é compilado e disponibilizado para uso. Esse tipo de composição é funcional quando os Serviços Web não sofrem mudanças, sejam internas ou externas, constantemente. As mudanças externas dizem respeito à disponibilidade, enquanto as internas são relacionadas às regras de negócio. Assim, a composição criada é imutável e permanece inalterada durante todo o funcionamento do sistema. Isso porque a composição está codificada e sempre que houver uma mudança na composição o código precisa ser

alterado por um desenvolvedor. Basta que um serviço da composição seja retirado do ar para que toda a composição deixe de funcionar.

A composição dinâmica, por sua vez, oferece mais flexibilidade no momento da criação da composição. A partir de um objetivo estabelecido (saída desejada) e uma lista de restrições (conjunto de entradas) a composição é construída em tempo de requisição (DUSTDAR *et al*., 2005). Isso significa que os Serviços Web são descobertos e o fluxo de execução criado de forma automática, sem a necessidade de envolver um desenvolvedor. Assim, a composição não fica susceptível a mudanças ocorridas com os Serviços Web que a compõe. Qualquer alteração ou problema que ocorrer com um desses serviços, a composição dinâmica se adapta e encontra um Serviço Web que o substitua.

O processo de composição pode ser dividido em duas etapas: síntese e orquestração. A etapa de síntese corresponde à formação propriamente dita da composição, desde a descoberta de serviços até a especificação de como executá-los. Como orquestração, entende-se o controle tanto do fluxo de execução como da troca de dados entre um serviço e outro. A orquestração tem como objetivo garantir a execução correta da composição. A abordagem proposta neste trabalho trata apenas da etapa de síntese, já que não é intenção deste trabalho realizar a execução das composições descobertas.

Por serem desenvolvidos por diferentes empresas e pessoas, cada qual com suas técnicas e modelos de desenvolvimento bem como visão de mundo e negócio, mesmo que dois serviços realizem a mesma tarefa, a descrição deles pode ser bastante diferente (RAO *et al*., 2004). Ferramentas que proveem mecanismos para compor serviços precisam, entre outras coisas, lidar com condições, requisitos e resultados diferentes fornecidos por Serviços Web distintos (CHARIF *et al*., 2006). Em outras palavras, a partir da requisição de um serviço e um conjunto de Serviços Web disponíveis, a ferramenta deve ser capaz de encontrar e orquestrar uma sequência de serviços a fim de que atendam à solicitação.

O processo de composição de Serviços Web engloba as etapas de descoberta, seleção e execução. A primeira tem como objetivo encontrar, nos repositórios de Serviços Web, aqueles que atendem completamente ou em partes a requisição por uma determinada tarefa. Encontrados os serviços, eles passam por uma triagem a fim de selecionar, com base nos atributos funcionais (entradas e saídas) e não funcionais (disponibilidade e tempo de resposta, por exemplo) apenas aqueles que melhor se adéquam ao contexto da requisição. Por fim, a

etapa de execução corresponde à invocação dos serviços que agregam a composição a fim de realizar a tarefa desejada.

O processo de descoberta, em muitos casos, é recorrente durante o processo de composição. À medida que a composição é construída, Serviços Web são encontrados e agregados a ela. Assim, a descoberta de Serviços Web é realizada repetidamente até que a composição esteja concluída. A composição parte de um conjunto de Serviços Web, armazenados em um único repositório ou distribuídos ao longo de diversos repositórios, e realiza uma série de iterações sobre eles. A cada iteração um ou mais serviços podem ser selecionados, com base nas suas entradas, saídas, pré-condições e efeitos, por exemplo. As iterações subsequentes buscam Serviços Web que sejam passíveis de serem conectados aos serviços selecionados anteriormente.

A realização automática do processo de descoberta de Serviços Web decorrente do surgimento dos Serviços Web Semânticos teve impacto no processo de composição, já que os processos estão intrinsecamente relacionados. Anteriormente ao surgimento dos Serviços Web Semânticos, os métodos para compor serviços não eram capazes de assegurar a automação e a dinamicidade durante a construção das composições. A razão para isso estava na insuficiência de técnicas e recursos para descrever formalmente e semanticamente os serviços. Com o advento da web semântica, as ferramentas para composição de serviços evoluíram para um estágio de maior autonomia no sentido de depender menos da interferência humana durante o processo (CHARIF *et al*., 2006).

Normalmente o processo de descoberta de composições tem início com a chegada de uma requisição de serviço por parte de um cliente (pessoa ou sistema). Essa requisição contém, por meio de alguma notação ou expressão, a especificação da tarefa a ser executada. Essa especificação é semelhante à especificação de um Serviço Web, uma vez que as duas definem as condições para execução e o resultado de uma determinada tarefa em termos de entradas e saídas. Uma requisição pode ser caracterizada de acordo com seu modo de interação e o grau de plenitude (BERARDI, 2005). A primeira avalia o modo como a requisição mapeia as características do serviço a ser encontrado, podendo ser classificada em atômica, determinística e não determinística. Requisições atômicas descrevem as operações dos serviços que devem ser executados em termos de entradas e saídas. As determinísticas descrevem o comportamento dos serviços por meio de uma sequência linear de ações, de modo que uma ação sucede a execução da outra de forma única. Assim como as determinísticas, as

não determinísticas também detalham o comportamento dos serviços em termos de ações, porém, cada ação pode ser sucedida por mais de uma ação, especificadas através de um conjunto de possibilidades. O grau de plenitude avalia a expressividade da requisição e pode ser classificada em completa e parcial. Quando completa, a requisição expressa as características de um único serviço. Quando parcial, expressam as características de um conjunto de serviços capazes de atendê-la (BERARDI, 2005).

Existem diferentes técnicas e métodos relacionados à descoberta e composição de Serviços Web com a intenção de fornecer mecanismos que facilitem a integração de sistemas heterogêneos (RAO *et al*., 2004). Essas abordagens divergem umas das outras quanto às tecnologias utilizadas, a forma como realizam o processo, e o modo como manipulam os Serviços Web. Contudo, a maioria delas converge para um mesmo ponto que é a redução da interferência humana e a consequente agilidade propiciada ao desenvolvimento de novas aplicações. Como técnicas empregadas para compor serviços destacam-se as baseadas em:

- Workflow;
- Planejamento em Inteligência Artificial;
- Regras;
- Lógica;

Métodos baseados em *workflow* partem do princípio de que uma composição pode ser mapeada para um *workflow* com o fluxo de invocação dos serviços bem definido (CASATI *et al*., 2001). Esse fluxo deve ser coordenado, sincronizado e ordenado. A troca de dados e mensagens entre os serviços é feita de acordo com requisitos definidos pelo *workflow* (entradas, saídas, pré-condições e efeitos) (SILVA *et al*., 2011). Busca-se, através da interação colaborativa entre os serviços, alcançar um objetivo global definido no momento da requisição. A construção do *workflow* pode ser feita de forma estática ou dinâmica (RAO *et al*., 2004). Na primeira, o solicitante estabelece o conjunto das tarefas que devem ser executadas bem como a dependência de dados entre elas. Cada tarefa está associada a uma *query* utilizada para encontrar o Serviço Web que irá realizá-la. Os *workflows* dinâmicos, por sua vez, encontram e estabelecem a relação entre as tarefas automaticamente a partir de restrições e condições impostas pela requisição. Grafos são exemplos de estruturas comumente utilizados para a construção de *workflows*.

Um grafo (G) é uma estrutura de dados genérica composta por um conjunto de nós (N) e vértices (V) onde G = (N, V). Os nós de um grafo são ligados pelos vértices de modo que o número de vértices é a metade do número de nós. Um grafo pode ser classificado em dirigido, quando os vértices são direcionados, ou não dirigido, quando os vértices não seguem uma ordem ou direção. No contexto de composição de serviços, os nós correspondem aos serviços e os vértices representam a ordem de execução dos mesmos. Grafos direcionados são mais adequados, visto que a execução dos serviços deve seguir uma ordem estabelecida. Em outras palavras, a relação entre os serviços estabelecida pelos vértices direcionados define o fluxo de execução de uma composição. Um grafo pode ser composto por diferentes caminhos (*paths*), cada qual correspondendo a uma sequência de um ou mais nós. Na composição de serviços, caminhos representam sequências de invocações de Serviços Web que podem ser executadas em paralelo ou de forma sequencial.

Outra classificação aplicada a grafos tem relação com a associação de valores aos vértices. Assim sendo, existem os grafos valorados, onde é associado um valor numérico a cada vértice chamado de peso, e os não valorados, onde não há associação de pesos e nem distinção entre os vértices. Para determinar o menor caminho entre dois nós em grafos não valorados basta contabilizar a quantidade de nós a fim de mensurar a profundidade do caminho. Já em grafos valorados esse cálculo requer uma compreensão do significado dos pesos para posteriormente determinar o menor caminho.

Planejamento em Inteligência artificial é uma abordagem recorrente na literatura para tentar resolver o problema de composição de Serviços Web de forma automática (MARCONI *et al.*, 2009). A construção de composições é vista sob o ponto de vista dessa abordagem como um problema de planejamento e é representada a partir de uma tupla composta por cinco elementos {S, S_0, G, A, I}. O elemento S corresponde ao conjunto de todos os estados possíveis de uma composição, S_0 indica o estado inicial ($S_0 \subset S$) definido no momento da requisição de um serviço, o G representa o objetivo ou o que se espera da composição (também definido no momento da requisição), o elemento A corresponde ao conjunto de Serviços Web disponíveis e, por fim, o elemento I especifica a função de mudança de estado ou de transição de cada Serviço Web (RAO *et al.*, 2004).

Técnicas baseadas em lógica costumam adotar autômatos e máquinas de estado como formas de representação de composições e de requisições. A ideia por trás dessa abordagem é utilizar especificações

lógicas para codificar as composições em fórmulas e assim reduzir o problema de composição à satisfatoriedade da fórmula. Nesse contexto, uma composição existe se e somente se uma fórmula que expressa um modelo de composição for considerada satisfatória.

Os métodos baseados em regras, como o próprio nome sugere, trabalham com regras de agregação entre os Serviços Web. Essas regras podem considerar tanto as propriedades sintáticas como as semânticas dos serviços. As regras aplicadas às propriedades sintáticas abrangem as restrições associadas ao protocolo de comunicação e o modo de interação entre dois ou mais serviços. Através das regras semânticas têm-se a possibilidade de aplicar restrições e condições: (1) às mensagens, de modo que dois serviços são passíveis de serem interligados se a saída de um deles for compatível com a entrada do outro; (2) à operação, de modo que o domínio e categoria dos serviços sejam avaliados e equiparados; (3) às preferências enviadas na requisição, de modo que sejam comparadas com a qualidade das operações realizadas na composição; (4) à eficiência da composição, através da sua avaliação. As regras propostas pelos métodos baseados em regras podem ser utilizadas por métodos baseados em planejamento de IA como um guia para a construção da composição.

3.3 TRABALHOS RELACIONADOS

Dada a quantidade de trabalhos existentes nessa área, o objetivo desta seção é apresentar trabalhos que abordam o processo de descoberta e composição de Serviços Web com foco na web semântica.

O trabalho de (PRAZERES, 2009) propõe uma solução para promover a descoberta e a composição automática de Serviços Web Semânticos baseada em workflow. O processo de descoberta é feito mediante análise semântica das funcionalidades providas pelos serviços bem como da categoria, disponibilidade e pré-condições para execução. Para tal, o autor utiliza OWL-S para descrever os serviços e divide o algoritmo de descoberta em cinco estágios, como visto na Figura 12: *matching* das entradas, *matching* das saídas, seleção baseada nas pré-condições, seleção baseada na categoria, e por fim seleção final dos serviços. O *matching* realizado nas duas primeiras etapas é baseado em relações de *subsumption*, ou seja, na hierarquia entre os conceitos de uma ontologia. A execução dessas cinco etapas visa selecionar, dentre os serviços disponíveis, aqueles que melhor atendem os requisitos.

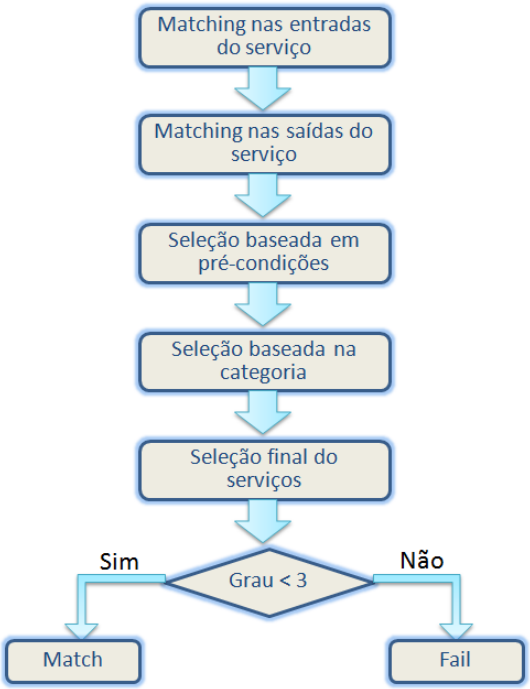

Figura 12 – Algoritmo para a descoberta de Serviços Web Semânticos (PRAZERES, 2009).

Quando nenhum serviço é descoberto, de modo que por si só atenda a requisição feita pelo usuário, o autor apresenta um modelo baseado em grafos direcionados e valorados para a composição automática de Serviços Web Semânticos. Os grafos são criados uma única vez e à medida que os Serviços Web são publicados ou removidos de um repositório UDDI, o grafo é atualizado e os serviços recebem um custo associado à quantidade de entradas e saídas que possuem. O cálculo para esses custos é refeito sempre que uma nova solicitação de serviço é realizada, ou seja, assim que o usuário envia as entradas e saídas desejadas. Isso significa que são dados descontos aos custos de um serviço caso ele possua alguma entrada disponível ou saída requerida informada na requisição. Esses custos são definidos pelo autor como custos funcionais, uma vez que estão relacionados às entradas e saídas dos serviços. Segundo ele, ainda é possível atribuir descontos aos custos do serviço de acordo com a sua complexidade, caracterizando os custos não funcionais. Ao final do processo de atribuição de custos, são

selecionados para a composição aqueles que possuírem o caminho no grafo com o custo mínimo.

O trabalho de (ZHANG *et al*., 2003) propõem um algoritmo baseado em grafos para a composição automática de Serviços Web Semânticos. O objetivo do algoritmo é encontrar composições que retornem a saída desejada com o menor tempo de execução e o melhor grau de similaridade entre as entradas e saídas. Os nós do grafo representam os Serviços Web e as arestas indicam a conexão entre eles. Pesos são atribuídos às arestas do grafo e são calculados de acordo com os seguintes fatores:

*Peso = (λ) * tempo de execução + (1-λ) * valor da similaridade.*

Outros fatores podem ser considerados pelos usuários e então substituídos pelo *λ*. A técnica empregada no algoritmo, chamada pelos autores de *Interface-Matching Automatic* (IMA) e representada na Figura 13, utiliza as entradas disponíveis e as saídas requeridas pelo usuário para percorrer as ontologias associadas aos Serviços Web. Os serviços que possuem as entradas compatíveis com as entradas do usuário são selecionados e a partir deles novos serviços não conectados até que se alcance a saída desejada. Assim, a composição inicia pelo estado inicial (SI) e termina no estado final (SF). Os graus de similaridade usados para comparar as entradas e saídas são os mesmos propostos em (PAOLUCCI *et al*., 2002).

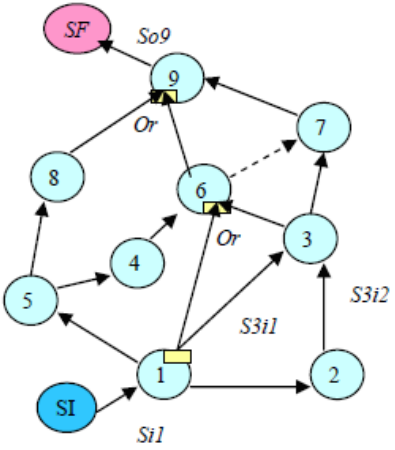

Figura 13 – Técnica de composição IMA (ZHANG *et al*., 2003).

Outro trabalho relacionado à descoberta de Serviços Web Semânticos foi apresentado por (TRAN *et al*., 2009). Apesar de não focar na composição e tratar apenas da descoberta de serviços, apresenta um algoritmo para o *matching* semântico de Serviços Web Semânticos baseado em SAWSDL. A ideia da proposta dos autores é comparar os requisitos funcionais do serviço solicitado com os dos serviços disponíveis e então identificar níveis de compatibilidade entre eles. Para isso utiliza a semântica associada à descrição dos serviços por meio do SAWSDL para realizar inferências e comparar entradas e saídas dos serviços.

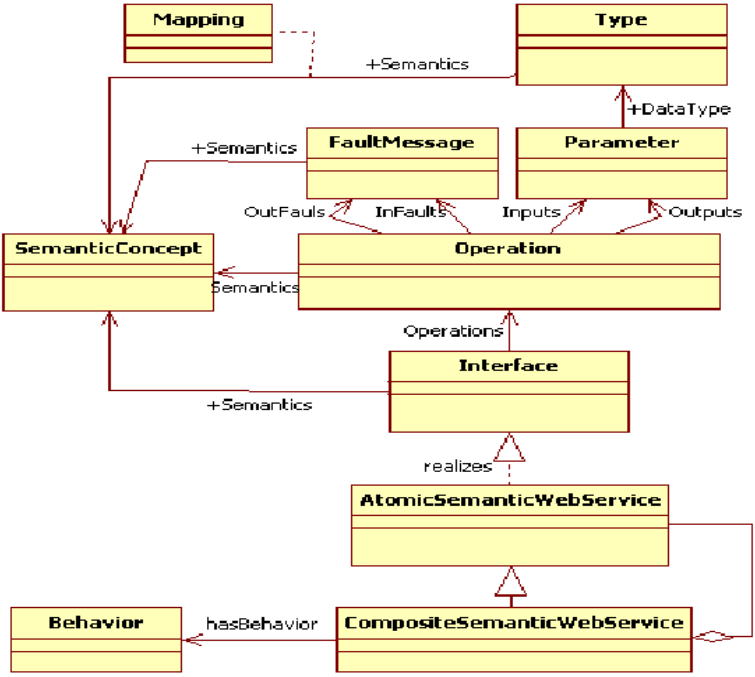

Figura 14 – Metamodelo para a composição de Serviços Web Semânticos (BELOUADHA *et al*., 2010).

Ainda relacionado a este trabalho está o trabalho de (BELOUADHA *et al*., 2010), que propõe uma abordagem baseada em UML (*Unified Modeling Language*), vista na Figura 14, para descrever e compor Serviços Web Semânticos. A descrição semântica dos serviços ocorre através um metamodelo especificado em UML e independente de

linguagem semântica. Posteriormente esse metamodelo é transformado, utilizando regras de transformação, em um WSDL anotado com SAWSDL. Por fim, os autores utilizam o BPMN (*Business Process Modeling Notation*) para gerar o plano de execução da composição de serviços. As composições não são descobertas e nem criadas automaticamente, caracterizando composições estáticas.

O trabalho de (MONTEIRO, 2008) propõe a composição de Serviços Web Semânticos de maneira semi-automática. Por meio de uma aplicação com interface gráfica, o usuário desenvolvedor é orientado a construir a composição. A aplicação funciona, portanto, como um guia para a composição de Serviços Web. Para tanto, os serviços precisam estar descritos com OWL-S. A ferramenta oferece estruturas de controle, como sequencia e condição, para acoplar um serviço ao outro. Ao selecionar um serviço, a ferramenta busca no repositório serviços que podem ser conectados a ele e o usuário, por sua vez, opta por um deles. A ferramenta proposta pelos autores não cria composições de forma automática, as composições são construídas estaticamente com base na interação com o usuário.

