UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

OTÁVIO CALAÇA XAVIER

Serviços Web Semânticos Baseados em RESTful

Um Estudo de Caso em Redes Sociais Online

Goiânia
2011

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Serviços Web Semânticos Baseados em RESTful – Um Estudo de Caso em Redes Sociais Online

Autor(a): Otávio Calaça Xavier

Goiânia, 26 de Setembro de 2011.

Otávio Calaça Xavier – Autor

Cedric Luiz de Carvalho – Orientador

OTÁVIO CALAÇA XAVIER

Serviços Web Semânticos Baseados em RESTful

Um Estudo de Caso em Redes Sociais Online

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Sistemas de Informação.

Orientador: Prof. Cedric Luiz de Carvalho

Goiânia
2011

OTÁVIO CALAÇA XAVIER

Serviços Web Semânticos Baseados em RESTful

Um Estudo de Caso em Redes Sociais Online

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Ciência da Computação, aprovada em 26 de Setembro de 2011, pela Banca Examinadora constituída pelos professores:

Prof. Cedric Luiz de Carvalho
Instituto de Informática – UFG
Presidente da Banca

Prof. Frederico Luiz Gonçalves de Freitas
Centro de Informática – UFPE

Prof. Ricardo Couto Antunes da Rocha
Instituto de Informática – UFG

Otávio Calaça Xavier

Graduou-se em Redes de Comunicação pelo Instituto Federal de Educação, Ciência e Tecnologia de Goiás - IFG. Durante sua graduação, desenvolveu o ClasseV, uma ferramenta livre para videoconferência no Moodle, para EaD. Mestre em Ciência da Computação, com linha de pesquisa em Web Semântica, pela Universidade Federal de Goiás - UFG. Possui experiência com desenvolvimento Web desde 2004. Professor e Consultor principalmente nos seguintes temas: Inteligência Artificial, Desenvolvimento Web, Orientação a Objetos, AJAX, PHP e MVC. Ministrou palestras em mais de 20 eventos nacionais e internacionais e é entusiasta do Software Livre e da Web 2.0. Atualmente é pesquisador em Tecnologia da Informação, na Requisito Tecnologia.

Dedico este trabalho à minha família, em especial meus pais, Maria Vitória e Waldir, que sempre preocuparam-se com minha educação e me fizeram ser a pessoa que sou.

Agradecimentos

A realização deste trabalho contou com a colaboração e apoio de algumas pessoas. Para estas, deixo os meus sinceros agradecimentos:

Ao professor Cedric L. de Carvalho, pela orientação, conselhos, sugestões, atenção e críticas construtivas.

Ao amigo chileno Jair Abú Bechir Láscar Alarcón por ter sido como um co-orientador para mim e pelo desenvolvimento do Airetama.

Ao pesquisador Otávio Freitas Ferreira Filho, pelo desenvolvimento da ontologia *RESTfulGrounding* e pela atenção em responder meus e-mails.

Ao pesquisador Thorsten Möller, principal mantenedor da OWL-S API, pelas dicas, informações, sugestões e atenção dispensados na elaboração deste trabalho.

À todos os amigos e parentes que colaboraram de alguma forma para a concretização deste trabalho.

A coisa mais bela que podemos experimentar é o mistério. Essa é a fonte de toda a arte e ciências verdadeiras.

Albert Einstein,
Físico criador da teoria da relatividade e eleito, em 2009, o mais memorável físico de todos os tempos.

Resumo

Xavier, Otávio C.. **Serviços Web Semânticos Baseados em RESTful**. Goiânia, 2011. 135p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

As pesquisas acerca de Serviços Web Semânticos são voltadas, em sua grande maioria, à arquitetura SOAP. Esta arquitetura é pouco utilizada na Web 2.0 e, logo, em Redes Sociais *Online*. Este trabalho apresenta uma abordagem para implementação prática de descrição semântica em Serviços Web, baseados na arquitetura REST. Trata-se de uma arquitetura simplificada e que ganhou muito enfoque na Web 2.0, substituindo cada vez mais a arquitetura SOAP. O desenvolvimento da ferramenta, apresentada aqui, poderá preencher uma lacuna no processo de implantação da Web Semântica. As soluções existentes expõem uma visão teórica do assunto e não possuem implementações práticas. A solução proposta neste trabalho, relaciona padrões e tecnologias já existentes para o desenvolvimento de uma ferramenta livre e integrada. A partir dela, serviços de uma Rede Social *Online* popular são descritos. Por fim, é mostrado como realizar a descoberta, composição e invocação automatizada de tais serviços.

Palavras–chave

Web Semântica, Redes Sociais *Online*, Serviços Web, Ontologias, Serviços Semânticos, RESTful.

Abstract

Xavier, Otávio C.. **Semantic Web Services based on RESTful: A Case Study in Online Social Networks.**. Goiânia, 2011. 135p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

The researches on Semantic Web Services are aimed, mostly, the SOAP architecture. This architecture is rarely used in Web 2.0 and therefore in Online Social Networks. This dissertation presents an approach for practical implementation of semantic descriptions in RESTful Web Services. It is a simplified architecture that gained much focus on Web 2.0, increasingly replacing the SOAP architecture. The development of the tool, presented here, will fill a gap in the process of implanting the Semantic Web. Existing solutions expose a theoretical view and have no practical implementations. The solution proposed in this paper, relates existing standards and technologies to develop an integrated and free tool. From which, services of a popular Online Social Network are described. Finally, the automatic discovery, composition and invocation of such services are made.

Keywords

Semantic Web, Online Social Networks, Web Services, Ontologies, Semantic Services, RESTful.

Sumário

Lista de Figuras

Lista de Abreviaturas e Siglas

API - Application Programming Interface

BPEL4WS - Business Process Execution Language for Web Services

CERN - Conseil Européen pour la Recherche Nucléaire (Organização Europeia para a Pesquisa Nuclear)

DAML - DARPA Agent Markup Language

EXPRESS - EXPressing REstful Semantic Services

FBML - FaceBook Markup Language

FOAF - The Friend of a Friend

FQL - Facebook Query Language

GRDDL - Gleaning Resource Descriptions from Dialects of Languages

hRESTS - HTML for RESTful Services

HTML - HyperText Markup Language

HTTP - Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

IETF - Internet Engineering Task Force

IOPE - Inputs, Outputs, Preconditions and Effects

JADE - Java Agent DEvelopment Framework

JAX-RS - Java API for RESTful Web Services

JAXB - Java Architecture for XML Binding

JPEG - Joint Photographic Experts Group

JSON - JavaScript Object Notation

LSDIS - Large Scale Distributed Information Systems

MD5 - Message-Digest Algorithm 5

METEOR-S - Managing End-To-End OpeRations Services

MIME - Multipurpose Internet Mail Extensions

MWSDI - METEOR-S Web Services Discovery Infrastructure

NCSA - National Center for Supercomputing Applications

OIL - Ontology Inference Layer

OWL - Web Ontology Language

OWL-S - Web Ontology Language for Services

PDF - Portable Document Format
PHP - PHP: Hypertext Preprocessor
RDF - Resource Description Framework
RDFa - RDF in attributes
RDFS - RDF Schema
ReLL - Resource Linking Language
REST - Representational State Transfer
RPC - Remote Procedure Call
SA-REST - Semantic Annotation of Web Resources
SAWSDL - Semantic Annotations for WSDL
SDK - Software Development Kit
SEREDASj - Semantic REstful DAta Services with JSON
SGML - Standard Generalized Markup Language
SHOE - Simple HTML Ontology Extensions
SIOC - Semantically-Interlinked Online Communities
SOA - Service-Oriented Architecture
SOAP - Simple Object Access Protocol
SPARQL - SPARQL Protocol and RDF Query Language
SQL - Structured Query Language
SWS - Semantic Web Services
UDDI - Universal Description, Discovery and Integration
UFG - Universidade Federal de Goiás
UIUC - Universidade de Illinois em Urbana e Champaign
URI - Uniform Resource Identifier
URL - Uniform Resource Locator
USP - Universidade de São Paulo
W3C - World Wide Web Consortium
WADL - Web Application Description Language
WSDL - Web Services Description Language
WSDL-S - Web Services Description Language with Semantics
WSMO - Web Service Modeling Ontology
WWW - World Wide Web
XHTML - eXtensible HyperText Markup Language
XML - eXtensible Markup Language
XOL - XML-based Ontology-exchange Language
XSD - XML Schema Definition
XSL - eXtensible Stylesheet Language
XSLT - eXtensible Stylesheet Language for Transformation

CAPÍTULO 1

Introdução

A Web foi construída com o objetivo de ser um mecanismo para indexação e relacionamento entre documentos. Desta forma, possuía o propósito de funcionar como uma ferramenta em que pessoas de todo o mundo pudessem compartilhar documentos, de forma independente de tipos de máquinas ou sistemas [12]. Notoriamente, a Web tornou-se algo maior que a proposta inicial. Ela tem sido um importante meio de comunicação e a principal fonte de informação da última década. A linha evolutiva da Web pode ser dividida em três gerações, marcadas por evoluções conceituais e não tanto tecnológicas.

A primeira fase da Web (chamada de Web 1.0 ou Web Sintática) trata-se da versão proposta inicialmente por seu idealizador, Tim Berners-Lee. Os documentos eram indexados, endereçados e acessados de qualquer computador ligado à Internet a partir de um navegador. Entretanto, não havia interação entre os usuários. Existiam dois tipos de usuários, os escritores, que disponibilizavam os documentos, e os leitores, estes apenas liam o conteúdo disponibilizado. Com o crescimento do número de documentos e usuários, tornou-se cada vez mais necessária a possibilidade de interação entre eles. Com isso, a Web iniciou uma nova geração.

A Web Social é a segunda geração da Web (também chamada de Web 2.0). Nesta geração, a Web não disponibilizava apenas documentos e sim aplicações. Utilizando as mesmas tecnologias da Web Sintática, os documentos ganharam cada vez mais funcionalidades e passaram a ser considerados serviços ou aplicações. Nesta geração, o enfoque não é mais em documentos mas, em pessoas. Ela possui os princípios de garantir a interatividade e a colaboração entre usuários. Neste contexto surgiram as Redes Sociais *Online*, entre outros serviços sociais. Com as vantagens propostas inicialmente pela Web Sintática (independência de sistemas e endereçabilidade) e a característica social, esta geração da Web transformou-a em um importante meio de comunicação.

O crescimento da Web, como um meio de comunicação em que todos os usuários podem gerar e consumir informações, levou à necessidade cada vez maior da utilização de Serviços Web. As Redes Sociais *Online* começaram a disponibilizar Serviços Web que fornecem suas funcionalidades para outras aplicações. Entretanto, a arquitetura utilizada pelos serviços da Web Social difere daquela consolidada na primeira geração da Web.

Trata-se de uma arquitetura simplificada e baseada em recursos, chamada REST[1] [40]. Com os Serviços Web, as Redes Sociais *Online* possibilitaram a construção de Aplicações Sociais. Trata-se de aplicações com enfoque no relacionamento e colaboração entre pessoas.

Concomitante à popularização da Web Social, uma nova abordagem começou a ser conceitualizada. Trata-se da terceira geração da Web, a Web Semântica. Esta geração propõe que os dados e documentos presentes na Web sejam descritos semanticamente, e não apenas sintaticamente. Com isso, os relacionamentos semânticos entre os recursos da Web possibilitariam uma série de novas possibilidades. Uma vez que os dados estejam descritos de forma semântica, eles poderão ser interpretados através de processos automatizados. Com os dados contextualizados, torna-se possível a realização de inferências acerca do que é cada recurso e como ele pode interagir com outros recursos da Web. Logo, os recursos não seriam apenas documentos mas, a representação de qualquer entidade do mundo real (pessoas, musicas e organizações, por exemplo).

1.1 Motivação e Justificativas

A Web, em seus dias atuais, vivencia a sua segunda geração. Um dos principais adventos dessa geração (a Web Social), foram as Redes Sociais *Online*. Elas garantem interação e colaboracionismo contínuo entre seus usuários. Por si só, uma Rede Social *Online* não fornece conteúdo algum. Todo o conteúdo é criado e manipulado pelos seus usuários. Com isso, as Redes Sociais *Online* são grandes fontes de informações sobre o que as pessoas procuram e desejam saber. Tais informações podem ser sociais, como a situação e notícias sobre uma determinada pessoa. Também podem ser de âmbito educacional, como manuais, apostilas, tutoriais, aulas e comentários acerca de um determinado tema. Podem ser científicas, como discussões e proposições de comunidades acadêmicas, e de muitos outros propósitos.

A grande massa de dados e informações geradas em Redes Sociais *Online* têm tornado-se cada vez mais difícil de ser gerenciada. É um desafio, na geração atual da Web, garantir que a informação adequada chegue ao destino correto. Entretanto, esta geração da Web possui alguns problemas, como a baixa precisão nas buscas. Ferramentas de busca não interagem entre si, mas apenas exibem uma possível localização de onde encontrar a informação procurada. As buscas são baseadas em palavras chaves e dependem do vocabulário utilizado. Esses problemas ocorrem porque os dados, na Web atual, estão descritos apenas sintaticamente. O significado do conteúdo não é acessível por máquinas.

[1] *Representational State Transfer* - Transferência de Estado Representacional

A Web Semântica visa sanar estes problemas, descrevendo os dados de maneira que o computador consiga interpretar.

Descrever os dados semanticamente é apenas uma etapa para solução dos problemas de requisição da informação apontados. Mesmo que todos os dados estejam descritos semanticamente, se eles não forem acessíveis pelo mecanismo de busca, de nada adiantará. Nesse contexto, são utilizados os Serviços Web. Um Serviço Web é uma aplicação que consegue receber requisições Web de outras aplicações, e gerar respostas que possam ser interpretadas por elas. Serviços Web podem ser considerados uma poderosa ferramenta de integração entre diferentes aplicações e ferramentas. Entretanto, os Serviços Web, como são implementados, não garantem a interação automatizada entre aplicações Web.

Uma aplicação, ao disponibilizar um Serviço Web, fornece apenas sua descrição sintática. Novamente, é gerado o problema da incapacidade da interpretação de tais serviços por máquinas. Logo, uma aplicação que necessite de tal serviço, deverá ser adaptada para consumi-lo. Notoriamente, essa adaptação não é automatizada e deverá ser realizada por uma pessoa. Uma proposta para automatização da descoberta e invocação de serviços Web é descrevê-los semanticamente. Uma vez que os serviços Web estejam descritos semanticamente, torna-se possível, para máquinas, a interpretação de suas funcionalidades. Esta característica permite às aplicações identificarem automaticamente os serviços Web mais adequados em um determinado processo.

Em Redes Sociais *Online* os Serviços Web são utilizados para muitos outros propósitos além da realização de buscas. Elas também os utilizam para disponibilizar a manipulação dos dados dos usuários e para criação de Aplicações Sociais. Tais aplicações utilizam uma Rede Social *Online* como plataforma. Fornecer a descrição semântica de tais serviços permitirá novas possibilidades para tais aplicações. A integração entre Redes Sociais *Online* e a otimização em buscas são algumas delas. Entretanto, as Redes Sociais *Online* utilizam uma nova arquitetura de Serviços Web, a arquitetura REST [89]. Esta arquitetura é orientada a recursos e baseada no funcionamento da Web, garantindo vantagens para aplicações interativas como Redes Sociais *Online*.

A utilização da arquitetura REST, em Serviços Web de Redes Sociais *Online*, gera um novo desafio. As pesquisas em Serviços Web Semânticos tem sido maiores na arquitetura já consolidada na primeira geração da Web. Trata-se da arquitetura orientada ao serviço, utilizando SOAP[2] e WSDL[3][50]. Logo, existem poucas soluções para a abordagem voltada às Redes Sociais *Online*. O presente trabalho tem como principal motivação diminuir a carência de soluções para descrição de Serviços Semânticos na

[2] *Simple Object Access Protocol* - Protocolo Simples de Acesso a Objetos
[3] *Web Services Description Language* - Linguagem de Descrição de Serviços Web

arquitetura REST. Com a construção de uma solução para tal arquitetura, será possível garantir as vantagens da descrição semântica de Serviços Web em Redes Sociais *Online*, já mencionadas.

1.2 Objetivos

O principal objetivo deste trabalho é garantir a descoberta, invocação e composição automatizadas de Serviços Web em arquitetura REST, através da implementação de descrição semântica nestes serviços, tendo como caso de uso Redes Sociais *Online*. Para atingir este objetivo, é prioritária a utilização de padrões preestabelecidos e formalizados pelas entidades competentes. São focos adicionais deste trabalho, transformar os dados disponibilizados por Redes Sociais *Online* em conceitos semânticos e utilizar sistemas multiagentes, visando um maior paralelismo e escalabilidade do processo. Por fim, para validar o projeto desenvolvido, consultas semânticas devem ser realizadas nos dados obtidos de uma Rede Social Online de maneira automatizada.

Para atingir o objetivo principal, os seguintes objetivos específicos foram realizados:

- Apresentar uma visão geral sobre Web Semântica, Redes Sociais *Online* e Serviços Web, descrevendo as principais tecnologias e conceitos presentes nessas áreas do conhecimento.
- Realizar uma comparação entre diferentes arquiteturas de Serviços Web, apontando vantagens e desvantagens de cada arquitetura, expondo os pontos fortes na utilização da arquitetura REST em Redes Sociais *Online*.
- Apresentar as metodologias disponíveis para a construção de Serviços Web Semânticos, identificando o estado da arte para a descrição Semântica de Serviços Web, especificamente seguindo a arquitetura REST.
- Definir uma abordagem para implementação de Serviços Web Semânticos na arquitetura REST. Tal abordagem deverá utilizar padrões já estabelecidos para descrição semântica de serviços Web em outas arquiteturas. Adicionalmente, poderá ser investigada a possibilidade de integração entre serviços de diferentes arquiteturas.
- Projetar e desenvolver uma ferramenta que permita a implementação de serviços semânticos na arquitetura REST. Esta ferramenta deverá utilizar ferramentas previamente desenvolvidas e conceitos preestabelecidos, evitando retrabalho.
- Utilizar a ferramenta desenvolvida para descoberta e invocação de Serviços Web em Redes Sociais *Online*.
- Viabilizar mecanismos de conversão entre os dados disponibilizados por Redes Sociais *Online* em formatos semânticos, propiciando consultas semânticas em dados de Redes Sociais *Online*.

1.3 Metodologia

A metodologia para realização deste projeto compreendeu as seguintes etapas:

- **Estudo e Fundamentação Teórica**. Nesta etapa foi realizada a pesquisa acerca dos fundamentos teóricos do trabalho. Foram estudadas as três áreas do conhecimento em que este trabalho está submetido: Redes Sociais *Online*, Serviços Web e Web Semântica. Com isso, foram selecionadas as referências bibliográficas para obtenção de informações correlatas à pesquisa.
- **Pesquisa Acerca de Serviços Semânticos**. Esta etapa completa o estudo teórico realizado na primeira etapa. Os conceitos estudados naquela etapa são utilizados para o estudo específico acerca de Serviços Web Semânticos.
- **Estado da Arte e Análise de Trabalhos Correlatos**. Nessa etapa, foi feito um estudo acerca do estado da arte para descrição Semântica de serviços Web seguindo a arquitetura REST. Foram levantados alguns trabalhos com objetivos semelhantes ao desta pesquisa.
- **Implementação de Descrição Sintática e Semântica**. Nesta etapa, foi realizada a implementação da descrição sintática em serviços Web da arquitetura REST, com padrões estabelecidos pelas entidades competentes. Em seguida, foi realizada a descrição semântica de serviços Web convencionais, seguindo os padrões já formalizados para tal arquitetura. Também foi estudada uma solução teórica para a descrição semântica de Serviços Web na arquitetura REST.
- **Projeto e Desenvolvimento**. Nesta etapa, foi projetada e desenvolvida uma ferramenta para implementação de Serviços Web Semânticos na arquitetura REST. Foram utilizados os formatos já preestabelecidos para outras arquiteturas.
- **Implementação em Redes Sociais *Online***. Nesta etapa, foi realizada uma implementação de Serviços Web Semânticos em uma Rede Social *Online*. Para isso, foi utilizada a ferramenta desenvolvida na etapa anterior.
- **Documentação**. Esta etapa consiste na documentação de todas as etapas e tem como produto esta dissertação.

1.4 Organização da Dissertação

Este trabalho encontra-se organizado em 7 capítulos, além desta introdução. O Capítulo 2 destina-se a apresentar os fundamentos teóricos acerca da Web Semântica. Para isso, é realizado um histórico da Web. São expostos os principais conceitos e tecnologias da Web Semântica, como ontologias e sistemas multiagentes. Tecnologias como XML, JSON, RDF, OWL e SPARQL também são abordadas.

No Capítulo 3, é realizado um estudo acerca de Serviços Web. Nele, duas arquiteturas (REST e SOAP) são descritas e comparadas. Em seguida, é feita uma pesquisa acerca de três abordagens para implementação de Serviços Web Semânticos na arquitetura baseada em SOAP e WSDL. Estas três abordagens são comparadas, levantando-se pontos fortes e fracos de cada uma. Também é realizado um estudo acerca do estado da arte na descrição semântica de Serviços Web na arquitetura REST. Ao final, é feita uma discussão acerca de qual abordagem se ajusta melhor à tal arquitetura. No Capítulo 4 são apresentados alguns trabalhos com objetivos semelhantes aos desta pesquisa. Estes trabalhos são comparados em uma discussão, levantando-se as vantagens de cada um, possíveis de serem implementadas na proposta deste trabalho.

No Capítulo 5, são exibidas formas e padrões para descrição sintática e semântica de serviços Web. É feita a aplicação dos conceitos obtidos nos capítulos anteriores para implementação prática de Serviços Web Semânticos. Ao final, é descrita uma solução teórica (ontologia) para descrição semântica de serviços Web na arquitetura REST. O Capítulo 6 é dedicado à descrição do projeto e desenvolvimento de uma solução para implementação prática de Serviços Web Semânticos na arquitetura REST. Para isso, é realizado um levantamento de requisitos e a modelagem do sistema. Por fim, é apresentado como o sistema foi desenvolvido, suas classes, métodos e funcionalidades.

O Capítulo 7 descreve como a ferramenta criada no capítulo anterior foi utilizada para a realização de descrição Semântica de Serviços Web em Redes Sociais *Online*. Inicialmente, há uma descrição dos conceitos de Redes Sociais *Online* e um breve histórico. Foram criadas e descritas algumas ontologias para descrição dos serviços de uma Rede Social *Online* popular. Também é apresentada uma solução para o problema de transformação de dados em redes sociais para formatos ontológicos. Problema este que encontra-se em aberto na comunidade da Web. Neste Capítulo, é utilizado um arcabouço multiagente com características semânticas, chamado Airetama, para armazenamento dos dados transformados. Através dele, foi feita a consulta semântica nos dados obtidos da Rede Social *Online* e transformados em formatos semânticos.

O Capítulo 8 apresenta algumas considerações finais acerca da pesquisa apresentada neste trabalho. Nele é feita uma análise acerca dos resultados obtidos, vinculando-os aos objetivos definidos. Também são levantadas as contribuições que este trabalho trouxe para a comunidade acadêmica e tecnológica. São apontadas algumas limitações e desafios que podem ser estudados em trabalhos futuros.

CAPÍTULO 2

Web Semântica

A Web foi criada com o propósito de ser uma grande rede de documentos interligados entre si. O principal objetivo de seu criador, Tim Berners-Lee, ao desenvolvê-la, foi criar um mecanismo para indexação e relacionamento entre documentos científicos para que físicos de todo o mundo, pudessem compartilhar dados, de forma independente de máquina ou sistema [12].

Como o conteúdo publicado na Web foi, principalmente, destinado a seres humanos, não houve uma preocupação com a semântica mas apenas com a sintaxe da informação. Entretanto, com o crescimento da Web, alguns problemas tornaram-se cada vez mais comuns. Entre eles estão a baixa precisão nas buscas, a alta dependência dos resultados em relação ao vocabulário usado nas buscas e a falta de relação entre páginas de um mesmo assunto [8]. Tem sido cada vez mais necessário o processamento e a interpretação do conteúdo presente na Web por computadores, através de processos automatizados.

A Web Semântica, uma extensão da Web convencional, surgiu com o propósito de suprir tal necessidade. Proposta por Berners-Lee, Hendler e Lassila em 2001, a Web Semântica possui mecanismos para tratar semanticamente a informação [99]. Os recursos presentes na Web são ligados através de *links*. Com a adição de semântica nessas ligações, cada uma pode representar uma forma de relacionamento entre dados, contextualizando-os. Então, os relacionamentos semânticos tornam possível o processamento do significado das palavras em seu contexto [77].

Para garantir a semântica do conteúdo Web é necessário um processo composto de três etapas. Primeiramente, é necessário especificar e descrever formalmente os conceitos e termos através de ontologias. A partir disso, é necessário referenciar o conteúdo semanticamente, a partir das ontologias previamente criadas. Para essas duas etapas existem alguns padrões já bem consolidados, como o RDF[1], o RDFS[2] e o OWL[3], que são

[1] *Resource Description Framework* - Arcabouço para Descrição de Recursos
[2] *Resource Description Framework Schema* - Esquemas de Arcabouço para Descrição de Recursos
[3] *Web Ontology Language* - Linguagem para criação de Ontologias Web

descritos mais detalhadamente na seção 2.2.2. A terceira etapa consiste na interpretação, a partir de agentes de software, dos conceitos previamente referenciados semanticamente. Para isso, são usados sistemas multiagentes, descritos na seção 2.5.

A seguir, é feito um breve histórico da Web, para contextualização da Web Semântica. Também são descritas as principais tecnologias e conceitos necessários para garantir semântica na Web.

2.1 Histórico da Web

Proposta por Tim Berners-Lee, no CERN[4], em 1989, como um grande banco de dados de hipertexto com *links* tipados, a Web (ou *World Wide Web*, como ficou conhecida) não atraiu muito interesse [12]. Em 1990, Berners-Lee encontrou o entusiasta Robert Cailliau que colaborou com sua pesquisa. Os dois pesquisadores uniram suas ideias e apresentaram-nas em setembro de 1990 para o *European Conference on Hypertext Technology*. Entretanto, não encontraram nenhum financiador. Em dezembro de 1990, Berners-Lee já havia desenvolvido todas as ferramentas para fazer a Web funcionar: o *HyperText Transfer Protocol* (HTTP)[5], a *HyperText Markup Language* (HTML)[6], o primeiro navegador Web (chamado WorldWideWeb Browser[13]), o primeiro software servidor HTTP (conhecido como CERN HTTPD e mais tarde renomeado para W3C HTTPD) e o primeiro servidor Web (`http://info.cern.ch`), com algumas páginas Web que descreviam o projeto [14].

A partir do protocolo HTTP e da linguagem HTML, desenvolvidas por Berners-Lee, logo outros pesquisadores começaram a desenvolver navegadores e servidores para outras plataformas, auxiliando na popularização da Web [25]. Em 1993, o navegador Web gráfico precursor dos navegadores modernos foi lançado. O Mosaic foi desenvolvido por um grupo de pesquisadores, liderados por Marc Andreessen, do *National Center for Supercomputing Applications* (NCSA) na *University of Illinois at Urbana-Champaign* (UIUC). Após o término da graduação, Andreessen aliou-se com James H. Clark, fundando a Mosaic Communications Corporation, para comercialização do Mosaic [6]. Em abril de 1994, a companhia mudou o nome para Netscape, e o navegador para Netscape Navigator.