A Tabela 1 apresenta uma comparação entre os trabalhos relacionados. São confrontadas as características de cada uma das abordagens vistas anteriormente. A forma como as abordagens apresentadas trabalham com Serviços Web Semânticos para realizarem os processos de descoberta e composição serviu de base para a solução construída neste trabalho. A combinação das técnicas e práticas apresentadas orientou e serviu de base para resolver algumas das limitações inerentes às abordagens.

Abordagens que trabalham com composição realizada em tempo de desenvolvimento, como é o caso dos trabalhos de (MONTEIRO, 2008) e (BELOUADHA *et al*., 2010), exigem o envolvimento de desenvolvedores sempre que uma composição precisa ser refeita e consequentemente requerem uma repetição do processo. A abordagem proposta neste trabalho pretende eliminar essa intervenção do usuário, transformando o processo de composição efetuado estaticamente em um processo dinâmico.

O algoritmo proposto no trabalho de (TRAN *et al.*, 2009) será utilizado como base para os *matchings* semânticos de entradas, saídas e operações na abordagem proposta. Enquanto a abordagem proposta neste trabalho contempla a descoberta e a composição de Serviços Web, o trabalho apresentado pelos autores consiste de um algoritmo para a descoberta de serviços isolados. Com relação ao trabalho de (PRAZERES, 2009), a abordagem proposta pretende eliminar os casos

onde são exigidas entradas indisponíveis pelo usuário quando uma composição é criada e trabalhar com uma linguagem de anotação semântica mais simples de ser utilizada quando comparada com o OWL-S, mesma linguagem utilizada no trabalho de (ZHANG *et al*., 2003).

O trabalho de (ZHANG *et al*., 2003) contribuiu para a construção tanto do módulo de descoberta como do módulo de composição da abordagem apresentada neste trabalho. Contudo, pretende-se incrementar a solução apresentada por (ZHANG *et al*., 2003) através da adição de repositórios de composições, com o objetivo de diminuir o tempo de resposta de uma requisição de serviço e avaliação das composições encontradas por meio de uma fórmula que avalia o grau de divergência semântica. Essa avaliação não é realizada por nenhum dos trabalhos relacionados e é ponto chave na abordagem apresentada. Além disso, os trabalhos relacionados não contemplam a avaliação das anotações relacionadas às operações dos Serviços Web, melhoria esta realizada neste trabalho.

Outras comparações com os trabalhos relacionados serão apresentadas no capítulo de Implementação e Avaliação da Abordagem.

Tabela 1- Comparação entre os trabalhos relacionados

	PRAZERES, 2009	**TRAN *et al.*, 2009**	**BELOUADHA *et al.*, 2010**	**MONTEIRO, 2008**	**ZHANG *et al.*, 2003**
Linguagem	OWL-S	SAWSDL	UML / SAWSDL	OWL-S	DAML-S
Descoberta	Sim	Sim	Sim	Sim	Sim
Composição	Sim	Não	Sim	Sim	Sim
Forma de Composição	Automática	-	Automática	Semi-Automática	Automática
Momento de Composição	Publicação/ Requisição	-	desenvolvimento	desenvolvimento	execução
Tipo de Composição	Dinâmica	-	Estática	Estática	Dinâmica
Técnica de composição	Grafos	-	UML e BPMN	Interação com o usuário	Grafos
Pré-Condições e Efeitos	Sim	Não	Sim	Sim	Não

4 ABORDAGEM AUTOMÁTICA PARA A DESCOBERTA E COMPOSIÇÃO DE SERVIÇOS WEB

Este trabalho visa à concepção e o desenvolvimento de uma abordagem que permite a descoberta e a composição automática de Serviços Web Semânticos em tempo de requisição. Nesta abordagem, anotações SAWSDL são extraídas a partir de descritores WSDL e empregues na construção de grafos de composições baseados na semântica dos serviços. A composição pode ser requisitada pelos clientes através de um pedido único. Em resposta à requisição, uma listagem de serviços, com a ordem e as operações pertencentes à composição, é retornada. A execução desses serviços, bem como o controle dos dados trocados entre eles, fica a cargo do cliente realizar.

O projeto teve origem a partir da ideia de construir um mecanismo capaz de descobrir e combinar Serviços Web automaticamente de modo a possibilitar a resolução de problemas advindos da dificuldade de encontrar Serviços Web e da complexidade de compô-los. Esses problemas podem ser minimizados com soluções já existentes, porém diferem-se da apresentada neste trabalho no que diz respeito à seleção de composições baseadas na qualidade semântica e na forma como os Serviços Web Semânticos são utilizados, motivo pelo qual nasceu este trabalho.

4.1 PUBLICAÇÃO DE SERVIÇOS WEB

Para que seja possível a realização dos processos de descoberta e composição de Serviços Web, estes precisam ser publicados em repositórios de serviços descritos semanticamente. Durante a publicação de um Serviço Web, as anotações semânticas das suas operações, entradas e saídas são extraídas do descritor WSDL e armazenadas em um banco de dados relacional. Essa estratégia é adotada a fim de otimizar o processo de descoberta, uma vez que ela é mais rápida do que extrair as anotações diretamente dos descritores WSDL (*parsing*) sempre que uma requisição for realizada.

Assume-se que as informações semânticas devem ser anotadas na descrição dos Serviços Web, por quem os desenvolve, utilizando SAWSDL. Em outras palavras, parte-se do princípio de que os serviços já são publicados devidamente anotados. Qualquer outra linguagem utilizada para descrever Serviços Web Semânticos não será aceita.

Como exemplo, considere o fragmento de um WSDL exibido na Figura 15. Os elementos destacados correspondem às URIs que serão armazenadas no banco de dados. No cabeçalho do WSDL definiu-se que *ontology* é o *namespace* que referencia a ontologia de domínio. Utiliza-se então o *namespace* como um atalho para referenciar os conceitos definidos na ontologia. No fragmento são utilizados os conceitos *CarPriceByRegion*, *Region* e *Price* para descrever a operação, a entrada e a saída, respectivamente.

```
<xsd:element name="price"
     sawsdl:modelReference="ontology#Price"/>
<xsd:element name="region"
     sawsdl:modelReference="ontology#Region"/>

<wsdl:message name="input" >
     <wsdl:part name="region" element="region"/>
</wsdl:message>

<wsdl:message name="output" >
     <wsdl:part name="price" element="price"/>
</wsdl:message>

<wsdl:operation name="getCarPriceByRegion"
     sawsdl:modelReference="ontology#CarPriceByRegion">
     <wsdl:input message="input"/>
     <wsdl:output message="output" />
</wsdl:operation>
```

Figura 15 – Fragmento de WSDL.

Vale ressaltar que uma operação pode ser composta por mais de uma entrada bem como por mais de uma saída. Cada entrada e saída podem estar associadas com elementos simples ou complexos definidos na sessão *Types* do WSDL. Normalmente as anotações semânticas das entradas e saídas são feitas nesses elementos já que são eles que representam os conceitos da ontologia. Quanto a anotações das operações, elas são adicionadas diretamente no elemento *operation* do WSDL. Quando o algoritmo encontra elementos simples durante a publicação de um Serviço Web, suas anotações são extraídas e associadas à entrada ou saída relacionada ao elemento. Quando surgem elementos complexos, o processo torna-se um pouco mais complicado uma vez que eles podem ser compostos por outros objetos, também complexos ou simples. Nesses casos, é preciso navegar, de forma recursiva, por cada um dos elementos aninhados e extrair as anotações

semânticas que são encontradas. Cada anotação é então associada com a entrada ou saída referenciada pelo elemento mais externo. Qualquer saída fornecida por uma operação, seja de forma direta, acessando um elemento simples, ou indireta, acessando um elemento aninhado a um elemento complexo, pode ser encontrada pelo algoritmo durante os processos de descoberta e composição.

Em virtude da quantidade de elementos internos a um elemento complexo, tanto uma entrada como uma saída podem apresentar diversas anotações, haja vista que cada elemento interno pode ter sua própria anotação. A adição de múltiplas anotações às operações, por exemplo, possibilita definir pré e pós-condições, mesmo que com uma baixa expressividade, para a sua execução.

4.2 REQUISIÇÃO DE DESCOBERTA DE SERVIÇOS WEB E COMPOSIÇÕES

Para localizar um serviço ou composição, o usuário (desenvolvedor) deve invocar uma operação que requer seis parâmetros:

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

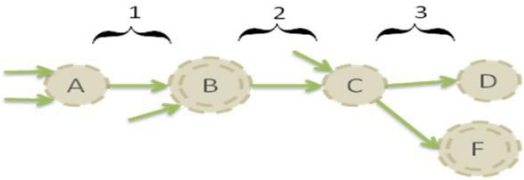

Figura 16 – Nível de profundidade de uma composição

Essa requisição é codificada por desenvolvedores durante a implementação de sistemas informacionais. As listas de entradas, saídas e operações são compostas por URIs que fazem referência aos conceitos definidos em ontologias de domínio. A profundidade máxima da composição determina quantos serviços que, sequencialmente, podem estar presentes em cada caminho do grafo de composições. Uma composição que possua uma sequência de quatro operações a serem

executadas apresenta três níveis de profundidade, conforme exemplificado na Figura 16. O parâmetro timeout corresponde ao intervalo de tempo que o cliente está disposto a aguardar pelo resultado do pedido. Finalmente, o parâmetro *allowRebuild* só é utilizado em casos onde não existe um único serviço que atenda a solicitação do usuário e especifica se composições previamente encontradas podem ser reconstruídas ou não.

Em resposta a uma requisição, um único serviço, uma composição de serviços ou uma mensagem informando que nada foi encontrado deve ser retornada. No caso de retornar uma composição, ela não necessariamente atenderá todos os objetivos especificados na requisição, visto que apenas algumas das saídas desejadas podem ter sido encontradas. Nesse caso, a composição é considerada válida porém parcialmente completa. Isso não acontece no caso em que apenas um serviço é retornado, já que para isso o serviço deve ser capaz de atender a todos os objetivos da requisição.

4.3 DESCOBERTA DE SERVIÇOS WEB

O modelo de descoberta de Serviços Web adotado é o *capability matching* (GAO *et al*., 2002). A adoção desse modelo está diretamente relacionada à capacidade de comparar conceitos ontológicos de forma eficiente com base em graus de similaridade. O algoritmo de descoberta proposto baseia-se nos requisitos funcionais de Serviços Web e leva em conta as anotações semânticas associadas com entradas, saídas e as operações do WSDL. Ele calcula a similaridade entre estes requisitos através de inferências baseadas em relações de *subsumption*. Apesar de a linguagem SAWSDL oferecer suporte básico à descrição de pré-condições e efeitos, esses atributos não foram considerados pelo algoritmo dada a baixa expressividade proporcionada pela linguagem para a descrição dos mesmos.

O algoritmo analisa o conteúdo semântico dos Serviços Web ao invés de apenas palavras-chaves. Para isso, parte-se da premissa de que os serviços sejam classificados como Serviços Web Semânticos e que seus descritores WSDL estejam anotados semanticamente utilizando SAWSDL. O algoritmo baseia-se na classificação proposta por (PAOLUCCI *et al*., 2002) para a comparação entre conceitos de ontologias, conforme apresentado na sessão 3.1. Assim sendo, a função de matching é dada por:

$$match(A,B) = \{x \mid x \in (exact, plugin, subsume, fail)\}$$

A partir do resultado da comparação entre dois conceitos A e B, o algoritmo atribui um valor que varia de acordo com a Tabela 2.

Tabela 2- Valor associado aos graus de similaridade

Grau de similaridade	Valor
Exact	0
Plugin	1
Subsume	2
Fail	3

O algoritmo proposto para a descoberta de serviços está dividido em quatro etapas: recuperação dos serviços publicados, seleção de operações de inicialização, seleção de operações alvo e seleção final de serviços, como mostrado na Figura 18.

Antes de detalhar as etapas da descoberta e a fim de fornecer uma visão geral, o algoritmo de descoberta é descrito através de um diagrama de atividades da UML 2, mostrado na Figura 17. O algoritmo tem como entrada três listas de URIs (entradasDisponiveis, saidasDesejadas e operacoesDesejadas) que especificam conceitos definidos em uma ontologia de domínio associados, respectivamente, com as entradas disponíveis, as saídas desejadas, e as operações desejadas. Inicialmente, os serviços armazenados no repositório são recuperados para que suas operações possam ser utilizadas (atividade "Recupera Operações Publicadas"). Então, as operações que podem ser invocadas usando as entradas disponíveis são escolhidas como potenciais operações de inicialização (atividade "Seleciona Operações Inicialização"), enquanto em paralelo operações que produzem pelo menos uma das saídas desejadas são escolhidas como potenciais operações-alvo (atividade "Seleciona Operações Alvo"). Posteriormente, é realizada uma análise a fim de verificar se um único serviço é capaz de atender a solicitação (decisão "Existe Único Serviço Suficiente"). Se mais de um serviço é capaz de fazê-lo, a seleção final do serviço é baseada naquele com o melhor SMD, índice que determina a diferença semântica entre o servido desejado e o serviço encontrado (atividade "Seleciona Serviço com melhor SMD").

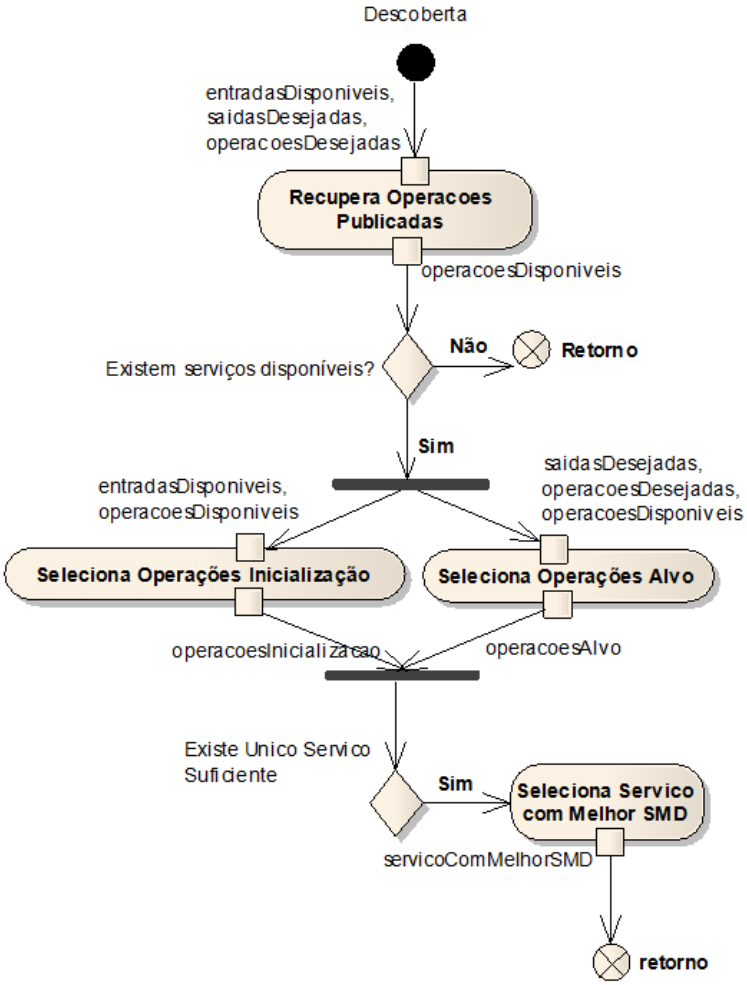

Figura 17 – Algoritmo para a composição de Serviços Web

4.3.1 Seleção de Operações de Inicialização

Após recuperar do repositório as operações disponíveis pelos serviços publicados, o algoritmo inicia sua execução com o *matching* das entradas, processo realizado durante a atividade "Seleciona Operações Inicialização". Ele seleciona, a partir de um repositório de serviços anotados semanticamente com SAWSDL, aqueles cujas

operações exigem entradas compatíveis com as entradas disponíveis na requisição do serviço. Portanto, todas as entradas de pelo menos uma das operações expostas pelo Serviço Web devem ser classificadas como *exact* ou *plugin*, a fim de que o serviço seja selecionado. Apenas esses dois graus de similaridade são aceitos, uma vez que os demais podem exigir entradas não disponíveis pelo cliente. Imagina-se uma operação que possui como parâmetro uma entrada do tipo *Veículo* e o cliente possui como entrada disponível um *Carro.* Nesse caso, a operação seria selecionada visto que *Carro* é uma especialização de *Veículo* e consequentemente a operação poderia ser executada. Já em um caso inverso, onde a operação possui um *Carro* como parâmetro de entrada e o cliente dispõem de um *Veículo*, não é possível assegurar a execução da operação, visto que nem todo *Veículo* pode ser mapeado para um *Carro.* A relação exposta pelo primeiro caso caracteriza-se como *plugin*, enquanto a demonstrada no segundo caso é do tipo *subsume.* A condição para a seleção da operação é dada por:

$$\forall x \in In : Match(Ed, Ee)_{op} <= 1$$

Onde:

- *In*: Conjunto de entradas de operação exposta pelo serviço;
- *Ed*: Entrada disponível pelo cliente;
- *Ee*: Entrada esperada pela operação pertencente ao conjunto *In*;
- *Match*(*Ed*,*Ee*): Valor resultante do *matching* da entrada disponível com a entrada esperada pela operação *op* do serviço, conforme a Tabela 2.

As operações selecionadas durante o *matching* das entradas são chamadas operações de inicialização. Elas podem ser selecionadas de Serviços Web distintos. A partir delas a execução da tarefa solicitada é iniciada, seja através de um único serviço ou através de uma composição deles. A função *selecionaOperacoesInicializacao*, exibida na linha 1 do Algoritmo 1, é responsável por selecionar as operações de inicialização. Ela recebe como parâmetros uma lista de todas as operações disponíveis no repositório de serviços e as entradas disponíveis pelo usuário. Ambas as listas são compostas por tipos de dados textuais (*Strings*) que correspondem a URI dos conceitos ontológicos. Da linha 3 até a linha 20, todas as operações são percorridas em busca daquelas que possuem todos os seus parâmetros de entrada disponíveis pelo usuário. Da linha 6 até a linha 17 todos os parâmetros de entrada de cada uma das operações

são comparados um a um com as entradas enviadas na requisição. A linha 11 verifica, por meio da função *verificaSeExactOuPlugin*, se essa comparação resulta em um grau de similaridade *exact* ou *plugin*. Se a condição, avaliada na linha 18, de que todos os parâmetros de entrada de uma operação tiverem uma entrada correspondente enviada na requisição for verdadeira, a operação é então adicionada à lista de operações de inicialização.

Algoritmo 1. *Matching* para seleção das operações de inicialização

```
1  Funcao selecionaOperacoesInicializacao(listaOperacoes,
   entradasDisponiveis)
2  Inicio
3     Para i = 1 Até listaOperacoes.tamanho Faça
4     Inicio
5        entradasCompativeis = 0;
6        Para j = 1 Até
   listaOperacoes[i].getEntradas().tamanho Faça
7        Inicio
8           existeCompativel = false;
9           Para k = i Até entradasDisponiveis.tamanho Faça
10          Inicio
               Se (verificaSeExactOuPlugin(
11                  listaOperacoes[i].anotacao,
                    entradasDisponiveis[k]))
12                   existeCompativel = true;
13          Fim
15          Se (existeCompativel)
16             entradasCompativeis = entradasCompativeis + 1;
17       Fim
18       Se (entradasCompativeis > 0 e entradasCompativeis ==
               listaOperacoes[i].getEntradas().tamanho)
19
   operacoesInicializacao.adiciona(listaOperacoes[i]);
20    Fim
21    retorna operacoesInicializacao;
22 Fim
```

4.3.2 Seleção de Operações Alvo

O *matching* semântico das operações, realizado durante a atividade "Seleciona Operações Alvo", não é realizado com os Serviços Web selecionados na etapa anterior e por isso é executado em paralelo à atividade “Seleciona Operações de Inicialização”. O algoritmo seleciona a partir dos serviços disponíveis, aqueles que têm pelo menos uma operação semelhante a uma das operações desejadas. Assim, o *matching* é executado para cada operação fornecida pelos serviços de modo a compará-las com as operações desejadas enviadas na requisição. Se o resultado dessa comparação for menor ou igual a 1 (graus de similaridade *exact* ou *plugin*), o Serviço Web é então selecionado.

A função *selecionaOperacoes*, demonstrada no Algoritmo 2, é responsável por realizar o *matching* de operações. Ela recebe como parâmetros uma lista de todas as operações disponíveis no repositório de serviços e as operações desejadas pelo usuário. Assim como no Algoritmo 1, ambas as listas são compostas pelas URIs dos conceitos ontológicos. Da linha 3 até a linha 14, todas as operações são percorridas em busca daquelas que possuem suas anotações compatíveis com ao menos uma das operações desejadas. A análise de compatibilidade é realizada na linha 8 através da comparação entre duas URIs, uma proveniente da anotação da operação e a outra da operação desejada. Caso a análise de compatibilidade resulte em um grau de similaridade *exact* ou *plugin*, a operação é então selecionada, conforme exibido na linha 13.

As anotações adicionadas ao elemento <operation> de uma descrição WSDL permitem expressar o significado semântico de cada operação oferecida pelo serviço correspondente. Assim, é possível distinguir Serviços Web que têm as mesmas características, com as mesmas entradas e saídas, mas têm significado semântico diferente. Considere um serviço que informa os carros fabricados em um determinado país e outro que retorna os carros mais vendidos no país. Ambos têm as mesmas entradas e saídas, no entanto, essas operações são funcionalmente diferentes. Por esta razão, as anotações associadas com as operações devem ser levadas em conta durante o processo de comparação. No exemplo dado, cada operação será associada com um conceito ontológico diferente que especifica a sua finalidade, permitindo que o processo de *matching* faça distinção entre elas.

Algoritmo 2. *Matching* baseado nas operações dos Serviços Web

```
1  Funcao selecionaOperacoes(listaOperacoes,
   operacoesDesejadas)
2  Inicio
3     Para i = 1 Até listaOperacoes.tamanho Faça
4     Inicio
5           existeCompativel = false;
6           Para k = i Até operacoesDesejadas.tamanho Faça
7           Inicio
               Se (verificaSeExactOuPlugin(
8                    listaOperacoes[i].anotacao,
                     operacoesDesejadas[k]))
9                         existeCompativel = true;
10
11          Fim
12          Se (existeCompativel)
13            operacoesSelecionadas.adiciona(listaOperacoes[i]);
14    Fim
15    retorna operacoesSelecionadas;
16 Fim
```

O *matching* das saídas, por outro lado, é realizado apenas com os Serviços Web selecionados durante o *matching* das operações e por isso é realizado durante a atividade "Seleciona Operações Alvo". Ou seja, a saída é estritamente relacionada à semântica da operação, uma vez que o serviço precisa ser selecionado na etapa 2 para posteriormente ser selecionado na etapa 3. As saídas das operações selecionadas são comparadas com os resultados solicitados na requisição. Se na lista de saídas solicitadas houver uma correspondência com valor inferior ou igual a 1, a operação é selecionada e consequentemente o Serviço Web também. No final desta etapa, há apenas Serviços Web que possuem operações classificadas como *exact* ou *plugin* que têm saídas também classificadas como *exact* ou *plugin*.

Algoritmo 3. *Matching* para a seleção das Operações Alvo

```
1   Funcao selecionaOperacoesAlvo(operacoesDisponiveis,
    saidasDesejadas, operacoesDesejadas)
2   Inicio
        operacoes = selecionaOperacoes(operacoesDisponiveis,
                            operacoesDesejadas)
3       Para i = 1 Até operacoes.tamanho Faça
4       Inicio
5           Para j = 1 Até operacoes[i].getSaidas().tamanho Faça
6           Inicio
7               existeCompativel = false;
8               Para k = i Até saídas.tamanho Faça
9               Inicio
10                  Se (verificaSeExactOuPlugin(
                            operacoes[i].anotacao, saidas [k]))
11                      existeCompativel = true;
13              Fim
14          Fim
15          Se (existeCompativel)
16                operacoesAlvo.adiciona(operacoes[i]);
17      Fim
18      retorna operacoesAlvo;
19  Fim
```

As operações selecionadas durante o *matching* das saídas são chamadas operações alvo. Essas operações são responsáveis por retornar o que foi solicitado na requisição. Caso uma única operação seja suficiente para atender uma requisição, ela será tanto uma operação de inicialização como uma operação alvo. No caso das composições, parte-se de uma ou mais operações de inicialização para encontrar as operações alvo. A função *selecionaOperacoesAlvo*, demonstrada no Algoritmo 3, é responsável por realizar o *matching* de saídas e selecionar as operações alvo. Ela recebe como parâmetros uma lista de todas as operações selecionadas durante o *matching* de operações bem como as saídas e operações desejadas pelo usuário. Assim como nos algoritmos anteriores, as listas são compostas pelas URIs dos conceitos ontológicos. Da linha 3 até a linha 17, todas as operações são percorridas em busca daquelas que possuem suas saídas compatíveis com ao menos uma das saídas desejadas. Basta que apenas uma das saídas possua um grau de similaridade *exact* ou *plugin* quando

comparadas com as saídas desejadas, conforme linha 10, para que a operação seja selecionada como operação alvo, conforme linha 16.

4.3.3 Seleção Final de Serviços Web

Após executar as três primeiras etapas, o algoritmo verifica se algum serviço foi selecionado em ambas as atividades "Seleciona Operações de Inicialização" e "Seleciona Operações Alvo". Essa é a atividade de "Seleção Final de Serviços" e é a quarta e última executada pelo algoritmo. Caso exista uma operação que tenha sido selecionada durante as duas atividades, o algoritmo verifica se a operação é capaz de retornar todas as saídas solicitadas. Se sim, o serviço que oferece a operação é elegível a retornar como resposta da requisição. Nos casos em que mais de um serviço é elegível, são calculados os graus de divergência semântica das entradas, das saídas e da operação de cada um deles de maneira isolada a fim compará-los e selecionar o melhor semanticamente, conforme as seguintes funções:

$$SMDI(O) = \frac{\sum_{i=1}^{Nin} Min}{Nin} \quad (1)$$

$$SMDO(O) = \frac{\sum_{i=1}^{Nout} Mout}{Nout} \quad (2)$$

$$SMDP(O) = Mop \quad (3)$$

Onde:

- *SMDI*(*O*): Grau de divergência semântica (*Semantic Mismatch Degree of Inputs*) das entradas da operação *O*;
- *SMDO*(*O*): Grau de divergência semântica (*Semantic Mismatch Degree of Outputs*) das saídas da operação *O*;
- *SMDP*(*O*): Grau de divergência semântica (*Semantic Mismatch Degree of Operation*) da operação *O*;
- *Nin*: Número de entradas da operação *O*;
- *Nout*: Número de saídas da operação *O*;
- *Min*: Valor resultante do *matching* semântico de cada entrada da operação *O*, de acordo com a Tabela 2;
- *Mout*: Valor resultante do *matching* semântico de cada saída da operação *O*, de acordo com a Tabela 2;
- *Mop*: Valor resultante do *matching* semântico da operação *O*, de acordo com a Tabela 2;

A equação (1) leva em conta o resultado do *matching* de cada entrada da operação para calcular o grau de divergência semântica das entradas. O mesmo é feito na equação (2), onde o resultado do *matching* de cada saída da operação é somado a fim de calcular o grau de divergência semântica das saídas. Por fim, a equação (3) utiliza o resultado do *matching* da operação para avaliar a divergência semântica da operação. Uma vez calculados os graus de divergência, eles são então comparados com os graus das demais operações selecionadas. O SMDI de uma operação é comparado com o SMDI das demais. O mesmo é feito com os graus SMDO e SMDP de cada operação. A operação que possuir os menores graus de divergência é retornada ao cliente. Considerando que a operação que possuir o menor SMDI, por exemplo, não necessariamente será a mesma com o menor SMDO, a operação selecionada será aquela que possuir o menor SMDO, seguida pelo menor SMDP e por fim o menor SMDI como critérios de desempate.

Quando o algoritmo não é capaz de encontrar um único serviço capaz de atender a requisição a construção de um grafo de composições é requerida a fim de encontrar caminhos que conduzam às saídas necessárias. Tem-se inicio então o algoritmo de composição de Serviços Web.

4.4 COMPOSIÇÃO DE SERVIÇOS WEB

A abordagem utilizada para a construção automática de composições de Serviços Web baseia-se em técnicas de workflow. Uma composição é caracterizada por caminhos em um grafo direcionado que iniciam com operações selecionadas durante o *matching* de entradas (operações de inicialização) e atingem pelo menos uma das operações selecionadas durante o *matching* de saídas (operações alvo). Assim, sabe-se por onde começar a execução de uma composição e quando ela termina. Cada nó do grafo é representado por um Serviço Web e os vértices abstraem as ligações semânticas entre eles.

Antes de iniciar a construção do grafo de composições, as saídas das operações de inicialização, agora utilizadas como entradas, são adicionadas e somadas à lista de entradas disponíveis. Inicialmente essa lista era composta apenas pelas entradas enviadas na requisição do serviço. Agora ela é composta pelas saídas das operações de inicialização bem como pelas entradas enviadas na requisição.

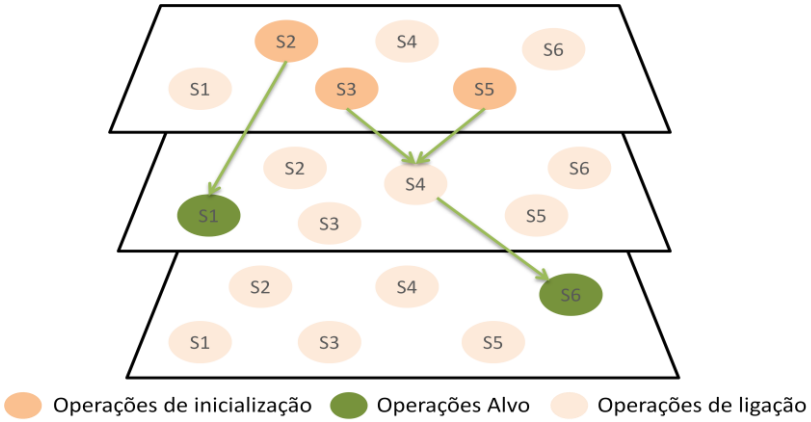

Figura 18 – Grafo de composições em camadas

O algoritmo usa camadas de operações, como demonstrado na Figura 18, para representar o grafo de composições. A primeira camada é composta pelas operações de inicialização. As outras camadas emergem na medida em que o algoritmo itera sobre o conjunto de serviços disponíveis. Uma iteração consiste em estabelecer a correspondência semântica das entradas necessárias por parte dos serviços com as entradas disponíveis. As operações que são selecionadas formam uma nova camada. Como resultado, suas saídas são adicionadas à lista de entradas disponíveis para utilização em iterações subsequentes. Assim, novos caminhos podem surgir no grafo a cada iteração.

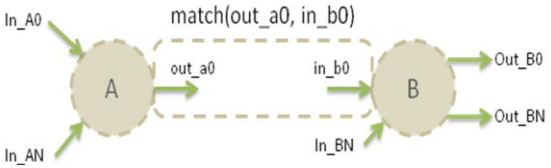

Figura 19 – Relação semântica entre duas operações de Serviços Web

As operações de uma camada estão relacionadas com as operações da camada superior por meio da origem das entradas. Como ilustrado na Figura 19, as entradas da operação B podem vir de uma operação A ou da lista de entradas enviadas na requisição. Eventualmente, as entradas de uma operação podem vir de uma ou mais operações, tal como ocorre com o serviço C, representado na Figura 20.

Isso ocorre quando serviços que estavam sendo executados em paralelo encontram um ponto de intersecção.

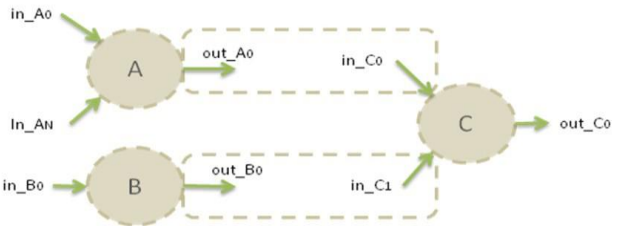

Figura 20 – Origem das entradas de uma operação de Serviço Web

Na medida em que o algoritmo progride, diversos caminhos existentes no grafo podem levar a uma das saídas desejadas. Cada caminho pode ou não encontrar uma operação que atende as especificidades da requisição. Uma vez que uma operação é selecionada para compor uma camada, o algoritmo verifica se ela pertence ou não ao conjunto das operações alvo. Em caso positivo, significa que uma composição foi encontrada. No entanto, o caminho do grafo continua a ser construído à procura de novas saídas, visto que nem todas as saídas desejadas podem ter sido encontradas ou até mesmo porque podem existir saídas semanticamente melhores para substituir alguma das que já foram encontradas.

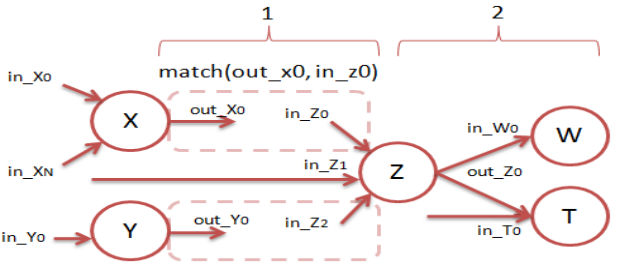

Figura 21 – Exemplo de caminhos do grafo de composições

Cada caminho no grafo conduz a uma única saída, mesmo que várias saídas desejadas estejam no mesmo caminho do grafo. A Figura 21 mostra uma situação deste tipo, onde as operações Z e T fornecem saídas desejadas e, apesar de estarem no mesmo caminho do grafo, o algoritmo os trata como dois caminhos diferentes, um composto pelas operações $Y \rightarrow Z$ e outro pelas operações $Y \rightarrow Z \rightarrow T$.

Um caminho é interrompido apenas quando não é possível estabelecer uma ligação semântica com outras operações, quando o

caminho atinge a profundidade máxima especificada no momento da requisição ou quando o tempo máximo de espera para a descoberta e composição de serviços é excedido. Ou seja, o algoritmo possui três critérios de parada. A ocorrência de loops poderia ser um quarto critério. Um loop é identificado quando uma mesma operação aparece repetidamente no mesmo caminho. Como o algoritmo evita que aconteçam loops de serviços, retirando os que já foram utilizados por um caminho a cada iteração, esse critério não é utilizado. Algumas abordagens utilizam ainda como critério de parada a condição de que todas as saídas tenham sido encontradas. Neste trabalho, esse critério não é utilizado visto que o objetivo é encontrar composições com a melhor qualidade semântica possível. Se a construção de uma composição é interrompida assim que todas as saídas forem encontradas, deixa-se de avaliar o restante dos serviços disponíveis. Com isso ganha-se em tempo de reposta em detrimento de saídas semanticamente mais adequadas do que as encontradas.

No final da construção do grafo, quer por atingir o tempo limite ou por não conseguir mais estabelecer relações semânticas entre os Serviços Web, o algoritmo analisa se dois ou mais caminhos levam à mesma saída. Caso isso aconteça, a seguinte função é aplicada para cada caminho, a fim de selecionar aquele com o menor grau de divergência semântica:

$$SMD(P) = \sum_{i=1}^{Ns} Min_s * \alpha + Mout_p * \beta + Mop_p * \omega$$

Onde:

- *SMD*(*P*): Grau de divergência semântica (*Semantic Mismatch Degree*) do caminho *P*;
- *Ns*: Número de operações no caminho *P*;
- *Mins*: Valor resultante do *matching* semântico das entradas de cada operação do caminho P, de acordo com a Tabela 2;
- *Moutc*: Valor resultante do *matching* semântico da saída do caminho *P*, de acordo com a Tabela 2;
- *Mops*: Valor resultante do *matching* semântico da operação final do caminho *P*, de acordo com a Tabela 2;
- α: Peso associado às entradas;
- β: Peso associado às saidas;
- ω: Peso associado às operações;

A equação *SMD*(*P*) leva em conta o resultado do *matching* das entradas de todos os serviços que pertencem ao caminho, mas apenas

considera o resultado do *matching* da operação e da saída do último serviço do caminho. Isto porque as saídas que servem como entradas para outras operações já são consideradas no valor Min. Além disso, a função aplica diferentes pesos aos critérios avaliados. A distribuição de peso ocorre de acordo com a importância de cada critério e podem ser atribuídos valores que variam de 0 a 1. A soma de todos os pesos deve ser igual a 1.

O resultado da equação *SMD* é representado através de um número decimal que indica o grau de divergência semântica entre os serviços que compõem o caminho. Quanto menor for esse número, mais similares e, portanto compatíveis, serão os serviços que interagem entre si. Em um caso onde a comparação semântica entre todos os conceitos envolvidos na composição resulta no grau de similaridade semântica *exact*, o resultado da função *SMD* seria zero. Em outras palavras, não existiria divergência semântica entre os Serviços Web. No lado oposto, o grau máximo de divergência semântica pode variar de acordo com a quantidade e a similaridade entre os conceitos avaliados durante a construção da composição.

O *matching* da saída bem como o *matching* das operações de um caminho são considerados igualmente importantes e possuem um peso associado maior do que o *matching* das entradas. Isso porque juntos eles definem o que irá retornar da execução da composição, que corresponde ao que o requerente deseja. Se após a aplicação da equação ainda assim existirem dois ou mais caminhos com o mesmo grau de divergência semântica, um quarto critério é aplicado:

$$Cost(P) = SMD(P) + Depth(P)$$

Onde:

- *Cost*(*P*): Custo do caminho *P*;
- *SMD*(*P*): Grau de divergência semântica do caminho *P*;
- *Depth*(*P*): Profundidade do caminho *P*.

A fim de priorizar caminhos semanticamente melhores, o critério de profundidade é aplicado apenas em um segundo momento. A profundidade é um critério de desempate para a qualidade da composição, uma vez que quanto maior o número de serviços que integram uma composição, maior é a probabilidade de um deles tornar-se indisponível e impedir a execução da composição. Atributos não funcionais de Qualidade de Serviço (QoS) não são considerados no escopo deste trabalho. Apesar disso, poderiam ser adicionados à função de custo critérios como o tempo de resposta de um serviço e a própria

disponibilidade do mesmo. Esses critérios ajudariam a filtrar e selecionar serviços com melhores performances, disponibilidades, confiabilidade, etc. De qualquer modo, avaliar e confrontar a profundidade de dois ou mais caminhos demonstrou-se suficiente para selecionar o melhor caminho em termos de qualidade da composição.