[4] *European Organization for Nuclear Research* é o maior laboratório de física de partículas do mundo. Possui 20 países europeus membros e 8 como observadores. Sua função principal é fornecer os aceleradores de partículas e outras infra-estruturas necessárias para a pesquisa em física de alta-energia.

[5] *Hypertext Transfer Protocol* é um protocolo da camada de aplicação de redes TCP/IP. Trata-se do principal protocolo da Web. Com ele, é possível fazer transferência de dados em hipermídia.

[6] *HyperText Markup Language* é uma linguagem de marcação para descrição de hipertexto. Trata-se da linguagem predominante para desenvolvimento de *sites* para Web.

Em maio de 1994, Robert Cailliau organizou a primeira Conferência Internacional WWW, no CERN, com o propósito de divulgar e discutir conceitos e tecnologias da Web [25]. Em setembro desse mesmo ano Berners-Lee fundou o *World Wide Web Consortium* (W3C) [26]. Tal consórcio reúne várias empresas para elaboração e publicação de padrões e recomendações com o objetivo de melhorar a qualidade da Web.

A partir de 1996, *sites* de comércio eletrônico começaram a se popularizar. Então surgiu uma tendência que ficou conhecida como "mania ponto com". Cada vez mais empresas virtuais eram criadas e supervalorizadas muito rapidamente. Muitas dessas empresas possuíam ações em bolsas de valores e, a partir de especulações, ganhavam valores irreais. Isso desencadeou uma crise que ficou conhecida como "estouro da bolha". Muitas empresas perderam capital e faliram [26]. A partir de então, a Web começou uma nova geração, chamada de Web 2.0 ou Web Social.

Web 2.0 é um termo criado por Tim O'Reilly da O'Reilly Media, em 2004, para descrever uma nova geração de *sites*, iniciada em 2001, com foco na interatividade e com características sociais. Na Web 2.0, o conteúdo é desenvolvido de forma colaborativa. Os internautas ganham a possibilidade de editar as informações e interagir com outros internautas, a partir da Web. Wikis, Redes Sociais, Blogs e *sites* de multimídia como Youtube, Flickr e Google Maps, são exemplos de *sites* desta geração da Web [88].

Web Semântica é a próxima geração da Web. Também chamada de Web 3.0, nela o foco é a semântica dos dados, e não mais a sintaxe como na primeira versão, ou o social, como na segunda. O objetivo, nesta nova geração, é garantir meios para que seja possível interpretar as informações presentes no conteúdo da Web, através de processos automatizados [16]. Esta geração ainda está em estudo, porém já existem padrões bem fundamentados e mecanismos que a viabilizam. Entre os *sites* que já utilizam alguns conceitos da Web Semântica, podem ser destacados o Facebook, Google, Lastfm e Wolframalpha.

2.2 Linguagens e Formatos da Web Semântica

Para que seja possível a implementação da Web Semântica, bem como das outras gerações da Web, são necessários alguns formatos padronizados. Cada formato ou linguagem possui um propósito específico no contexto na Web Semântica. Alguns, como o XML e JSON, que já são utilizados nas outras gerações da Web, possuem propósitos mais estruturais como a transmissão e como base para formatos mais complexos. Nesta sessão serão vistos os principais formatos utilizados para viabilização da Web Semântica.

2.2.1 XML e JSON

A XML (*eXtensible Markup Language*) é uma linguagem de marcação para propósitos especiais, usada para descrever e organizar dados para aplicações. Diferentemente da HTML suas etiquetas não são pré-definidas. O desenvolvedor poderá criar suas próprias etiquetas. Essa característica torna-a capaz de descrever diversos tipos de dados [20]. Recomendada pelo W3C, tal linguagem é utilizada para os mais diversos fins como formatação de texto, armazenamento das propriedades de mídias como áudio ou vídeo, imagens vetoriais e até para armazenamento de dados, como um banco de dados. Como será visto na seção 2.2.2, a linguagem XML é a base para os principais padrões presentes na Web Semântica, como o RDF, o RDFS e a OWL. Em Web Services, a XML é usada para transferência e representação de dados.

A XSL (*Extensible Stylesheet Language*) é uma linguagem para realização de transformações e apresentação dos dados contidos em um arquivo XML. Trata-se de uma linguagem que define regras de estilo para arquivos XML. Essa linguagem possui uma extensão chamada XSLT (*XSL Transformation*)[72]. Com ela é possível criar regras de transformação entre arquivos XML. Como vários formatos da Web são baseados em XML, esta tecnologia torna possível a transição entre dois formatos, através da elaboração de apenas um único arquivo XSLT.

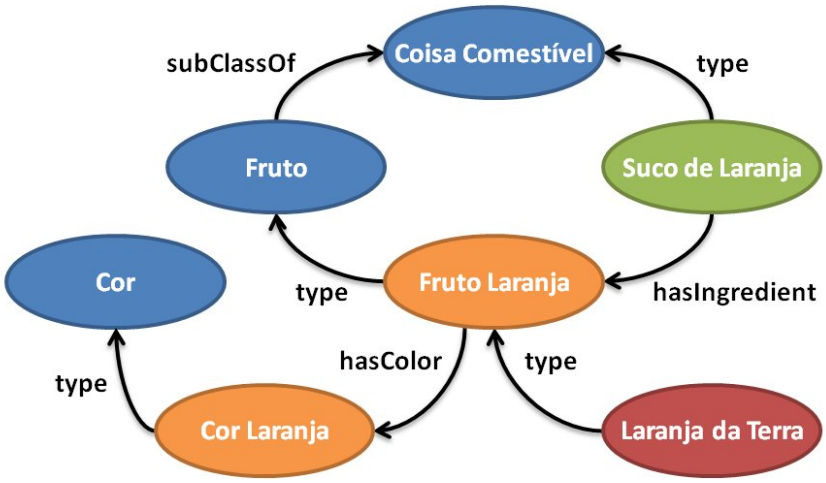
JSON (*JavaScript Object Notation*) é um formato para representação textual de dados, em objetos. Suas principais características vantajosas são os fatos de ser leve e simplificada [29]. Baseada na linguagem JavaScript, essa notação consegue descrever diferentes tipos de estruturas de dados (como objetos e arrays), seguindo a sintaxe da própria Javascript. Desta forma, torna-se muito familiar aos desenvolvedores em linguagens com sintaxe parecida com C (C++, C#, Java, JavaScript, PHP, Perl, Python, etc). Ideal para transferência de dados, consome menos banda e é mais intuitiva que a XML para essa tarefa. Devido a tais vantagens, é comum a utilização de JSON em *Web Services* desenvolvidos na arquitetura RESTful (veja mais detalhes na seção 3.2.2). Também há uma representação de RDF em JSON, com foco na interoperabilidade e na simplicidade de definição (mais detalhes serão vistos na seção 2.2.2)[3].

2.2.2 RDF, RDFS e OWL

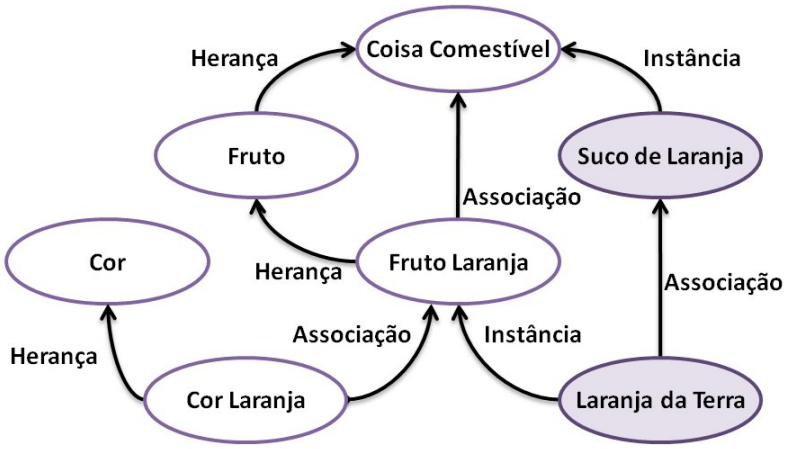
A Web Semântica pode ser considerada uma implementação dos conceitos de Redes Semânticas na Web. Uma Rede Semântica é uma forma de representação do conhecimento, definida em grafos direcionais. Nestes grafos, os vértices representam conceitos e as arestas definem relações semânticas entre os conceitos. Além da Web Semântica, podem ser citados os mapas mentais como implementação de redes semânticas. A principal diferença entre estas redes e grafos conceituais convencionais está nos relacionamentos.

Grafos definidos em sistemas de *frames* (como por exemplo em sistemas orientados a objetos) ou em banco de dados relacionais, são exemplos de grafos conceituais convencionais. Nestas abordagens, os relacionamentos são limitados, disponibilizando muito pouca semântica (na orientação a objetos, por exemplo, os relacionamentos podem ser divididos em apenas dois grupos: Associação e Herança).

A Figura 2.1 exibe uma comparação de representação do mesmo conhecimento em uma rede semântica (2.1(a)) e em grafos conceituais convencionais (2.1(b)). É possível observar que na rede semântica há uma maior expressividade e mais informações podem ser extraídas. Para representação das redes semânticas na Web, faz-se uso de três principais tecnologias: RDF, RDFS e OWL. Tais tecnologias serão descritas, nesta seção, a seguir.



(a) *Exemplo de Rede Semântica*



(b) *Exemplo de grafo conceitual convencional.*

Figura 2.1: *(a) e (b) representam os mesmos conceitos. Entretanto, (a) é mais expressiva por ser uma rede semântica.*

RDF (*Resource Description Framework*) é uma linguagem para representação de informações acerca de recursos, na Web. Tais informações normalmente referem-se a

metadados dos recursos, como título, autor, data de modificação, idioma, entre outros. Um recurso pode ser qualquer coisa, real ou virtual, que possa ser identificada na Web, mesmo que este recurso não possa ser requisitado diretamente pela Web. Exemplos incluem informações acerca de itens à venda em uma loja *online*, ou informações acerca das preferências de um usuário em uma rede social *online* [80].

RDF possui como propósito a descrição dos recursos com informação que possa ser interpretada por aplicações, e não apenas exibidas para seres humanos [80]. Para isso, cada recurso é identificado com um URI[7]. As informações acerca de um recurso são representadas como propriedades. A descrição das propriedades dos elementos, o relacionamento entre eles e entre suas propriedades, são feitos através de triplas.

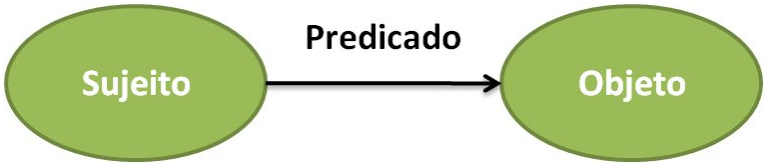


Figura 2.2: *Grafo do conceito de tripla, presente na RDF.*

O conceito de triplas pode ser considerado um conceito chave da Web Semântica. Uma tripla é uma declaração composta por três elementos: sujeito, predicado e objeto. Sendo assim, cada arco do grafo de uma rede semântica é representado por uma tripla. A Figura 2.2 exibe um grafo que ilustra o conceito de tripla. Nesta notação, sujeito e objeto são recursos (representados pelos vértices) e o predicado (ou propriedade) é o relacionamento semântico entre eles (representado sempre por uma aresta direcionada ao objeto). O sujeito, o predicado e o objeto são representados por URIs. Entretanto, o objeto pode vir a ser um literal e, nestes casos, é informado o próprio literal [80].

A Figura 2.3 representa uma parte do grafo da figura 2.1(a) em triplas RDF. É possível observar que, nessa notação, os URIs dos recursos são repetidos várias vezes. Para facilitar o uso de URIs, estes são substituídos por QNames (XML *qualified names*). Em um arquivo RDF podem ser definidos *namespaces* e os QNames utilizam-os para simplificar os URIs. A Figura 2.4 exibe as mesmas triplas definidas na figura 2.3, utilizando QNames.

[7] *Uniform Resource Identifier*

```
<http://www.exemplo.com/ontologias/frutos#laranja>
<http://www.exemplo.com/ontologias#ingredientOf>
<http://www.exemplo.com/ontologias#CoisaComestivel>

<http://www.exemplo.com/ontologias/frutos#laranja>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.exemplo.com/ontologias#Fruto>

<http://www.exemplo.com/ontologias/frutos#laranjaTerra>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.exemplo.com/ontologias/frutos#laranja>

<http://www.exemplo.com/ontologias/cores#laranja>
<http://www.exemplo.com/ontologias#colorOf>
<http://www.exemplo.com/ontologias/frutos#laranja>
```

Figura 2.3: *Exemplo de documento RDF em notação N-TRIPLES.*

```
@prefix : <http://www.exemplo.com/ontologias#>.
@prefix f: <http://www.exemplo.com/ontologias/frutos#>.
@prefix c: <http://www.exemplo.com/ontologias/cores#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

f:laranja :ingredientOf :CoisaComestivel.
f:laranja rdf:type :Fruto.
f:laranjaTerra rdf:type f:laranja.
c:laranja :colorOf f:laranja.
```

Figura 2.4: *Exemplo de documento RDF em notação N-TRIPLES, utilizando QNames.*

Existem diferentes notações para implementar o RDF. As mais comuns são RDF/XML, N-TRIPLES, Turtle, N3 e RDF/JSON. Cada uma atende a um propósito específico e há formas de conversão entre notações. A notação N-TRIPLES, utilizada nas Figuras 2.3 e 2.4, evidencia as triplas. Entretanto, esta notação possui como desvantagem a repetição de URIs. As notações N3 e Turtle são muito semelhantes e possuem foco na abreviação do código. Um exemplo destas notações pode ser visto na Figura 2.5. A notação RDF/XML descreve as triplas em XML. Nesta notação é mais difícil visualizar as triplas. Entretanto, não há repetições de URIs. Um exemplo de RDF/XML pode ser visto na Figura 2.6. A notação RDF/JSON é abreviada e mais fácil de ser interpretada por sistemas Web que a Turtle/N3. Um exemplo pode ser visto na figura 2.7.

```
@prefix : <http://www.exemplo.com/ontologias#>.
@prefix f: <http://www.exemplo.com/ontologias/frutos#>.
@prefix c: <http://www.exemplo.com/ontologias/cores#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

f:laranja
        :ingredientOf :CoisaComestivel;
        rdf:type :Fruto.
f:laranjaTerra rdf:type f:laranja.
c:laranja :colorOf f:laranja.
```

Figura 2.5: *Exemplo de documento RDF em notação Turtle/N3, utilizando QNames.*

```
<?xml version="1.0"?>
<rdf:RDF xmlns:f="http://www.exemplo.com/ontologias/frutos#"
         xmlns:c="http://www.exemplo.com/ontologias/cores#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:o="http://www.exemplo.com/ontologias#">
        <f:laranja rdf:about="http://www.exemplo.com/ontologias/frutos#laranjaTerra" />
        <rdf:Description rdf:about="http://www.exemplo.com/ontologias/cores#laranja">
                <o:colorOf>
                        <o:Fruto rdf:about="http://www.exemplo.com/ontologias/frutos#laranja">
                                <o:ingredientOf
                                   rdf:resource="http://www.exemplo.com/ontologias#CoisaComestivel" />
                        </o:Fruto>
                </o:colorOf>
        </rdf:Description>
</rdf:RDF>
```

Figura 2.6: *Exemplo de documento RDF em notação RDF/XML.*

```
{
  "http:\/\/www.exemplo.com\/ontologias\/frutos#laranjaTerra": {
    "http:\/\/www.w3.org\/1999\/02\/22-rdf-syntax-ns#type": [
      {
        "value": "http:\/\/www.exemplo.com\/ontologias\/frutos#laranja",
        "type": "uri"
      }
    ]
  },
  "http:\/\/www.exemplo.com\/ontologias\/cores#laranja": {
    "http:\/\/www.exemplo.com\/ontologias#colorOf": [
      {
        "value": "http:\/\/www.exemplo.com\/ontologias\/frutos#laranja",
        "type": "uri"
      }
    ]
  },
  "http:\/\/www.exemplo.com\/ontologias\/frutos#laranja": {
    "http:\/\/www.w3.org\/1999\/02\/22-rdf-syntax-ns#type": [
      {
        "value": "http:\/\/www.exemplo.com\/ontologias#Fruto",
        "type": "uri"
      }
    ],
    "http:\/\/www.exemplo.com\/ontologias#ingredientOf": [
      {
        "value": "http:\/\/www.exemplo.com\/ontologias#CoisaComestivel",
        "type": "uri"
      }
    ]
  }
}
```

Figura 2.7: *Exemplo de documento RDF em notação RDF/JSON.*

RDF Schema (também abreviado para RDFS, RDF-S, RDF(S) ou RDF/S) é uma extensão da RDF. Enquanto que RDF possui como principal propósito a descrição dos relacionamentos entre recursos e suas propriedades, RDFS objetiva a descrição de metadados acerca dos recursos e das propriedades [71]. Com RDFS é possível criar o conceito de herança (tanto de recursos, com `rdfs:subClassOf` como de propriedades, com `rdfs:subPropertyOf`), e também definir o âmbito (sujeitos possíveis) e o domínio (objetos possíveis) de uma propriedade (ou predicado) [76].

Utilizando RDF e RDFS é possível desenvolver modelos de representação do conhecimento bem elaborados. A partir da propriedade `rdf:type`, por exemplo, é possível definir um indivíduo (instância) a partir de uma classe. Em uma tripla com essa propriedade, o sujeito e o objeto podem ser uma classe, um indivíduo ou uma propriedade. É possível, então, criar indivíduos instâncias de outros indivíduos e propriedades instâncias de outras propriedades. Entretanto, RDFS possui algumas limitações na descrição de

recursos, como por exemplo [71]:

- Não há restrições localizadas de domínio e âmbito. Não é possível, por exemplo, dizer que a propriedade `hasChild` terá como domínio `Person` quando o âmbito for `Person` e `Eagle` quando o âmbito for `Eagle`.
- Não há restrições de existência e cardinalidade. Não é possível dizer, por exemplo, que todas as instâncias de `Person` possuem uma mãe que também é `Person`, ou que todas as pessoas possuem exatamente dois pais.
- Não há propriedades transitivas, simétricas ou inversas. Não é possível dizer, por exemplo, que `isPartOf` é uma propriedade transitiva, que `hasPart` é o seu inverso ou ainda que elas são simétricas.
- Dificuldade em prover implementação de raciocínio. Só é possível raciocinar a partir de lógica de primeira ordem.

OWL (*Web Ontology Language*) é uma linguagem para definição e implementação de ontologias na Web. Baseada em XML, RDF e RDFS, é mais expressiva que suas predecessoras, para definição de ontologias e relacionamentos entre elas. A OWL foi criada para suprir as necessidades de raciocínio eficiente (a partir da descrição dos recursos Web) e a conveniência de uma linguagem que alia as características do RDFS com todos os tipos de lógica. Essas necessidades fizeram com que o W3C definisse três sub-linguagens da OWL [7]:

- **OWL *Full***: O conjunto completo com toda a descrição da linguagem é definido como OWL Full. Ela implementa todas as características da RDF e RDFS e é totalmente compatível com elas. Além disso, dispõe de propriedades lógicas que suprem as necessidades apresentadas anteriormente.
- **OWL DL (*Description Logic*)**: Esta sub-linguagem não é totalmente compatível com RDF e RDFS. Para utilizar todas as características da RDF, poderá ser necessário criar outro documento. Entretanto, esta sub-linguagem promove a implementação de raciocínio eficiente.
- **OWL *Lite***: Uma versão básica e limitada da linguagem. Possui como vantagem a fácil compreensão e implementação. Como desvantagem, está a baixa expressividade.

O poder da OWL está na lógica descritiva. Com ela é possível fazer construções lógicas complexas, facilmente. A Figura 2.8 exibe um exemplo de restrição lógica e de cardinalidade, implementado em uma ontologia desenvolvida com OWL.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:someValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasVintageYear"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Figura 2.8: *Exemplo de utilização de OWL para implementação de restrições.*

2.2.3 Microformatos

Para implementação de marcações semânticas em arquivos HTML ou XHTML[90], são utilizados **microformatos**. Projetados para dar prioridade aos humanos e depois às máquinas, os microformatos são um conjunto de formatos, abertos e simples, de dados. A partir desta tecnologia, é possível reutilizar as tags HTML para adicionar metadados acerca da parte textual presente nestas. Esta abordagem permite que *softwares* processem a informação que inicialmente era destinada a seres humanos. A principal vantagem disso, é centralizar em um único lugar o conteúdo disponibilizado para humanos e máquinas [4]. A Figura 2.9 apresenta um exemplo de aplicação para microformatos.

```
<div class="vcard">
   <div class="fn">João da Silva</div>
   <div class="org">Companhia Exemplo</div>
   <div class="tel">604-555-1234</div>
   <a class="url" href="http://examplo.com/">http://examplo.com/</a>
</div>
```

Figura 2.9: *Exemplo de microformato para descrição pessoal (hCard).*

2.3 Ontologias

O termo "ontologia"originou-se no campo da filosofia e está relacionado com o estudo do ser e da existência. Introduzido por Aristóteles (384-222 a.C) como uma teoria sobre a natureza da existência, na filosofia, as ontologias são utilizadas para categorizar o

conhecimento. Na Ciência da Computação, o termo ontologia refere-se à artefatos designados ao propósito de modelar o conhecimento acerca de algum domínio, real ou imaginário [49]. O termo foi adotado pelos primeiros pesquisadores em Inteligência Artificial, na década de 80, aplicando-o em conjunto com a lógica matemática. Estes pesquisadores argumentavam que a partir das ontologias seria possível criar novos modelos computacionais que permitissem certos tipos de raciocínio automatizado [86].

Em 1993, Gruber, em uma das principais referências sobre o tema, descreve uma ontologia como sendo uma especificação formal e explícita de uma conceituação compartilhada [48]. Isto é, a conceituação de um modelo abstrato de algum fenômeno do mundo real, com elementos claramente definidos e passíveis de serem interpretados por máquinas. Este conceito deve ser compartilhado, de forma que uma ontologia representa um conhecimento consensual, ou seja, de grupo.

Hendler, em 2001, definiu o termo ontologia de forma mais simplificada e voltada à Web Semântica [58]. Ele descreve uma ontologia como sendo um conjunto de termos do conhecimento, incluindo o vocabulário, as interconexões semânticas e algumas regras simples de inferência e lógica, para alguns tópicos em particular. Hendler cita como exemplo uma ontologia para cozinhar. Neste exemplo seriam especificados os ingredientes, como agitar ou combiná-los, as diferenças entre ferver e fritar, a expectativa de como o produto deve ser consumido (comendo ou bebendo) e assim por diante.

Os principais componentes de ontologias são: **Classes**, que descrevem conceitos ou conjuntos de objetos de um mesmo tipo; **Atributos**, que são aspectos, propriedades ou características dos conjuntos de objetos ou conceitos descritos pelas classes; **Relações**, responsáveis por definir semanticamente os relacionamentos entre classes, indivíduos e atributos; **Indivíduos**, que representam entidades do mundo real (implementações de classes). Entretanto, as ontologias ainda possuem como componentes os **termos de função**, que definem estruturas complexas formadas por certas relações que podem ser utilizadas em lugar de termos individuais de uma declaração. As **Restrições**, para validação das relações entre classes e indivíduos. **Regras** lógicas (do tipo *if-then*) para inferência de novos conceitos e relações. **Axiomas**, que, como os axiomas da matemática, descrevem de forma lógica um domínio e são aceitos como verdadeiros. **Efeitos**, que estão relacionados com a mudança dos atributos e das relações [7].

Diversas linguagens já foram criadas para implementação e descrição de ontologias para a Web Semântica. Dentre elas, podem ser citadas SHOE[8], XOL[9], DAML[10],

[8] *Simple HTML Ontology Extensions* (SHOE) é um conjunto de extensões HTML projetadas para dar significado semântico à páginas Web.

[9] *Ontology Exchange Language* (XOL) é uma linguagem desenvolvida pelo BioOntology Core Group, tendo como propósito principal a troca de ontologias relacionadas com bioinformática.

[10] *DARPA Agent Markup Language* (DAML) é uma linguagem de marcação, baseada em RDF, para criação de representações para a Web, passiveis de serem interpretadas por máquinas.

OIL[11] e OWL. Entretanto, a única recomendada pelo W3C é a OWL. Trata-se de uma revisão da junção das linguagens DAML e OIL (DAML+OIL[12]). Como OWL é baseada em RDF e RDFS, muitas ontologias podem ser desenvolvidas utilizando apenas RDF e RDFS. Entretanto, a OWL possui propósito específico para a descrição de ontologias e consegue ser mais expressiva que suas predecessoras. Na próxima seção, estas três tecnologias serão abordadas mais detalhadamente.

2.4 Consultas Semânticas com SPARQL

SPARQL é um acrônimo recursivo para *SPARQL Protocol and RDF Query Language*. Trata-se de uma linguagem para consulta e manipulação de modelos RDF. Tornou-se uma recomendação pelo W3C em 2008 e, desde então, é a principal linguagem para consulta na Web Semântica [92]. Apesar de seu foco na consulta, alguns membros do W3C têm feito esforços acerca da manipulação de triplas RDF com SPARQL [94]. A Figura 2.10 exibe um exemplo de consulta e a Figura 2.11 de inserção de triplas RDF, em um repositório semântico, utilizando SPARQL.

```
1  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2  SELECT ?name ?website
3  FROM   <http://planetrdf.com/bloggers.rdf>
4  WHERE { ?person foaf:weblog ?website ;
5                  foaf:name ?name .
6          ?website a foaf:Document
7        }
```

Figura 2.10: *Exemplo de consulta em RDF com SPARQL.*

```
1  PREFIX dc: <http://purl.org/dc/elements/1.1/>
2  INSERT { <http://example/egbook3> dc:title  "This is an example title" }
```

Figura 2.11: *Exemplo de inserção em RDF com SPARQL/Update.*

A especificação da SPARQL ainda é composta por um protocolo que define um meio de transporte para as consultas SPARQL entre clientes e processadores [27]. Para isso, são utilizados Web Services que podem ser implementados em REST ou SOAP/WSDL. A partir desse protocolo é possível transmitir a consulta SPARQL para diferentes repositórios semânticos, transformando a Web em um grande e único repositório de informações.

[11] *Ontology Inference Layer* ou *Ontology Interchange Language* é uma linguagem para geração de ontologias, utilizando conceitos de lógica descritiva. Esta linguagem é parte do projeto IST OntoKnowledge.

[12] A DAML+OIL é a linguagem precursora da OWL. Possui raízes na DAML, OIL e SHOE e é desenvolvida utilizando XML e RDF.

2.4.1 Jena

Jena é um arcabouço desenvolvido em Java para construção de aplicações baseadas na Web Semântica. Ele provê um ambiente de programação para RDF, RDFS, OWL e SPARQL e inclui um motor de inferência baseado em regras. O Jena possui código fonte aberto e foi originado no *HP Labs Semantic Web Programme* [98]. Além das APIs para RDF e OWL e do motor de busca para SPARQL, o Jena ainda inclui dois sistemas para repositório semântico: TDB[98] e SDB[98].

Um repositório semântico é um sistema para armazenamento de triplas RDF, no qual podem ser realizadas consultas complexas com SPARQL e inferência com motores baseados em regras. No caso do Jena, as triplas podem ser gravadas em bancos de dados relacionais, utilizando SDB, ou em arquivos que podem ser acessados diretamente, utilizando TDB. A utilização de SDB possui vantagens como a maior administração, segurança e robustez do repositório. Além de viabilizar o uso de transações. Entretanto, TDB possui desempenho mais rápido e eficiente.