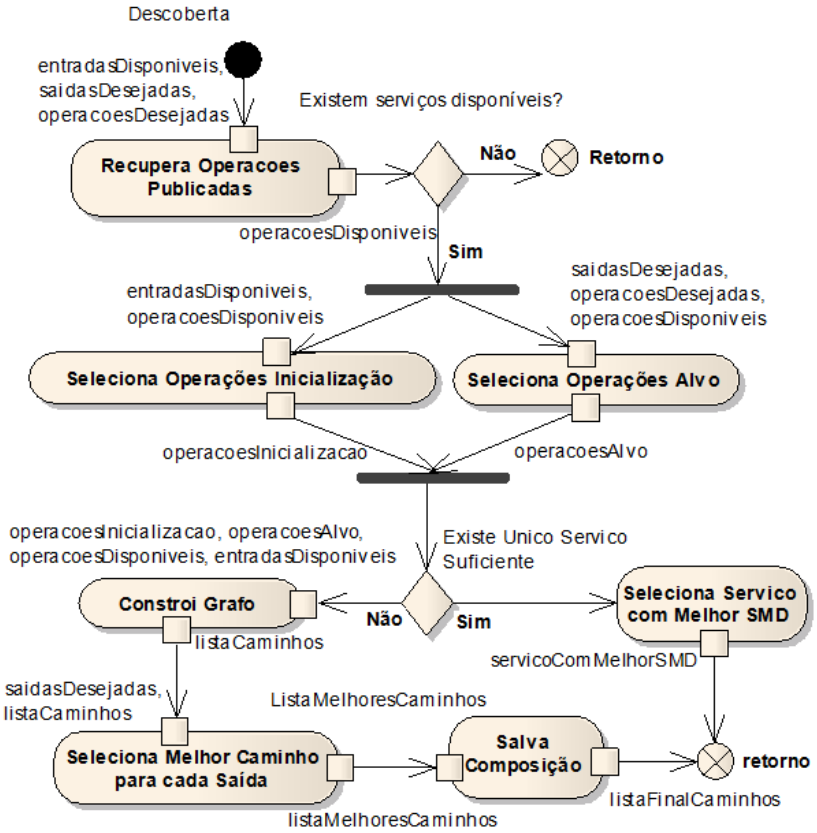

Figura 22 – Algoritmo para a composição de Serviços Web

Uma vez calculado o custo de cada caminho, o algoritmo seleciona aquele com o menor valor. Isso significa que o caminho que fornecer o menor grau de divergência semântica das suas entradas, da sua saída e de suas operações, juntamente com a menor profundidade, será selecionado. Esse cálculo é realizado apenas quando há dois ou mais caminhos que fornecem saídas semelhantes. Caso contrário, o

cálculo dessas métricas é desnecessário e o único caminho obtido é selecionado diretamente.

A Figura 22 complementa a Figura 17 com as atividades necessárias para a realização da composição. Quando a análise, a fim de verificar se um único serviço é capaz de atender a solicitação (decisão "Existe Único Serviço Suficiente"), retorna falso, o algoritmo parte para a construção de um grafo de composições. Um grafo é construído para identificar os caminhos que ligam uma operação de inicialização a uma operação alvo (atividade "Constroi Grafo"). Para cada saída desejada, o algoritmo seleciona apenas o caminho com o melhor SMD (atividade "Seleciona Melhor Caminho para cada Saída"). Finalmente, a composição resultante é obtida a partir da união dos melhores caminhos que produzem os resultados desejados e é armazenada para ser utilizada em futuras requisições (atividade "Salva Composição"). O SMD global da composição C é calculado de acordo com a seguinte função:

$$SMD(C) = \sum_{i=1}^{N} SMD(P_i)$$

Onde:

- $SMD(C)$: Grau de divergência semântica da composição C;
- $SMD(P)$: Grau de divergência semântica do caminho P_i;
- N: Número de caminhos da composição C.

O somatório do SMD de cada caminho pertencente à composição resulta no SMD global. O SMD global é utilizado apenas como uma referência para mensurar o grau de incompatibilidade. O Algoritmo 4 representa, através de um pseudocódigo, o algoritmo para a descoberta e composição de Serviços Web Semânticos. Na linha 3 do algoritmo, todas as operações dos serviços publicados no repositório são recuperadas do banco de dados relacional. Nenhum critério é utilizado para filtrar as operações, visto que inicialmente não é possível determinar quais estarão ou não envolvidas em uma composição. A partir de então as funções *selecionaOperacoesInicializacao* (linha 4) e *selecionaOperacoesAlvo* (linha 5), vistas anteriormente, são executadas para encontrar as operações de inicialização e as operações alvo, respectivamente. Em seguida a função *existemServicosUnicosSuficientes*, demonstrada na linha 6, verifica se existem serviços que por si só retornam todas as saídas desejadas. Independentemente de o processo de descoberta evoluir ao ponto de recorrer à composição de serviços para atender a uma requisição, o algoritmo sempre irá realizar a sua execução até a linha 6 do algoritmo.

Algoritmo 4. Composição de Serviços Web

```
1   Funcao discoveryComposition(entradasDisponiveis,
                    saidasDesejadas, operacoesDesejadas)
2   Inicio
3       operacoesDisponiveis = getOperacoesPublicadas();
4       operacoesInicializacao = selecionaOperacoesInicializacao(
                operacoesDisponiveis, entradasDisponiveis)
5       operacoesAlvo = selecionaOperacoesAlvo(
                operacoesDisponiveis, saidasDesejadas,
                operacoesDesejadas);
6       servicosUnicos = existemServicosUnicosSuficientes(
                operacoesInicializacao, operacoesAlvo,
    saidasDesejadas)
7       Se (servicosUnicos.tamanho > 0)
8           retorna getServicoComMelhorSMD(servicosUnicos);
9       Senao
10      Inicio
11          caminhos = constroiGrafo(operacoesInicializacao,
                operacoesAlvo, saidasDesejadas,
                entradasDisponiveis, operacoesDesejadas)
12          Para i = 0 Até saidasDesejadas.tamanho Faça
13          Inicio
14              melhoresCaminhos.adiciona(
                        getCaminhoComMelhorSMD(
                            caminhos, saidasDesejadas[i]));
15          Fim
16          salvaComposicoesEncontradas();
17          retorna melhoresCaminhos;
18      Fim
19  Fim
```

Caso exista mais de um Serviço Web capaz de fornecer o que foi solicitado com base no que existe disponível, o algoritmo seleciona aquele que possuir o melhor SMD e o retorna como resposta da requisição. Se isso acontecer, o algoritmo não precisará iniciar o processo de composição. Caso contrário, o algoritmo inicia a construção do grafo de composições através da função *constroiGrafo* da linha 11. Essa função, como o próprio nome sugere, é responsável por efetivamente estabelecer as relações semânticas entre as operações dos serviços a fim de identificar possíveis composições. Ao final da

construção do grafo, para cada uma das saídas desejadas o algoritmo verifica se existe algum caminho que a retorne. Se para uma determinada saída existir apenas um caminho capaz de fornecê-la, esse caminho será retornado para o cliente que realizou a requisição. Caso exista mais de um caminho que produza a mesma saída, o algoritmo aplica a fórmula do SMD, durante a execução da função *getCaminhoComMelhorSMD* da linha 14, para selecionar aquele com o menor grau de divergência semântica.

Algoritmo 5. Construção do grafo de composições de Serviços Web

```
1   Funcao constroiGrafo(operacoesInicializacao,
                operacoesDisponiveis, entradasDisponiveis)
2   Inicio
3       Para i = 0 Até operacoesInicializacao.tamanho Faça
4       Inicio
5           entradasDisponiveis.adiciona(
                operacoesInicializacao[i].getSaidas())
6           grafo.adicionaNo(operacaoInicializacao[i]);
7           operacoesDisponiveis.remove(operacaoInicializacao [i]);
8           constroiCaminho(grafo, operacoesDisponiveis,
                entradasDisponiveis)
9       Fim
10  Fim
```

A função *constroiGrafo*, dada a sua importância para o algoritmo de composição, foi detalhada no Algoritmo 5. Inicialmente, ela recebe como parâmetros de entrada uma listagem com todas as operações de inicialização identificadas na descoberta, todas as operações oferecidas pelos serviços publicados no repositório e todas as entradas disponíveis até o momento. Cada operação de inicialização corresponde a um ponto de partida do grafo, ou seja, os nós dos quais originarão os demais nós. Elas constituem a primeira camada de serviços do grafo e são, portanto, os nós de origem. O algoritmo interage com as operações desse conjunto e a cada iteração uma operação é selecionada a fim de criar um novo caminho no grafo. Somente após criar esse caminho, que pode se ramificar em outros novos caminhos, o algoritmo volta para iniciar uma nova iteração. Ao final da iteração, cada operação de inicialização terá dado início a nenhum, um ou mais de um caminho que, eventualmente, poderão ter nós em comum com outros caminhos. Em outras palavras, pode não existir uma operação capaz de se integrar com uma operação

de inicialização e assim não dar início a um novo caminho, como também poderão existir múltiplos caminhos partindo de uma mesma operação de inicialização.

Algoritmo 6. Construção de caminho do grafo de composições

```
1   Funcao constroiCaminho(grafo, operacoesDisponiveis,
                entradasDisponiveis, ultimaOperacao)
2   Inicio
        Se (naoAtingiuTempoLimite)
        Inicio
3           Para i = 0 Até operacoesDisponiveis.tamanho Faça
4           Inicio
5               Se (possuiEntradasRequisitadas(
                        operacoesDisponiveis[i], entradasDisponiveis))
6               Inicio
7                   grafo.adicionaNo(operacoesDisponiveis[i]);
8                   grafo.adicionaVertice(ultimaOperacao,
                        operacoesDisponiveis[i])
9                   entradasDisponiveis.adiciona(
                        operacoesDisponiveis[i].getSaidas())
10                  operacoesDisponiveis.remove(
                        operacoesDisponiveis[i]);
11                  constroiCaminho(grafo, operacoesDisponiveis,
                        entradasDisponiveis, operacoesDisponiveis[i]);
12              Fim
13      Fim
14  Fim
```

As operações que são utilizadas para compor um caminho do grafo que não são nem operações de inicialização e nem operações alvo são chamadas operações de ligação. Sua função consiste em interligar operações de modo que uma operação de inicialização possa alcançar uma operação alvo. Durante a construção do grafo, uma operação de inicialização pode ser utilizada como operação de ligação. Isso acontece em casos onde uma operação de inicialização aparece no caminho iniciado por outra. O mesmo acontece com as operações alvo, que em determinados momentos podem assumir a condição de operações de ligação. Isso acontece quando duas operações alvo aparecem no mesmo caminho. Apesar de estarem no mesmo caminho, o algoritmo os separa em dois caminhos diferentes, cada um terminando em uma das

operações alvo. Consequentemente, em um desses caminhos uma das operações alvo irá ser utilizada como operação de ligação. Um caminho formado por três operações, $Y \rightarrow Z \rightarrow T$, onde Z e T são consideradas operações alvo, será separado nos caminhos $Y \rightarrow Z$ e $Y \rightarrow Z \rightarrow T$. No segundo caminho a operação Z assume a condição de operação de ligação.

A cada iteração, as saídas da operação que inicia o caminho são adicionadas a lista de entradas disponíveis, conforme especificado na linha 5 do Algoritmo 5, para que possam ser utilizadas como entradas para as próximas operações. Em seguida um novo nó é adicionado ao grafo, linha 6, caracterizando o nó de origem. Na linha 7 a operação, que já foi adicionada ao grafo, é removida da lista de operações disponíveis para que não seja mais utilizada no caminho que está iniciando. Isso evita que sejam identificados loops durante a construção da composição. Por fim, na linha 8, a função *constroiCaminho* tem o objetivo de identificar todas as operações que podem ser interligadas ou conectadas semanticamente ao nó que foi adicionado ao grafo.

A função *constroiCaminho*, exibida pelo Algoritmo 6, é chamada recursivamente até que um dos critérios de parada do algoritmo seja satisfeito. Inicialmente é feita uma verificação sobre o parâmetro timeout enviado na requisição para identificar se ele já foi ultrapassado. O algoritmo só continua com a construção do caminho caso essa condição seja falsa. Para cada uma das operações pertencentes ao conjunto das operações disponíveis o algoritmo avalia se existe alguma operação cujas entradas estão presentes na lista de entradas disponíveis, função *possuiEntradasRequisitadas* da linha 5. Essa função é bastante similar à função *selecionaOperacoesInicializacao*, visto que as duas realizam o *matching* de entradas para selecionar as operações. Uma vez que exista uma operação que tenha sido selecionada, ela é então adicionada ao grafo na forma de um novo nó e um vértice é estabelecido entre ele e o nó adicionado anteriormente, nas linhas 7 e 8 respectivamente. Em seguida, na linha 9, as saídas dessa operação são adicionadas à lista de entradas, e a operação é removida da lista de operações disponíveis. Com isso, uma nova camada do grafo é criada e o nível de profundidade do caminho é aumentado. Por fim, a função é invocada novamente a fim de encontrar novas operações que se interliguem ao último nó adicionado ao grafo. Esse processo ocorre até que uma das condições de parada seja satisfeita. Vale ressaltar que essa função é capaz de criar caminhos que posteriormente serão executados em paralelo.

Após finalizar a seleção dos melhores caminhos com base no índice do SMD, o algoritmo os armazena em um repositório de composições, por meio da função *salvaComposicoesEncontradas* (linha 16 do Algoritmo 4), a fim de reutilizá-los em futuras requisições. Em outras palavras, todas as composições descobertas como resultado de uma requisição são armazenadas em um repositório de composições. Antes de iniciar todo o processo de composição quando serviço é solicitado, a abordagem verifica a existência de alguma composição que atenda às necessidades de solicitação. Se existir, esta composição é retornada para o solicitante logo após uma validação.

A validação de composições consiste em verificar a existência dos serviços envolvidos. Este processo é necessário porque os Serviços Web podem ser removidos do repositório onde foram inicialmente publicados. Se todos os serviços estiverem presentes no repositório, a composição é retornada, caso contrário, um processo de nova descoberta é iniciado.

Há também o caso em que novos Serviços Web são publicados e, de alguma forma, melhoram as composições já criadas. Em tais casos, quando é solicitado um serviço para o qual já foi criada uma composição, o algoritmo verifica se algum novo serviço foi publicado após a criação da composição e reexecuta o processo de descoberta, a fim de identificar qualquer eventual melhoria. No entanto, esta reconstrução pode ser evitada através do parâmetro *allowRebuild* enviado na requisição. Se o valor especificado nesse parâmetro for verdadeiro, a reconstrução da composição é realizada, caso contrário, a composição existente é retornada.

Não faz parte do escopo deste trabalho realizar a execução das composições. Elas são descobertas e retornadas para quem as solicitou no formato de XML, conforme modelo exemplificado na Figura 23. Uma composição, representada pelo elemento <composicao>, é formada por uma listagem de elementos <saida> que abstraem cada uma das saídas desejadas. Cada elemento <saida> possui uma referência para a URI da saída e é composto por uma série de operações, elemento <operacao>, responsáveis por retornar a saída em questão. O algoritmo trabalha com apenas um caminho para cada saída em virtude da aplicação da função SMD. Cada elemento <operação> é composto por três atributos: ordem, nome e wsdl. O primeiro corresponde à ordem de execução da operação dentro do caminho, o segundo refere-se ao nome da operação no WSDL e o terceiro contém a referência para o descritor WSDL do Serviço Web que disponibiliza a operação. A Figura 23

mostra o exemplo de uma composição formada por dois caminhos, um composto por quatro operações e outro por três.

```
<composicao>
    <saida uri="http://ontologia.com.br/mestrado.owl#SolicitacaoDefesa">
        <operacao
            ordem="1" nome="entregarDocumentacao"
            wsdl="http://mestrado.com.br/defesa.wsdl">
        </operacao>
        <operacao
            ordem="2" nome="agendarData"
            wsdl="http://mestrado.com.br/defesa.wsdl">
        </operacao>
        <operacao
            ordem="3" nome="enviarConviteMembrosBanca"
            wsdl="http://mestrado.com.br/defesa.wsdl">
        </operacao>
        <operacao
            ordem="4" nome="enviarCopiaDissertacaoMembrosBanca"
            wsdl="http://correios.com.br/sedex.wsdl">
        </operacao>
    </saida>
    <saida uri="http://ontologia.com.br/mestrado.owl#ReservaSala">
        <operacao
            ordem="1" nome="selecionarSala"
            wsdl="http://mestrado.com.br/reservasalas.wsdl">
        </operacao>
        <operacao
            ordem="2" nome="agendarData"
            wsdl="http://mestrado.com.br/defesa.wsdl">
        </operacao>
        <operacao
            ordem="3" nome="marcarHorario"
            wsdl="http://mestrado.com.br/reservasalas.wsdl">
        </operacao>
    </saida>
</composicao>
```

Figura 23 – Exemplo de resposta com uma composição formada por dois caminhos

Percebe-se que a operação **agendarData** aparece na listagem dos dois caminhos, representando um paralelismo. Uma operação pode estar presente em um ou mais caminhos, desde que sejam diferentes. Cada caminho possui sua própria ordem de execução. Não importa a quantidade de caminhos retornados como resposta da requisição, pois cada caminho é responsável por fornecer, de maneira única, uma das saídas desejadas. O objetivo da abordagem limita-se a descobrir as composições, informar quais as operações que a constituem e a sequência na qual devem ser executadas. A forma como serão executadas e o controle sobre a troca de dados entre elas fica a critério do cliente que realizou a requisição.

5 IMPLEMENTAÇÃO E AVALIAÇÃO DA ABORDAGEM

Com o objetivo de avaliar o funcionamento da abordagem proposta, o protótipo de uma ferramenta para a descoberta e composição de serviços, chamado de SWSComposer, foi desenvolvido. Esse capítulo irá apresentar a arquitetura dos componentes que compõem o protótipo, bem como os resultados de testes de avaliação realizados com o ambiente desenvolvido.

5.1 METODOLOGIA DE DESENVOLVIMENTO

Durante o desenvolvimento do SWSComposer, buscou-se verificar a viabilidade de construir um mecanismo que fosse capaz de descobrir Serviços Web em tempo de requisição e posteriormente realizar composições. Comprovado através de testes que seria possível o desenvolvimento do mecanismo utilizando a linguagem Java, passou-se a definir os requisitos do projeto.

O levantamento dos requisitos começou pela análise de projetos e abordagens já existentes. Tomou-se como base uma série de trabalhos e buscou-se identificar características comuns a eles a fim de mapear as formas como cada um realizava os processos de descoberta e composição de Serviços Web. Em seguida, foram identificadas falhas e pontos de melhorias que poderiam ser tratados pelo projeto.

5.2 ARQUITETURA DO PROTÓTIPO

O protótipo foi projetado para ser compatível tanto com a versão 1.1 como a 2.0 do WSDL. Isso quer dizer que o protótipo é capaz de extrair as anotações semânticas e compor Serviços Web que estejam descritos com qualquer versão da linguagem WSDL. A preocupação de manter essa compatibilidade é justificada pelo aumento da abrangência da ferramenta, visto que mais Serviços Web podem ser considerados durante os processos de descoberta e composição. A estrutura e as tecnologias utilizadas para desenvolver o protótipo asseguram a sua usabilidade por sistemas desenvolvidos em diferentes linguagens, além de garantir o baixo acoplamento entre eles. Em outras palavras, o protótipo desenvolvido é independente de linguagem de programação e por isso proporciona que diferentes sistemas compartilhem e utilizem as funcionalidades oferecidas por ele.

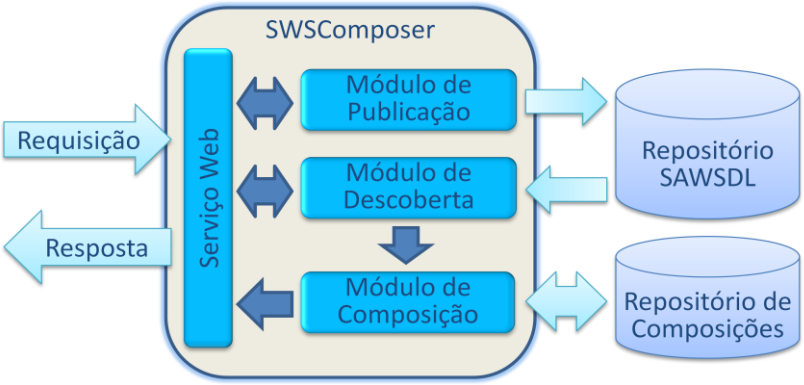

Figura 24 – Arquitetura do SWSComposer

A arquitetura do SWSComposer é ilustrada na Figura 24. O componente Serviço Web disponibiliza apenas duas operações: uma para a solicitação de serviços e a outra para a publicação de serviços. O fluxo de comunicação entre os componentes se dá a partir do recebimento de uma requisição. A operação de publicação de serviços recebe como parâmetro o WSDL do serviço a ser publicado e o encaminha para o Módulo de Publicação. Esse módulo realiza, dentre as tarefas já vistas anteriormente, a validação do arquivo recebido. São aceitos tanto serviços descritos com WSDL 1.1 como 2.0. Se o arquivo for válido, o *parsing* é feito e as anotações extraídas são armazenadas no Repositório SAWSDL ilustrado na Figura 24. Como resultado da operação, uma mensagem de sucesso retornará caso tenha sido possível publicar o serviço ou uma mensagem de erro em caso de insucesso.

A operação de solicitação de serviços, por sua vez, aceita requisições que estejam no modelo definido na Seção 4.3. Vale destacar que os valores aceitos para o parâmetro profundidade podem ser -1, 1 ou mais de um. Caso seja passado o valor -1, o protótipo não levará em conta esse parâmetro como critério de parada do algoritmo. Em outras palavras, serão aceitas composições com qualquer profundidade. Outro ponto a se destacar são os valores e a medida do parâmetro *timeout*. Os valores devem ser expressos em milissegundos e, caso esse parâmetro não deva ser considerado como critério de parada durante a composição, o valor -1 deve ser utilizado a exemplo do parâmetro profundidade.

Uma vez recebida a requisição, ela é encaminhada para o Módulo de Descoberta. Como atribuições desse módulo estão a recuperação das

informações dos serviços publicados no Repositório SAWSDL, a realização do *matching* de operações, entradas e saídas dos serviços e a verificação da existência de um único serviço que atenda a solicitação. Em caso positivo, esse módulo retorna ao Serviço Web que o invocou o serviço encontrado e encerra o processo de descoberta. Caso contrário, as informações recuperadas do repositório são mantidas em memória e encaminhadas para o Módulo de Composição.

Quando nenhum serviço sozinho é capaz de satisfazer o pedido, o Módulo de Composição é requisitado para gerar/criar composições com base nas relações entre os conceitos associados às entradas, saídas e operações. Antes de iniciar o processo de composição propriamente dito, uma consulta ao Repositório de Composições é realizada a fim de verificar se existe alguma capaz de oferecer as saídas desejadas. Se houver, a composição é retornada, senão a construção do grafo de composições tem inicio. No final do processo, as composições descobertas são armazenadas no Repositório de Composições e devolvidas ao cliente através do Serviço Web no formato de um documento XML visto anteriormente. O resultado de uma solicitação pode conter nenhum, um ou múltiplos caminhos que conduzem às saídas desejadas. Vários caminhos podem ser encontrados e selecionados, desde que cada um atenda parte da solicitação. Se dois caminhos que conduzem à mesma saída são encontrados, apenas um será selecionado e devolvido com base nos critérios propostos e detalhados na Seção 4.5.

A infraestrutura proposta visa, através da integração entre os módulos, que o cliente que a utiliza possa encontrar Serviços Web de modo que não dependa única e exclusivamente de um único serviço. Caso o cliente utilize um Serviço Web que porventura se torne inexistente, o protótipo saberá encontrar um novo serviço capaz de substituí-lo.

A estrutura do protótipo está organizada em dez classes organizadas em cinco pacotes:

- *org.swscomposer.webservice;*
- *org.swscomposer.publish;*
- *org.swscomposer.discovery;*
- *org.swscomposer.composition;*
- *org.swscomposer.repository.*

Com o objetivo de auxiliar na construção do protótipo proposto, além de obter o máximo de aproveitamento na reutilização de código, foi utilizada a linguagem de programação Java, mais especificamente a

plataforma Java EE, e selecionadas ferramentas e bibliotecas para integrar o processo de desenvolvimento. A IDE Eclipse Indigo[6] auxiliou na construção dos componentes bem como no desenvolvimento de aplicações de testes para as funcionalidades.

O pacote *org.swscomposer.webservice* contém as classes Java necessárias para disponibilizar o Serviço Web do SWSComposer que receberá as requisições de descoberta de composição e publicação. A proposta deste serviço é receber requisições que obedeçam a quantidade, a sequência e os tipos de parâmetros definidos e retornar o resultado de acordo com o tipo de solicitação. O servidor de aplicações Tomcat 6[7] foi utilizado para manter o serviço ativo, à espera de requisições. A escolha desse servidor deve-se ao fato deste ser muito leve e por ter suporte às bibliotecas utilizadas durante o desenvolvimento do Serviço Web.

Para a construção do Serviço Web, foi utilizado o *framework* Axis2[8], desenvolvido em um projeto de software livre mantido pela *Apache Software Foundation*, cujo objetivo é oferecer uma plataforma ágil para o desenvolvimento de aplicações baseadas em serviços. Existem duas formas de utilizar o Axis2, uma delas é para o desenvolvimento de aplicações clientes que invocarão os Serviços Web, e a outra é utilizá-lo como um servidor onde serão publicados os serviços. Neste trabalho, as duas formas de utilização foram adotadas, a primeira para disponibilizar o serviço do SWSComposer e a segunda para criar Serviços Web utilizados para testar os processos de descoberta e composição. Para configurar o Axis2 com o Tomcat 6, basta transferir o arquivo *axis2.war* para dentro da pasta *webapps* do Tomcat 6.

Conforme ilustrado na Figura 25, a operação *serviceRequest* do Serviço Web do SWSComposer exige seis parâmetros que possuem os tipos: os três primeiros correspondem a listas de *Strings* compostas por URIs com referência a conceitos ontológicos (*input*, *outputs* e *operations*). O quarto parâmetro refere-se à profundidade máxima da composição e é representada por um número inteiro. O quinto parâmetro corresponde ao tempo limite de resposta da requisição e é representado em milisegundos através de um número inteiro. Por fim, o último parâmetro é do tipo booleano e define a permissão para reconstruir uma composição. O retorno da operação é constituído por uma *String* com o XML apresentado na seção 4.4. A operação *servicePublish* possui apenas um parâmetro de entrada do tipo *array* de bytes que representa o

[6] http://www.eclipse.org
[7] http://tomcat.apache.org/
[8] http://axis.apache.org/

arquivo WSDL anotado semanticamente com SAWSDL. Uma variável booleana é utilizada como retorno da operação para informar se a publicação foi realizada com sucesso.

```
▼<wsdl:types>
  ▼<xs:schema targetNamespace="http://webservice.swscomposer.org">
    ▼<xs:element name="serviceRequest">
      ▼<xs:complexType>
        ▼<xs:sequence>
            <xs:element minOccurs="0" name="inputs" nillable="true" type="xs:anyType"/>
            <xs:element minOccurs="0" name="outputs" nillable="true" type="xs:anyType"/>
            <xs:element minOccurs="0" name="operations" nillable="true" type="xs:anyType"/>
            <xs:element minOccurs="0" name="deep" type="xs:int"/>
            <xs:element minOccurs="0" name="timeout" type="xs:long"/>
            <xs:element minOccurs="0" name="allowRebuild" type="xs:boolean"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    ▼<xs:element name="serviceRequestResponse">
      ▼<xs:complexType>
        ▼<xs:sequence>
            <xs:element minOccurs="0" name="return" nillable="true" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    ▼<xs:element name="servicePublish">
      ▼<xs:complexType>
        ▼<xs:sequence>
            <xs:element minOccurs="0" name="wsdl" nillable="true" type="xs:base64Binary"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    ▼<xs:element name="servicePublishResponse">
      ▼<xs:complexType>
        ▼<xs:sequence>
            <xs:element minOccurs="0" name="return" type="xs:boolean"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>
```

Figura 25 – Tipos de dados das operações do Serviço Web do SWSComposer

O pacote *org.swscomposer.publish* contém as classes necessárias para realizar o processo de publicação de Serviços Web no repositório de serviços do protótipo. A biblioteca EasySAWSDL[9] é uma extensão da EasyWSDL[10] e é usada para manipular anotações semânticas em descritores WSDL. Ela suporta as versões 1.1 e 2.0 do WSDL e permite ler, escrever e adicionar anotações em qualquer elemento do WSDL ou XML Schema. No contexto do SWSComposer, essa biblioteca foi utilizada pelas classes do pacote *org.swscomposer.publish* para ler e processar os elementos de arquivos WSDL, técnica conhecida como *parsing*, e extrair anotações que remetem a conceitos de ontologias de

[9] http://easywsdl.ow2.org/extensions-sawsdl.html
[10] http://easywsdl.ow2.org/

domínio. Após a extração das anotações, a biblioteca Hibernate[11], utilizada para facilitar o armazenamento e a recuperação de dados de bancos de dados relacionais através do mapeamento objeto-relacional, é utilizada para armazenar informações sobre os serviços disponíveis conforme a estrutura apresentada na Figura 25. O modelo do banco de dados permite que operações, entradas e saídas sejam associadas e armazenadas com múltiplas anotações. O banco de dados MySQL[12] foi utilizado para implementar a estrutura de tabelas definida no modelo entidade-relacionamento.

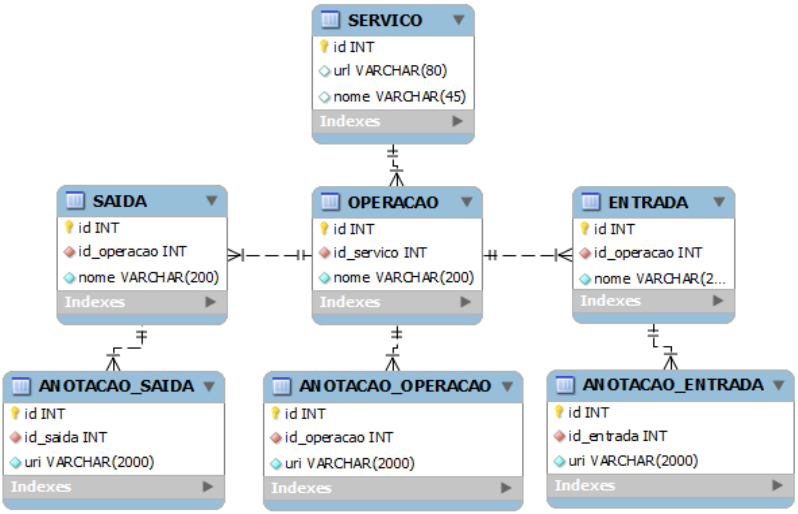

Figura 26 – Modelo entidade-relacionamento do banco de dados de serviços publicados.

O modelo entidade-relacionamento do banco de dados é composto por 7 tabelas, das quais 3 são responsáveis por armazenar as anotações semânticas: ANOTACAO_SAIDA, ANOTACAO_OPERACAO e ANOTACAO_ENTRADA. Elas armazenam as URIs associadas às anotações das saídas, operações e entradas, respectivamente. As tabelas SAIDA, ENTRADA e OPERACAO são utilizadas para armazenar o nome de cada saída, entrada e operação descritos no WSDL, respectivamente. Por fim, a tabela SERVICO é utilizada para armazenar o nome e a localização do WSDL através da coluna url.

[11] http://www.hibernate.org/
[12] http://www.mysql.com/

Para que o componente de publicação possa realizar o *parsing* do WSDL e extrair as anotações, estas devem seguir o formato: **http://host_da_ontologia/nome_da_ontologia.owl#conteito_da_ontologia**. A ontologia precisa obrigatoriamente ser especificada utilizando a linguagem OWL e o conceito deve ser referenciado logo após o nome da ontologia, por meio do nome do conceito precedido pelo símbolo #.

O pacote *org.swscomposer.discovery* contém as classes necessárias para realizar o processo de descoberta de Serviços Web. Dentro desse pacote está a classe responsável por realizar o *matching* entre os Serviços Web. Durante o *matching*, são utilizadas as bibliotecas Jena[13] e Pellet[14]. O *framework* Jena, projeto da *Apache Software Foundation*, conta com um conjunto de recursos para o desenvolvimento de aplicações relacionadas à Web Semântica. Tais recursos possibilitam ler, escrever e processar arquivos RDF, manipular ontologias OWL e RDF e realizar inferências sobre conceitos ontológicos. A biblioteca Pellet fornece serviços de raciocínio e inferência por meio de uma máquina de inferências. O Jena e o Pellet foram usados dentro do pacote *org.swscomposer.discovery* para carregar conceitos ontológicos e inferir significados sobre eles.

O pacote *org.swscomposer.composition* contém as classes necessárias para realizar o processo de composição de Serviços Web. O *framework* JUNG[15], acrônimo de *Java Universal Network/Graph Framework*, é uma biblioteca com recursos para a modelagem, análise e visualização de dados através de grafos e/ou redes. Uma variedade de representações é suportada pela biblioteca tais como: grafos direcionados, grafos não direcionados, grafos multimodais e hipergrafos. Além disso, dispõem de uma série de algoritmos para cálculo de distâncias, análises estatísticas, agrupamento, etc. Ela foi utilizada pelas classes do pacote *org.swscomposer.composition* para construir os grafos de composições de acordo com a relação entre os conceitos ontológicos.

O pacote *org.swscomposer.repository* contém as classes necessárias para armazenar e recuperar as composições de Serviços Web do repositório de composições. As classes desse pacote também fazem uso da biblioteca Hibernate para armazenar as composições descobertas na estrutura de tabelas apresentada na Figura 26.

A Figura 26 contém o modelo entidade-relacionamento do banco de dados criado para armazenar as composições descobertas. Composto

[13] http://jena.apache.org/
[14] http://clarkparsia.com/pellet/
[15] http://jung.sourceforge.net/

por sete tabelas, o repositório de composições é utilizado tanto no final do processo, para salvar uma nova composição através da função *salvaComposicoesEncontradas* (linha 16 do Algoritmo 4) como no início do processo, para verificar se já existe uma composição que atenda a requisição.

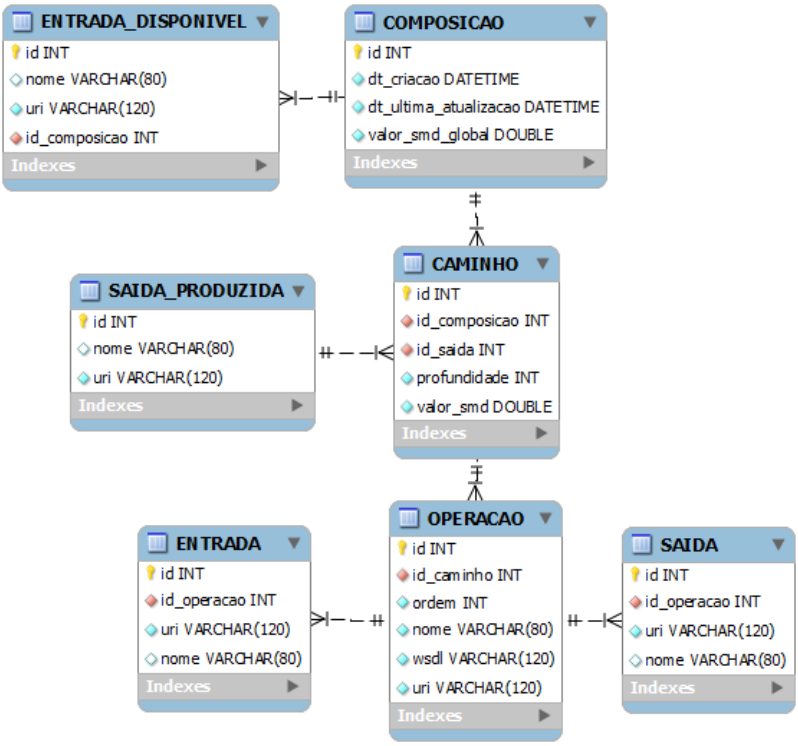

Figura 27 – Modelo entidade-relacionamento do banco de dados de composições

Toda composição (tabela COMPOSICAO) está associada a uma listagem de entradas enviadas na requisição e que deram origem a ela (tabela ENTRADA_DISPONIVEL) e um ou mais caminhos (tabela CAMINHO), cada qual responsável por prover uma das saídas desejadas (tabela SAIDA_PRODUZIDA). Cada caminho, por sua vez, é composto por um conjunto de operações (tabela OPERACAO) e cada operação está associada a uma ou mais entradas e saídas. A coluna ordem da tabela OPERACAO indica a sequência de execução das operações que

compõem o caminho a fim de prover a saída desejada. A coluna wsdl, da mesma tabela, faz referência à URL onde está localizado o descritor WSDL do serviço ao qual pertence a operação. As tabelas CAMINHO e COMPOSICAO armazenam ainda os valores resultantes da função SMD do caminho e da composição global, respectivamente.

Por fim, a tabela COMPOSICAO dispõe de duas colunas que indicam, respectivamente, a data de criação e descoberta da composição e a data da sua última atualização. Uma composição pode ser atualizada sempre que um novo serviço for publicado no repositório ou removido do mesmo e o parâmetro *allowRebuild* enviado na requisição permitir que a composição seja refeita.

Pode-se observar que o SWSComposer conta com estruturas para realizar *parsing* de WSDLs e criar composições em tempo de requisição. A fim de determinar qual o impacto dessas estruturas no desempenho de aplicações que utilizam o SWSComposer, foram elaborados testes que medem o tempo de resposta para requisições de serviços. Isso permite mensurar quando é válido utilizar o SWSComposer para descobrir e compor Serviços Web dada as características de uma aplicação que possua requisitos de performance a serem cumpridos.

5.3 AVALIAÇÃO DO PROTÓTIPO

A fim de avaliar a infraestrutura proposta no que diz respeito a desempenho, tempo de resposta e validade das composições obtidas pelo algoritmo de composição, o protótipo desenvolvido foi utilizado para realizar experimentações e os resultados encontrados são apresentados nesta seção. Os experimentos foram realizados em um PC com CPU Intel ® Core 2 Duo 2,10 GHz, 3 GB de RAM, rodando o sistema operacional Windows 7, usando o JDK 1.6.0 e MySQL 5.5. O link de Internet utilizado era de 10Mbps.

Inicialmente, 1.000 Serviços Web Semânticos foram obtidos a partir do repositório www.semwebcentral.org e replicados até atingirem 5.000 Serviços Web. Em seguida eles foram então publicados e as suas anotações foram extraídas e armazenadas no banco de dados de acordo com o modelo apresentado na Seção 5.1.