Jena é largamente utilizado no desenvolvimento da Web Semântica e na comunidade de pesquisadores. Por exemplo, o próprio W3C utiliza-o em seu serviço de validação de RDF, que checa se um documento RDF/XML é válido e gera um grafo a partir dele [82]. Na seção 5.3 será descrita uma API para desenvolvimento de *Web Services* Semânticos que utiliza o Jena em seu núcleo.

2.5 Sistemas Multiagentes

Sistemas Multiagentes são compostos por agentes que interagem entre si. Esta interação garante que as funcionalidades de um sistema possam ser distribuídas em vários agentes criando um maior paralelismo e possibilitando mais escalabilidade. Os agentes podem possuir comportamento cooperativo ou concorrente, tornando possível a modelagem de sistemas complexos [10]. Os sistemas multiagentes são uma subárea da Inteligência Artificial Distribuída. A ideia central destes sistemas é a de que, mesmo que os agentes não possuam inteligência, um conjunto de agentes pode alcançar um comportamento global inteligente.

Existem diversas definições para um agente, no contexto de sistemas multiagentes. Entretanto, todas as definições convergem para o fato de um agente ser essencialmente um software autônomo e com comportamentos semelhantes aos de um indivíduo em uma sociedade [45]. Para isso, o agente deve **perceber** o ambiente em que está, através de sensores, e **agir** nele, a partir de atuadores. Além da autonomia, um agente também deve ter **habilidade social**, que consiste na capacidade de comunicar-se com outros agentes [75].

Os agentes, em um sistema multiagentes, podem ser **reativos** ou **pró-ativos**. Agentes reativos percebem o ambiente em que estão imersos e respondem a estímulos provindos deste. Os pró-ativos possuem a capacidade de tomar a inciativa, ou seja, podem executar ações mesmo que não recebam estímulo externo algum [103].

Em um sistema multiagentes, podem existir várias sociedades de agentes que trabalham em conjunto, seguindo um comportamento colaborativo ou competitivo. Em um sistema colaborativo, os agentes cooperam na resolução de problemas complexos que estão além da capacidade de resolução de um agente isolado. Por outro lado, um sistema concorrente é composto por agentes com um objetivo em comum, entretanto cada um trabalha isoladamente. Esse tipo de sistema pode ser útil para simular alguma concorrência do mundo real.

Para implementação de sistemas multiagentes, uma das principais ferramentas é o JADE. JADE, acrônimo para *Java Agent DEvelopment framework*, é um arcabouço desenvolvido em Java para a construção de aplicações baseadas em agentes autônomos inteligentes [10]. O JADE provê funcionalidades para desenvolvimento de agentes que comunicam-se em redes remotas, e em dispositivos com recursos limitados (como dispositivos móveis, por exemplo).

CAPÍTULO 3

Serviços Web Semânticos

Fornecer serviços na Web tornou-se comum com a sua popularização. Entretanto, nem tudo que está na Web pode ser considerado um serviço. Para ser considerada um serviço, a aplicação Web deve ser capaz de receber requisições de outras aplicações (Web ou não) e gerar respostas possíveis de serem interpretadas por estas. Para isso, são usadas algumas técnicas descritas nessa seção. Para melhor entendimento da necessidade de utilização dos *Web Services* e viabilização destes, um histórico é apresentado na seção a seguir.

3.1 Histórico de Serviços Web

A História dos *Web Services* começa na criação da XML pelo W3C. Derivada da *Standard Generalized Markup Language* (SGML), a XML (mais detalhes na seção 2.2.1) data da década de 60. Entretanto, só ganhou popularidade na década de 90, época em que as linguagens *server-side*[1] começavam a surgir, viabilizando os negócios pela Internet. Esse movimento ficou conhecido como *eBusiness* [33]. Com o uso da XML, tornou-se possível atribuir significado e contexto a qualquer parte de informação transmitida através dos protocolos da Internet.

Além de garantir uma forma padronizada para representação de dados, XML foi usada na definição de várias outras especificações, como XSLT (seção 2.2.1) e WSDL (seção 3.2.1). Nos *Web Services*, XML é usada para definição do formato e da estrutura das mensagens trocadas. Sendo que as próprias mensagens podem ser arquivos XML.

A próxima tecnologia chave na história dos *Web Services* é o protocolo SOAP (mais detalhes na seção 3.2.1). Em 2000, o W3C recebeu uma submissão da especificação do SOAP. Tal protocolo possuía como principal objetivo, a unificação e substituição da comunicação RPC, que é proprietária. A ideia central do protocolo é encapsular os dados, serializados, a serem transmitidos entre as aplicações, em arquivos XML [33].

[1] Linguagens de programação usadas em uma arquitetura cliente-servidor, executadas no servidor. Exemplos: PHP, Java, Python, etc.

O movimento *eBusiness* logo avançou, com a definição padronizada de tecnologias livres, na Internet, para comunicação de aplicações. Isso levou à criação de uma tecnologia pura, baseada na Web e distribuída, que pode alavancar um novo conceito de arcabouço para comunicações padronizadas. Esse conceito foi chamado de *Web Services* [33].

Para que um Web Service possa ser usado, é essencial uma interface pública. Nela, são descritos e identificados os serviços disponíveis em um servidor. A primeira iniciativa nesse sentido foi a WSDL (detalhes na seção 3.2.1). Em 2001, o W3C recebeu a primeira submissão dessa linguagem e desde então passou a revisar sua especificação [33].

A primeira geração dos *Web Services* ainda contava com a especificação UDDI[11]. Originalmente desenvolvida pela UDDI.org, foi submetido ao OASIS[2], que continuou o desenvolvimento em colaboração com o UDDI.org. O UDDI foi criado com o objetivo de fornecer um registro centralizado com a descrição dos serviços disponíveis pela Web. Esse registro ficaria fora do domínio das organizações criadoras dos *Web Services*. Diferentemente do WSDL e do SOAP, o UDDI não foi muito aceito, tornando-se opcional e cada vez menos usado [33].

Em 2000, Roy Fielding, definiu um conceito que ele chamou de REST (veja seção 3.2.2), em sua tese de doutorado. Entretanto, o modelo apresentado por Fielding só ganhou popularidade em 2006 [89], com a migração da Web para o conceito Web 2.0 (mais detalhes na seção 2.1). O REST possui foco em recursos e não serviços. Baseado inteiramente na Web, utiliza-a como plataforma e não apenas como meio de acesso, como a implementação com SOAP e WSDL. Atualmente, grandes *sites* (como Google, Yahoo e Facebook) estão migrando para REST, devido à facilidade de uso e ao enfoque específico para Web.

3.2 Principais Arquitetura de Serviços Web

Como visto no histórico apresentado, existem duas principais arquiteturas para a implementação de serviços Web. A primeira, focada nas operações e em um contrato formal entre as partes, é baseada nos padrões SOAP e WSDL. A segunda, mais recente, focada em recursos e baseada no funcionamento da Web, é baseada nos principios REST. Nesta seção, cada arquitetura é descrita em maiores detalhes e em seguida é feita uma comparação entre elas.

[2]Mais informações em http://www.oasis-open.org

3.2.1 Arquitetura SOAP e WSDL

A arquitetura orientada a serviços, ou SOA, implementada com *Web Services*, utiliza comumente as tecnologias WSDL e SOAP. Como dito anteriormente, nessa arquitetura, a Web (ou mais especificamente o protocolo HTTP) é usada apenas como uma camada de transporte. O cliente e o servidor abstraem todas as tecnologias Web empregadas. Com isso, o desenvolvimento, utilizando *Web Services*, torna-se muito semelhante ao que utiliza métodos e classes locais. Ao requisitar um serviço, o objeto que representa o cliente passa a ter um método (conforme descrito em um contrato) que executa-o. Dessa forma, todas as rotinas remotas são tratadas como locais ao sistema orientado a objetos.

A *Web Services Description Language* (WSDL) é uma linguagem que define o contrato entre o servidor e o cliente. Nesse contrato, são especificados quais são os serviços, a identificação dos métodos que serão chamados nos clientes para execução destes, os tipos complexos (como classes, objetos e estruturas de dados) que os serviços poderão utilizar ou retornar, e opcionalmente o caminho para a classe que implementará os serviços. Esse contrato é fornecido em um arquivo pelo servidor, para o cliente. Através dele, o cliente passa a "conhecer"os métodos disponíveis no servidor e como chamá-los. A expressão WSDL pode fazer referência ao arquivo de contrato ou para a linguagem em que ele é escrito.

Existem duas abordagens distintas para o desenvolvimento de *Web Services* utilizando WSDL: *contract-first* e *contract-last*. Na abordagem *contract-first*, o contrato (arquivo WSDL) é criado antes, sem que seja escrito nenhum código de implementação do serviço. Logo, quando os serviços forem implementados, deverão seguir exatamente a especificação previamente definida no contrato. Na abordagem *contract-last*, o código fonte é escrito primeiro. Apenas no final, com os serviços já desenvolvidos, o contrato é criado [101]. É comum, nessa abordagem, o contrato ser criado automaticamente a partir da análise do código.

A abordagem *contract-last* é popular no desenvolvimento de *Web Services* por ser fácil. Não é necessário conhecimentos em WSDL, SOAP e XML Schema (XSD) para implementação desta abordagem. Isso se dá porque o contrato é feito a partir de reflexão nas classes do código fonte. Entretanto, isso pode se tornar um problema. O código fonte da aplicação é mais volátil que o contrato. É muito mais comum alterações no código do que no contrato da aplicação. Com a abordagem *contract-last*, o contrato passa a ser tão volátil quanto o código fonte da aplicação. Toda vez que este sofre alterações, será gerado um novo contrato. Com isso, alterações constantes nos clientes podem vir a ser necessárias. Para contornar esse problema, costuma-se criar várias versões para um mesmo serviço (métodos com nomes diferentes), mantendo a compatibilidade com clientes antigos. Outra solução baseia-se em não mudar o contrato mesmo que o código fonte mude. Quando for necessário mudá-lo, a modificação não deve causar

incompatibilidade com versões anteriores. Porém, é difícil fazer isso quando o contrato é gerado automaticamente [101].

Devido aos problemas na abordagem *contract-last*, a abordagem *contract-first* foi proposta. Entretanto, essa abordagem é mais trabalhosa. O desenvolvedor deverá escrever o contrato e conhecer a linguagem WSDL. Outro fator de complicação para essa abordagem é a necessidade de conhecimento de toda a arquitetura e como serão os serviços, antes de que eles sejam criados. Nessa abordagem o foco principal é **o que são** os serviços e não **como eles são** [101].

SOAP é um protocolo que encapsula as mensagens trocadas pelo servidor e o cliente. Tais mensagens são arquivos XML definidos de acordo com o padrão estabelecido por esse protocolo. Sempre que um cliente requisita um serviço, a mensagem é encapsulada no protocolo SOAP que, por sua vez, é encapsulado no protocolo HTTP. Ao chegar no servidor, o protocolo SOAP se encarrega de desencapsular as mensagens para tratamento pela aplicação (qual serviço foi chamado e quais são os parâmetros passados). A mensagem de resposta do servidor segue o mesmo fluxo.

3.2.2 Arquitetura REST

REST, como dito anteriormente, foi introduzido originalmente como um estilo arquitetural para construção de aplicações hipermídia de grande escala distribuídas. Nessa técnica, as mensagens trocadas são encapsuladas diretamente no protocolo HTTP. Há um foco nos recursos e não nas chamadas aos procedimentos/serviços, como em outras abordagens. No REST são feitas requisições para recursos e não serviços [40]. Essa abordagem pode ser interessante para aplicações em que é mais importante a interoperabilidade do que um contrato formal entre as partes [100]. Os recursos são representados por URIs e também podem ser chamados de métodos, caso representem alguma ação. Devido à grande escalabilidade do protocolo HTTP, REST quase sempre é usado em conjunto com este [89].

A arquitetura REST é baseada em 4 princípios [89, 40]:

- **Identificação de recursos através de URI**. A arquitetura REST foca principalmente em recursos para interação com os clientes. Tais recursos são identificados através de URIs [15]. Com isso, cada recurso tem um endereço global que pode ser usado para descoberta de serviços.
- **Interface Uniforme**. Os recursos são manipulados utilizando um conjunto de quatro operações: inserção, atualização, exclusão e listagem/leitura. Cada operação utiliza um método de requisição HTTP diferente. O método **GET** retorna o estado corrente do recurso em alguma representação. **PUT** insere um novo recurso, que

pode ser excluído com **DELETE**. **POST** transfere um novo estado para um recurso (atualização).

- **Mensagens auto-descritivas**. Os recursos estão desassociados de suas representações. Dessa maneira, estes podem ser acessados em diferentes formatos (HTML, XML, PDF, JPEG, texto puro, etc.). Metadados em relação aos recursos estão disponíveis e são usados, por exemplo, para negociar o formato apropriado da representação.
- **Interações de estado através de *hyperlinks***. Todas as interações com os recursos são sem estado (como o protocolo HTTP). Entretanto, os estados podem ser explicitamente transferidos. Existem várias técnicas para isso, por exemplo, o uso de cookies, campos *hidden* em formulários, ou reescrita de URI. Os estados podem ser embutidos nas mensagens de resposta para garantir a futura interação com estes.

Web Services REST costumam ser implementados de duas formas diferentes: *Hi-REST* e *Lo-REST*. Hi-REST é a implementação conforme descrito por Fielding, também chamada de RESTful. A implementação Lo-REST, por outro lado, utiliza apenas os métodos HTTP GET e POST. Isso se dá porque alguns *proxies* e *firewalls* não permitem outros métodos que não esses. Além disso, formulários XHTML admitem apenas esses métodos. Nessa implementação, GET é usado para requisições independentes e POST para qualquer outra. Para identificar a ação (inserção, atualização ou exclusão), pode ser usado um cabeçalho especial do protocolo HTTP (*X-HTTP-Method-Override*). Também pode ser passada, na requisição, uma variável específica para tal identificação (_method por exemplo) [89].

A arquitetura de *Web Services* em REST é simples, visto que são utilizados padrões W3C/IETF[3] bem conhecidos (HTTP, XML, URI, MIME por exemplo). A implementação, tanto do servidor quanto do cliente, é possível na grande maioria de linguagens de programação e sistemas operacionais. Tendo isso em vista, a barreira para adoção dessa arquitetura é muito pequena. O desenvolvedor pode testar os serviços no próprio navegador Web. A implantação é bem similar à de um Web site dinâmico. A arquitetura REST é escalável para um grande número de clientes, possui suporte a *caching*, clusterização e balanceamento de carga.

Para a implementação de Web Services REST, uma das principais ferramentas é o Restlet. Restlet é um arcabouço leve e de fácil compreensão desenvolvido em Java. Possui um núcleo pequeno, extensível através de *plugins*. Tal arcabouço inclui todos os conceitos da arquitetura REST, sendo possível o desenvolvimento de clientes e servidores.

[3] *Internet Engineering Task Force* (IETF) é uma comunidade internacional aberta com o foco na padronização e evolução da arquitetura da Internet. Site oficial: http://www.ietf.org.

Pode ser usado com a maioria dos padrões Web (como JSON, XML, WADL e Atom[4]). Com sua arquitetura extensível, o Restlet consegue integrar-se com Servlets [91], Spring [64], Jetty [102], JAXB [83] e muitas outras tecnologias. Também estão disponíveis *plugins* para a especificação JAX-RS [23] e até para tecnologias da Web Semântica (como RDF) [73].

3.2.3 Comparação entre Arquiteturas

A escolha da arquitetura REST, em relação à baseada em SOAP e WSDL, remove a necessidade de muitas decisões futuras [89]. *Web Services* SOAP/WSDL possuem várias camadas e complexidade, em alguns casos, supérflua. Caso seja necessária, a implementação de *Web Services* SOAP/WSDL em aplicações que implementam REST é bem mais trabalhosa que o contrário.

Devido ao maior número de camadas e protocolos, *Web Services* SOAP/WSDL consomem maior banda que a arquitetura REST [100]. Com isso, recomenda-se o uso de REST em aplicações que possam ser acessadas por dispositivos móveis ou de maneira ubíqua. Entretanto, *Web services* REST não possuem um contrato formal bem definido. Sendo assim, é necessário que o produtor (servidor) do serviço e seu consumidor (cliente) conheçam-o e estejam contextualizados. Porém, é possível usar WSDL com REST [79] ou utilizar WADL [51].

Baseada em XML, a *Web Application Description Language*, ou WADL, é uma linguagem para descrição de aplicações baseadas no protocolo HTTP, como *Web Services* REST. De acordo com Hadley (criador da WADL), a utilização deste padrão garante vantagens como a possibilidade de geração automática de código, baseada na especificação, para manipulação dos recursos e a configuração do cliente e do servidor a partir de um único formato portável.

Os arquivos gerados pela WADL possuem vários elementos para descrição de aplicativos Web. Entre eles, podem ser destacados os que fazem um mapeamento direto aos conceitos da arquitetura REST: Recurso (`Resource`), Requisição (`Request`), Resposta (`Response`), Representação (`Representation`) e Método (`Method`). Logo, conclui-se que o padrão WADL descreve sintaticamente *Web Services* RESTful assim como o WSDL descreve *Web Services* SOAP.

Como dito anteriormente, um serviço Web consiste em uma aplicação Web capaz de receber requisições e gerar respostas, interagindo com outras aplicações. Isso garante o aspecto dinâmico da Web. Entretanto, apenas a descrição semântica de tais serviços, possibilita a descoberta, composição, invocação, e o monitoramento dinâmicos destes.

[4] Atom é um formato para publicação de fontes Web que são periodicamente atualizadas, como por exemplo Blogs. É baseado em XML e metadados.

Existem muitas tecnologias sendo estudadas e desenvolvidas para garantir a descrição semântica de serviços Web. OWL-S[5], WSMO[6] e METEOR-S[7] podem ser destacadas como as mais maduras e desenvolvidas. Este capítulo destina-se à exposição destas três tecnologias. É dado enfoque em suas diferenças e semelhanças, propiciando uma comparação.

Objetivando facilitar a comparação, cada tecnologia será descrita em quatro funcionalidades necessárias para viabilizar serviços semânticos: **Descoberta de Serviços**, que define como o serviço é descrito para possibilitar buscas e descoberta. **Interoperabilidade**, a capacidade que a tecnologia tem para propiciar interoperabilidade entre serviços, descrevendo entradas e saídas. **Composição**, a descrição de como os serviços são formados, caracterizando seus sub-processos, caso existam. **Invocação**, que define como é descrito o processo de chamada para os serviços. Esta metodologia foi baseada em [18].

3.3 METEOR-S

METEOR é um acrônimo para *Managing End-To-End OpeRations*. Trata-se de um projeto desenvolvido no LSDIS Lab, na Universidade de Georgia, focado em técnicas de gerenciamento de fluxos de trabalho transacionais. O principal subprojeto do METEOR é voltado à descrição de fluxos de trabalho em serviços Web Semânticos. Este é chamado METEOR-S. A principal funcionalidade deste projeto, consistia em descrever semanticamente os ciclos de vida de serviços Web, por completo. Estendendo as especificações e normas da arquitetura SOA, METEOR-S adota uma abordagem evolutiva para dar suporte a semântica.

3.3.1 Descoberta de Serviços

Sistemas de descoberta eficientes visam sanar duas grandes necessidades principais em Serviços Web. A primeira, está relacionada com o registro dos serviços Web. Na abordagem disponibilizada pelo UDDI, o registro é feito de forma centralizada. Logo, requisitar um serviço específico, de acordo com a necessidade de alguma aplicação, pode se tornar uma tarefa crítica. A outra necessidade está relacionada com a capacidade de encontrar o serviço mais apropriado para uma determinada requisição. Para suprir essas duas necessidades, é disponibilizado a METEOR-S *Web Services Discovery Infrastructure* (MWSDI).

[5]OWL-S é uma ontologia para descrição de serviços Web, baseada em OWL.

[6]WSMO é um acrônimo para *Web Service Modeling Ontology*. Trata-se de uma ontologia criada para descrever vários aspéctos de serviços Web.

[7]METEOR é um acrônimo para *Managing End-To-End OpeRations*. METEOR-S é a principal pesquisa deste projeto.

MWSDI consiste em uma infraestrutura escalável, para publicação e descoberta semânticas de serviços Web. Nesta tecnologia, os registros de serviços Web são categorizados em domínios. Com isso, a busca por serviços de um determinado domínio podem ser direcionadas diretamente à ele. Por exemplo, um registro com serviços Web apenas relacionados com Viagens. Caso algum serviço de Viagem seja solicitado, o sistema de descoberta direcionaria a busca diretamente para tal registro. Em complemento a isto, a MWSDI adiciona semântica ao relacionamento entre o registro e o domínio. Isso provê maior eficiência na localização do registro correto, baseado nos requisitos da descoberta. Para isso é usada uma ontologia especializada, chamada *Registries Ontology*.

A necessidade de obter-se o serviço mais apropriado, em uma descoberta, também é suprida com semântica. Os registros UDDI armazenam arquivos WSDL que descrevem os serviços. Entretanto, tais arquivos possuem apenas a descrição sintática. A solução proposta pela MWSDI envolve a adição de semântica na descrição dos serviços Web. Tais descrições são armazenadas nos registros. Estas descrições são implementadas através de ontologias. Nesta infraestrutura, os registros podem ser disponibilizados por diferentes operadores[8]. Cada operador, pode ter seu conjunto de ontologias específicas relacionadas com os domínios de seu registros de serviços.

A infraestrutura de descoberta de serviços Web do METEOR-S é baseada em redes *peer-to-peer*. Tais redes possuem alta escalabilidade e autonomia, visto que cada *peer* é uma entidade independente. Com isso, cada registro é mantido por um *peer* que, pode ter suas próprias regras. Desta forma, cada registro pode conter serviços e ontologias distintas.

3.3.2 Interoperabilidade

Para garantir a interoperabilidade entre serviços Web, é necessário descrever semanticamente quatro elementos presentes nestes. Para cada operação de um serviço Web, existem **entradas**, **saídas**, **pré-condições** e **efeitos**. As entradas e saídas dizem respeito aos parâmetros de requisição e ao retorno das operações, respectivamente. As pré-condições especificam os requisitos necessários para que a operação seja invocada. Os efeitos descrevem as mudanças, em todo o sistema, que são esperadas depois da invocação de uma operação. Para descrever semanticamente esses elementos o METEOR propôs a WSDL-S. O conjunto de entradas, saídas, pré-condições e efeitos de um serviço é comumente chamado de IOPEs (*Inputs, Outputs, preconditions, effects*).

WSDL-S é um enriquecimento semântico da WSDL. Mais especificamente, o objeto *ModelReference* é usado para mapear um a um os elementos presentes no

[8]Operadores de registros de serviços Web são organizações que possuem uma instância de um registro público.

WSDL para ontologias. Para os tipos complexos exitem duas alternativas para adição de anotações semânticas. A abordagem de baixo nível e a de alto nível. Na abordagem de baixo nível, os significados dos elementos de tipos complexos, são caracterizados usando a *ModelReference*. Já na abordagem de alto nível, os tipos complexos são anotados utilizando *SchemaMapping* ou *ModelReference*. *SchemaMapping* é usado para fazer mapeamento muitos-para-um e um-para-muitos com modelos semânticos [37].

As pré-condições e os efeitos também são descritos pela WSDL-S. Uma operação com requisição e resposta pode estar associada com, no máximo, uma pré-condição e zero ou mais efeitos. METEOR-S também usa WSDL-S na descoberta de serviços. Os relacionamentos entre os registros e seus domínios é feito através da WSDL-S. Esta linguagem deu origem ao padrão *Semantic Annotation of WSDL* (SAWSDL) recomendado pelo W3C.

3.3.3 Composição

A etapa de ciclo de vida de serviços Web, envolve a criação de uma representação para processos Web. METEOR-S utiliza BPEL4WS para criar processos abstratos. Esta tecnologia provê um rico conjunto de funcionalidades para padrões de modelagem de fluxos de trabalho. O projeto de processos abstratos envolve três etapas.

A primeira etapa diz respeito à criação do fluxo do processo. Para isso são usadas ferramentas de controle de fluxo providas pelo BPEL4WS. Em seguida, é feita a representação dos requisitos de cada serviço no processo. Para isso, são especificados modelos do serviço que permitem, ao projetista dos processos, especificar uma descrição semântica para o serviço Web. Por fim, são especificadas as restrições do processo, para otimização.

3.3.4 Invocação

METEOR-S não implementa nenhum mecanismo específico para invocação de serviços Web. Na medida em que os serviços são acessados e invocados, são usados os mecanismos já definidos em serviços Web convencionais. Tais mecanismos envolvem padrões como WSDL e SOAP, já descritos em sessões anteriores.

3.4 WSMO

WSMO é um acrônimo para *Web Service Modeling Ontology*. Trata-se de uma iniciativa europeia com o intuito de prover um padrão para descrição semântica de serviços Web [18]. WSMO é uma ontologia baseada no *Web Service Modelling Framework*

(WSMF) [38]. WSMF descreve Serviços Web Semânticos em quatro elementos diferentes [31]:

- **Ontologias**, que proveem a descrição semântica das terminologias utilizadas nos outros elementos;
- **Metas**, que definem os problemas que são solucionados através do serviço Web em questão;
- **Serviços**, que descrevem os serviços Web e seus vários aspectos;
- **Mediadores**, que tratam da solução de problemas de interoperabilidade entre serviços.

Estes elementos são definidos por classes ontológicas na WSMO. Os relacionamentos entre essas classes podem ser vistos na Figura 3.1. A classe `NonFunctionalProperties` é utilizada para descrever propriedades não funcionais. Estas propriedades envolvem desde tipos de entradas e saídas até parâmetros de qualidade do serviço (QoS).

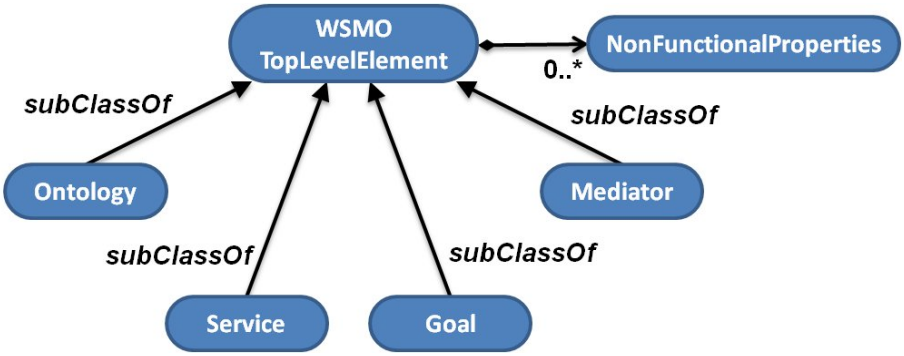


Figura 3.1: *Classes principais da ontologia WSMO, e seus relacionamentos.*

3.4.1 Descoberta de Serviços

WSMO pode implementar múltiplos mecanismos de descoberta. Com isso, o usuário pode especificar, através de propriedades não funcionais de sua Meta, qual mecanismo utilizar em uma requisição. WSMO disponibiliza diferentes níveis de descoberta [95]:

- **Descoberta por Palavra-Chave**, que realiza a busca apenas nas propriedades não funcionais;
- **Descoberta Funcional**, que consiste em uma descoberta leve baseada apenas nas pré-condições do serviço;
- **Descoberta baseada em Lógica Descritiva**, que usa uma ontologia dedicada para anotar a lista de serviços descobertos;

- **Descoberta baseada em Instâncias**, que consiste na manipulação da descrição do serviço, em nível de instância. Com isso, é possível buscar, dinamicamente, informações complementares durante o processo de descoberta. Isso é realizado, através de uma interface para contrato de servico, que integra dinamicamente as informações obtidas, no contexto de raciocínio;
- **Descoberta baseada Qualidade do Serviço (QoS)**, que provê um mecanismo que coincide os requisitos de QoS do usuário com a descrição semântica dos serviços providos.