Diferentes conjuntos de serviços foram usados durante os testes a fim de comparar o tempo necessário para publicá-los e o tempo necessário para encontrar uma composição em meio a eles. Vários

fatores foram observados por exercerem influência direta sobre o desempenho do algoritmo de composição, tais como:

- Número de serviços envolvidos;
- Número de operações de um serviço,
- Número de entradas e saídas de uma operação,
- Número de anotações semânticas em operações, entradas e saídas;
- Número de ontologias envolvidas;
- Número de conceitos e relações em uma ontologia;
- Comprimento do caminho mais curto a partir de uma operação de inicialização até uma operação alvo.

Para realizar os testes, os Serviços Web foram agrupados em conjuntos de 1000, 2000, 3000, 4000 e 5000 serviços. Antes de mensurar os tempos envolvidos nos processos de descoberta e composição, o tempo de publicação dos serviços foi mensurado e está exibido na Tabela 3. Em média, os serviços eram compostos por uma operação com duas entradas e uma saída. Cada entrada, saída e operação era associada a uma única anotação semântica no respectivo arquivo WSDL.

Tabela 3- Tempo gasto na publicação dos serviços

Quantidade de Serviços Web	Tempo de publicação (ms)
1	4311
1000	492555
2000	1165902
3.000	2094112
4.000	3380537
5.000	4525671

Os resultados obtidos a partir dos testes demonstram que o tempo de publicação aumenta proporcionalmente na medida em que aumenta o número de Serviços Web. A publicação de serviços é um processo relativamente demorado, visto que o *parsing* é realizado com o arquivo WSDL em memória. Assim sendo, o arquivo WSDL precisa ser

carregado inteiramente para a memória para somente depois os seus elementos serem lidos, de forma hierárquica, pelo *parser*. Essa técnica utilizada pela API EasyWSDL para realizar o *parsing* é conhecida como DOM (*Document Object Model*) *parser*. Segundo (BOISSEL-DALLIER *et al.*, 2009), quando comparada com outras APIs para a realização de *parsing*, a API EasyWSDL apresenta melhores resultados de um modo geral, porém não possui a melhor performance. A Tabela 4 demonstra essa comparação realizada com a API WSDL4J[16].

Tabela 4- Testes de desempenho entre as APIs WSDL4J e EasyWSDL

API	WSDL4J	EasyWSDL
Número de arquivos testados	40.257	40.257
Erros encontrados	51	**9**
Taxa de sucesso (%)	98.87	**99.98**
Tempo médio de *parsing* (ms)	**24**	30
Vazamento de memória	Sim	**Não**

Além do tempo de publicação, foram mensurados o consumo de memória e de processamento durante a publicação dos serviços. A ferramenta VisualVM[17], disponível no JDK, foi utilizada para monitorar o uso de recursos pelo protótipo. Cada serviço a ser publicado tem o seu WSDL lido para a memória, em seguida o *parsing* é realizado e posteriormente as anotações são armazenadas no repositório. Ao final disso, o WSDL é liberado da memória e o próximo serviço a ser publicado é carregado. Dessa forma, o consumo de memória bem como o de processamento tende a ser o mesmo, independentemente da quantidade de serviços a serem publicados. O que exerce influência direta sobre a memória e o processamento é o tamanho do WSDL a ser lido. A Figura 27 apresenta o consumo de processamento durante a publicação de 5000 Serviços Web.

[16] http://sourceforge.net/projects/wsdl4j/
[17] http://visualvm.java.net/

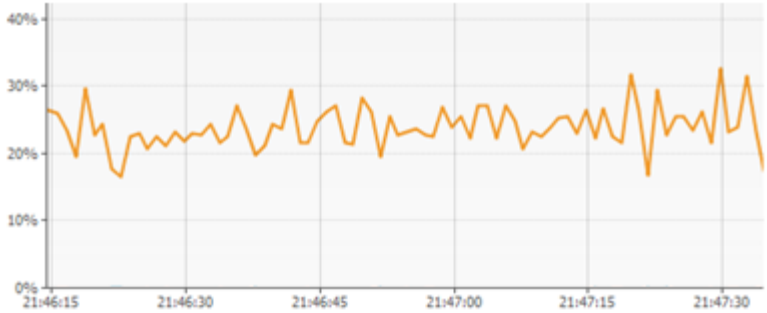

Figura 28 – Consumo de processamento durante publicação de Serviços Web

A Figura 28 apresenta o consumo de memória durante a publicação dos mesmos Serviços Web. Através do gráfico é possível perceber que o consumo oscila entre 5MB e 14MB de um total de 268MB disponível de memória *heap* para alocação de objetos na máquina virtual Java. Tanto o processamento quanto a memória não foram exigidos excessivamente durante os testes.

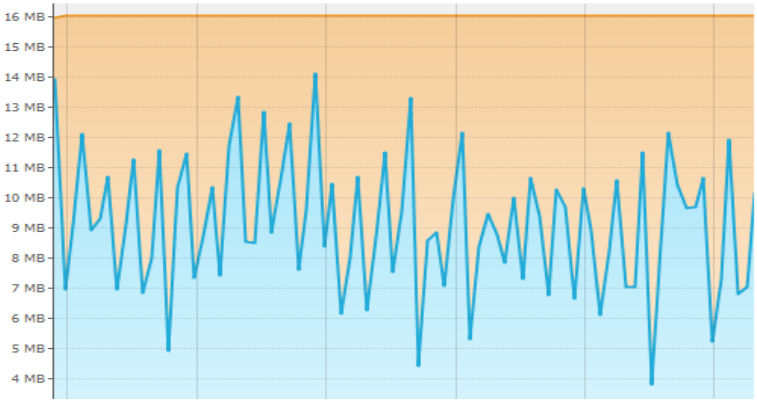

Figura 29 – Consumo de memória durante publicação de Serviços Web

Após avaliar o tempo de publicação, o tempo de resposta para a descoberta e a construção de uma composição foi avaliado em função da quantidade de serviços disponíveis em cada um dos três conjuntos. Para cada um dos conjuntos foi criada uma requisição de teste cujo objetivo era encontrar 3 saídas específicas a partir de 2 entradas disponíveis. As operações e saídas bem como as entradas foram escolhidas a partir dos Serviços Web publicados. Os conceitos utilizados para compor a

requisição pertencem todos à ontologia disponível em *http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/ontologies/rosetta.owl*. Esta ontologia dispõe de conceitos e relações que representam o domínio de compra e venda de produtos. A requisição final, com a lista completa de parâmetros ficou da seguinte forma:

- Lista de entradas disponíveis (2 entradas - 2 URIs)
  - #PurchaseOrderRequest
  - #ShoppingCart

- Lista de saídas desejadas (3 saídas - 3 URIs)
  - #PurchaseOrderResponse
  - #TotalPrice
  - #PartnerDescription

- Lista de operações desejadas (4 operações - 4 URIs)
  - #Quote
  - #ReturnsAndFinance
  - #BusinessDescription
  - #ProductPriceAndAvailability

- Profundidade máxima da composição;
  - -1

- Tempo máximo de espera pela resposta (*timeout*);
  - -1

- Permissão para reconstruir composições (*allowRebuild*)
  - falso

O propósito da requisição é encontrar o preço total, bem como possíveis fornecedores e informações acerca deles, para efetuar a compra de uma lista de produtos. Para isso, tem-se a disposição a solicitação de ordem de compra (#PurchaseOrderRequest) e a coleção com as descrição dos produtos desejados, quantidades e preços que compõem a intenção do comprador (#ShoppingCart). Espera-se que a partir dessas informações, seja possível encontrar o preço total dos produtos desejados (#TotalPrice), informações sobre os fornecedores desses produtos (#PartnerDescription) e o resultado da ordem de compra (#PurchaseOrderResponse). Essas saídas devem ser retornadas a partir

de operações que sejam capazes de informar propriedades comerciais que descrevem uma oferta para fornecer uma quantidade de produtos a um preço acordado (#Quote), de efetuar a emissão de pagamento, faturamento, créditos e notas fiscais, bem como suportem a devolução de produtos e do investimento realizado (#ReturnsAndFinance), forneçam informações sobre unidades de negócios, sua localização (#BusinessDescription), preço praticado e disponibilidade de produtos oferecidos (#ProductPriceAndAvailability).

Uma vez definida a requisição, execuções foram realizadas até que fosse possível ser encontrada uma composição. A função de SMD foi avaliada com pesos diferentes durante a realização dos testes, a fim de selecionar a melhor combinação de valores. Depois de executar o processo de composição numerosas vezes, os melhores resultados em termos de qualidade semântica foram obtidos de acordo Tabela 5.

Tabela 5- Pesos associados aos critérios avaliados.

	Peso
Entrada	0,3
Saída	0,35
Operação	0,35

O *matching* semântico das operações e da saída de cada caminho foram considerados igualmente importantes e têm maior peso do que o *matching* das entradas. Isso porque juntos eles definem o que irá retornar após a execução do caminho e correspondem ao que o requerente está buscando.

A avaliação dos resultados se deu através de 30 execuções sucessivas para cada conjunto de serviços do processo de descoberta e composição a fim de mensurar e calcular a média dos tempos de cada etapa envolvida no processo. Não foi necessário executar um número maior de testes visto que os tempos giravam sempre em torno do mesmo valor. As 15 primeiras execuções de cada conjunto estão exibidas no Anexo A. O tempo de resposta de cada etapa é detalhado na Tabela 6 e ilustrado na Figura 29. A primeira etapa, representada por "*Descoberta de Operações de Inicialização*", realiza o *matching* de entradas e requer a análise de cada entrada de cada operação de um serviço, a fim de selecionar a operação como uma operação de inicialização. A segunda etapa, denominada "*Descoberta de Operações Alvo*" foi subdividida no *matching* de operações e *matching* de saídas. Basta que uma saída seja compatível para que a operação seja selecionada como uma Operação

Alvo. A etapa rotulada "*Construção do Grafo*" é responsável por consumir a maior parte do tempo necessário para executar o algoritmo. Este resultado já era esperado, uma vez que este procedimento executa sucessivos *matching* de entradas para identificar as relações entre os serviços. Por fim, a etapa final "*Seleção do melhor Caminho*" aplica as fórmulas de SMD e *Cost*, quando preciso, para selecionar os caminhos que retornam as melhores saídas semanticamente.

Tabela 6- Tempo gasto em cada etapa do processo de composição

Etapas	1.000 serviços (ms)	2.000 serviços (ms)	3.000 serviços (ms)	4.000 serviços (ms)	5.000 serviços (ms)
Descoberta de Operações de Inicialização	2185	2195,13	2173,26	2332,6	2373,06
Matching de entradas	2185	2195,13	2173,26	2332,6	2373,06
Descoberta de Operações Alvo	31,13	36,86	45	52,86	64,53
Matching de operações	27,8	34,6	41,2	47,4	58
Matching de saídas	1	1,1	2,1	2,4	2,8
Construção do Grafo	13619,13	21639,73	31492,53	39658,66	51298,8
Seleção do melhor Caminho	711,46	701,6	712,73	689,8	751,33
Tempo total para a Descoberta e Composição	16546,72	24573,32	34423,52	42733,92	54487,72

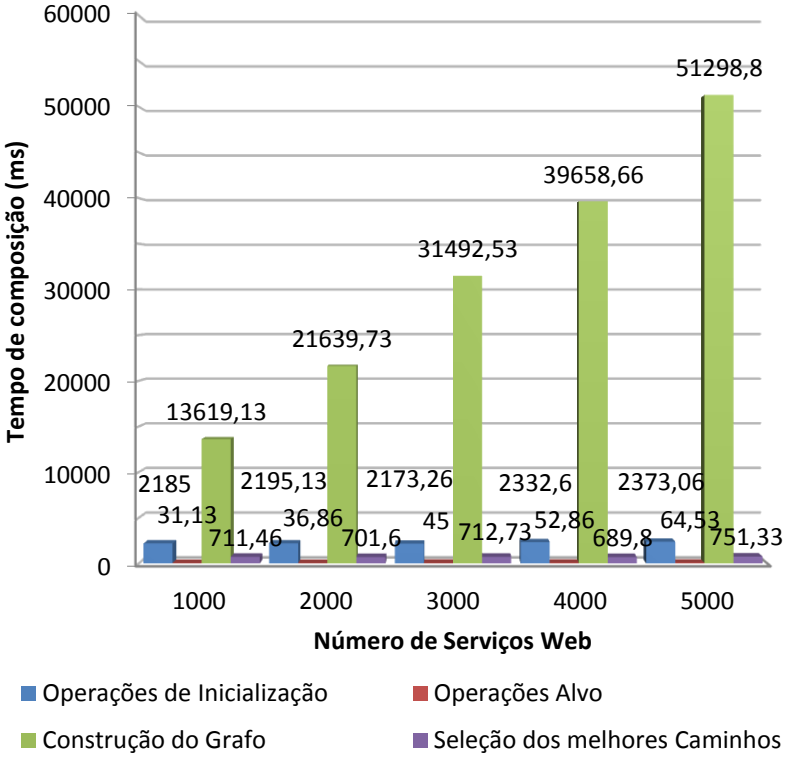

Figura 30 – Tempo de composição medido durante o experimento

Dentre os tempos apresentados na Tabela 6, chama atenção o tempo da etapa "*Descoberta de Operações Alvo*". Se comparado com o tempo de execução da etapa "*Descoberta de Operações de Inicialização*" ele é muito menor, apesar de o algoritmo ser semelhante. Em ambas as etapas todos os serviços do repositório são analisados e comparados com os conceitos enviados na requisição. Enquanto na primeira etapa basta que uma saída seja encontrada para a operação ser selecionada, na segunda todas as entradas de cada operação precisam ser analisadas para evitar que seja selecionada uma operação que exija entradas indisponíveis pelo cliente. Apesar dessa diferença, o que realmente difere entre as duas etapas é o carregamento das ontologias para o motor de inferências.

Ainda analisando o gráfico, é possível perceber que o tempo para a execução da etapa "*Seleção dos melhores Caminhos*" se mantém

constante independentemente da quantidade de Serviços Web. Esse comportamento ocorre em função de os conjuntos de serviços testados possuírem os mesmos serviços, que foram apenas replicados. Como os serviços são os mesmos, os caminhos encontrados também são os mesmos, resultando em tempos similares.

Para que seja possível comparar conceitos e identificar o grau de similaridade entre eles, as ontologias nas quais os conceitos estão especificados precisam ser carregadas para a máquina de inferências, provida pelo Pellet. Esse carregamento exige que seja feito o *download* da ontologia, e o tempo necessário para efetuar o *download* pode variar em função do seu tamanho, do tráfego na rede e da velocidade do link de Internet disponível. A fim de minimizar esse tempo durante as etapas, a partir do momento que uma ontologia é utilizada, ela é mantida em *cache* para futuras consultas. Como a etapa "*Descoberta de Operações Alvo*" é executada após a etapa "*Descoberta de Operações de Inicialização*", a maior parte das ontologias já está em *cache*, tornando a execução mais rápida. A ontologia só não estará em *cache* caso nenhuma entrada tenha sido associada a ela, fazendo com que seja carregada somente quando for utilizada por uma alguma operação ou saída. Durante os testes, as ontologias estavam hospedadas em diferentes servidores, inclusive algumas delas em um servidor local.

O Módulo de Descoberta não é capaz de identificar se duas ontologias hospedadas em diferentes servidores com URLs (*Uniform Resource Locator*) diferentes são na verdade a mesma ontologia. Para lidar com esse caso, antes de comparar um conceito e verificar se ele pertence a uma ontologia, as URLs são comparadas e fim de evitar que uma ontologia seja carregada sem necessidade e demande tempo de carregamento. Assim sendo, uma ontologia só é carregada quando se sabe que o conceito a ser comparado faz parte dela.

Considerando o cenário com cinco mil Serviços Web, o protótipo encontrou duas operações de inicialização e duas operações alvo durante o processo de composição. Como nenhuma operação foi encontrada de forma que sozinha fosse capaz de atender à requisição, o grafo foi construído a fim de identificar composições. Durante a construção do grafo foram encontrados 1492 caminhos, dos quais 692 foram eliminados por não alcançarem uma operação alvo. Os 800 caminhos restantes, todos iniciando por uma operação de inicialização e finalizando em uma operação alvo, foram separados e agrupados de acordo com a saída retornada. Das 3 saídas solicitadas na requisição, apenas 1 delas foi encontrada (#PartnerDescription) e era fornecida por duas das 5000 operações disponíveis no repositório. A API JUNG

possui componentes gráficos que permitem exibir uma composição. O grafo parcial da composição, obtido a partir dos 800 caminhos sem selecionar os melhores semanticamente, está ilustrado na Figura 30. Apesar de parecer pelo grafo que existem caminhos que iniciam em uma das operações de inicialização e alcançam uma das operações alvo sem a necessidade de operações de ligação, todos os caminhos passam por operações de ligação até encontrar uma das saídas desejadas. Nos vértices do grafo, onde uma operação de inicialização está interligada diretamente a uma operação alvo, o que acontece é que a operação de inicialização está, na verdade, atuando como uma operação de ligação. Os valores nos vértices correspondem à ordem de execução dos serviços.

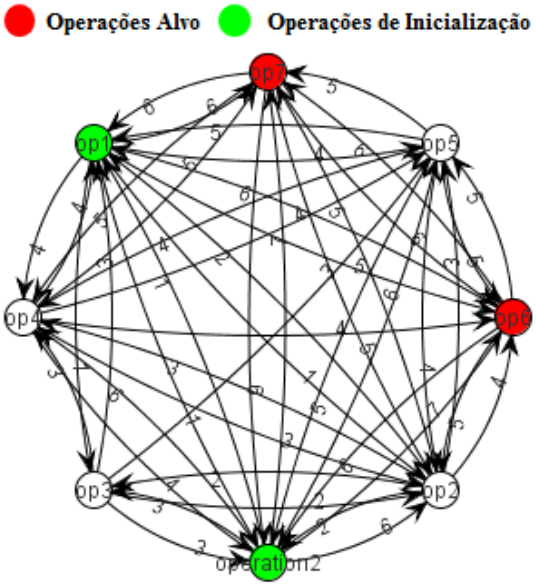

Figura 31 – Composição parcial resultante do experimento.

As operações rotuladas op6 e op7 são responsáveis por retornar apenas uma das três saídas requisitadas. A composição encontrada não foi capaz de retornar todas as saídas requisitadas e atende a requisição de forma parcial. Após filtrar e selecionar somente os melhores caminhos com base no cálculo da função SMD e na profundidade de cada um, um novo grafo foi construído e está exibido na Figura 31. Como as operações op6 e op7 retornavam a mesma saída, apenas uma se

manteve na composição final. Em resumo, o caminho final não exige entradas que o cliente não dispõe e retorna apenas uma das saídas requeridas.

A composição final, ilustrada na Figura 31, teve o SMD calculado em 0.0 juntamente com outras 437 do conjunto de 800 composições. Para selecioná-la foi necessário considerar a profundidade das composições como critério de desempate. O resultado foi uma composição de 4 operações provenientes de 2 Serviços Web diferentes.