3.4.2 Interoperabilidade

Como dito anteriormente, a ontologia WSMO provê o elemento Serviço, representado pela classe *Service*. A função dessa classe é representar as operações e transformações de dados do serviço. Um serviço WSMO está associado com um elemento Capacidade (*Capability*), que descreve seu funcionamento. Também está relacionado com uma ou mais Interfaces, que descrevem como a capacidade será cumprida. Esses relacionamentos podem ser vistos na Figura 3.2.

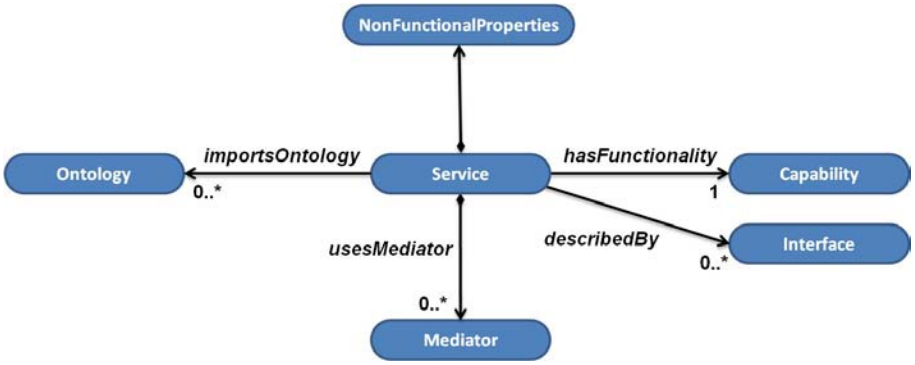


Figura 3.2: *Classe Service da ontologia WSMO, e seus relacionamentos.*

Como pode ser visto na Figura 3.3, cada Interface está associada com um elemento de Coreografia (*Choreography*), um de Orquestração (*Orchestration*) e um Mediador (*Mediator*). A Coreografia diz respeito às informações necessárias para comunicação com um serviço. A Orquestração trata de como o serviço utiliza outros serviços para realizar sua Capacidade. A Mediação provê soluções para conflitos e problemas de interoperabilidade [18].

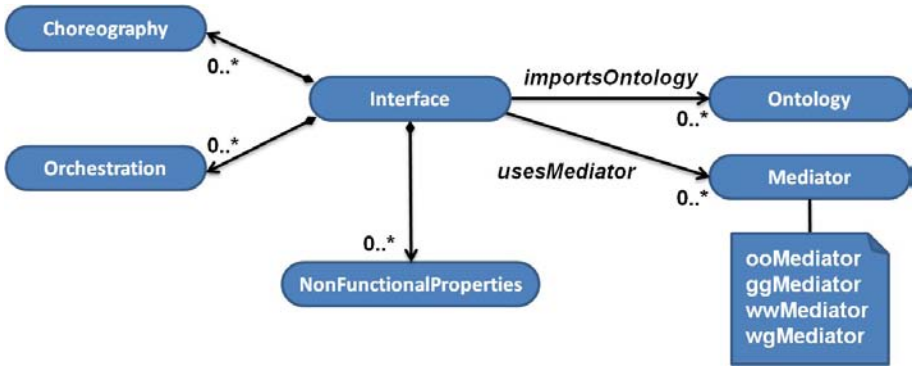


Figura 3.3: *Classe Interface da ontologia WSMO, e seus relacionamentos.*

A mediação presente na WSMO, é dividida em quatro tipos [31]:

- ***ooMediator***, que relaciona duas ontologias, solucionando possíveis problemas de incompatibilidade entre elas;
- ***ggMediator***, que associa duas metas no contexto de refinamento da meta de origem na meta alvo;
- ***wwMediator***, que relaciona dois serviços que se complementam ou possuem funcionalidade semelhante;
- ***wgMediator***, que associa os serviços com suas metas. Isso significa que o serviço cumpre com a meta associada com ele.

3.4.3 Composição

Para a descrição da composição dos serviços, também são utilizados os conceitos de Coreografia e Orquestração. Na WSMO, eles são baseados nas metodologias de Máquinas de Estado Abstrato [17]. Este modelo cumpre com três princípios. Primeiramente, ele deve ser baseado em estados. Em segundo, cada estado é representado por uma álgebra. Por fim, o estado do modelo muda através de regras de transição previamente guardadas, que mudam os valores das funções e relações definidas pela assinatura da álgebra [18].

3.4.4 Invocação

WSMO provê a invocação de serviços Web através de *Grounding* [67]. As informações presentes nas ontologias da WSMO são mapeadas, bidirecionalmente, com os dados presentes nas respostas em XML de serviços Web. As descrições de funcionalidades e comportamentos de um serviço são mapeados no arquivo WSDL que o descreve sintaticamente [18]. Enquanto que a Coreografia apenas diz que o cliente pode ler os dados, cabe ao serviço enviá-los ao cliente em forma de mensagens. O *Grounding* ainda

provê detalhes sobre a rede, como por exemplo qual é o protocolo para transferência (SOAP ou HTML, por exemplo). Também é o *Grounding* que informa como os dados são encapsulados no XML e se há necessidade de serialização [95].

3.5 OWL-S

OWL-S é uma ontologia, baseada em OWL, com o objetivo específico de permitir descrição semântica de serviços Web, através de anotações. Esta ontologia também é chamada de linguagem pelos seus autores, visto que ela fornece um vocabulário padronizado para descrição de serviços, que pode ser utilizado em conjunto com outros aspectos da OWL [24]. A proposta apresentada pela OWL-S define uma ontologia principal, denominada *Service*, e outras três sub-ontologias: *Profile*, *Process* e *Grounding*. Estas três sub-ontologias descrevem, respectivamente, o que o serviço provê ao seu consumidor, como o serviço pode ser usado e como interagir com o serviço. A ontologia principal, *Service*, possui a função de ligação entre as três ontologias. Para isso, ela define classes abstratas que posteriormente são especializadas pelas sub-ontologias [81] [41].

Como pode ser visto na Figura 3.4, a ontologia *Service* define a classe *Service* e outras três classes *ServiceProfile*, *ServiceModel* e *ServiceGrounding*. Especializações concretas de *ServiceProfile* devem descrever as funcionalidades disponibilizadas pelo serviço em questão [81]. Especializações concretas de *ServiceModel* devem descrever como um cliente pode usar o serviço e como ele funciona. Especializações concretas de *ServiceGrounding* devem especificar os detalhes de como um agente pode acessar o serviço, descrevendo detalhes como os protocolos de comunicação utilizados, os formatos das mensagens, entre outros [18].

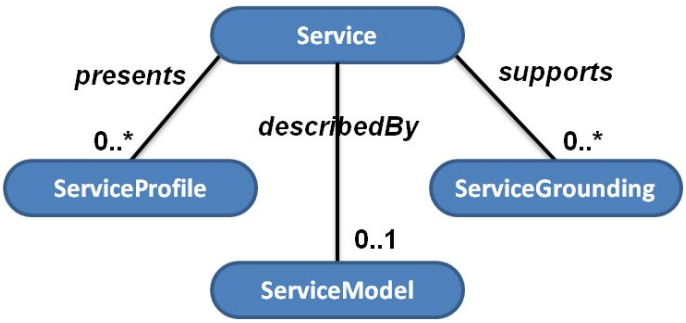


Figura 3.4: *Estrutura da ontologia Service presente na OWL-S.*

3.5.1 Descoberta de Serviços

Como dito anteriormente, a classe *ServiceProfile* é abstrata. Sua especialização concreta é realizada a partir da classe *Profile*, da ontologia de mesmo nome. Esta classe

representa propriedades não funcionais do provedor do serviço. Também representa elementos funcionais e algumas propriedades adicionais, como por exemplo características de QoS [18]. A Figura 3.5 apresenta os relacionamentos entre a classe *Profile* e elementos que auxiliam na descrição do perfil do serviço. Através do perfil definido na classe *Profile*, é possível viabilizar sistemas de descoberta eficientes, denominados *Matchmakers*. Um *Matchmaker* permite ao usuário encontrar serviços, através de um mecanismo para registro das capacidades dos serviços. Tais informações são registradas na forma de anúncios [95].

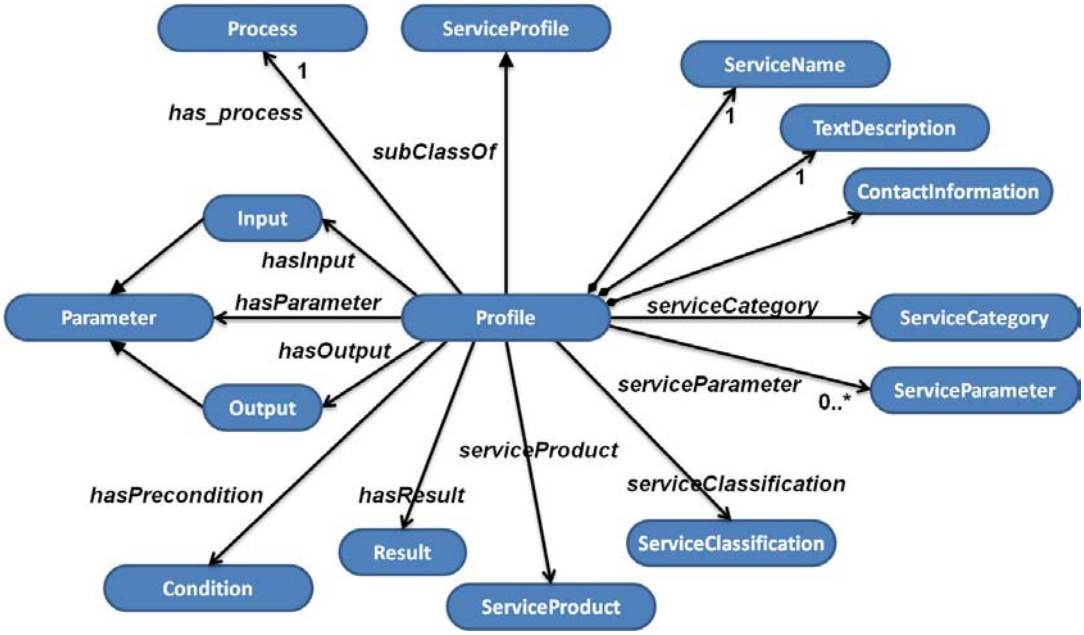


Figura 3.5: *Classe Profile da ontologia de mesmo nome, e seus relacionamentos, na OWL-S.*

Quando um usuário envia uma requisição ao *Matchmaker*, ele busca em seu banco de dados dinâmico de anúncios, por agentes que possam realizar a requisição enviada. Com isso, um *Matchmaker* realiza a comunicação entre o requisitante e o provedor do serviço. Para realizar a descoberta, o *Matchmaker* utiliza técnicas para análise de similaridades semânticas e sintáticas entre as capacidades dos serviços e as necessidades requisitadas. O motor para combinação de similaridades contêm cinco diferentes filtros. Em um *Matchmaker* é possível filtrar por comparação de *namespace*, comparação pela frequência de palavras, combinação por similaridades ontológicas, combinação por classificações ontológicas e combinação por restrições. O usuário pode configurar tais filtros para alcançar a relação entre desempenho e qualidade da descoberta desejados [95].

3.5.2 Interoperabilidade

A classe abstrata *ServiceModel*, citada anteriormente, possui especialização concreta na classe *Process*, da ontologia de mesmo nome. Ao descrever como um serviço é utilizado, a classe *Process* se relaciona com os IOPEs deste. Na Figura 3.6 são exibidos os relacionamentos de um processo com suas entradas (*Inputs*), saídas (*Outputs*), pré-condições (*Conditions*) e efeitos (*Results*). Estes relacionamentos podem ser vistos na Figura 3.6. É possível observar que entrada (*input*) e saída (*output*) herdam de parâmetro (*parameter*). Através da definição das IOPEs é possível garantir a interoperabilidade entre processos de serviços Web, descritos semanticamente em OWL-S. Entretanto, outros elementos são introduzidos na ontologia *Process*, para garantir a transformação de dados e a transição entre estados. Como pode ser visto na Figura 3.6, a classe *Process* se relaciona com os elementos:

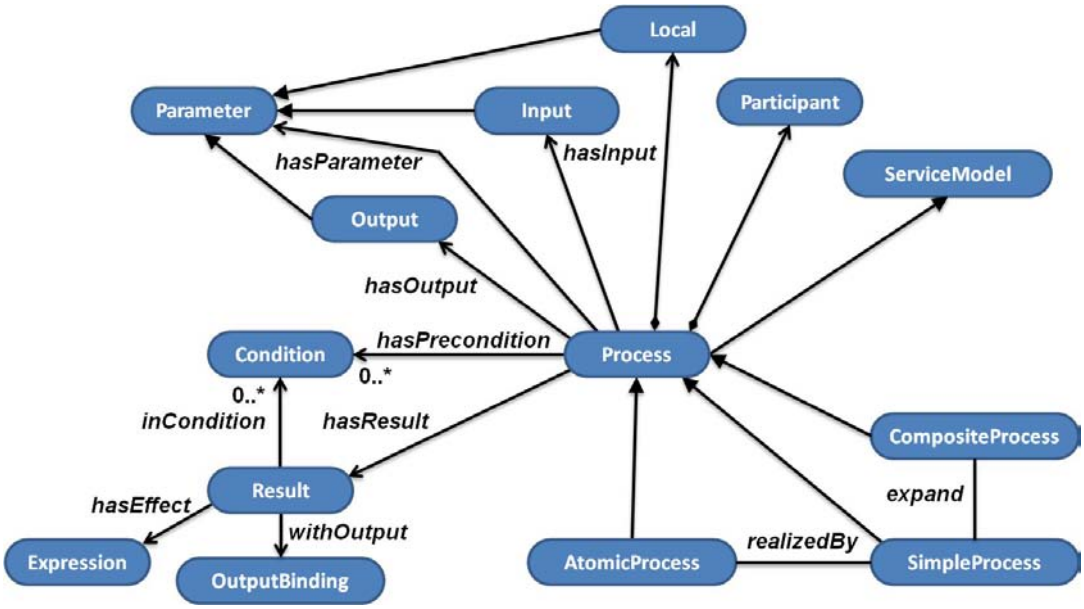


Figura 3.6: *Classe Process da ontologia de mesmo nome, e seus relacionamentos, na OWL-S.*

- **Participantes (*Participants*)**, que são entidades envolvidas no processo, que não sejam nem o cliente (requisitador) nem o servidor (provedor) do serviço;
- **Locais (*Locals*)**, que são parâmetros locais. Isto é, variáveis de escopo local do serviço, que não estão disponibilizadas externamente. Parametros locais só podem ser utilizados em processos atômicos (mais detalhes na seção 3.5.3);

Os parâmetros de entrada, saída e locais, são descritos por classes ontológicas da OWL, ou baseadas nela. Com isso, não é necessário fazer mediação entre parâmetros de diferentes serviços. Mesmo que os parâmetros sejam de classes diferentes e representem o mesmo conceito, o fato de serem classes da mesma ontologia (OWL) faz com que um

acabe herdando do outro [24]. As pré-condições e efeitos são representados por fórmulas lógicas. Para isso, OWL-S recomenda a linguagem SWRL[59] para a descrição das regras lógicas.

3.5.3 Composição

A ontologia OWL-S especifica três tipos de processos, como pode ser visto na figura 3.6. Cada tipo é uma subclasse da classe *Process*. Tais subclasses são [41]:

- ***AtomicProcess***. Essa classe descreve processos atômicos. Tais processos são unitários, ou seja, não possuem sub-processos;
- ***CompositeProcess***. Tal classe consiste na especialização da classe *Process* para definição de processos compostos. Estes processos são caracterizados por conterem múltiplos passos. Cada passo pode ser decomposto em um subprocesso. Estes subprocessos podem ser compostos ou não. OWL-S permite um número ilimitado de processos compostos aninhados;
- ***SimpleProcess***. Trata-se de uma especialização de *Process* para descrever processos simples. Esse tipo de processo não é invocável e é utilizado apenas como elemento de abstração. Processos simples podem ser utilizados com o intuito de fornecer versões simplificadas de processos compostos, para fins de planejamento. Também são úteis para viabilizar uma representação especial de utilização de um serviço atômico.

Além destas três classes que descrevem a composição do processo de um serviço, OWL-S ainda provê mecanismos para descrição de fluxo de dados. Através das classes *InputBinding* e *OutputBinding*, é possível prover regras de fluxo entre entradas e saídas de processos, mesmo que estas sejam de tipos diferentes.

3.5.4 Invocação

Na OWL-S, a classe abstrata *ServiceGrounding* possui como especialização concreta a classe *Grounding*. Essa classe especifica detalhes de como acessar e invocar o serviço. Esses detalhes incluem quais protocolos são utilizados, os formatos das mensagens, formas de serialização, transporte e endereçamento. Os desenvolvedores da OWL-S, ao propô-la, selecionaram WSDL como linguagem para descrição sintática do serviço. Entretanto, são enfáticos em relação à abertura para outras linguagens [24]. Para invocação de serviços Web baseados em SOAP e WSDL, é utilizada uma especialização da classe *Grouding*, denominada *WsdlGrouding*. Deste modo, são criadas relações entre os elementos do WSDL e as ontologias da OWL-S.

Para relacionar a descrição semântica de um serviço Web, com sua descrição sintática já existente em WSDL, são realizados três mapeamentos. As operações descritas em WSDL são mapeadas para processos atômicos em OWL-S. As mensagens de entrada e saída de uma operação em WSDL, são mapeadas para as entradas e saídas de um processo em OWL-S. Por fim, os tipos abstratos descritos em WSDL, são mapeados para as classes ontológicas que descrevem os tipos das entradas e saídas do processo. Com isso, é gerado um novo arquivo com uma instância da classe *WsdlGrounding* que inclui o arquivo WSDL e faz os mapeamentos necessários.

3.5.5 Ontologia *RESTfulGrouding*

Como dito anteriormente, a ontologia *Grounding* presente na OWL-S permite que novas formas de acesso e invocação de serviços sejam construídas. A partir da especialização de tal classe, é possível construir fundamentações para implementações específicas. Com base nisso, e observando a deficiência que existia para descrição semântica de serviços Web RESTful, um pesquisador da USP (Otávio Ferreira) desenvolveu a *RESTfulGrounding* [41]. Trata-se de uma ontologia que especifica especializações das classes de *Grounding* da OWL-S para fundamentação com descrição sintática em WADL. Deste modo, serviços RESTful podem ser descritos semanticamente, utilizando OWL-S.

3.6 Descrição Semântica de Serviços RESTful

A proposta apresentada neste trabalho visa a implementação prática da ontologia *RESTfulGrounding* (mais detalhes no capítulo 5), garantindo a descrição semântica de serviços Web, através da abordagem OWL-S (seção 3.5). Para tal, faz-se necessária a criação de arquivos WADL, que descrevam sintaticamente os serviços Web RESTful. Devido à simplicidade e agilidade, objetivadas pela arquitetura REST, é comum que serviços Web RESTful não tenham nenhuma descrição sintática. Os provedores destes serviços costumam descrevê-los em HTML ou XHTML. Desta forma, a descrição de tais serviços, seus métodos e componentes, são passíveis de serem interpretadas apenas por seres humanos, não possuindo descrições (sintática ou semântica) interpretáveis por máquinas [96].

Para descrever serviços Web semanticamente, antes é necessário que uma descrição sintática, que possa ser interpretada por máquinas, exista. Existem duas abordagens para suprir tal necessidade. A primeira é a criação de um arquivo que descreva sintaticamente o serviço. Tal abordagem pode ser implementada com arquivos WADL, por exemplo. A segunda, mais simplificada, visa implementar alguns atributos nas tags HTML, anotando-as semanticamente. Desta forma, os arquivos HTML que descrevem para hu-

manos os serviços Web RESTful, também descreveriam-nos para máquinas. Esta última abordagem é a utilizada pelas tecnologias apresentadas nesta seção.

Esta seção destina-se a apresentar três tecnologias relacionadas com a descrição semântica de serviços Web RESTful, através de microformatos (apresentados na seção 2.2.3). São elas: hRESTS[9][66], SA-REST[10][46] e MicroWSMO[11][68]. Em seguida, será realizada uma discussão, com o objetivo de expor as vantagens de cada abordagem a fim de implementá-las na proposta desta pesquisa.

3.6.1 hRESTS

HTML for RESTful Services (hRESTS) [66] é um microformato com o propósito de prover uma representação, que possa ser interpretada por máquinas, da descrição de serviços e APIs Web. Seguindo os conceitos e princípios dos microformatos, hRESTS utiliza alguns atributos das *tags* HTML/XHTML como `class` e `rel`. Nestas *tags* são colocados metadados acerca dos serviços, descritos originalmente apenas em HTML/XHTML. Para serviços RESTful, hRESTS possui o mesmo objetivo que WSDL para serviços baseados em SOAP. Entretanto, a abordagem do hRESTS inclui a descrição para máquinas e humanos em um mesmo arquivo [47], seguindo o princípio *Don't Repeat Yourself* [61].

A Figura 3.7 exibe triplas em RDF e RDFS que formam as classes e propriedades disponíveis em hRESTS. Na sintaxe apresentada na Figura, podem ser definidas várias triplas com o mesmo sujeito, sem que ele seja repetido. Para isso, são utilizados sinais de ponto-e-vírgula, identificando que o sujeito de uma tripla será sujeito das próximas também. Nas linhas de 6 à 8, é possível ver a definição de 3 triplas, sendo que as linhas 7 e 8 possuem apenas predicado e objeto. Ao final das definições de triplas para um determinado sujeito, é colocado um sinal de ponto, como na linha 8. As triplas do hRESTS são chamadas de *Service Model*.

Cada classe, definida neste modelo, é implementada a partir do atributo `class` das *tags* HTML/XHTML. As propriedades, presentes no modelo, especificam quais classes podem estar contidas em outras classes. A Figura 3.8 exibe um exemplo de código HTML que descreve um serviço RESTful para seres humanos. Na Figura 3.9 é apresentado o mesmo exemplo, porém com a adição de *tags* HTML com o atributo `class`. Tais *tags* com o atributo `class` é a implementação do hRESTS. É importante observar que hRESTS, por si só, não descreve semanticamente os serviços. Seu objetivo está voltado à descrição sintática [47].

[9] hRESTS é um acrônimo para *HTML for RESTful Services*. Trata-se de um microformato para descrição de APIs Web como simples modelos de serviços.

[10] SA-REST ou *Semantic Annotation of Web Resources* é um *POSHformat*[63] para adição de metadados acerca de descrições de APIs REST, em HTML ou XHTML. Entretanto, não é limitado a apenas isso.

[11] MicroWSMO é uma extensão de hRESTS para adição de anotações semântica em serviços RESTful.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix hr: <http://www.wsmo.org/ns/hrests#> .

hr:Service a rdfs:Class .
hr:hasOperation a rdf:Property ;
rdfs:domain hr:Service ;
rdfs:range hr:Operation .
hr:Operation a rdfs:Class .
hr:hasInputMessage a rdf:Property ;
rdfs:domain hr:Operation ;
rdfs:range hr:Message .
hr:hasOutputMessage a rdf:Property ;
rdfs:domain hr:Operation ;
rdfs:range hr:Message .
hr:Message a rdfs:Class .
hr:hasAddress a rdf:Property ;
rdfs:domain hr:Operation ;
rdfs:range hr:URITemplate . # a datatype for URI templates
hr:hasMethod a rdf:Property ;
rdfs:domain hr:Operation ;
rdfs:range hr:HTTPMethod .
hr:HTTPMethod a rdfs:Class .
hr:GET a hr:HTTPMethod .
hr:POST a hr:HTTPMethod .
hr:PUT a hr:HTTPMethod .
hr:DELETE a hr:HTTPMethod .
```

Figura 3.7: *Service model do hRESTS, implementado em RDFS/N3.*

```
<h1>ACME Hotels service API</h1>
<h2>Operation <code>getHotelDetails</code></h2>
<p> Invoked using the method GET at http://example.com/h/{id} <br/>
<strong>Parameters:</strong> <code>id</code> - the identifier of the particular hotel
<br/>
<strong>Output value:</strong> hotel details in an <code>ex:hotelInformation</code> document
</p>
```

Figura 3.8: *Exemplo de descrição de um serviço Web RESTful em HTML.*

```
1  <div class="service" id="svc">
2     <h1><span class="label">ACME Hotels</span> service API</h1>
3     <div class="operation" id="op1">
4        <h2>Operation <code class="label">getHotelDetails</code></h2>
5        <p> Invoked using the <span class="method">GET</span>
6        at <code class="address">http://example.com/h/{id}</code><br/>
7           <span class="input">
8              <strong>Parameters:</strong>
9              <code>id</code> - the identifier of the particular hotel
10          </span>
11          <br/>
12          <span class="output">
13             <strong>Output value:</strong> hotel details in an
14             <code>ex:hotelInformation</code> document
15          </span>
16       </p>
17    </div>
18 </div>
```

Figura 3.9: *Exemplo de Utilização de hRESTS em HTML.*

hRESTS possui seis principais classes. São elas [66]:

- **Classe *service***, como exibido na linha 1 da Figura 3.9, indica que a *tag* em questão descreverá um serviço Web.
- **Classe *operation***, indica que a *tag* em questão descreve uma operação de um serviço Web. Esta classe deve ser utilizada em uma *tag* que esteja contida em outra com a classe *service*. É possível verificar a utilização dela na linha 3 da Figura 3.9.
- **Classe *address***, indica o URL de uma operação de um serviço Web. Logo, deve ser colocada em uma *tag* que esteja contida em outra com a classe *operation*. Esta classe pode ser vista na linha 6 da Figura 3.9.
- **Classe *method***, como visto na linha 5 da Figura 3.9, indica o método HTTP utilizado pela operação.
- **Classes *input* e *output***, indica que a *tag* descreve uma mensagem de entrada ou de saída de uma operação. Há exemplos destas classes nas linhas 7 e 12 da Figura 3.9. hRESTS não suporta anotações semânticas para as entradas e saídas. Logo, todas as mensagens são do tipo *Message*.

Como todo microformato, para implementação do hRESTS, é necessário extrair os metadados inseridos nas *tags*. Para isso, é utilizado o GRDDL[12][52]. Trata-se de um padrão recomendado pelo W3C para extração de metadados em arquivos XML. Este padrão requer a utilização de arquivos XSLT (mais detalhes na Seção 2.2.1). Com esta

[12]GRDDL é um acrônimo para *Gleaning Resource Descriptions from Dialects of Languages*.

tecnologia é possível gerar, a partir do exemplo da Figura 3.9, as triplas exibidas na Figura 3.10.

No exemplo apresentado nas Figuras 3.9 e 3.10, os elementos representados por *tags* na Figura 3.9 ganharam o prefixo `jk:`. Na Figura 3.10, há dez triplas e apenas dois sujeitos diferentes. Os sujeitos são os elementos representados por *tags* com atributo `id` na Figura 3.9. O serviço `jk:svc` é sujeito das triplas nas linhas 5 à 8 e a operação `jk:op1` das triplas nas linhas 9 à 14. Apesar de terem sido extraídas triplas RDF da descrição disponibilizada pelo hRESTS, não há uma descrição semântica. As triplas apenas descrevem sintaticamente os serviços e seus elementos.

```
1  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
2  @prefix hr: <http://www.wsmo.org/ns/hrests#> .
3  @prefix jk: <http://members.sti2.at/~jacekk/hrests/src.xhtml#> .
4
5  jk:svc a hr:Service;
6  rdfs:isDefinedBy <http://members.sti2.at/~jacekk/hrests/src.xhtml>;
7  rdfs:label "ACME Hotels";
8  hr:hasOperation jk:op1 .
9  jk:op1 a hr:Operation;
10 rdfs:label "getHotelDetails";
11 hr:hasMethod hr:GET;
12 hr:hasAddress "http://example.com/h/{id}"^^hr:URITemplate;
13 hr:hasInputMessage [ a hr:Message ];
14 hr:hasOutputMessage [ a hr:Message ] .
```

Figura 3.10: *Exemplo de RDF em notação N3, extraido da implementação de hRESTS da Figura 3.9*

3.6.2 MicroWSMO

Como descrito na Seção anterior, o microformato hRESTS identifica as partes chave de um arquivo HTML que descreve um serviço RESTful para humanos. Para isso, são inseridos metadados acerca do serviço, efetivamente criando uma solução análoga ao WSDL para serviços RESTful. hRESTS forma a base para que, posteriormente, extensões possam incluir informações adicionais acerca dos serviços e suas partes. O MicroWSMO [68] consiste em uma extensão do hRESTS para adição de anotações semânticas à descrição sintática de serviços RESTful.

MicroWSMO adota as mesmas propriedades presentes no SAWSDL para adição de anotações semânticas. Ele define três atributos de arquivos XML, junto com propriedades RDF com os mesmos nomes. São elas [68]:

- ***modelReference***, que é usada em qualquer componente do modelo de serviço para fazer o mapeamento para o conceito semântico apropriado. Este apontamento é feito através de URIs e indicam ontologias.

- ***liftingSchemaMapping*** e ***loweringSchemaMapping***, que são utilizados para associar as mensagens com as transformações apropriadas. Estas transformações relacionam o formato de base, como o XML, com o formato de representações semânticas do conhecimento, como o RDF. Essas transformações são identificadas por URIs.

A Figura 3.11 exibe a implementação de MicroWSMO no exemplo mostrado na Figura 3.9. Como MicroWSMO é implementado através do hRESTS, a extração das anotações semânticas é realizada da mesma forma que no hRESTS. Através de GRDDL, os metadados são extraídos do arquivo HTML/XHTML, gerando um documento RDF. A Figura 3.12 exibe o RDF extraído para o exemplo da Figura 3.11. Pode-se observar que as triplas formadas agora possuem informações semânticas acerca dos serviços e seus elementos. É utilizada a ontologia SAWSDL (linhas 10, 18 e 19) e os sujeitos (`jk:svc` e `jk:op1`) das triplas são descritos como instâncias de classes ontológicas específicas. Para implementação de MicroWSMO é utilizada a ontologia WSMO-Lite, versão simplificada da WSMO, que será detalhada a seguir.

```
1  <div class="service" id="svc">
2      <h1><span class="label">ACME Hotels</span> service API</h1>
3      <p>This service is a
4          <a rel="model" href="http://example.com/ecommerce/hotelReservation">
5             hotel reservation</a> service.
6      </p>
7      <div class="operation" id="op1">
8          <h2>Operation <code class="label">getHotelDetails</code></h2>
9          <p> Invoked using the <span class="method">GET</span>
10             at <code class="address">http://example.com/h/{id}</code><br/>
11             <span class="input">
12                 <strong>Parameters:</strong>
13                 <a rel="model" href="http://example.com/data/onto.owl#Hotel">
14                    <code>id</code></a> - the identifier of the particular hotel
15                 (<a rel="lowering" href="http://example.com/data/hotel.xsparql">lowering</a>)
16             </span><br/>
17             <span class="output">
18                 <strong>Output value:</strong> hotel details in an
19                 <code>ex:hotelInformation</code> document
20             </span>
21         </p>
22     </div>
23 </div>
```

Figura 3.11: *Implementação de MicroWSMO em hRESTS, de acordo com a Figura 3.9.*

```
@prefix hr: <http://www.wsmo.org/ns/hrests#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .
@prefix wsl: <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix ex: <http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/example.xhtml#> .

ex:svc a wsl:Service ;
  rdfs:isDefinedBy <http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/example.xhtml> ;
  rdfs:label "ACME Hotels" ;
  sawsdl:modelReference <http://example.com/ecommerce/hotelReservation> ;
  wsl:hasOperation ex:op1 .
ex:op1 a wsl:Operation;
  rdfs:label "getHotelDetails" ;
  hr:hasMethod "GET" ;
  hr:hasAddress "http://example.com/h/{id}"^^hr:URITemplate ;
  wsl:hasInputMessage [
    a wsl:Message ;
    sawsdl:modelReference <http://example.com/data/onto.owl#Hotel> ;
    sawsdl:loweringSchemaMapping <http://example.com/data/hotel.xsparql>
  ];
  wsl:hasOutputMessage [ a wsl:Message ] .
```

Figura 3.12: *RDF extraido do exemplo de MicroWSMO apresentado na Figura 3.11.*

WSMO-Lite é uma ontologia que simplifica a WSMO. Um dos objetivos em sua construção, foi facilitar o uso da WSMO. Na WSMO-Lite, o serviço é decomposto nos seguintes componentes [39]:

- ***Information Model***, que define o modelo de dados para entradas, saídas e mensagens de falha.
- ***Functional Descriptions***, que descrevem as funcionalidades do serviço. Ou seja, o que ele oferece para o cliente, quando é invocado.
- ***Non-Functional Descriptions***, que definem detalhes secundários, específicos do provedor do serviço, da implementação do serviço ou do ambiente de execução.
- ***Behavioral Descriptions***, que definem comportamentos externos (coreografia pública) e internos (fluxos privados).
- ***Technical Descriptions***, que definem os detalhes das mensagens. Como a serialização, protocolos de comunicação e pontos de acesso ao serviço.

3.6.3 SA-REST

SA-REST é um *POSHformat*[63]. A definição de *POSHformat* é semelhante à dos microformatos. O conjunto dos microformatos está contido no conjunto dos *POSHformats*. Então, um *POSHformat* é como um microformato. Porém, não segue os princípios[84] e conceitos[85] dos microformatos. O padrão SA-REST passou por muitas

alterações desde sua criação. Sua primeira versão consistia em uma extensão do hRESTS, para descrição das facetas de serviços RESTful. Diferentemente dos serviços Web baseados em SOAP, onde o XML é o único formato, em serviços RESTful os dados podem ser retornados em vários formatos diferentes. Logo, as entradas e saídas de serviços RESTful poderiam ser anotadas com o padrão SA-REST [66].

Além dos formatos, SA-REST também previa, em sua primeira versão, anotações que informassem para qual linguagem de programação a API provia bibliotecas clientes. Estas bibliotecas visam facilitar o uso de APIs e serviços RESTful, como visto na Seção A.1. A Figura 3.13 exibe um exemplo desta versão do SA-REST. Entretanto, com o avanço das pesquisas, em 2007 o *W3C SWS Testbed incubator group*[96] lançou uma versão do SA-REST baseada em RDFa[32]. O padrão RDFa consiste em colocar as triplas RDF embutidas em arquivos XHTML, possibilitando semântica em arquivos feitos para leitura de humanos.

```
<div class="service" id="svc">
    <p>The output format of the operations of the
        <code class="label">ACME Hotels</code> service is
        <span class="data-format">JSON</span>.
        Client libraries are available in
        <span class="p-lang-binding">Java</span> and
        <span class="p-lang-binding">PHP</span>.
    </p>
</div>
```

Figura 3.13: *Implementação da primeira versão do SA-REST.*

A diferença entre RDFa e microformatos está no fato de RDFa utilizar atributos, das *tags*, e *namespaces* específicos. Com isso, ele não fica limitado aos atributos padrões da HTML. Entretanto, é compatível apenas com XHTML. Pode-se destacar ainda como vantagens do RDFa em relação aos microformatos, o mecanismo de extração mais eficiente (visto que não é preciso usar GRDDL) e a maior padronização (os microformatos não são padronizados pelo W3C). A Figura 3.14 exibe um exemplo de utilização do SA-REST, tendo como base RDFa. Esta versão do SA-REST já era mais completa e possuía as mesmas características da MicroWSMO, implementadas através de SAWSDL. Com isso, SA-REST não era mais considerada uma extensão de hRESTS.

```
<p about="http://craigslist.org/search/">
    The logical input of this service is an
    <span property="sarest:input">
        http://lsdis.cs.uga.edu/ont.owl#Location_Query
    </span>
    object.The logical output of this service is a list of
    <span property="sarest:output">
        http://lsdis.cs.uga.edu/ont.owl#Location
    </span>
    objects.This service should be invoked using an
    <span property="sarest:action">HTTP GET</span>
    <meta property="sarest:lifting" content="http://craigslist.org/api/lifting.xsl"/>
    <meta property="sarest:lowering" content="http://craigslist.org/api/lowering.xsl"/>
    <meta property="sarest:operation" content="http://lsdis.cs.uga.edu/ont.owl#Location_Search"/>
</p>
```

Figura 3.14: *Implementação da segunda proposta apresentada para o SA-REST, utilizando RDFa.*

Em 2010, o padrão SA-REST passou a ser especificado pelo W3C, como *Member Submission*. Esta especificação é baseada em uma nova versão. Nesta, SA-REST voltou a ser um *POSHformat*. Entretanto, de propósito mais geral. O padrão especificado no W3C, possui três propriedades básicas. Estas propriedades utilizam os atributos `class` e `title` das *tags* HTML para inserção de anotações semânticas acerca dos serviços. São elas [46]:

- **`domain-rel`**: Trata-se de uma propriedade de bloco. Ou seja, a *tag* com essa marcação pode conter outras *tags* com marcações SA-REST. Esta propriedade permite informar uma descrição de domínio, acerca do recurso. Caso o recurso pertença a mais de um domínio, podem ser inseridos vários domínios em uma mesma *tag*.
- **`sem-rel`**: Uma propriedade de elemento. Ou seja, a *tag* que possui essa marcação não pode conter outras *tags* com marcações SA-REST. Esta propriedade provê a adição de anotações externas em documentos de terceiros. Ela só pode ser usada em *tags* de ancora (links `<a>`).
- **`sem-class`**: Uma propriedade de elemento. Pode ser usada para marcar uma única entidade em um recurso. Esta entidade pode ser um fragmento de texto ou um vídeo, por exemplo.

A Figura 3.15 exemplifica a utilização dessas propriedades. Para a descrição semântica do serviço, é utilizado um modelo de serviço parecido com o fornecido no hRESTS. Este modelo pode ser visto na Figura 3.16. Com isso, SA-REST independe do hRESTS e pode ser utilizado em conjunto com outras linguagens. Essa nova abordagem é baseada no SAWSDL e visa uma implementação semelhante, em serviços RESTful [46].

```
<div class="section domain-rel" lang="en" title="sarest:Service" >
   <span class="domain-rel" title="sarest:Operation" >
   <div class="titlepage">
      <div>
         <div>
            <h3 id="JSON-RPCEndpoint">JSON-RPC Endpoint</h3>
         </div>
      </div>
   </div>
   <p>
   The JSON-RPC endpoint implements the
   <a href="http://foo.xsd" class="sem-rel"
      title="http://taxonomy.org/computerscience#json" >JSON-RPC spec</a> on
   top of the Web service. Requests are serialized JavaScript following a specific data format.
   Each serialized JavaScript object contains the following properties:
   </p>

   <div class="itemizedlist domain-rel" title="sarest:InputMessage">
      <ul>
         <li class="bullist sem-class" title="sarest:Parameter">
         <code class="code">method</code>:
             name of the API method being called.
         </li>
         ...
      </ul>
   </div>
   </span>
</div>
```

Figura 3.15: *Implementação da última proposta apresentada para o SA-REST, utilizando POSHformat.*

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sarest: <http://www.knoesis.org/research/srl/standards/sa-rest/#> .

sarest:hasParameter rdf:type owl:ObjectProperty ;
              rdfs:range sarest:Parameter ;
              rdfs:domain sarest:Message .

sarest:hasAddress rdf:type owl:ObjectProperty ;
                  rdfs:domain sarest:Operation ;
                  rdfs:range sarest:URITemplate .

sarest:hasInputMessage rdf:type owl:ObjectProperty ;
                       rdfs:range sarest:InputMessage ;
                       rdfs:domain sarest:Operation .

sarest:hasMethod rdf:type owl:ObjectProperty ;
                 rdfs:range sarest:HTTPMethod ;
                 rdfs:domain sarest:Operation .

sarest:hasOperation rdf:type owl:ObjectProperty ;
                    rdfs:range sarest:Operation ;
                    rdfs:domain sarest:Service .

sarest:hasOutputMessage rdf:type owl:ObjectProperty ;
                        rdfs:range sarest:OutputMessage ;
                        rdfs:domain sarest:Operation .

sarest:InputMessage rdf:type owl:Class ;
              rdfs:subClassOf sarest:Message .

sarest:OutputMessage rdf:type owl:Class ;
               rdfs:subClassOf sarest:Message .

sarest:Parameter rdf:type owl:Class .
sarest:HTTPMethod rdf:type owl:Class .
sarest:Message rdf:type owl:Class .
sarest:Operation rdf:type owl:Class .
sarest:Service rdf:type owl:Class .
sarest:URITemplate rdf:type owl:Class .

sarest:DELETE rdf:type owl:Thing ,
                       sarest:HTTPMethod .
sarest:GET rdf:type owl:Thing ,
                    sarest:HTTPMethod .
sarest:POST rdf:type owl:Thing ,
                     sarest:HTTPMethod .
sarest:PUT rdf:type owl:Thing ,
                    sarest:HTTPMethod .
```

Figura 3.16: *Service model para SA-REST.*

3.6.4 Discussão

Os padrões apresentados nesta seção são baseados em microformatos. Esta abordagem traz a vantagem de possuir apenas um arquivo que descreve o serviço para humanos e máquinas. Entretanto, é menos madura e padronizada que WADL [47]. Todas as abordagens apresentadas utilizam SAWSDL, ou se baseiam nele, para a definição dos tipos de mensagem de entrada e saída, dos serviços. Logo, é interessante que a abordagem introduzida neste trabalho o aceitasse também. Isto é possível, visto que SAWSDL não se limita apenas a arquivos WSDL, podendo ser inserido também em arquivos WADL.

Os microformatos para descrição de serviços Web, apresentados neste capítulo, são extraídos, formando triplas RDF. É possível que tais microformatos possam formar arquivos WADL que descrevam os serviços Web sintaticamente. Isso garantiria a interoperabilidade entre o hRESTS ou SA-REST e a ontologia *RESTfulGrounding* da OWL-S. A implementação destes extratores poderá ser feita em trabalhos futuros.

3.7 Discussão em Implementação de Serviços Web Semânticos

As três abordagens para a realização de serviços semânticos apresentadas nesse capítulo, possuem características distintas. Ao compará-las, o objetivo desta pesquisa não é assegurar qual é a melhor, mas apresentar as vantagens e desvantagens de cada abordagem, exibindo em quais contextos cada uma é mais especializada. É possível observar também que, características de diferentes abordagem podem ser utilizadas em conjunto. Todas as três abordagem surgiram na mesma época, no ano de 2004. Entretanto, as mais pesquisadas recentemente são OWL-S e WSMO.

Enquanto que METEOR-S destina-se a descrever semanticamente todo o ciclo de vida de um serviço Web, OWL-S e WSMO são mais especializadas nas funcionalidades do serviço Web. Com isso, OWL-S e WSMO descrevem mais detalhadamente os processos de invocação, interoperabilidade e composição de serviços. Na WSMO há uma grande preocupação com a interoperabilidade, através dos mediadores. Isso se dá porque a WSMO oferece um mecanismo mais flexível, menos ditatorial que OWL-S, para importação de ontologias. Como OWL-S restringe a importação apenas de ontologias criadas em OWL, não é necessária tamanha preocupação com a interoperabilidade.

Apesar de METEOR-S atrair menor interesse, a partir dela surgiu um padrão importante para os serviços semânticos, o SAWSDL. Iniciado em 2006, como *Working Draft* pelo W3C, o SAWSDL foi inteiramente baseado no WSDL-S do METEOR-S. Trata-se de um padrão, recomendado pelo W3C, para definição de anotações semânticas em arquivos WSDL. O foco deste padrão é modesto. Ele não visa caracterizar o serviço

viabilizando a descoberta, composição e outras características relacionadas à descrição semântica de serviços. O principal objetivo do SAWSDL é descrever semanticamente os tipos das variáveis de entrada e saída, bem como as operações [81].

SAWSDL pode ser utilizada em conjunto com qualquer mecanismo para descrição semântica de serviços Web. Este padrão pode ser utilizado em conjunto com o OWL-S, em busca de uma maior flexibilidade em relação à importação de ontologias. Apesar de ter sido desenvolvido para trabalhar em conjunto com WSDL, SAWSDL pode ser utilizado em outros tipos de arquivos. A presente pesquisa dará enfoque na utilização de SAWSDL com OWL-S e a ontologia *RESTfulGrounding*. Como é possível observar, a única abordagem que possibilita a implementação de serviços Web RESTful é a OWL-S. Entretanto, a ontologia *RESTfulGrounding* é muito recente (sua primeira versão foi lançada em 2009), não tendo uma implementação desta ainda.

CAPÍTULO 4

Trabalhos Relacionados

Os trabalhos apresentados nessa seção destinam-se a solucionar o mesmo problema proposto por essa pesquisa. Para tal, os autores de cada trabalho identificaram características necessárias para uma solução em descrição semântica de serviços RESTful. As abordagens apresentadas divergem entre si e em relação à proposta nessa pesquisa. Isso ocorre visto que o tema ainda é muito novo e as abordagens existentes ainda muito imaturas.

4.0.1 EXPRESS

EXPRESS é um acrônimo para *EXPressing REstful Semantic Services*. Trata-se de uma abordagem simplificada para Serviços Web Semânticos, que necessita apenas da definição ontológica do domínio [5]. EXPRESS elimina a necessidade de utilização de uma descrição semântica separada. Isso se dá porque ele provê os recursos através de uma interface uniforme (um dos princípios do REST). Logo, não é necessária a utilização de tecnologias simplificadas (como o SAWSDL), tampouco as mais complexas (como OWL-S e WSMO).

Os recursos disponibilizados pelo EXPRESS são entidades descritas semanticamente em ontologias definidas em OWL. Combinando a expressividade e a semântica das ontologias e provendo uma interface uniforme para elas, é proposta a criação de Serviços RESTful Semânticos. Um provedor de serviços, utilizando EXPRESS, provê um arquivo OWL que descreve os recursos presentes no Serviço Web. Há também um motor de implantação que cria URIs automaticamente para as classes, instâncias e propriedades.

Os autores deste trabalho, levantam algumas características negativas das tecnologias propostas para Serviços Web Semânticos (como OWL-S e WSMO). Entre elas está o fato de que, mesmo com a utilização de tecnologias focadas em Serviços RESTful (como hRESTS, SA-REST e MicroWSMO), a abordagem é baseada em RPC. Os recursos são tratados como serviços, limitando as vantagens do REST. Também é mencionado que tais abordagens possuem alta complexidade pela forte dependência de raciocínio lógico

para a automação de descoberta e composição. Os autores apontam que esse obstáculo será um grande desafio para disponibilizar os recursos em escala Web.

Apesar das vantagens apontadas pelos autores, o EXPRESS possui alguns pontos negativos. A abordagem proposta não contempla nenhum tipo de interoperatividade entre serviços semânticos RESTful e SOAP. O fato de não haver vínculo com nenhuma tecnologia contemporânea de descrição semântica de serviços Web, isola-a. Outro ponto fraco está no motor de implantação, que gera automaticamente os URIs de recursos, a partir da ontologia. Dessa forma, não é possível implantar a descrição semântica em ambientes que já estão em execução. Poderia ser inviável a mudança de todos os URIs de recursos do serviço.

4.0.2 ReLL

ReLL[2], ou *Resource Linking Language*, é uma linguagem para descrição de serviços RESTful. Por ser focada em RESTful, os autores preocuparam-se em atender todos os princípios do REST. A linguagem é baseada em XML e possui um esquema (XSD) que a padroniza. Possui elementos semelhantes àqueles presentes na WADL. Entretanto, tem uma abordagem mais voltada à semântica. A abordagem proposta por ReLL possui foco no relacionamento (*links*) entre os recursos. Tais relacionamentos podem ser descritos semanticamente, sem a necessidade de tecnologias complexas de descrição semântica de serviços.

A partir da ReLL é possível, facilmente, gerar triplas RDF. Isso é feito com GRDDL. A partir das triplas RDF, as consultas e relacionamentos semânticos entre recursos podem ser realizados. A ReLL pode interagir com microformatos, ampliando as possibilidades de descrição semântica dos serviços. Para implementação da ReLL, foi criado um indexador automático (*crawler*), denominado **RESTler**. Este indexador recebe como entrada um documento XML com a descrição em ReLL e um conjunto de URIs (sementes). Então, produz um grafo dos recursos rastreados e seus relacionamentos.

O grafo retornado pelo RESTler é enviado a um tradutor que converte-o em triplas RDF e armazena-as em um repositório. A partir dai, consultas compostas para descoberta de recursos podem ser realizadas. Ao contrário do EXPRESS, a ReLL disponibiliza a descoberta em serviços já existentes. Atualmente ela aceita apenas o método HTTP GET. Isso restringe significantemente o uso de tal tecnologia. Como é possível observar, ReLL também não implementa interoperabilidade com outras tecnologias de descrição semântica de serviços.

4.0.3 SEREDASj

SEREDASj é uma metodologia de descrição de serviços RESTful, baseada em JSON. Trata-se do trabalho relacionado mais recente. SEREDAS significa *SEmantic REstful DAta Services* [70]. O "j"ao final do nome é uma referência ao JSON. A abordagem proposta pelos autores visa atender todos os princípios REST. Os aspectos considerados mais importantes são como um recurso pode ser acessado, como ele é representado e como ele está interligado. Para tal, SEREDASj especifica um esquema JSON para descrição semântica de recursos representados em JSON.

A estrutura sintática de representações em JSON, proposta pelo SEREDASj, permite referenciar elementos JSON à conceitos em ontologias. Para isso, a descrição SEREDASj consiste de metadados e uma descrição da estrutura da representação em JSON da instância a ser descrita. Nos metadados há informações sobre os *hyperlinks* relacionados com os dados da instância e prefixos para abreviar URIs longos. A descrição do *link* contém toda a informação necessária para que o cliente requisite e manipule os dados da instância.

O projeto do SEREDASj apresenta três casos de uso aos quais ele é proposto como solução. O primeiro, diz respeito à documentação de serviços RESTful. Visto que a grande parte de tais serviços estão descritos apenas em HTML para seres humanos, SEREDASj visa prover um mecanismo com o qual seja possível gerar a descrição para seres humanos automaticamente a partir da descrição processável por máquinas. Isso é realizado a partir dos comentários em elementos de ontologias que podem ser incluídas na descrição SEREDASj.

Outro caso de uso é a habilidade de criar clientes dinâmicos e flexíveis. Utilizando anotações semânticas, o cliente consegue interpretar o que é um determinado *hyperlink* e o que ele representa para o recurso. Integração de dados é um outro caso de uso. Com a descrição semântica proposta pelo SEREDASj é possível fazer integração de dados e identificar tipos sem que sejam necessárias técnicas complexas de mediação. Além disso, é possível gerar triplas RDF a partir da descrição em JSON de uma representação.

SEREDASj propõe-se a fazer algo audacioso: descrever semanticamente serviços RESTful, utilizando apenas JSON. Entretanto, a abordagem proposta ainda não possui nenhuma implementação prática. Outra desvantagem é o fato de que apenas recursos com representações em JSON podem ser descritos. Entretanto, tal abordagem pode auxiliar na implantação de semântica em serviços RESTful, por ser simples e possuir muitas características atraentes para desenvolvedores de aplicações convencionais.

4.0.4 Discussão

As três abordagens apresentadas aqui possuem focos semelhantes, porém cada uma possui um objetivo diferente. EXPRESS destina-se a fornecer Serviços Web, com URIs padronizadas, para manipulação de recursos previamente descritos semanticamente em OWL. Com isso, atende bem as necessidades de composição, com instâncias de classes ontológicas criadas a partir de OWL, e a invocação, que pode ser realizada através da requisição de URIs geradas automaticamente, a partir de cada classes ontológica. Entretanto, não trabalha com descoberta e interoperabilidade dos serviços. Também não é possível implementá-la com Web Services já existentes, visto que as URIs teriam que ser alteradas.

A abordagem proposta por ReLL é focada na descoberta e composição dos serviços. A interoperabilidade é limitada a serviços que estejam descritos em ReLL. Isto é, para interoperar com outro serviço, é necessário descreve-lo em ReLL previamente. A invocação é feita externamente, não sendo foco desta abordagem. O fato de suportar apenas o método GET do protocolo HTTP restringe as possíveis utilizações desta abordagem. A metodologia SEREDASj, por outro lado, é focada em descrever as representações em JSON de recursos de Serviços Web na arquitetura REST. Esta abordagem não trabalha com as funcionalidades de descoberta, composição, interoperabilidade e invocação especificamente. Seu foco está em vincular as representações às classes ontológicas de domínio.

Nenhum dos trabalhos relacionados disponibiliza um mecanismo em que seja possível a interoperabilidade entre Serviços Web REST e SOAP. Também não há uma abordagem que apresente todas as funcionalidades necessárias para a implementação de Serviços Web Semânticos: descoberta, composição, interoperabilidade e invocação automatizadas de serviços. Ainda há o fato de que as abordagens relacionadas não utilizam os padrões para descrição semântica de serviços Web, já discutidos. A abordagem apresentada neste trabalho visa oferecer as quatro funcionalidades de serviços Web semânticos utilizando um dos padrões já estabelecidos para descrição semântica de serviços Web.

Como pode ser visto neste capítulo, nenhuma das soluções para descrição semântica de serviços Web RESTful, atuais, implementam OWL-S. Nota-se, então, que há uma carência de implementações nesta área. Tendo isto em vista, a presente pesquisa objetiva o preenchimento desta lacuna. Para tal, foi desenvolvida uma implementação, utilizando a ontologia *RESTfulGrounding*, para garantir a descoberta, composição, interoperabilidade e invocação de serviços Web RESTful semânticos. Os próximos capítulos focam o desenvolvimento de tal implementação.

CAPÍTULO 5

Implementação de Serviços Semânticos

Como visto no Capítulo 3, existem várias formas de implementar serviços web semânticos. Este trabalho foca na implementação com OWL-S, pela sua maturidade e pela carência de implementação para serviços Web RESTful em tal abordagem. Esse Capítulo tem como objetivo a explicação de como é feita a descrição sintática e semântica de serviços Web RESTful. Para isso, inicialmente é descrito e exemplificado o padrão WADL, bem como a Ontologia *RESTfulGrounding*. Em seguida, é exibido o funcionamento da OWL-S API [87]. Trata-se de uma API, de código fonte aberto, desenvolvida em Java para implementação da OWL-S.