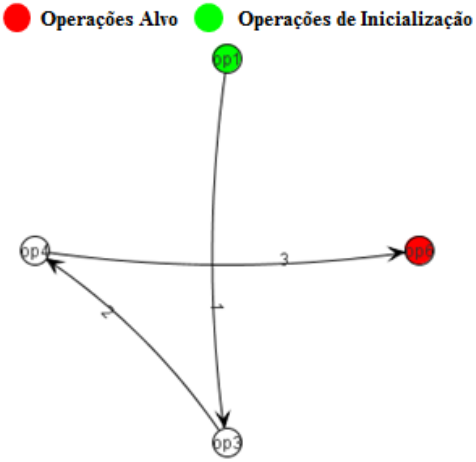

Figura 32 – Composição final resultante do experimento.

Precisão e *recall* são duas medidas de desempenho utilizadas para a medição da eficácia da descoberta de composições. Dada uma requisição, a precisão é a proporção das composições relevantes que foram encontradas mediante todas as composições descobertas. Já o *recall* é a proporção de composições relevantes que foram encontradas mediante todas as composições relevantes (WEI *et al*., 2011). Devido à utilização de um algoritmo não probabilístico decorrente da ausência de métodos estatísticos para descoberta de composições, os resultados não se alteram ao longo das execuções. O *recall* e a precisão das composições retornadas pelo protótipo dependem do nível de detalhamento das anotações adicionadas aos serviços bem como dos parâmetros enviados na requisição.

A partir de uma requisição específica, o protótipo retorna sempre as mesmas composições. Nesse caso, diz-se que o algoritmo é preciso visto que as composições relevantes obtidas pela repetição do processo

são sempre as mesmas. Nos testes realizados, o protótipo foi capaz de encontrar todas as composições possíveis para determinada requisição. Como composições possíveis entendem-se todos os pontos de integração entre os diferentes serviços disponíveis a partir de um conjunto de entradas. Além de encontrar todas as composições, o algoritmo foi capaz de filtrar e selecionar somente as composições julgadas relevantes, de modo que restaram apenas composições que levavam a alguma saída desejada, apresentando assim uma taxa de *recall* com valor máximo.

Testes de *benchmark* com as ferramentas apresentadas nos trabalhos relacionados não foram possíveis de serem realizados em virtude da falta de acesso a elas. As medidas de *recall* e precisão poderiam prover indicadores de desempenho quando comparadas com outros trabalhos a fim de apresentar vantagens competitivas.

Além dos tempos de resposta para descobrir a composição, foram medidos também o consumo de processamento e memória durante a execução do processo. A Figura 32 apresenta a taxa de uso do processador. Em grande parte do tempo essa taxa se mantém entre 40% e 50% da capacidade máxima de processamento. O ponto que se destaca ocorre no final do processo, onde o grafo precisa ser construído. Nesse momento a taxa sobe para 76% e atinge o pico de processamento. Após a construção do grafo, o processamento decai e atinge o patamar mais baixo, visto que não são mais executadas operações que exigem elevado processamento.

Figura 33 – Consumo de processamento durante composição de Serviços Web

A Figura 33 tem como objetivo apresentar o consumo de memória durante a realização do processo de composição. Assim como a avaliação de processamento, a avaliação do uso de memória foi realizada durante a composição realizada com 5000 serviços. Pode-se verificar que, apesar de o processo ser realizado em tempo de requisição, o consumo de memória é aceitável, visto que foram consumidos em média 25MB de um total de 268MB de memória disponíveis. Isso ocorre porque apenas as informações básicas de cada serviço, nesse caso as anotações, são mantidas em memória.

Uma vez que a composição é descoberta e salva no repositório de composições, o tempo de resposta da busca pela composição se resume ao tempo de uma consulta ao banco de dados. Uma composição que demanda cerca de 50 segundos para ser descoberta na primeira requisição, quando for utilizada em futuramente levará em torno de 65 milissegundos para ser selecionada no repositório de composições.

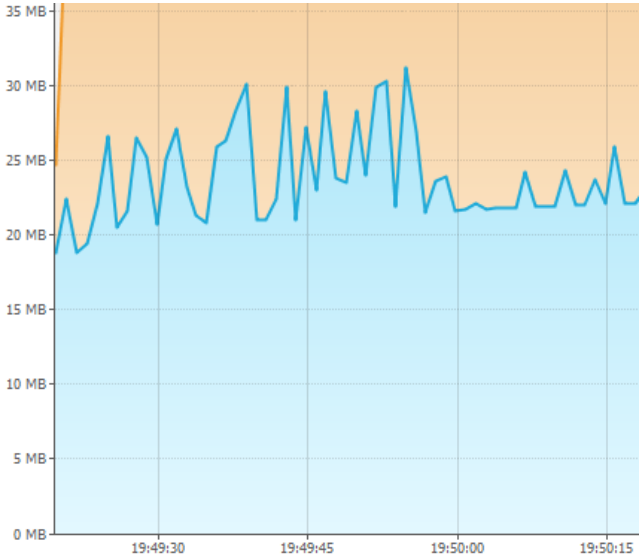

Figura 34 – Consumo de memória durante composição de Serviços Web

Na medida em que a quantidade de serviços aumenta, o tempo de resposta também aumenta, de modo que o tempo de resposta é diretamente proporcional à quantidade de serviços disponíveis. A Tabela 7 apresenta o tempo de resposta envolvendo as atividades inerentes ao processo de composição considerando os piores casos. Com exceção da

atividade “Construção do Grafo”, as demais apresentam complexidade do algoritmo linear. A atividade “Seleção de Operações de Inicialização” percorre todas as operações publicadas no repositório SAWSDL (*Ns)* a fim de identificar aquelas que possuem todas as suas entradas (N_{inS}) compatíveis com as entradas disponíveis (N_{inR}). Para cada operação, o algoritmo recupera a lista com as entradas exigidas por ela e, para cada uma dessas entradas, percorre a lista de entradas disponíveis, comparando-as. O algoritmo é considerado linear e, portanto, com complexidade *O(N).* O mesmo se aplica à atividade “Seleção de Operações Alvo”, onde as operações publicadas são percorridas e, para cada uma delas, a lista de operações requisitadas (N_{opR}) é consultada a fim de compará-las. Após selecionar as operações compatíveis, a mesma análise é realizada com as saídas fornecidas pelas operações (N_{outS}) e as saídas requisitadas (N_{outR}). O algoritmo responsável por realizar essa atividade é linear e demanda complexidade *O(N).*

A atividade de “Seleção de Serviço com o Melhor SMD” percorre a listagem de todas as operações alvo, sendo que no pior caso seria a lista de todas as operações publicadas, e verifica se elas são também operações de inicialização. Considerando ainda o pior caso e que todas as operações publicadas são tanto operações de inicialização como operações alvo, o algoritmo verifica se cada operação é capaz de fornecer todas as saídas desejadas. De mesma forma que as atividades anteriores, a complexidade do algoritmo é O(N). A atividade “Construção do Grafo”, por ser recursiva, apresenta complexidade maior e demanda mais tempo. Tendo em vista o pior caso, a lista com todas as operações publicadas seria percorrida aos pares, com um laço dentro do outro. Por esse motivo, o algoritmo é polinomial e apresenta complexidade $O(N^2)$. Como o algoritmo para a construção do grafo gera todos os caminhos possíveis, pode acontecer de caminhos não levarem a nenhuma das saídas desejadas. Nesse caso, após a construção do grafo os caminhos que não levam a nenhuma operação alvo são eliminados. Esse procedimento faz parte da atividade “Seleção do Melhor Caminho para Cada Saída”, que ainda percorre, para cada saída desejada, todos os caminhos (N_c) que levam a ela para realizar o calculo do SMD. Essa atividade é considerada linear e apresenta complexidade *O(N).* Por fim, a atividade “Armazenamento da composição” recupera todos os caminhos responsáveis por retornar ao menos uma das saídas requisitadas. O algoritmo percorre cada um desses caminhos e salva as operações pertencentes a cada caminho no banco de dados de composições, conforme modelo apresentado anteriormente. A

complexidade desse algoritmo é *O(N)*. Os resultados apresentados demonstram que, de modo geral, o tempo associado à execução do algoritmo de composição é polinomial.

Tabela 7- Complexidade das atividades envolvidas na composição

Atividade	Complexidade
Seleção Operações de Inicialização	$O(N_s * N_{inS} * Ni_{nR}) = O(N)$
Seleção Operações Alvo	$O(N_s * N_{opR}) + O(N_s * N_{outS} * N_{outR}) = O(N)$
Seleção Serviço com Melhor SMD	$O(N_s) + O(N_s * N_{outR}) = O(N)$
Construção do Grafo	$O(N^{2} * N_{in}) = O(N^2)$
Seleção Melhor Caminho para cada Saída	$O(Ns * N_{outS} * N_{outR}) + O(N_{outR} * N_c) = O(N)$
Armazenamento da Composição	$O(N_c * N_s) = O(N)$

Uma vez que a qualidade das composições depende única e exclusivamente do *matching* entre os parâmetros enviados no pedido e as anotações dos serviços, quanto mais detalhados forem os parâmetros e as anotações dos serviços, melhor a qualidade das composições. O algoritmo mostra-se capaz de selecionar todas as saídas desejadas existentes no repositório com base numa lista de entradas. Além disso, ele prova ser capaz de identificar composições e relações semânticas entre operações de diferentes Serviços Web em tempo de requisição e de forma automática.

5.4 COMPARAÇÕES COM TRABALHOS RELACIONADOS

Comparado com os projetos relacionados encontrados na literatura que têm objetivos semelhantes, a abordagem proposta difere-se em alguns critérios. A Tabela 8 complementa a Tabela 1 com a adição das características propostas neste trabalho.

A abordagem proposta neste trabalho distingue-se da abordagem de (PRAZERES, 2009) em alguns aspectos. Tanto a linguagem utilizada para descrever a semântica dos Serviços Web como o momento de construção das composições são diferentes. Em função de a composição ser realizada em tempo de publicação dos Serviços Web, durante uma

requisição por uma saída qualquer podem ser solicitadas entradas indisponíveis pelo cliente. Como as composições são construídas sem conhecimento das entradas disponíveis pelos clientes e são reajustadas no momento da requisição para contemplar essas entradas, a abordagem não combina as entradas enviadas na requisição com as saídas de outros Serviços Web para servirem de entrada para os demais serviços existentes. As entradas enviadas na requisição são utilizadas com o objetivo de selecionar o serviço inicial das composições já criadas. Essa situação não acontece com a abordagem proposta visto que a composição é construída com base somente nas entradas enviadas na requisição. Por outro lado, a abordagem de (PRAZERES, 2009) apresenta como diferencial a análise de pré-condições para a execução de um serviço e efeitos da sua execução durante o processo de composição. Apesar de a linguagem SAWSDL prover meios primitivos para a especificação de pré-condições e efeitos, eles não foram utilizados durante o desenvolvimento do protótipo.

O trabalho de (TRAN *et al*., 2009) não realiza o processo de composição. Apesar disso, ele foi utilizado como referência para a construção do algoritmo de descoberta de Serviços Web já que a linguagem semântica utilizada em ambos os trabalhos é a mesma. O trabalho de (TRAN *et al*., 2009) utiliza graus de similaridade diferentes dos utilizados neste trabalho para o *matching* entre conceitos e não avalia a qualidade semântica dos serviços encontrados a fim de filtrar e selecionar os melhores.

O trabalho de (BELOUADHA *et al*., 2010), diferente da abordagem proposta neste trabalho, realiza a composição estática de Serviços Web Semânticos. A composição estática não permite que Serviços Web pertencentes a uma composição sejam substituídos por outros caso fiquem indisponíveis por exemplo. Nesses casos a composição precisa ser manualmente refeita para que possa continuar em funcionamento, diferente do que foi proposto neste trabalho, onde as composições são criadas à medida que forem utilizadas.

O trabalho de (MONTEIRO, 2008) cria composições através de um processo semiautomático. Por meio de uma interface gráfica,o usuário é guiado durante a montagem da composição de serviços. A partir de um Serviço Web inicial escolhido pelo usuário dentre os Serviços Web publicados, a ferramenta sugere serviços passíveis de serem integrados. Essa abordagem difere da abordagem proposta por depender da intervenção humana e pelo fato de as composições criadas serem estáticas, a exemplo das composições criadas no trabalho de (BELOUADHA *et al*., 2010).

O trabalho de (ZHANG *et al*., 2003) é provavelmente o que mais se assemelha à abordagem proposta. Em ambos os trabalhos as composições são construídas em tempo de requisição através da utilização de grafos. A diferença entre eles está na linguagem utilizada para descrever os Serviços Web semanticamente, na forma de construção do grafo e na forma de seleção das composições. A abordagem de (ZHANG *et al*., 2003) considera apenas anotações nas entradas e saídas das operações e não avalia as anotações das próprias operações. Além disso, não utiliza critérios de avaliação da qualidade semântica das composições encontradas em casos onde existem mais de uma capaz de fornecer os mesmos resultados. Outro detalhe da abordagem de Zhang é a inexistência de um repositório para armazenar as composições descobertas e agilizar o processo em futuras requisições, como é feito na abordagem proposta neste trabalho.

Em resumo, as composições são obtidas pelo algoritmo proposto em tempo de requisição e sem a assistência do usuário. Além disso, nunca requerem insumos indisponíveis pelo cliente. Os resultados das experiências demonstraram que o algoritmo é capaz de combinar serviços únicos para formar composições com um tempo de resposta diretamente proporcional à quantidade de serviços disponíveis no repositório.

Tabela 8- Comparação com os trabalhos relacionados

	PRAZERES, 2009	**TRAN *et al.*, 2009**	**BELOUADHA *et al.*, 2010**	**MONTEIRO, 2008**	**ZHANG *et al.*, 2003**	**Abordagem proposta**
Linguagem	OWL-S	SAWSDL	UML / SAWSDL	OWL-S	DAML-S	SAWSDL
Descoberta	Sim	Sim	Sim	Sim	Sim	Sim
Composição	Sim	Não	Sim	Sim	Sim	Sim
Forma de Composição	Automática	-	Automática	Semi-Automática	Automática	Automática
Momento de Composição	Publicação/ Requisição	-	desenvolvimento	desenvolvimento	execução	requisição
Tipo de Composição	Dinâmica	-	Estática	Estática	Dinâmica	Dinâmica
Técnica de composição	Grafos	-	UML e BPMN	Interação com o usuário	Grafos	Grafos
Pré-Condições e Efeitos	Sim	Não	Sim	Sim	Não	Não

6 CONCLUSÕES E TRABALHOS FUTUROS

A construção de composições de Serviços Web não é uma tarefa trivial, especialmente com a grande quantidade de serviços disponíveis atualmente. A composição é necessária quando um único serviço não é capaz de executar a tarefa desejada. Tecnologias semânticas podem desempenhar um papel importante no presente processo, permitindo a automatização do processo de composição.

O presente trabalho abordou a possibilidade de composição dinâmica de Serviços Web baseada na descrição semântica dos serviços. Por meio de anotações semânticas SAWSDL adicionadas aos elementos do WSDL foi possível extrair a semântica associada às entradas, saídas e operações dos serviços (objetivo específico 1). Um protótipo da abordagem proposta foi implementado a fim de usar as informações semânticas extraídas anteriormente para estabelecer uma correspondência semântica com os dados enviados em uma requisição. Por meio dos critérios de comparação de conceitos ontológicos proposto em (PAOLUCCI *et al*., 2002) e com o auxilio de bibliotecas para manipulação de ontologias, foi possível identificar relações semânticas entre conceitos ontológicos a mensurar o grau de similaridade entre eles (objetivo específico 2). A utilização de tecnologias capazes de inferir classificações e relacionamentos entre conceitos de ontologias e com isso interpretar a semântica associada às funcionalidades dos serviços sem a intervenção humana permite automatizar os processos de descoberta e composição de Serviços Web (objetivo específico 3). O protótipo apresentado, com sua arquitetura e tecnologias, comprova através dos resultados obtidos a partir da sua execução a viabilidade de compor Serviços Web de forma automática e em tempo de requisição (objetivo específico 4).

A construção de composições de Serviços Web de forma automática retira a responsabilidade dos desenvolvedores de procurar serviços em repositórios distribuídos pela web e compô-los manualmente. Quando esse processo é feito de forma *ad hoc*, qualquer alteração ou mudança realizada em um dos serviços pertencentes à composição implica diretamente na sua execução. Quando um dos serviços de uma composição deixa de existir, por exemplo, um novo Serviço Web precisa ser encontrado a fim de substituí-lo. Isso exige retrabalho por parte dos desenvolvedores sempre que um serviço que integra composição deixa de ser oferecido. Um dos benefícios oferecidos pelo protótipo apresentado é a facilidade da substituição de serviços que, por algum motivo, tornam-se inexistentes. Nesses casos, o

protótipo automaticamente constrói uma nova composição com outros serviços, caso isso seja possível, e mantém o serviço oferecido em funcionamento.

Outro benefício oferecido pelo protótipo remete à agilidade para a construção de composições. Sem a utilização de uma ferramenta que automatize esse processo, desenvolvedores precisariam analisar individualmente cada Serviço Web a fim de compreender a semântica associada às entradas e saídas de cada operação para somente depois realizar a integração entre os serviços. Com o uso do protótipo proposto, esse processo tende a ser realizado mais rapidamente, poupando tempo de desenvolvimento.

Em resumo, a abordagem proposta, implementada em um protótipo para validar e avaliar os resultados mostrou-se capaz de encontrar composições que atendam determinadas requisições e com isso, atingindo os objetivos propostos e respondendo a pergunta de pesquisa.

6.1 CONTRIBUIÇÕES

O trabalho apresentado nesta dissertação apresenta contribuições relacionadas à área de pesquisa de Serviços Web, mais especificamente Serviços Web Semânticos. As contribuições abrangem aspectos do modelo de requisição e de descoberta de Serviços Web bem como uma nova abordagem para a composição automática de Serviços Web Semânticos. Como principais contribuições destacam-se:

- Um algoritmo para a seleção de composições baseado na qualidade semântica das mesmas. Após construir o grafo com as conexões entre os Serviços Web Semânticos, o algoritmo analisa se dois ou mais caminhos levam à mesma saída. Caso isso aconteça, uma função é aplicada para cada caminho, a fim de selecionar aquele com o menor grau de divergência semântica;
- Um protótipo para a descoberta e a composição de Serviços Web Semânticos. O protótipo implementa uma infraestrutura proposta para oferecer as funcionalidades de publicação, descoberta e composição de Serviços Web Semânticos. Esse protótipo possibilitou validar os algoritmos apresentados bem como avaliar os tempos de resposta para todas as funcionalidades oferecidas;
- Uma abordagem para a descoberta e a composição automática de Serviços Web Semânticos em tempo de requisição, baseada

em anotações SAWSDL e com tempo de resposta polinomial. A partir de um modelo de requisição apresentado, Serviços Web Semânticos são solicitados e retornados, caso sejam encontrados. A abordagem torna a resposta à requisição transparente para o usuário, de modo que pode ser retornado apenas um Serviço Web como também uma composição de serviços. O tempo associado à execução do algoritmo de composição é polinomial. Algoritmos com complexidade polinomial são considerados computacionalmente tratáveis e exigem um tempo de execução limitado por uma função polinomial.

- Apesar de desempenho não ser foco desse trabalho, pode-se observar que o consumo de memória e processamento demandado pelos algoritmos de descoberta e composição são relativamente baixos, se considerada a quantidade de operações publicadas no repositório e a forma como são identificadas as composições. Isso porque os algoritmos carregam para memória apenas as informações necessárias para a construção das composições, além de realizarem poucos procedimentos que exijam muito poder de processamento.