5.1 Descrição Sintática de Serviços RESTful com WADL

Como visto na Seção 3.2.3, o padrão WADL define arquivos, baseados em XML, para a descrição sintática de aplicações Web. A Figura 5.1 exibe um exemplo de documento WADL. O elemento (ou *tag*) principal de um arquivo WADL é o `application` [51]. Este elemento representa a aplicação Web a ser descrita. Em um arquivo WADL, todos os elementos podem conter o elemento `doc`. Este elemento é destinado à documentação dos demais elementos. O `application` pode possuir o elemento opcional `grammars`. Este elemento atua como um contêiner para definições de formatos de dados trocados durante a execução do protocolo descrito pelo documento WADL. Tais definições podem ser incluídas por referência usando o elemento `include`.

O elemento `application` pode conter zero ou mais elementos `resources`. Este tem como objetivo agrupar vários recursos com um mesmo URI base. Tais recursos são representados através do elemento `resource`. Cada `resource` pode possuir vários elementos `param`. Estes elementos descrevem os parâmetros possíveis para a requisição de um recurso. Cada elemento `resource` também poderá conter zero ou mais elementos `method`. Tal elemento objetiva a descrição dos possíveis métodos de requisição HTTP para o recurso.

```
<?xml version="1.0"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
  xmlns:tns="urn:yahoo:yn" xmlns:yn="urn:yahoo:yn" xmlns:ya="urn:yahoo:api"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://wadl.dev.java.net/2009/02">
   <grammars>
     <include href="NewsSearchResponse.xsd"/>
     <include href="Error.xsd"/>
   </grammars>
   <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
     <resource path="newsSearch">
       <method name="GET" id="search">
         <request>
           <param name="appid" type="xsd:string" style="query" required="true"/>
           <param name="query" type="xsd:string" style="query" required="true"/>
           <param name="type" style="query" default="all">
             <option value="all"/>
             <option value="any"/>
             <option value="phrase"/>
           </param>
           <param name="results" style="query" type="xsd:int" default="10"/>
           <param name="start" style="query" type="xsd:int" default="1"/>
           <param name="sort" style="query" default="rank">
             <option value="rank"/>
             <option value="date"/>
           </param>
           <param name="language" style="query" type="xsd:string"/>
         </request>
         <response status="200">
           <representation mediaType="application/xml" element="yn:ResultSet"/>
         </response>
         <response status="400">
           <representation mediaType="application/xml" element="ya:Error"/>
         </response>
       </method>
     </resource>
   </resources>
</application>
```

Figura 5.1: *Exemplo de arquivo WADL para o serviço de busca de notícias do Yahoo! [51].*

Cada elemento `method` poderá conter um elemento `request` e zero ou mais elementos `response`. O elemento `request` descreve a requisição para o recurso. Nele poderão ser definidos parâmetros específicos para o método de requisição HTTP em questão, através de elementos `param` como filhos. O elemento `response` representa uma resposta possível para tal recurso. Cada `response` poderá conter zero ou mais elementos `representation` e `param`. As possíveis representações do recurso requisitado a partir do método em questão, são representadas por elementos `representation`. Os elementos `param` filhos de `response` tem como objetivo descrever parâmetros do cabeçalho da resposta HTTP.

No documento WADL, é possível criar links para elementos que são utilizados mais de uma vez. Para tal, é utilizado o atributo `href`. O elemento `application` poderá possuir como filho direto qualquer outro elemento que será referenciado no contexto apropriado. Os elementos `param` poderão conter opcionalmente elementos `option` que informam os possíveis valores para aquele parâmetro. A Figura 5.2 mostra um diagrama com todos os elementos do WADL e os relacionamentos entre eles.

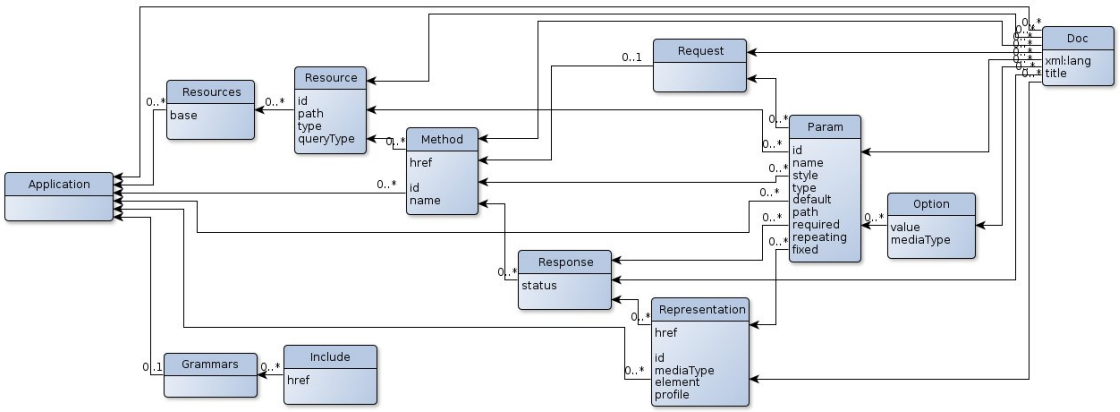


Figura 5.2: *Diagrama com principais elementos do padrão WADL e seus relacionamentos.*

É possível observar que, através dos elementos presentes no padrão WADL, pode-se fazer a descrição sintática de todas as características e princípios dos serviços Web RESTful, descritos anteriormente. A utilização de documentos WADL para a representação de serviços RESTful garante algumas vantagens como a possibilidade de geração de código fonte automaticamente. Entretanto, apenas com a descrição sintática não é possível fazer a descoberta e a invocação de serviços ou requisição de recursos dinamicamente. Como dito anteriormente, para isso, é necessário utilizar a descrição semântica, através de ontologias. A partir desta, não é necessário criar código fonte automaticamente, visto que o propósito é requisitar novos serviços ou recursos em tempo de execução[1].

5.2 Descrição Semântica com *RESTfulGrounding*

Como visto na Seção 3.5, a ontologia *RESTfulGrounding* é uma extensão da OWL-S para descrição semântica de serviços Web RESTful. Para tal, é necessário especializar apenas as classes ontológicas da camada de fundamentação (ou *Grounding*) da OWL-S. O mapeamento *Grounding* OWL-S/WADL, presente na ontologia

[1] Entende-se como tempo de execução o período em que o *software* permanece em execução. Tal termo é um contraponto para o termo tempo de compilação, que refere-se ao período em que o código é compilado para gerar um programa executável.

RESTfulGrounding, especializa a camada abstrata composta pelas classes ontológicas `Grounding` e `AtomicProcessGrounding` da OWL-S, com as classes `WadlGrounding` e `WadlAtomicProccessGrounding`. A classe `WadlGrounding` herda a propriedade `hasAtomicProcessGrounding` e discrimina que ela só poderá receber instâncias da classe `WadlAtomicProcessGrounding`.

Cada processo atômico define uma única operação, sem sub-processos. Entretanto, na arquitetura RESTful não há o conceito de operação, visto que o foco está em recursos. Considera-se uma operação cada par recurso/método presente na descrição sintática. Deste modo, a ontologia `RESTfulGrounding` define a classe `WadlResourceMethodRef` para descrição semântica de um par recurso/método. Essa classe possui duas propriedades: `resource` e `method`, que possuem os URIs para a descrição do recurso e do método, respectivamente, no arquivo WADL. A classe `WadlAtomicProcessGrouding` possui a propriedade `wadlResourceMethod` que aceita uma instância da classe `WadlResourceMethodRef`.

Para representar um parâmetro descrito sintaticamente pelo elemento `param` do padrão WADL, a ontologia *RESTfulGrounding* define a classe `WadlMessageParamMap` que herda da classe `MessageMap` da OWL-S. Esta classe possui a propriedade `wadlMessageParam` que aceita o URI para a descrição do parâmetro no arquivo WADL. A ontologia também define que os parâmetros poderão ser de requisição (entrada) ou de resposta (saída). Parâmetros de requisição são descritos pela classe `WadlRequestParamMap` enquanto que os de resposta são descritos pela classe `WadlResponseParamMap`, ambas herdam da classe `WadlMessageParamMap`. Tais classes também herdam da classe `InputMessageMap` e da classe `OutputMessageMap`, respectivamente, presentes na ontologia `Grouding` da OWL-S. Observa-se que ontologias podem ter herança múltipla.

A herança múltipla das classes `WadlRequestParamMap` e `WadlResponseParamMap` é feita visto que em cada uma delas é necessária uma propriedade que especifique qual a instância da classe `Input` ou da classe `Output` (presentes na ontologia `Process` da OWL-S), respectivamente, está vinculada. Isso é feito com a propriedade `owlsParameter` presente na classe `InputMessageMap` e na classe `OutputMessageMap`. A diferença é que na classe `InputMessageMap` essa propriedade aceita instâncias da classe `Input` enquanto que na classe `OutputMessageMap` ela aceita instâncias da classe `Output`.

Para referenciar os parâmetros de requisição e resposta a classe `WadlAtomicProcessGrounding` possui as propriedades `wadlRequestParam` e `wadlResponseParam` que aceitam instâncias da classe `WadlRequestParamMap` e `WadlResponseParamMap`, respectivamente. A classe `WadlAtomicProcessGrounding` ainda possui as propriedades `wadlDocument` e `wadlVersion` que aceitam URIs para o documento WADL e para a versão do padrão WADL que foi utilizado, respectivamente. Um diagrama com as principais classes e propriedades da ontologia `RESTfulGrounding`

pode ser visto na Figura 5.3.

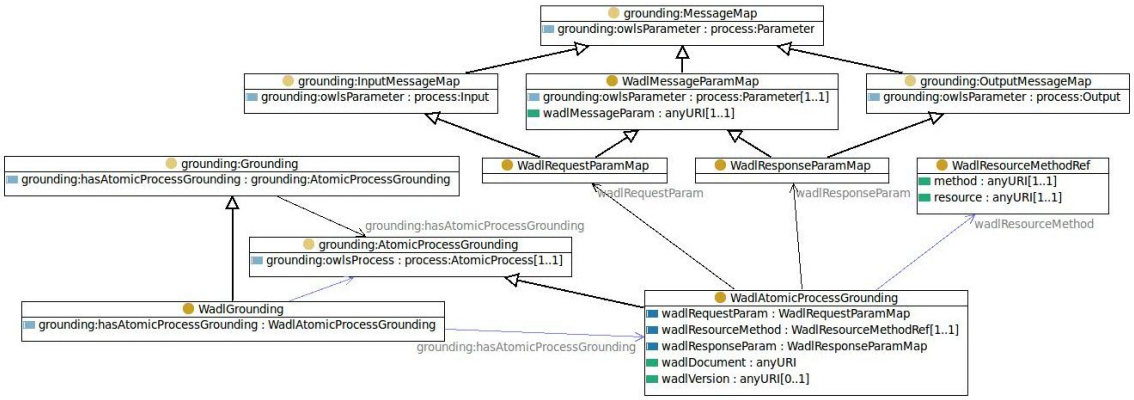


Figura 5.3: *Diagrama com principais classes e propriedades da ontologia RESTfulGrounding.*

A Figura 5.5 exibe um fragmento da ontologia que descreve semanticamente a fundamentação (*grounding*) do serviço descrito sintaticamente na Figura 5.1. O prefixo `&ypc` refere-se ao arquivo que descreve uma instância da classe ontológica `Process` da ontologia de mesmo nome, presente na OWL-S. O prefixo `&wadl` refere-se ao arquivo WADL apresentado na Figura 5.1 e o prefixo `grounding` refere-se à classe ontológica `Grounding` da ontologia de mesmo nome, presente na OWL-S. O diagrama apresentado na Figura 5.4 sintetiza os relacionamentos entre o arquivo de descrição sintática (WADL) e os arquivos de descrição semântica (OWL-S e *RESTfulGrounding*).

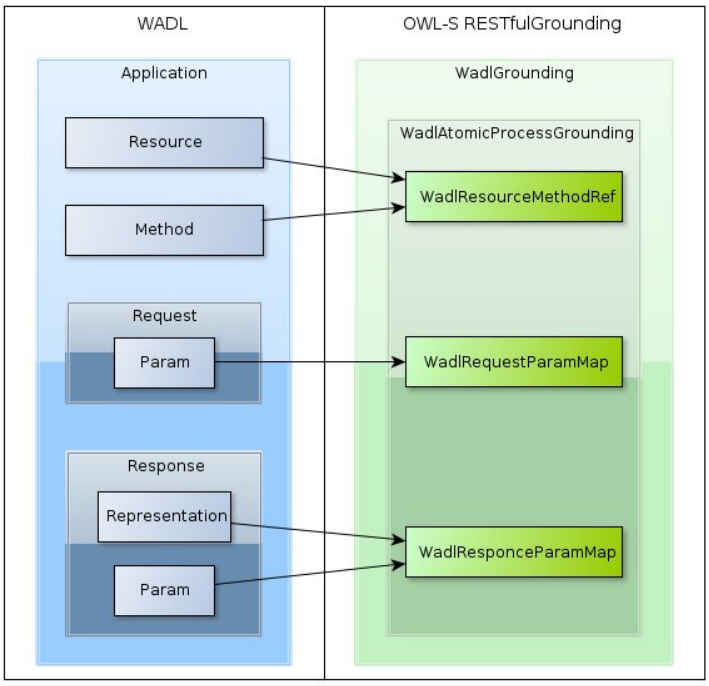


Figura 5.4: *Relação entre o padrão WADL e a ontologia OWL-S RESTfulGrounding.*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- (...) Cabeçalho da ontologia omitido -->

<owl:Ontology rdf:about="">
  <!-- (...) Descrição e importações da ontologia omitidas -->
</owl:Ontology>

<restful:WadlGrounding rdf:ID="YNS-Grounding">
  <grounding:hasAtomicProcessGrounding rdf:resource="#YNS-AtomicProcessGrounding"/>
</restful:WadlGrounding>

<restful:WadlAtomicProcessGrounding rdf:ID="YNS-AtomicProcessGrounding">
  <grounding:owlsProcess rdf:resource="&ypc;#YNS-AtomicProcess"/>
  <restful:wadlResourceMethod>
    <restful:WadlResourceMethodRef>
      <restful:resource rdf:datatype="&xsd;#anyURI">&wadl;#YNS-Resource</restful:resource>
      <restful:method rdf:datatype="&xsd;#anyURI">&wadl;#YNS-Method-GET</restful:method>
    </restful:WadlResourceMethodRef>
  </restful:wadlResourceMethod>
  <restful:wadlVersion rdf:datatype="&xsd;#anyURI">
    https://wadl.dev.java.net/wadl20090202.xsd
  </restful:wadlVersion>
  <restful:wadlDocument rdf:datatype="&xsd;#anyURI">
    https://sites.google.com/site/ocxmestrado/YNS.wadl
  </restful:wadlDocument>

  <!-- Parâmetros de Requisição -->
  <restful:wadlRequestParam>
    <restful:WadlRequestParamMap>
      <grounding:owlsParameter rdf:resource="&ypc;#YNS-AppID"/>
      <restful:wadlMessageParam rdf:datatype="&xsd;#anyURI">&wadl;#appid</restful:wadlMessageParam>
    </restful:WadlRequestParamMap>
  </restful:wadlRequestParam>
  <restful:wadlRequestParam>
    <restful:WadlRequestParamMap>
      <grounding:owlsParameter rdf:resource="&ypc;#YNS-Query"/>
      <restful:wadlMessageParam rdf:datatype="&xsd;#anyURI">&wadl;#query</restful:wadlMessageParam>
    </restful:WadlRequestParamMap>
  </restful:wadlRequestParam>
  <!-- (...) Definições dos demais parâmetros omitidas -->

  <!-- Parâmetros de Resposta -->
  <restful:wadlResponseParam>
    <restful:WadlResponseParamMap>
      <grounding:owlsParameter rdf:resource="&ypc;#YNS-Output"/>
      <restful:wadlMessageParam rdf:datatype="&xsd;#anyURI">
                &wadl;#YNS-Rep-Success
          </restful:wadlMessageParam>
    </restful:WadlResponseParamMap>
  </restful:wadlResponseParam>
</restful:WadlAtomicProcessGrounding>
</rdf:RDF>
```

Figura 5.5: *Fragmento do código OWL para ontologia que descreve semanticamente o serviço de busca de notícias do Yahoo! (adaptado de [41]).*

5.3 Implementação de Serviços Semânticos com OWL-S API

OWL-S API provê a possibilidade de acesso, a partir de linguagens de programação, à descrição semântica de serviços Web em OWL-S. Através dela, é possível implementar a leitura, escrita, criação e execução de processos OWL-S atômicos e compostos. A versão da OWL-S API utilizada foi a 3.1, ainda em desenvolvimento e a mais recente até o momento. Esta versão admite ontologias escritas no formato OWL 1.2.

Como as ontologias disponibilizadas pela OWL-S, a API é dividida em pacotes que possibilitam a extensão e adição de novas características. Ela possui um mecanismo de execução de processos (chamado `ProccessExecutionEngine`) que pode executar processos atômicos utilizando fundamentação em WSDL, Java, ou UPnP. Também pode executar processos compostos que usam construtores de controles como `Choice`, `Sequence`, `AnyOrder`, `Split` e `Split-Join`. Esses construtores serão explicados na Sessão 5.3.2.

A estrutura de classes da API foi definida para seguir o mais próximo possível as definições das ontologias OWL-S. Para tal, foram definidos quatro pacotes principais: `service`, `profile`, `process` e `grounding`. Também há o pacote `vocabulary` que possui classes com as definições de URIs para as classes e propriedades ontológicas da OWL-S. Para garantir a interoperabilidade, a API implementa polimorfismo das classes. Há interfaces que abstraem as funcionalidades previstas nas ontologias. Por exemplo, a classe ontológica `AtomicGrouding` possui uma interface que a representa. Para cada tipo de fundamentação há uma classe que implementa tal interface, garantindo o polimorfismo e a interoperabilidade.

5.3.1 Invocação

Uma vez que a definição sintática e a definição semântica estão construídas, a invocação de serviços Web com a OWL-S API é relativamente simples. A Figura 5.6 exibe um exemplo de código para invocação de serviços semânticos com a OWL-S API. Inicialmente, uma `ProcessExecutionEngine` é criada. Como dito anteriormente, este é o motor pelo qual, qualquer serviço é invocado na OWL-S API. Em seguida é adicionado um monitor ao motor de execução. Um monitor tem o papel de realizar auditoria em todas as ações realizadas no motor de execução de processos. O monitor `DefaultProcessMonitor` tem como objetivo, exibir as ações realizadas pelo motor na saída padrão do Java, facilitando a depuração.

```
1         ProcessExecutionEngine exec = OWLSFactory.createExecutionEngine();
2         exec.addMonitor(new DefaultProcessMonitor());
3         final OWLKnowledgeBase kb = OWLFactory.createKB();
4
5         // Carregando ontologias que descrevem o serviço
6         URI uri = new URI("http://on.cs.unibas.ch/owl-s/1.2/ZipCodeFinder.owl");
7         Service service = kb.readService(uri);
8         Process process = service.getProcess();
9
10        // Definindo entradas
11        ValueMap<Input, OWLValue> inputs = new ValueMap<Input, OWLValue>();
12        inputs.setValue(process.getInput("City"), kb.createDataValue("College Park"));
13        inputs.setValue(process.getInput("State"), kb.createDataValue("MD"));
14
15        // Criando conjunto de saídas e executando processo
16        ValueMap<Output, OWLValue> outputs = new ValueMap<Output, OWLValue>();
17        outputs = exec.execute(process, inputs, kb);
18
19        // Exibindo a saída em RDF
20        System.out.println(process.getOutput().toRDF(true, true));
21
22        // Exibindo a saída em String
23        final OWLValue outputValue = outputs.getValue(process.getOutput());
24        System.out.println(outputValue.toString());
```

Figura 5.6: *Exemplo de código Java para requisição e execução de um serviço semântico através da OWL-S API.*

Na linha 3 do código apresentado na Figura 5.6 é criada uma base de conhecimento OWL (*OWLKnowledgeBase*). Isto é realizado através da fábrica `OWLFactory`. A `OWLKnowledgeBase` disponibiliza uma série de métodos para leitura de ontologias para dentro da base de conhecimento. Com isso, é possível ler ontologias em arquivos externos através de URIs e importá-las para dentro da base de conhecimento. O método utilizado para ler a ontologia do serviço para a base é `readService`. Através da execução de tal método, a base passa a conter as ontologias de descrição do Serviço, do Processo, do Perfil e da Fundamentação, bem como classes ontológicas que descrevem tipos complexos que o serviço pode receber ou retornar.

Após o carregamento das ontologias na base de conhecimento, é possível acessar o Processo, bem como suas entradas. Para definir os valores das entradas é necessário converter os dados em `OWLDataValue`. Para isso, é utilizado o método `createDataValue` da `OWLKnowledgeBase`. Com a definição das entradas, é possível executar o motor, através do método `execute`. Tal método invoca o serviço passando o processo, as entradas e a base de conhecimento. O retorno do método são as saídas retornadas pelo serviço. Um serviço pode retornar uma ou mais saídas, dependendo de sua composição. Cada saída é uma instância da classe `Output` que pode ser transformada em RDF, para consultas semânticas elaboradas, `String` ou em um objeto do java que represente o tipo complexo

da saída.

5.3.2 Composição

Como dito na Seção 3.5.3, um serviço OWL-S pode ter um processo atômico, composto ou simplificado. Esses três tipos de processos são representados pelas interfaces `AtomicProcess`, `CompositeProcess` e `SimpleProcess`, respectivamente, na OWL-S API. A composição de um processo é feita pela ontologia. Entretanto, a API disponibiliza mecanismos para que tal composição seja feita no próprio código em Java. Para criar um processo composto é necessário utilizar um construtor de controle. Os construtores de controle descrevem como um processo composto poderá ser decomposto. Logo, um processo composto não descreve um comportamento do que o serviço fará, mas sim um comportamento do que o cliente pode realizar, enviando e recebendo uma série de mensagens. Na OWL-S os construtores de controle podem ser[24]:

- ***Sequence***: Neste construtor de controle, os sub-processos são chamados em ordem sequencial, um após o outro;
- ***Split***: Em tal construtor de controle, os sub-processos são chamados simultaneamente. O processo completa-se assim que todos seus sub-processos tiverem sido agendados para execução;
- ***Split+Join***: Neste construtor de controle, os sub-processos são chamados simultaneamente, com sincronização de barreira. Isso significa que o processo conclui-se apenas quando todos os sub-processos forem concluídos. Com *Split* e *Split+Join* é possível definir processos com sincronização parcial;
- ***Any-Order***: Os sub-processos são executados em uma ordem não especificada, mas não simultaneamente. Como no *Spli+Join*, a conclusão de todos os sub-processos é necessária. A execução dos sub-processos não pode ser sobreposta;
- ***Choice***: Neste construtor, o processo é composto por um conjunto de construtores de controle. Um único construtor é escolhido e executado. Qualquer um dos construtores presentes na lista poderá ser escolhido;
- ***If-Then-Else***: Este construtor de controle possui três propriedades: `ifCondition`, `then` e `else`. Sua semântica tem o propósito de testar a condição em `ifCondition`. Caso seja verdadeira a definição em `then` é executada. Caso contrário, executa-se a definição em `else`. As propriedades `then` e `else` aceitam construtores de controle.
- ***Iterate***: Este construtor é abstrato. Logo, não é possível criar instâncias de tal construtor. Ele tem como objetivo a definição de um modelo para criação de construtores com iteração. Os já definidos pela ontologia são `Repeat-While` e `Repeat-Until`. Entretanto, novos construtores de iteração podem ser definidos;

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<process:CompositeProcess rdf:ID="CP">
   <process:hasInput rdf:ID="I1"/>
   <process:hasOutput rdf:ID="O1"/>
   <process:composedOf>
      <process:Sequence rdf:ID="CP">
         <process:components rdf:parseType="Collection">
            <process:Perform rdf:ID="Step1">
               <process:process rdf:resource="&aux;#S1"/>
               <process:hasDataFrom>
                  <process:InputBinding>
                     <process:theParam rdf:resource="&aux;#I11"/>
                     <process:valueSource>
                        <process:valueOf>
                           <process:theParam rdf:resource="#I1"/>
                           <process:fromProcess rdf:resource="#TheParentPerform"/>
                        </process:valueOf>
                     </process:valueSource>
                  </process:InputBinding>
                  <process:InputBinding>
                     <process:theParam rdf:resource="&aux;#I12"/>
                     <process:valueData xsd:datatype="&xsd;#string">Academic</process:valueData>
                  </process:InputBinding>
               </process:hasDataFrom>
            </process:Perform>
            <process:Perform rdf:ID="Step2">
               <process:process rdf:resource="&aux;#S2"/>
               <process:hasDataFrom>
                  <process:Binding>
                     <process:theParam rdf:resource="&aux;#I21"/>
                     <process:valueSource>
                        <process:ValueOf>
                            <process:theParam rdf:resource="#O11"/>
                            <process:fromProcess rdf:resource="#Step1"/>
                        </process:ValueOf>
                     </process:valueSource>
                  </process:Binding>
               </process:hasDataFrom>
            </process:Perform>
         </process:components>

         <process:Produce>
            <process:producedBinding>
               <process:OutputBinding>
                   <process:theParam rdf:resource="#O1"/>
                   <process:valueSource>
                      <process:valueOf>
                         <process:theParam rdf:resource="#O21"/>
                         <process:fromProcess rdf:resource="#S2"/>
                      </process:valueOf>
                   </process:valueSource>
               </process:OutputBinding>
            </process:producedBinding>
      </process:Sequence>
  </process:composedOf>
</process:CompositeProcess>
```

Figura 5.7: *Exemplo de ontologia que descreve um processo composto na OWL-S.*

- ***Repeat-While*** e ***Repeat-Until***: Estes construtores repetem operações enquanto uma condição está verdadeira ou falsa, seguindo as convenções de laços de repetição em linguagens de programação. `Repeat-While` testa uma condição e enquanto ela for verdadeira executa a operação, caso falsa, para a execução. Este construtor possui as propriedades `whileCondition` que recebe a condição e, `whileProcess`, que recebe o processo no qual será feita a iteração. `Repeat-Until` repete uma operação enquanto a condição for falsa e para quando verdadeira. Ele possui as propriedades `untilCondition` para a condição e `untilProcess` para o processo.

```
1  // Criando Ontologia e Processo composto
2  final OWLKnowledgeBase kb = OWLFactory.createKB();
3  OWLOntology ont = kb.createOntology(baseURI);
4  final CompositeProcess compositeProcess = ont.createCompositeProcess(processURI);
5  final Input inputI1 = ont.createInput(I1URI);
6  final Output outputO1 = ont.createOutput(O1URI);
7
8  // Criando Step1 (S1 é o processo externo do qual Step1 é uma realização)
9  Process S1;
10 final Perform performStep1 = ont.createPerform(null);
11 performStep1.setProcess(S1);
12 // Definindo entradas de Step1
13 performStep1.addBinding(S1.getInput("I11"), OWLS.Process.ThisPerform, inputI1);
14 performStep1.addBinding(S1.getInput("I12"), ont.createValueConstant(
15                 ont.createDataValue("Academic"), ont.createInputBinding(null)));
16
17 // Criando Step2 (S2 é o processo externo do qual Step2 é uma realização)
18 Process S2;
19 final Perform performStep2 = ont.createPerform(null);
20 performStep2.setProcess(S2);
21 // Definindo saída O11 do Step1 como entrada I21 do Step2
22 performStep2.addBinding(S2.getInput("I21"), performStep1,
23                 performStep1.getProcess().getOutput("O11"));
24
25 // Criando Sequence e adicionando as realizações a ele
26 final Sequence sequence = ont.createSequence(null);
27 sequence.addComponent(performStep1);
28 sequence.addComponent(performStep2);
29
30 // Definindo saída do processo composto
31 final Result result = ont.createResult(null);
32 result.addBinding(outputO1, performStep2, performStep2.getProcess().getOutput("O21"));
33 compositeProcess.addResult(result);
```

Figura 5.8: *Exemplo de código Java para criar processo composto com a OWL-S API.*

Para que um processo seja tratado como um sub-processo de outro maior, é necessário definir uma "realização"de tal processo. Uma realização de um processo é uma representação dele em um determinado escopo. Isso é feito através da classe `Perform`.