6.2 LIMITAÇÕES

O trabalho apresentado nesta dissertação apresenta limitações referentes aos processos de descoberta e composição automática de Serviços Web Semânticos. Dentre os principais entraves destacam-se:

- Ao comparar dois conceitos ontológicos, para que os mesmos sejam classificados como *Exact* ou *Plugin*, eles devem pertencer à mesma ontologia. Em outras palavras, a URI da ontologia deve ser a mesma para os dois conceitos. O algoritmo não é capaz de identificar o mesmo conceito ontológico em ontologias diferentes e nem mesmo identificar um conceito presente em duas ontologias iguais, porém localizadas em diferentes lugares da Web;
- O algoritmo de composição não considera pré-condições para a execução de uma operação, tampouco os efeitos decorrentes da execução dela. Isso porque a linguagem SAWSDL oferece recursos limitados para expressar tais características. Além disso, requisitos de qualidade dos serviço (QoS) não são analisados no momento da descoberta e nem da composição.

Eles não foram considerados nesse primeiro momento por aumentarem o escopo do trabalho e, consequentemente, a complexidade do algoritmo;

- Antes de analisar as saídas fornecidas pelas operações publicadas, o algoritmo de descoberta avalia se a anotação adicionada às operações é similar a algumas das operações requisitadas. Caso não exista similaridade entre elas, a operação é descartada e a saída fornecida por ela sequer é avaliada. Nesses casos, pode haver o descarte de operações cujo resultado interessa à requisição;
- Não foram previstos no presente trabalho mecanismos que permitam executar as composições encontradas;
- Estruturas de controle e paralelismo não são identificadas durante a criação de composições. Consequentemente não é possível criar composições complexas, com estruturas condicionais por exemplo.

6.3 TRABALHOS FUTUROS

Os trabalhos futuros vão concentrar-se na execução das composições e na utilização de mediadores de ontologias. O protótipo apresentado faz parte de uma plataforma, atualmente em desenvolvimento, que também será capaz de realizar a execução das composições descobertas. Nesta plataforma, as composições serão descritas usando a linguagem WS-BPEL[18], permitindo que a composição seja executada por um motor BPEL[19] e invocada pelos consumidores de serviços de modo transparente, como se fosse um Serviço Web único.

Pretende-se estender a arquitetura proposta e adicionar novos módulos que contemplem a execução das composições. Tão logo as composições são descobertas e salvas no repositório, elas são processadas por um gerador de WS-BPEL. Este componente gera os arquivos WS-BPEL que descrevem essa composição e os armazena em um repositório de WS-BPEL. Finalmente, o cliente pode invocar uma composição através de um único pedido dirigido ao Serviço Web da plataforma. A solicitação será expedida por um módulo Gerenciador de Requisições, que gerencia as requisições em andamento. Este módulo ativa o Módulo Orquestrador, que executa o arquivo WS-BPEL

[18] http://www.oasis-open.org/committees/wsbpel
[19] http://ode.apache.org/

correspondente. Os resultados produzidos pela composição são então devolvidos ao cliente. Com isso, a plataforma passa a ser parte de um *framework* completo de composição de serviços, capaz de descobrir, compor e executar composições automaticamente.

A implementação do protótipo está sendo melhorada de modo a permitir a identificação de composições que possuem estruturas de controle e paralelismo durante a execução. Outra melhoria futura é a adição de mediadores de ontologias. Devido a diferentes contextos e pontos de vista, espera-se que pessoas e organizações acabem desenvolvendo diferentes ontologias para representar o mesmo domínio, ou partes dele. Como resultado, a plataforma irá tornar-se capaz de mediar ontologias e identificar os mesmos conceitos presentes em ontologias diferentes.

6.4 PUBLICAÇÕES

A pesquisa decorrente do presente trabalho resultou na publicação de dois artigos sobre a abordagem citada para a descoberta e composição de Serviços Web Semânticos:

- O primeiro artigo, intitulado "*Discovery of Semantic Web Services Compositions based on SAWSDL Annotations*", foi publicado e apresentado no *IEEE International Conference on Web Services (ICWS) 2012*, com qualificação A1 atribuída pelo Qualis, da CAPES.
- O segundo artigo, intitulado "*A Platform for Discovery and Execution of Semantic Web Services Compositions*", foi publicado e apresentado no evento *The 2012 International Conference on Semantic Web and Web Services* (SWWS), que possui qualificação A2.

REFERÊNCIAS

ALFARO, Luca de; HENZINGER, Thomas. **Interface Theories for Component-Based Design**. Proceedings of the First International Workshop on Embedded Software (EMSOFT). Volume 2211, pp. 148-165. 2001.

AL-MASRI, Eyhab; MAHMOUND, Qusay. **Discovering Web Services in Search Engines**. IEEE Internet Computing, volume 12 (3), pp 74-77. 2008.

ANKOLEKAR, Anupriya. **OWL-S: Semantic Markup for Web Services**. Disponível em: http://www.daml.org/services/owl-s/1.0/, 2003. Acesso em: 16 ago. 2012.

AZMEH, Zeina; HUCHARD, Marianne; MESSAI, Nizar; TIBERMACINE, Chouki; URTADO, Christelle; VAUTTIER, Sylvain. **Many-Valued Concept Lattices for Backing Composite Web Services**. 4th European Conference on Software Architecture (ECSA). Copenhagen, 2010.

BALANCIERI, Renato. **Um Método Baseado Em Ontologias Para Explicitação De Conhecimento Derivado Da Análise De Redes Sociais De Um Domínio De Aplicação**. Tese de Doutorado em Engenharia e Gestão do Conhecimento – Programa de Pós-Graduação em Engenharia e Gestão do Conhecimento, Universidade Federal de Santa Catarina. 2010.

BELOUADHA, Fatima-Zahra; OMRANA, Hajar; ROUDIÈS, Ounsa. **A model driven approach for Composing SAWSDL semantic Web Services**. IJCSI International Journal of Computer Science Issues. Volume 7. 2010.

BERARDI, Daniela. **Automatic Service Composition. Models, Techniques and Tools**. Tese para obtenças do título de PhD na Universidade de Roma “La Sapienza”. 2005.

BERNERS-LEE, Tim; HENDLER, James; LASSILA, Ora. **The Semantic Web: A New Form of Web Content that Is Meaningful to Computers Will Unleash a Revolution of New Possibilities**. Scientific American, 2001.

BOISSEL-DALLIER, Nicolas; LORRÉ, Jean-Pierre; BENABEN, Frédérick. **Management Tool for Semantic Annotations in WSDL**. On the Move to Meaningful Internet Systems: OTM 2009 Workshops. Volume 5872, pp. 898–-906. 2009.

CARDOSO, Jorge. **Semantic Web Services: Theory, Tools and Applications**. Estados Unidos, 2006.

CASATI, Fabio; SAYAL, Mehmet; SHAN, Ming-Chien. **Developing e-services for composing e-services**. Proceedings of 13th International Conference on Advanced Information Systems Engineering (CAiSE). Suiça, 2001.

CHARIF, Yasmine; SABOURET, Nicolas. **An Overview of Semantic Web Services Composition Approaches**. Electrical Notes Theory Computing Science, Vol. 146 (1), pp. 33–41. 2006.

CHINNICI, Roberto; MOREAU, Jean-Jacques; RYMAN, Arthur; WEERAWARANA, Sanjiva. **Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language**. Disponível em: http://www.w3.org/TR/wsdl20/, 2007. Acesso em: 16 ago. 2012.

CORCHO, Óscar; FERNÁNDEZ-LÓPEZ, Mariano; GÓMEZ-PÉREZ, Assunción. **Methodologies, tools and languages for building ontologies: where is their meeting point?**. Data and Knowledge Engineering, volume 46, p. 41-64. 2003.

CURBERA, Francisco; KHALAF, Rania; MUKHI, Nirmal; TAI, Stefan; WEERAWARANA, Sanjiva. **The next step in Web Services**. Communications of the ACM - Service-oriented computing. Volume 46, pp 29-34. 2003.

DUSTDAR, Schahram; SCHREINER, Wolfgang. **A survey on web services composition**. International Journal of Web and Grid Services, volume 1 (1), p.1-30. 2005.

FONSECA, André. **Desenvolvimento de um plugin para composição de serviços web utilizando a plataforma Eclipse**. Monografia para obtenção do grau de Bacharel em Ciência da Computação – Universidade Federal da Bahia. Salvador, Bahia, 2008.

GAO, Xiang, YANG, Jian; PAPAZOGLOU, Michael. **The Capability Matching of Web Services**. Proceedings of the IEEE International Symposium on Multimedia Software Engineering. pp. 56-63. California, USA, 2002.

HASHEMIAN, Seyyed Vahid; MAVADDAT, Farhad. **A Graph-Based Approach to Web Services Composition.** Proceedings of the The 2005 Symposium on Applications and the Internet (SAINT 2005). Itália, 2005.

KELLER, Uwe; RUBÉN, Lara; LAUSEN, Holger; FENSEL, Dieter. **Semantic Web Service Discovery in the WSMO Framework**. Cardoses, J. (ed.) Semantic Web: Theory, Tools and Applications, Idea Publishing, USA (2006).

KELLER, Uwe; RUBÉN, Lara; LAUSEN, Holger; POLLERES, Axel; PREDOIU, Livia; TOMA, Ioan. **Semantic Web Service Discovery, WSMX working draft**. Disponível em http://www.wsmo.org/TR/d10/v0.2/. 2005. Acesso em: 16 ago. 2012.

KOIVUNEN, Marja-Riitta; MILLER, Eric. **W3C Semantic Web Activity**. Semantic Web Kick-off Seminar. Finlândia, 2001. Disponível em http://www.w3.org/2001/12/semweb-fin/w3csw. 2005. Acesso em: 16 ago. 2012.

KOPECKÝ, Jacek; VITVAR, Tomas; BOURNEZ, Carine; FARELL, Joel. **SAWSDL: Semantic Annotations for WSDL and XML Schema**. IEEE Internet Computing, vol. 11, no. 6, pp. 60-67, 2007.

KOURTESIS, Dimitrius; PARASKAKIS, Iraklis. **Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery**. ESWC'08 Proceedings of the 5th European semantic web conference on The semantic web: research and applications. pp. 614-628. Berlin, 2008

LÉCUÉ, Freddy; WAJID, Usman; MEHANDJIEV, Nikolay. **Negotiating Robustness in Semantic Web Service Composition.** Proceedings of the European Conference on Web Services (ECOWS). pp.75-84. 2009.

LI, Hongqi; WU, Zhuang. **Research on Distributed Architecture Based on SOA**. International Conference on Communication Software and Networks. Pp. 670-674. 2009.

LU, Gehao; WANG, Tengfei; ZHANG, Guojin; LI, Shijin. **Semantic Web Services Discovery Based on Domain Ontology.** World Automation Congress (WAC). Mexico, 2012.

MACHADO, Guilherme Bertoni; SIQUEIRA, Frank; MITTMANN, Robinson; VIEIRA, Carlos Augusto Vieira E. **Embedded Systems Integration Using Web Services**. In: V INTERNATIONAL CONFERENCE ON NETWORKING - ICN'06, 2006, Mauritius. IEEE Computer Society Press, 2006.

MARCONI, Annapaola; PISTORE, Marco. **Synthesis and Composition of Web Services**. Livro Formal Methods for Web Services. pp 89-157. 2009.

MARTIN, David; BURSTEIN, Mark; MCDERMOTT, Drew; MCILRAITH, Sheila; PAOLUCCI, Massimo; SYCARA, Katia; MCGUINNESS, Deborah; SIRIN, Evren; SRINIVASAN, Naveen. **Bringing Semantics to Web Services with OWL-S.** World Wide Web. Volume 10 (3), pp. 243-277. Estados Unidos, 2004.

MCILRAITH, Sheila; SON, Tran Cao; ZENG, Honglei. **Semantic Web Services.** IEEE Intelligent Systems. Special Issue on the Semantic Web. Março, 2001.

MESMOUDI, Amin; MRISSA, Michaël; HACID, Mohand-Saïd. **Combining configuration and query rewriting for Web service composition.** Proceedings of the International Conference on Web Services (ICWS). pp.113-120. 2011.

MONTEIRO, Filipe Luiz Mélo da Costa. **Web Semântica na Automação de Composição de Web Services**. Monografia para obtenção do grau de Bacharel em Engenharia da Computação – Universidade Federal de Pernanbudo. Recife, 2008.

OLIVEIRA, Douglas; MENEGAZZO, Cinara; CLARO, Daniela Barreiro. **Uma Análise Conceitual das Linguagens Semânticas de Serviços Web focando nas composições: comparação entre OWL-S,**

WSMO e SAWSDL. IADIS Conferência IberoAmericana WWW/Internet (CIAWI 2009). Espanha, 2009.

OVERHAGE, Sven; THOMAS, Peter. **WS-Specification: Specifying Web Services Using UDDI Improvements**. Web, Web-Services, and Database Systems. Lecture Notes in Computer Science, 2003, Volume 2593/2003, 100-119.

PAOLUCCI, Massimo; KAWAMURA, TAKAHIRO; PAYNE, Terry R.; SYCARA, Katia. **Semantic Matching of Web Services Capabilities**. First International Semantic Web Conference on The Semantic Web. Estados Unidos, 2002.

PAUTASSO, Cesare; ZIMMERMAN, Olaf; LEYMANN, Frank. **Restful web services vs. "big"' web services: making the right architectural decision**. 17th International World Wide Web Conference Archieves. Pequim, p. 805-814, 2008.

PRAZERES, Cássio Vinicius Serafim. **Serviços Web Semânticos: da modelagem à composição.** Tese de Doutorado em Ciências – Ciências da Computação e Matemática Computacional. Instituto de Ciências Matemáticas e de Computação – ICMC. Universidade de São Paulo. 2009.

PRAZERES, Cássio Vinicius Serafim; TEIXEIRA, César A. C.; PIMENTEL, Maria da Graça Campos. **Semantic Web Services discovery and composition: paths along workflows**. ECOWS'09: Proceedings of the 7th IEEE European Conference on Web Services. Holanda, 2009.

PREIST, Chris. A **conceptual architecture for semantic web services**. Proceedings of the International Semantic Web Conference (ISWC). 2004.

RAMALHO, Rogério Aparecido. **Web Semântica: aspectos interdisciplinares da gestão de recursos informacionais no âmbito da Ciência da Informação**. Dissertação de Mestrado em Ciência da Computação – Faculdade de Filosofia e Ciências, Universidade Estadual Paulista. Marília, 2006.

RAO, Jinghai; SU, Xiaomeng. **A Survey of Automated Web Service Composition Methods**. Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC). California, Estados Unidos, 2004.

RODRIGUEZ-MIER, Pablo; MUCIENTES, Manuel; LAMA, Manuel. **Automatic web service composition with a heuristic-based search algorithm.** Proceedings of the International Conference on Web Services (ICWS). pp.81-88. 2011.

ROMAN, Dimitru; DOMINGUE, John; STOLLBERG, Michael. **Web Service Modeling Ontology (WSMO) - An Ontology for Semantic Web Services**. W3C Workshop on Frameworks for Semantics in Web Services. Austria, 2005.

SENA, Vanessa Aline dos Santos. **Incorporação da similaridade semântica no owl-s composer**. Monografia para obtenção do grau de Bacharel em Ciência da Computação – Universidade Federal da Bahia. Salvador, Bahia, 2009.

ELGAZZAR, Khalid; HASSAN, Ahmed E.; MARTIN, Patrick. **Clustering WSDL Documents to Bootstrap the Discovery of Web Services.** Proceedings of the International Conference on Web Services (ICWS). pp.147-154. 2010.

SHADBOLT, Nigel; BERNERS-LEE, Tim; HALL, Wendy. **The Semantic Web Revisited**. Journal IEEE Intelligent Systems, volume 21 (3). pp. 96-101. Estados Unidos, 2006.

SILVA, Laryssa; BRAGA, Regina; CAMPOS, Fernanda. **Scientific Workflow Composition in E-Science**. Proceedings of the 2011 25th Brazilian Symposium on Software Engineering (SBES). pp 273-282. 2011.

TAMILARASE, K.; RAMAKRISHNAN, M.. **Design of an intelligent search engine-based UDDI for web service discovery**. International Conference on Recent Trends In Information Technology (ICRTIT), 2012.

TRAN, Vuong Xuan; PUNTHEERANURAK, Sutheera; TSUJI, Hidekazu. **A new service matching definition and algorithm with SAWSDL**. DEST '09 3rd IEEE International Conference on Digital Ecosystems and Technologies. pp.371-376. 2009.

VALLET, David; CANTADOR, Ivan; FERNANDEZ, Miriam, CASTELLS, Pablo. **A Multi-Purpose Ontology-Based Approach for Personalized Content Filtering and Retrieval**. Proceedings of the First International Workshop on Semantic Media Adaptation and Personalization, p.19-24. Estados Unidos, 2006.

WEI, Dengping; WANG, Ting; WANG, Ji; BERNSTEIN, Abraham. **SAWSDL-iMatcher: A customizable and effective Semantic Web Service matchmaker.** Web Semantics: Science, Services and Agents on the World Wide Web 9. Pp 402- 417. 2011.

ZHANG, Ruoyan; ARPINAR, I. Budak; ALEMAN-MEZA, Boanerges. **Automatic Composition of Semantic Web Services**. Proceedings of the International Conference on Web Services (ICWS). pp.38-41. 2003.

ANEXO A – Tempos de composição avaliados

1000			
Startup (ms)	Target (ms)	Graph (ms)	Selection (ms)
2136	30	13513	684
2149	31	13553	724
2297	30	14332	745
2118	37	13605	731
2091	30	13451	696
2189	30	13230	702
2134	33	13288	760
2147	30	13300	680
2165	28	13639	732
2159	27	13835	736
2581	30	13472	684
2188	28	13533	684
2115	30	13638	689
2146	38	13588	715
2160	35	14310	710

2000			
Startup (ms)	Target (ms)	Graph (ms)	Selection (ms)
2251	36	21662	698
2209	40	21795	730
2154	35	21439	680
2111	34	21650	727
2202	41	21609	742
2296	44	21978	692
2111	35	22337	688
2486	35	21260	685
2138	43	21302	771
2128	34	22399	704

2335	34	21048	686
2114	36	21246	684
2155	36	21847	694
2109	35	20888	669
2128	35	22136	674

3000			
Startup (ms)	Target (ms)	Graph (ms)	Selection (ms)
2169	41	31205	698
2083	41	30006	668
2120	42	30919	724
2143	42	31136	710
2201	41	30981	702
2229	43	30943	693
2120	45	32465	677
2121	52	31272	724
2091	42	30809	726
2128	43	31051	728
2244	42	33018	768
2135	41	31069	723
2495	54	33440	743
2191	63	32502	714
2129	43	31572	693

4000			
Startup (ms)	Target (ms)	Graph (ms)	Selection (ms)
2117	52	40269	681
2116	48	39203	695
2303	51	42880	679
2209	49	38848	686
2176	48	38125	678

2610	75	40082	726
3173	62	40142	702
2494	50	39195	672
2402	51	39655	678
2218	49	39408	687
2283	48	39672	689
2184	48	39181	707
2214	50	38616	676
2192	53	39113	682
2298	59	40491	709

5000			
Startup (ms)	Target (ms)	Graph (ms)	Selection (ms)
2678	73	52100	733
2307	68	53808	729
2316	61	50479	711
2253	61	49843	747
2164	63	49457	697
2212	61	49944	765
2184	74	55708	784
2367	64	50838	730
2197	60	51790	830
2329	71	48744	719
2436	61	51618	802
2479	65	54259	764
2858	61	49768	734
2619	64	50444	735
2197	61	50682	790