Trata-se de uma classe filha da classe `ControlConstruct`, que define os construtores de controle. A Figura 5.7 exibe um exemplo de processo composto definido através de ontologias. Neste exemplo, o construtor de controle é `Sequence`, sendo que o processo possui uma entrada e uma saída e, executa dois sub-processos (*Step1* e *Step2*), um após o outro. O processo da realização *Step1* possui duas entradas. A primeira é preenchida com o valor da entrada do processo principal. A segunda é fixada com o valor *"Academic"*. A realização do segundo sub-processo (*Step2*), possui uma entrada que é preenchida com a saída do primeiro sub-processo (*Step1*). A saída do *Step2* é enviada para a saída do processo principal.

No processo composto da Figura 5.7, para que o *Step2* receba como entrada a saída de *Step1*, é necessário ambos aceitem instâncias da mesma classe ontológica. Os processos compostos também podem ser construídos através em código Java, através da OWL-S API. A Figura 5.8 exibe o mesmo processo composto, gerado através da OWL-S API.

5.3.3 Descoberta

Como dito anteriormente, a OWL-S utiliza *matchmakers* para descoberta dos serviços. A OWL-S API disponibiliza uma classe `Matchmaker` como exemplo básico dessa abordagem. Entretanto, é possível fazer *matchmakers* personalizados de acordo com a necessidade da aplicação. Também existem aqueles desenvolvidos por outras equipes e grupos de estudo. O mais robusto e popular é o OWLS-MX[65]. Trata-se de um abordagem para correspondência híbrida de serviços Web, utilizando "raciocinadores"(*reasoners*) baseados em lógica e técnicas de recuperação de informação baseada no conteúdo.

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX profile: <http://www.daml.org/services/owl-s/1.2/Profile.owl#>
3  PREFIX exp: <http://www.exemplo.com.br/ontologias/process>
4  SELECT *
5  WHERE {
6          ?profile rdf:type profile:Profile.
7          ?profile profile:hasInput ?param.
8          ?param rdf:type exp:Hotel.
9  }
```

Figura 5.9: *Exemplo de consulta SPARQL para descoberta de serviços descritos semanticamente em OWL-S.*

A maneira mais fácil de realizar a descoberta de serviços é através de consultas SPARQL. Uma vez que as descrições semânticas de serviços, processos e tipos de dados estão na base de conhecimento, uma consulta semântica poderá fazer a descoberta de

um serviço adequado para uma determinada ação. A identificação das entradas e saídas dos serviços também poderá ser feita em tal consulta. A Figura 5.9 exibe um exemplo de consulta SPARQL para a descoberta de serviços semânticos. Neste exemplo, são descobertos todos os serviços que possuem o parâmetro hipotético `exp:Hotel` como entrada. Para descobertas mais inteligentes faz-se necessária a utilização de um motor de inferências. A OWL-S utiliza, por padrão, o "raciocinador"*Pellet*[97] para tal atividade.

5.4 Discussão

A partir da ontologia *RESTfulGrounding*, é possível descrever serviços Web RESTful, com fundamentação sintática em arquivos WADL. A descrição sintática de serviços Web RESTful pode ser feita utilizando outros padrões (como WSDL 2.0). Entretanto, a abordagem do WADL é mais simplificada e atende os princípios da arquitetura REST. Utilizando a OWL-S API é possível a implementação prática de serviços Web semânticos. Entretanto, tal API não abrange serviços RESTful. Isso impede que APIs de Redes Sociais Online sejam descritas semanticamente. É necessário, então, o desenvolvimento de um módulo da OWL-S API para serviços RESTful. O próximo capítulo destina-se a exibir o projeto de tal módulo e descrever como ele foi desenvolvido.

CAPÍTULO 6

Projeto do Módulo RESTful Para OWL-S API

O desenvolvimento do Módulo para descrição de serviços Web RESTful foi dividido em quatro partes, seguindo os conceitos da Engenharia de Software. Inicialmente foram levantados os requisitos, em uma etapa de análise. Em seguida foi realizada a etapa de projeto (*design*), criando-se a modelagem do sistema, a partir de diagramas, e a arquitetura. As etapas finais consistem no desenvolvimento e testes do código fonte do Módulo gerado. Na etapa de desenvolvimento foram descritas as principais classes, interfaces e métodos do código criado. A etapa de testes exemplificará a utilização do Módulo desenvolvido.

Para realizar o levantamento de requisitos, inicialmente é necessário definir os atores e casos de uso do Módulo. Um caso de uso é uma unidade funcional coerente provida pelo sistema. Atores do sistema são os agentes (humanos ou máquinas) que interagem e realizam os casos de uso. Como trata-se de uma API, não há seres humanos como atores. Os atores definidos foram: **Cliente** e **Servidor**. O Cliente requisita e consome os serviços semânticos. O Servidor disponibiliza o serviço, seus recursos e as descrições sintática e semântica deste. Foi definido apenas um caso de uso: **Implementar Descrição Semântica em Serviços RESTful**. Este caso de uso sintetiza a funcionalidade provida pelo Módulo.

6.1 Levantamento de Requisitos

A etapa de levantamento de requisitos consiste na definição das necessidades do sistema. Normalmente, essa etapa é executada por meio de questionários e entrevistas com os usuários do *software*. Entretanto, por tratar-se de uma API, não foi possível a utilização destas técnicas. Os requisitos foram levantados com base nas funcionalidades necessárias para atingir o objetivo do caso de uso definido. Essa etapa foi dividida em duas partes: Requisitos Funcionais e Não Funcionais.

Os Requisitos Funcionais descrevem as funcionalidades necessárias para que o caso de uso seja colocado em prática. A partir deles, é possível identificar as ações que devem ser executadas para que o sistema atinja seu objetivo. Eles servem de base

para a geração das regras de negócio do sistema. Os Requisitos Não Funcionais definem parâmetros de qualidade e restrições para todo o sistema. Eles descrevem as necessidades acerca da finalidade do produto, questões de usabilidade, características relacionadas ao desempenho, manutenção e qualidade do sistema.

6.1.1 Requisitos Funcionais

Para definição dos requisitos funcionais, foram validados os princípios REST, os conceitos de Serviços Web Semânticos e a utilização de ontologias. Foram definidos 7 requisitos funcionais para o Módulo RESTful da OWL-S API:

- **Garantir o princípio da endereçabilidade**. Cada recurso disponibilizado por um serviço RESTful deve possuir uma identificação única. Essa identificação é obtida através de URIs.
- **Prover uma interface uniforme para manipulação dos recursos**. Para isso, faz-se necessária a disponibilização dos recursos através do protocolo HTTP. A interface unificada deste protocolo garantirá que um consumidor qualquer possa manipular os recursos através dos métodos de requisição (GET, POST, PUT, DELETE).
- **Garantir o princípio do estado não-presente**. As interações com os recursos não devem armazenar estado. É necessário que as requisições contenham todos os dados necessários para que o serviço seja executado corretamente e o recurso retornado.
- **Garantir o princípio da auto-descrição de mensagens**. As representações de um determinado recurso devem se auto-descreverem, através do cabeçalho de resposta do protocolo HTTP. É necessário, também, garantir a conectividade entre recursos. Para isso, as representações devem conter apontadores para outras representações, conectando os recursos.
- **Prover a descoberta de serviços semânticos RESTful**. Através da descrição semântica dos serviços RESTful, poderá ser feita a descoberta dinâmica de serviços. Para isso, faz-se necessária a utilização de ontologias (OWL-S e *RESTfulGrounding*).
- **Prover a composição de serviços semânticos RESTful**. Serviços com processos compostos, que envolvam mais de um serviço RESTful, poderão ser construídos. Isso poderá ser feito em ontologias ou programaticamente. Este requisito, em conjunto com o anterior, garantirá que serviços sejam descobertos dinamicamente e novos serviços compostos automaticamente, em tempo de execução.
- **Invocar automaticamente serviços RESTful**. Um serviço poderá ser invocado a partir de sua descrição sintática. Para isso, deverá ser utilizado o padrão WADL. Os recursos, suas representações e URIs, deverão ser extraídos de um documento

WADL que descreve o serviço. Esta invocação poderá ser realizada automaticamente, a partir da descoberta e composição de serviços semânticos.

6.1.2 Requisitos Não Funcionais

A elaboração dos requisitos não funcionais foi baseada na Norma ISO 9126 [62]. Essa norma propõe um Modelo de Qualidade de Software baseado em 6 atributos de qualidade: Funcionalidade, Confiabilidade, Usabilidade, Eficiência, Manutenibilidade e Portabilidade. Esses são atributos gerais para o desenvolvimento de qualquer software com qualidade. Para o desenvolvimento do Módulo da API, algumas adaptações foram necessárias.

Para o atributo Funcionalidade, foram definidos o requisito Interoperabilidade e o requisito Segurança. Como Confiabilidade foi definido o requisito Tratamento de Exceções. Como trata-se de uma API, um *software* que provê serviços para outro *software*, o atributo Usabilidade não foi utilizado. No lugar dele foi definido o requisito Documentação. Como Eficiência, foi definido o requisito Paralelismo. Para o atributo Manutenibilidade definiu-se o requisito Estabilidade. Como Portabilidade foi definido o requisito Flexibilidade. Por fim, foi definido o requisito Semântica que não possui correlato na norma.

- **Semântica**. Por tratar-se de um sistema para descrição semântica de serviços Web, é necessário garantir o acesso, armazenamento, processamento e comunicação dos dados de forma semântica. Isso deverá ser realizado através dos recursos já utilizados pela API.
- **Interoperabilidade**. O Módulo, a ser desenvolvido, deverá possuir funcionalidades que permitam a interação com outros sistemas de maneira simplificada. Esses sistemas poderão ser outras APIs, arcabouços e interfaces com o usuário, por exemplo.
- **Tratamento de Exceções**. O Módulo deve permitir diversas formas de uso, identificar possíveis exceções e tratá-las adequadamente.
- **Documentação**. É necessário que haja uma boa documentação. A documentação deverá ser composta de duas partes. Uma para o usuário, que irá utilizar a API e outra para o desenvolvedor, que irá construir novas características para a API.
- **Paralelismo**. É necessário que haja a capacidade de divisão de processos para execução paralela, aumentando a velocidade de resolução de problemas. Isso poderá ser garantido através de um sistema multiagentes externo.
- **Estabilidade**. Os efeitos colaterais decorrentes de modificações introduzidas devem ser reduzidos. É necessário garantir que novos módulos poderão ser introduzidos

sem modificações na estrutura existente. Como trata-se de um Módulo para uma API, os padrões e tecnologias utilizadas deveram ser seguidos e aproveitados.

- **Flexibilidade**. O sistema deve ser flexível e permitir a substituição de elementos externos utilizados. Para isso, é necessário definir camadas coesas garantindo o baixo acoplamento entre elas.

6.2 Construção da Modelagem e Arquitetura

Esta seção destina-se ao detalhamento da etapa de projeto (*design*) e da arquitetura do sistema. Inicialmente, é exibida e descrita a arquitetura do sistema. Esta arquitetura foi baseada nos requisitos levantados e nas funcionalidades providas pela API. Em seguida, é feita a modelagem do sistema, através de diagramas de classe e sequência. Através da arquitetura e da modelagem do sistema é possível ter uma visão clara de como ele funciona e quais as funcionalidades providas por ele.

6.2.1 Arquitetura

Por se tratar de algo bem específico, a arquitetura é simplificada, como pode ser visto na Figura 6.1. Basicamente, o Módulo é dividido em duas partes. Uma destinada à fundamentação do serviço semântico e outra destinada à invocação deste. Como é possível observar, na arquitetura não há os elementos para descoberta e composição de serviços semânticos. Isso se dá pois tais funcionalidades serão abstraídas pela API, não sendo necessária implementação no Módulo para tal tarefa.

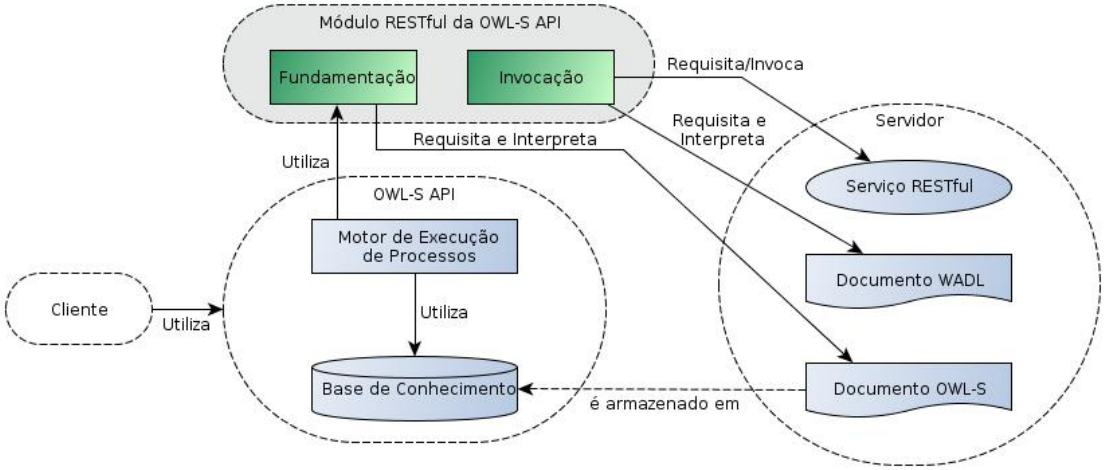


Figura 6.1: *Arquitetura do Módulo RESTful para OWL-S API.*

A camada de fundamentação do serviço destina-se a interagir com o servidor, requisitando as descrições semântica e sintática do serviço. É responsável, também, por

armazenar as ontologias de descrição do serviço RESTful e de seus recursos, requisitadas em formato OWL, na base de conhecimento da API. Para isso, é utilizado o arcabouço Jena, descrito na Seção 2.4.1. A parte de fundamentação ainda possui o papel de interpretar o documento WADL, identificando os recursos disponibilizados pelo serviço, os possíveis métodos de requisição desses recursos, seus parâmetros de entrada, e as possíveis representações.

A camada de invocação é responsável por requisitar os recursos de um serviço RESTful. A invocação é feita após a fundamentação. Uma vez que as ontologias de descrição do serviço, seu processo, seu perfil e a fundamentação estejam armazenadas na base de conhecimento da API, a descoberta poderá ser feita a partir da própria API. Com a informação de qual serviço requisitar e a partir dos dados, já interpretados, presentes no documento WADL, é possível invocar o serviço, requisitando o recurso adequado de maneira correta. O tratamento do retorno do serviço também é feito nessa etapa. Caso tenha ocorrido alguma falha na requisição do recurso, ela será identificada neste momento.

Através da arquitetura proposta é possível validar os três últimos requisitos funcionais descritos na Seção 6.1.1. Esses requisitos dizem respeito à invocação, descoberta e composição de serviços semânticos RESTful. Os demais requisitos serão validados na etapa de modelagem, na seção a seguir.

6.2.2 Modelagem

A modelagem do sistema foi definida por dois tipos de diagramas: diagrama de sequência e de classes. O diagrama de sequência tem como objetivo representar a sequência de processos necessários para realizar as atividades providas pelo sistema. A análise do fluxo de dados e atividades da aplicação pode ser realizada através dos métodos das classes do sistema. Entretanto, esta análise é bem mais facilmente realizada com a utilização de um diagrama de sequência. O diagrama de classes exibe as classes que compõem o sistema orientado a objetos. Através do diagrama de classes torna-se mais fácil a definição das classes com alta coesão. Tal diagrama também auxilia na visualização de acoplamentos desnecessários, com propósito de reduzi-los.

Na elaboração do diagrama de sequência foram identificados dois fluxos de atividades e, portanto, dois diagramas foram definidos. O primeiro diagrama, apresentado na Figura 6.2 representa a sequência de processos necessários para compor o Serviço. Nessa etapa, as ontologias são requisitadas e armazenadas na Base de Conhecimento, disponibilizada pela API. A Figura 6.3 apresenta o segundo diagrama de sequência. Neste, é representada a sequência de processos com o propósito de requisitar um recurso disponibilizado pelo serviço RESTful. Essa etapa inicia-se na descoberta dos serviços, em seguida, a fundamentação é realizada e, por fim, o serviço é invocado e o recurso

requisitado.

O primeiro diagrama (Figura 6.2) começa com a criação da Base de Conhecimento. Nela serão armazenadas as ontologias que descrevem o serviço. Em seguida, o Cliente especifica o URI para a ontologia do serviço. Tal ontologia faz referência às ontologias complementares (do processo, do perfil e de fundamentação). Através do URI informado pelo Cliente, a API requisita a ontologia de descrição do serviço e todas aquelas que ela fizer referência. Uma vez requisitadas, as ontologias são armazenadas na Base de Conhecimento. Entretanto, a ontologia de fundamentação necessita de ser processada pelo Módulo antes de ser armazenada. Isso é necessário visto que as ontologias de fundamentação para RESTful não são originalmente esperadas pela API. Por fim, a Base de Conhecimento com as ontologias requisitadas é disponibilizada ao Cliente.

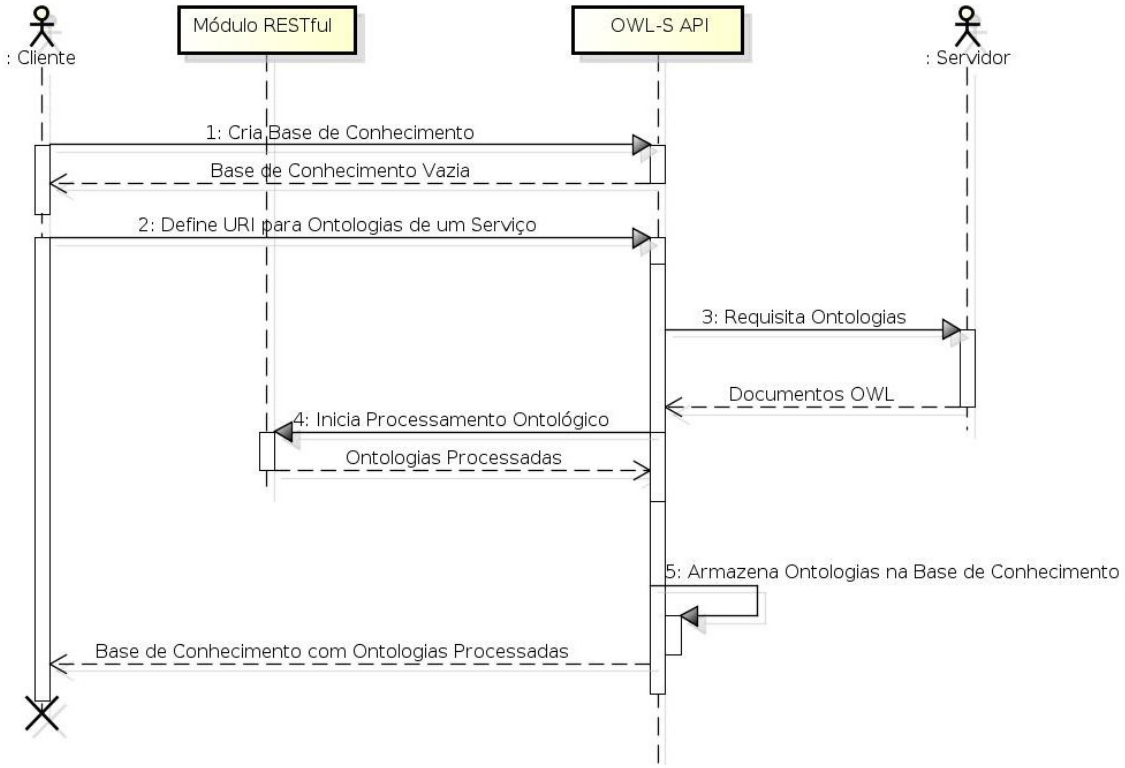


Figura 6.2: *Diagrama de Sequência para requisição de ontologias que descrevem serviços.*

O segundo diagrama de sequência (Figura 6.3) inicia-se com a descoberta do serviço. Essa descoberta poderá ser feita através de consultas SPARQL, de *Matchmakers* mais elaborados ou selecionando o serviço manualmente (a partir de sua ontologia). Na etapa de descoberta, a API retorna para o Cliente o serviço que melhor combina com a consulta. É possível que seja retornado mais de um serviço, dependendo da consulta realizada. Caso isso ocorra, a escolha de qual serviço utilizar é de responsabilidade do Cliente. Com o serviço definido, o Cliente especifica os valores de entrada, caso o serviço

necessite. Em seguida, o Motor de Execução de Processos é iniciado e executado pelo Cliente, para a API. A partir dai, a API envia o fluxo da aplicação para o Módulo RESTful.

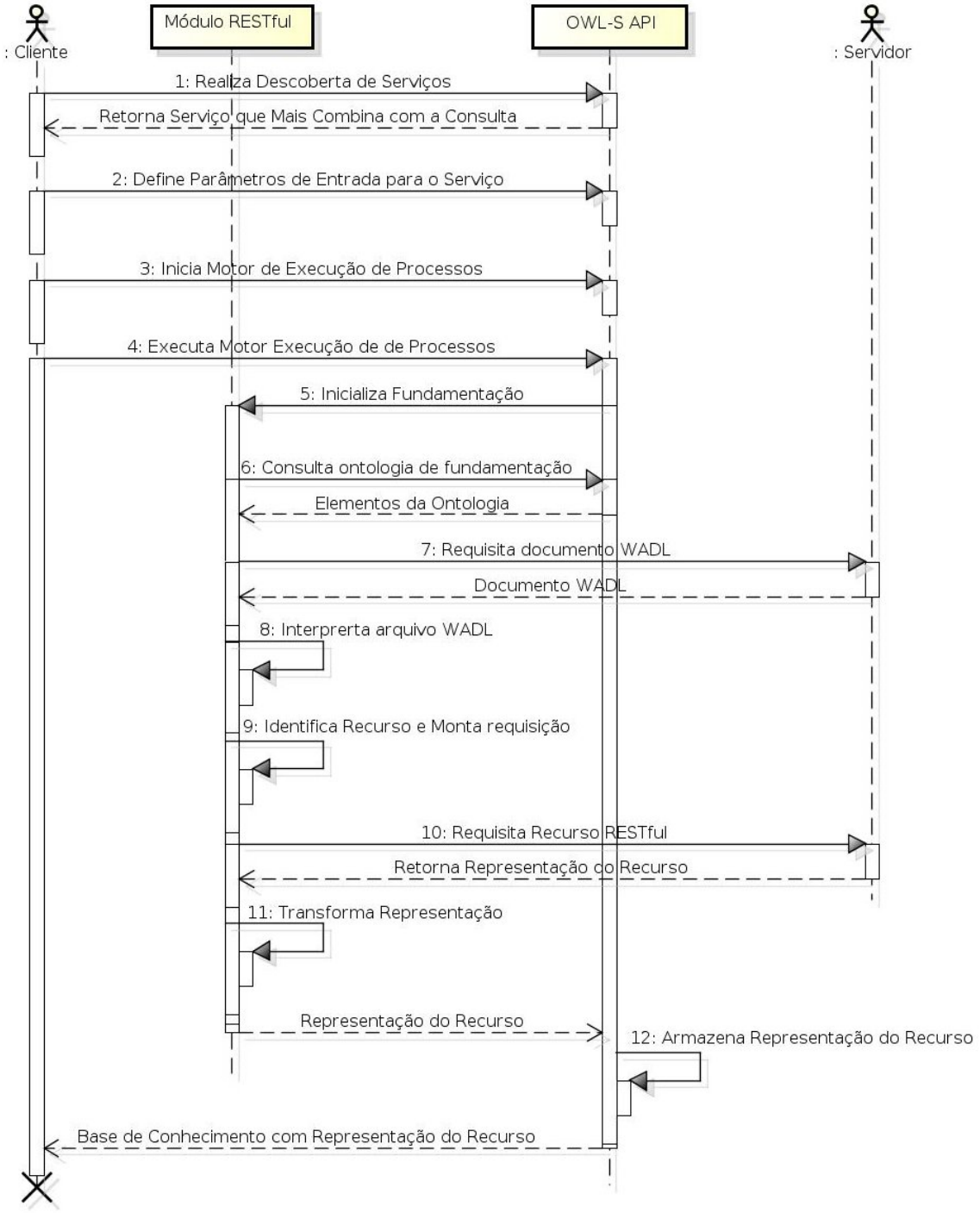


Figura 6.3: *Diagrama de Sequência para descoberta, fundamentação e invocação de serviços RESTful.*

O Módulo RESTful é responsável pela fundamentação do serviço RESTful. Para isso, ele consulta a ontologia de fundamentação, armazenada na Base de Conhecimento, até que o fluxo seja enviado novamente para a API. Na ontologia, o Módulo RESTful identifica o URI do documento WADL e requisita-o ao Servidor. Ao completar a requi-

sição, o Módulo RESTful interpreta o arquivo WADL, em conjunto com a ontologia de fundamentação. Em seguida, o recurso a ser requisitado, suas representações e seus parâmetros de entrada são identificados. A requisição do recurso então é montada e enviada ao Servidor. Como resposta, o Servidor retorna uma representação do recurso. O Módulo RESTful, então, valida a representação e a transforma, em triplas RDF ou Objeto do Java, caso seja necessário. Em seguida, o recurso é armazenado na Base de Conhecimento da API e disponibilizado para o Cliente.

A segunda sequência de processos deverá ser realizada sempre após a primeira. Elas estão separadas visto que a primeira pode ser realizada várias vezes antes de a segunda acontecer. Com muitas ontologias, que descrevam serviços diferentes, a descoberta poderá ser mais elaborada e eficiente. Nos dois diagramas é possível observar que o fluxo começa e termina no Cliente. Isso garante que ele tenha controle de todo o processo. Nota-se, também, que o Cliente sempre se comunica diretamente com a API e nunca com o Módulo ou com o Servidor. Isso garante a abstração de qual é a fundamentação que está sendo utilizada. Dessa forma, para o Cliente, a requisição de serviços SOAP ou RESTful será exatamente igual e transparente.

Foram definidos dois diagramas de classes. O primeiro diagrama, apresentado na Figura 6.4, exibe as classes que estendem a OWL-S API para a realização da fundamentação de serviços RESTful. Os atributos e métodos foram suprimidos para economizar espaço. Esse diagrama define uma interface que representa o *grounding* em WADL (`WADLGrounding`) e outra que representa o processo atômico para esse *grounding* (`WADLAtomicGrounding`). É definida, também, uma interface que representa um recurso e um método (`WADLResourceMethodRef`) fazendo referência à classe ontológica de mesmo nome. Essas interfaces possuem suas implementações em classes com terminação `Impl`.

O diagrama ainda define as classes `WADLGroundingFactory` e `WADLMessageParamMap`. A primeira disponibiliza instâncias das outras classes, quando necessário, conforme o padrão de projeto *Factory* da orientação a objetos. A segunda, representa os parâmetros de entrada e as representações dos recursos a serem requisitados. Por fim, é definida a interface `WADLGroundingProvider` e a classe que a implementa. Ela é utilizada pela fábrica com o propósito de prover as instâncias e de armazenar os dados para conversão de classes Java em classes ontológicas. Por fim, há a classe `OWLSRestful` que realiza a função de vocabulário, provendo informações sobre as classes e propriedades da ontologia *RESTfulGrounding*.

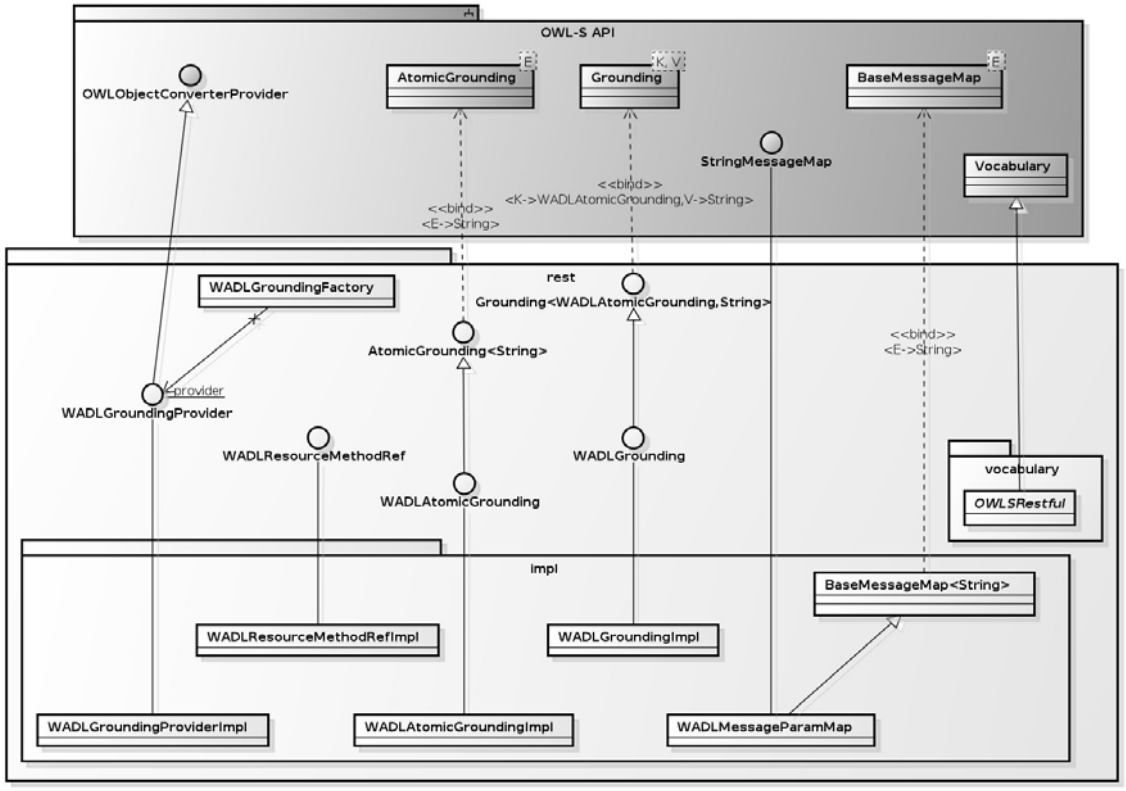


Figura 6.4: *Diagrama de Classes da parte de fundamentação do Módulo RESTful para a OWL-S API.*

A Figura 6.5 apresenta um diagrama com as classes que compõem a parte de invocação do serviço RESTful. A classe `WADLApplication` representa um elemento `application` do WADL. Ela possui o papel de realizar a interpretação do documento WADL. Para isso ela utiliza a biblioteca Wadl2Java[1]. Essa classe também funciona como uma fábrica para criação de objetos que representam recursos. Tais objetos são instâncias da classe `WADLResource`.

[1] Wadl2Java é uma biblioteca desenvolvida pelos mesmos criadores do padrão WADL. Mais informações podem ser obtidas em http://wadl.java.net/wadl2java.html

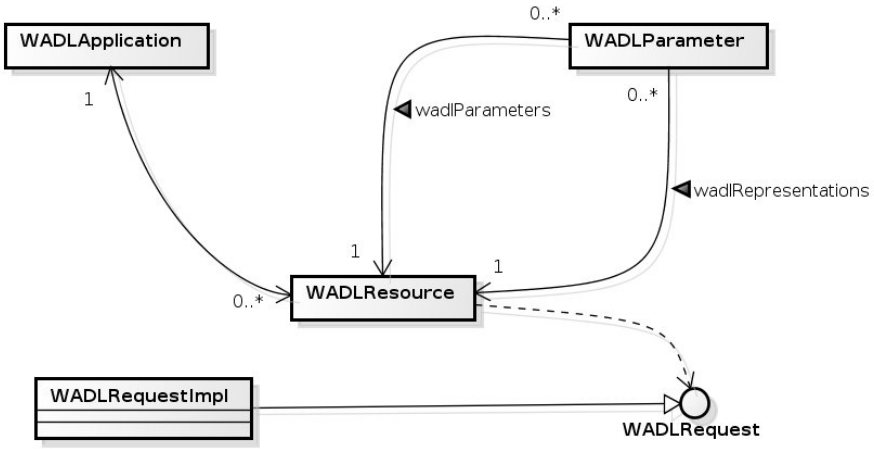


Figura 6.5: *Diagrama de Classes para invocação do serviço RESTful.*

Além de representar os recursos, a classe `WADLResource` possui o papel de invocar o serviço. Para isso ela utiliza um objeto do tipo `WADLRequest`. Essa interface garante a flexibilidade do sistema ao requisitar o recurso. Caso seja necessário requisitá-lo com um arcabouço mais elaborado (como por exemplo Restlet), basta criar uma classe que a implemente. A classe `WADLParameter` representa os parâmetros de requisição para um determinado recurso. Visando a simplificação e facilidade de uso, esta classe também pode expressar as representações de um serviço. Os métodos destas classes serão descritos na Seção 6.3.

A partir da modelagem proposta, os requisitos não funcionais são atendidos. A Interoperabilidade é garantida, visto que, por tratar-se de uma API ela obrigatoriamente irá interagir com outros sistemas. Como toda API, ela sozinha não produz nenhum resultado. O Tratamento de Exceções e a Documentação serão vistOs com mais ênfase na etapa de desenvolvimento (Seção 6.3). O Paralelismo poderá ser garantido a partir da utilização da API em conjunto com sistemas multiagentes, como será visto no Capítulo 7. A Estabilidade e Flexibilidade são garantidos a partir da alta coesão das classes (cada classe possui um propósito específico e especializado) e do baixo acoplamento entre as camadas.

6.3 Desenvolvimento do Módulo

Esta seção destina-se a descrever as principais tarefas realizadas no desenvolvimento do Módulo RESTful para a OWL-S API. A implementação a ser apresentada, é baseada na análise e projeto realizados nas seções anteriores deste Capítulo. Foram aplicados grande parte dos conceitos descritos nos Capítulos 2 e 3. Inicialmente, são detalhadas as funcionalidades das principais classes (definidas na Seção 6.2.2) e de seus métodos. Al-

guns métodos essenciais serão exemplificados com códigos fonte. São descritas, também, as ferramentas e tecnologias utilizadas. Em seguida, é exibido um exemplo de utilização do Módulo desenvolvido, validando o propósito principal desta pesquisa.

Para o desenvolvimento do Módulo, foi utilizado o ambiente de desenvolvimento Eclipse Helios, o servidor de aplicações Tomcat 6.0, e a máquina virtual Java 1.6. Todas essas ferramentas são de código fonte aberto e livres. Para estender a OWL-S API, foi utilizada a versão 3.1 desta. Essa versão utiliza o Jena 2.6.4, o Pellet 2.1.1 e o ARQ 2.8.7. ARQ é o processador de consultas SPARQL do Jena. Ele é responsável também por persistir as ontologias (disponibilizando os sistemas TDB e SDB, vistos na Seção 2.4.1). Foi utilizada, também, a biblioteca Wadl2Java, em sua última revisão (308) obtida a partir do repositório de versões do código fonte. O desenvolvimento de todo o Módulo foi realizado no sistema operacional Linux (Ubuntu 10.04), sendo que testes foram realizados nos sistemas operacionais *Windows* 7 e XP.

De acordo com o projeto do Módulo, realizado na Seção anterior, ele foi dividido em duas partes. Essas partes foram separadas através de pacotes no código fonte em Java. Foram definidos os pacotes `br.ufg.inf.rest` e `br.ufg.inf.sws.rest`. Dentro do segundo pacote foram criados mais dois pacotes: `vocabulary` e `impl`. A classe `OWLSRestful`, presente no pacote `vocabulary`, tem como objetivo carregar a ontologia *RESTfulGrounding* para a Base de Conhecimento da API. Ela possui atributos que referenciam diretamente os elementos da ontologia, podendo ser considerada uma camada responsável por fazer uma ponte entre a ontologia e o Módulo.

A Figura 6.6 exibe uma versão simplificada da classe `OWLSRestful`. Os atributos dela podem ser dos tipos `OWLClass` (Classe ontológica), `OWLDataProperty` (Propriedade ontológica que referencia um literal), `OWLObjectProperty` (Propriedade ontológica que referencia uma instância de classe ontológica). Esses tipos são interfaces disponibilizadas pela OWL-S API. As implementações de tais tipos são feitas utilizando o Jena. A generalização em interfaces garante o desacoplamento da API ao Jena. Logo, caso o usuário queira utilizar outro arcabouço da Web Semântica, é possível sem alterações no código da API.

As classes presentes no pacote `br.ufg.inf.sws.rest.impl` são responsáveis por realizar a fundamentação em WADL. As principais classes responsáveis por essa tarefa são `WADLAtomicGroundingImpl` e `WADLGroundingProviderImpl`. Como visto na Seção 6.2.2 (Figura 6.4), elas implementam as interfaces `WADLAtomicGrounding` e `WADLGroundingProvider`, respectivamente. `WADLAtomicGrounding` é uma especialização de `AtomicGrounding` que representa uma fundamentação em WADL para um processo atômico. No diagrama visto na Figura 6.4 é possível observar que, na especialização, é vinculado um parâmetro de modelo, do tipo `String`. Esse parâmetro define o tipo do transformador. Este é responsável por interpretar o recurso requisitado e gerar uma

representação semântica deste. Ele será visto com mais detalhes mais adiante nesta seção.

```
1  public abstract class OWLSRestful extends Vocabulary {
2          public static final String URI = "http://solutio.in/owlsrestful/";
3          public static final String GROUNDING_NS = URI + "RESTfulGrounding.owl#";
4          public static final OWLOntology ont = loadOntology(GROUNDING_NS);
5
6          public static OWLClass WadlGrounding =
7                  ont.getClass(URIUtils.createURI(GROUNDING_NS + "WadlGrounding"));
8          // Outras definições de classes suprimidas (...)
9          public static OWLDataProperty method =
10                 ont.getDataProperty(URIUtils.createURI(GROUNDING_NS + "method"));
11         // Outras definições de propriedades suprimidas (...)
12         public static OWLObjectProperty wadlResourceMethod =
13                 ont.getObjectProperty(URIUtils.createURI(GROUNDING_NS + "wadlResourceMethod"));
14         // Outras definições de propriedades suprimidas (...)
15 }
```

Figura 6.6: *Versão simplificada do código fonte da classe* `OWLSRestful`.

Os principais métodos disponibilizados pela classe `WADLAtomicGroundingImpl` podem ser vistos na Figura 6.7. O método mais importante dessa classe é o `invoke()`. Ele configura a requisição e invoca o serviço. Para isso, utiliza informações providas por objetos das classes `WADLMessageMapParam` e `WADLResourceMethodRefImpl`. A primeira, disponibiliza informações de parâmetros de entrada e representação. A segunda, dos recursos e seus possíveis métodos de requisição. Para invocar o serviço, o método `invoke()` da classe `WADLAtomicGroundingImpl` instancia um objeto da classe `WADLResource`, abstraindo a invocação e a manipulação do arquivo WADL.

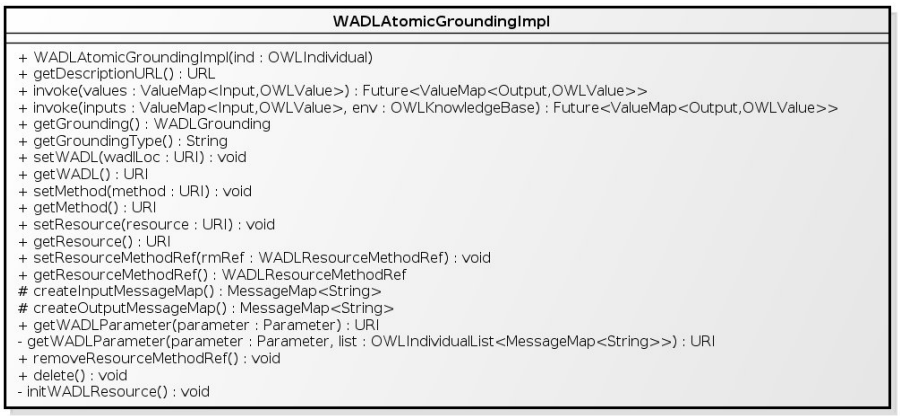


Figura 6.7: *Classe* `WADLAtomicGroundingImpl` *com seus métodos principais.*

A classe `WADLGroundingProviderImpl` possui o importante papel de realizar a conversão entre classes java e classes ontológicas da *RESTfulGrounding*. Para isso, ela

utiliza a classe `OWLSRESTful` para acessar as informações presentes na ontologia. Além disso, a classe `WADLGroundingProviderImpl` ainda registra quais classes serão utilizadas para realização da fundamentação (implementações das interfaces `Grounding` e `AtomicGrounding`). Isso é feito através do método `registerConverters`, que é apresentado em uma versão simplificada na Figura 6.8. A classe `WADLGroundingProviderImpl` pode ser considerada uma camada entre o Módulo e a API. A Figura 6.9 exibe os métodos e atributos da interface desta classe.

```
1 public void registerConverters(final OWLObjectConverterRegistry registry) {
2         final OWLObjectConverter<WADLAtomicGrounding> agc =
3                 new GenericOWLConverter<WADLAtomicGrounding>(
4                         WADLAtomicGroundingImpl.class, OWLSRestful.WadlAtomicProcessGrounding);
5
6         registry.registerConverter(WADLAtomicGrounding.class, agc);
7         registry.extendByConverter(AtomicGrounding.class, agc);
8         // (...) Outras definições foram suprimidas
9 }
```

Figura 6.8: *Versão simplificada do código fonte do método `registerConverters` da classe `WADLGroundingProviderImpl`.*

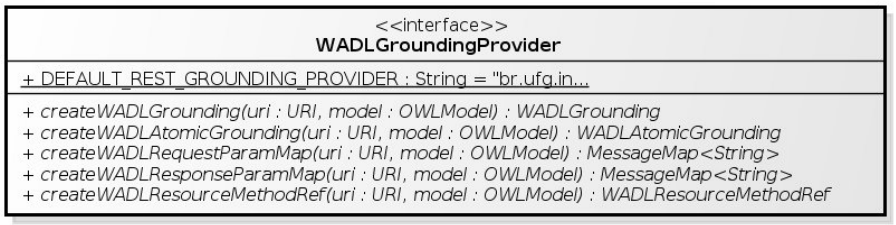


Figura 6.9: *Interface `WADLGroundingProvider`.*

A interface `WADLGroundingProvider` possui o atributo constante `DEFAULT_REST_GROUNDING_PROVIDER`. Esse atributo informa a implementação padrão de tal interface (no caso `WADLGroundingProviderImpl`). Tal atributo é utilizado pela classe `WADLGroundingFactory` que abstrai a classe provedora do restante da aplicação. É possível implementar uma nova fundamentação para RESTful, aproveitando a estrutura já construída no Módulo. Para que a API utilize a classe provedora definida no Módulo foi necessário criar um arquivo com o nome `org.mindswap.owl.OWLObjectConverterProvider` na pasta `/META-INF/services` do Módulo. Esse arquivo foi preenchido com o nome completo para a classe provedora (`br.ufg.inf.sws.rest.impl.WADLGroundingProviderImpl`).

As classes discutidas acima formam a camada de fundamentação do Módulo. Ou seja, elas trabalham com a parte semântica dos Serviços Semânticos RESTful (descrita na

Seção 5.2). A parte sintática (Seção 5.1) é realizada pelas classes de invocação do serviço. Para tal, foram desenvolvidas quatro classes (`WADLApplication`, `WADLResource`, `WADLParameter` e `WADLRequestImpl`) e uma interface (`WADLRequest`), como visto na Seção 6.2.2 (Figura 6.5).

A classe `WADLParameter`, utilizada para armazenar e descrever parâmetros de requisição e representações de recursos, é apresentada na Figura 6.10. Ela é composta apenas de atributos e os respectivos métodos de encapsulamento. Tais métodos foram suprimidos na Figura. Os atributos `name`, `type`, `style`, representam os atributos de mesmo nome no arquivo WADL e armazenam o nome, tipo e estilo do parâmetro, respectivamente. Os estilos de um parâmetro podem ser `query` (parâmetro de requisição), `template` (parâmetro na composição do URI), `matrix` (parâmetro de matriz, no URI), `header` (parâmetro no cabeçalho da requisição HTTP).

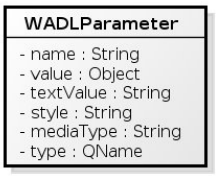


Figura 6.10: *Classe `WADLParameter`.*

Apenas uma classe foi desenvolvida para definição de parâmetros e representações por estes serem parecidos. A maior diferença está no atributo `mediaType` que armazena o tipo de arquivo esperado para uma representação. Esse atributo só é utilizado para representações. A diferenciação entre parâmetros e representações é feita pelo atributo `style`. Caso ele assuma o valor `representation`, indica que trata-se de uma representação. O atributo `value` armazena o valor de um parâmetro, ou o conteúdo de uma representação. Este valor pode ser armazenado pelo atributo `textValue` quando ele puder ser representado em forma de texto (`String`).

Para realizar a interpretação do arquivo WADL, a classe `WADLApplication` utiliza a biblioteca Wadl2Java. Trata-se de uma ferramenta para geração de classes Java a partir de um documento WADL. Entretanto, o Módulo RESTful para a OWL-S API não possui foco na criação de classes Java. Isso acabaria com o perfil dinâmico dos Serviços Web Semânticos, visto que seria necessária compilação das classes geradas. Contudo, para geração das classes, a biblioteca Wadl2Java possui mecanismos internos para interpretação de documentos WADL. Com uma pequena modificação na biblioteca, foi possível a utilização desta para tal interpretação. Com isso, a classe `WADLApplication` pode ser visualizada como uma camada entre a descrição sintática e o Módulo.

A Figura 6.11 exibe os métodos da classe `WADLApplication`. Ela utiliza o padrão de projeto *Factory Method* através do método estático `createApplication()`. Outro método importante dessa classe é o `getResource()`. A partir dos dados extraídos

do documento WADL, tal método cria instâncias da classe `WADLResource` de acordo com o recurso em questão. As instâncias são criadas por demanda e armazenadas dentro da classe `WADLApplication`. O método `getResource()` possui duas assinaturas, sendo que uma define o método de requisição para o recurso requisitado.

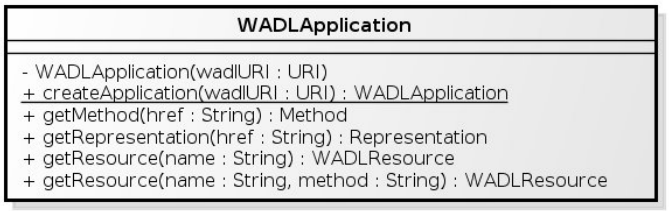


Figura 6.11: *Classe `WADLApplication`.*

A classe `WADLResource` está descrita na Figura 6.12. Os principais métodos desta classe são `invoke()` e `doResponse()`. O primeiro é utilizado pela classe `WADLAtomicGroundingImpl` para invocar o serviço RESTful. Ele é responsável por montar a requisição de acordo com as definições presentes no documento WADL e os dados informados pela `WADLAtomicGroundingImpl`. Este método necessita de um objeto do tipo `WADLRequest` para fazer a requisição. Depois que a requisição é realizada, o método `doResponse()` é chamado. Ele possui o papel de disponibilizar o recurso requisitado de uma forma que possa ser utilizada pela `WADLAtomicGroundingImpl`. Logo, a classe `WADLResource` define uma camada entre o Módulo e o recurso.

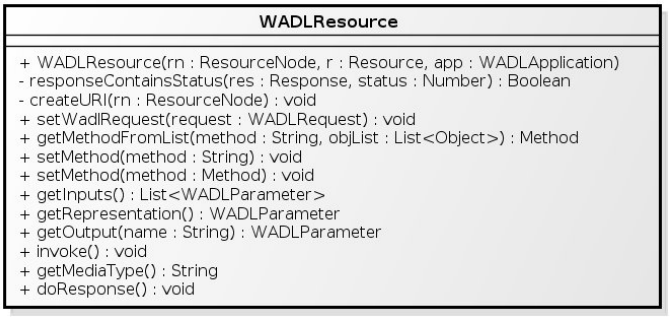


Figura 6.12: *Classe `WADLResource`.*

A interface `WADLRequest` abstrai a requisição do recurso. Os métodos definidos por ela podem ser visualizados na Figura 6.13. O método `request` possui várias assinaturas e é responsável por fazer a requisição. Ele admite qualquer método de requisição HTTP e recebe os parâmetros de requisição através de uma representação (`WADLParameter`). É possível, também, enviar parâmetros de cabeçalho HTTP e o `mediaType` esperado para a representação a ser retornada. A classe `WADLRequestImpl` é uma implementação simplificada desta interface. Entretanto, implementações mais elaboradas podem ser facilmente realizadas.

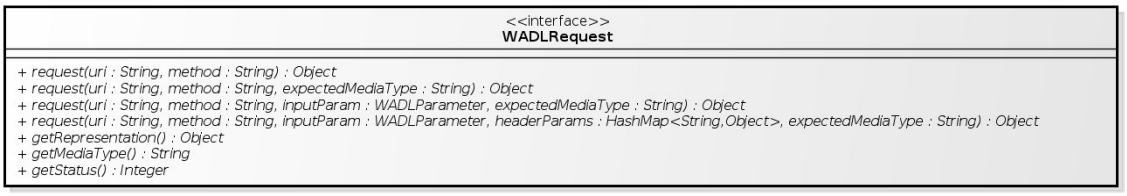


Figura 6.13: *Interface* `WADLRequest`.

Para transformar as representações requisitadas em elementos semânticos é necessária a utilização de um transformador. Quando as representações estão em formato XML é utilizada a linguagem XSL. Como OWL e RDF/XML são baseadas em XML, através de um arquivo escrito em XSL, é possível transformar de XML para OWL ou RDF/XML. A transformação de representações em JSON para formatos semânticos ainda é um problema em aberto.

Para atender o requisito de Tratamento de Exceções, foram utilizadas exceções do Java, com mensagens bem definidas para o usuário. O requisito de Documentação foi satisfeito a partir de comentários (no estilo JavaDoc) no código fonte e através da descrição realizada neste Capítulo. A Figura 6.14 sintetiza a arquitetura do Módulo desenvolvido. A Parte de invocação é dividia em duas. Uma responsável por interpretar o documento WADL e outra por requisitar os recursos. Da fundamentação são extraídos o provedor, que integra o Módulo na API e o vínculo com a ontologia *RESTfulGrounding*.

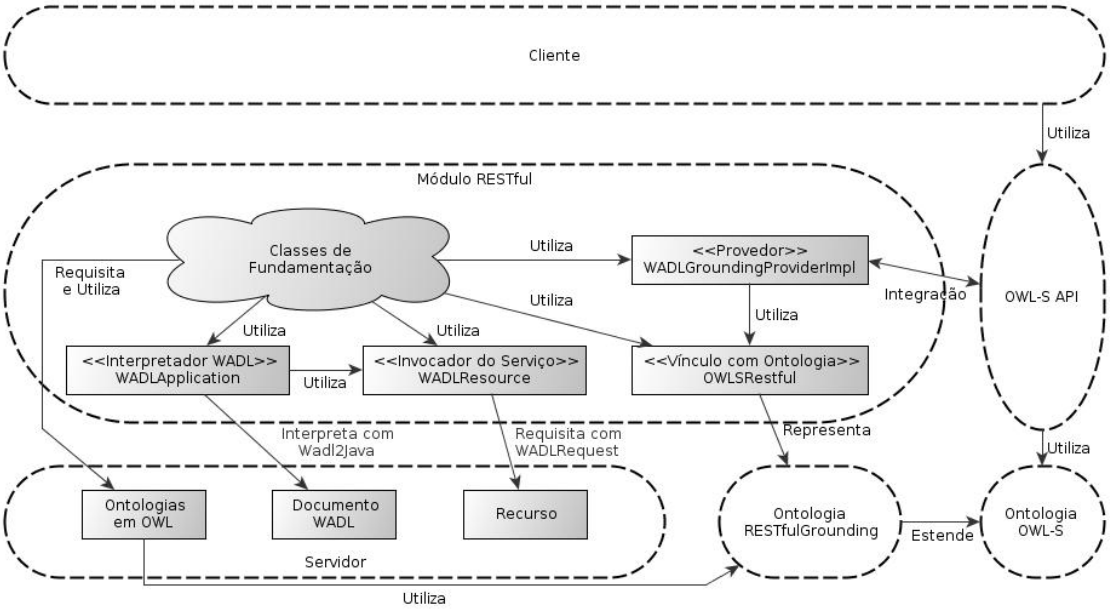


Figura 6.14: *Arquitetura em camadas do Módulo desenvolvido.*

6.3.1 Exemplo de Utilização

A última etapa no desenvolvimento do Módulo consiste na realização de testes para validar seu funcionamento. Para isso, essa seção apresenta um exemplo de utilização do Módulo com Serviços Semânticos RESTful. Como estudo de caso, foi escolhido o serviço de buscas de notícias do Yahoo! visto que este já foi descrito semanticamente em [41]. Foram realizadas algumas adaptações nas ontologias desenvolvidas naquele trabalho, visando maximizar a utilização dos recursos desenvolvidos. Para esse exemplo, serão utilizadas quatro ontologias (`YNS-Service`[2], `YNS-Process`[3], `YNS-Grounding`[4], `YNS-Profile`[5]) e um documento WADL[6].

Foram realizados dois testes com as ontologias disponibilizadas. No primeiro, o serviço foi requisitado sem a realização da descoberta de serviços. Ele foi definido manualmente a partir do seu URI. Este exemplo pode ser visto na Figura 6.15. A escolha do serviço é feita através do método `readService()`, na linha 7. É possível observar que o código exibido para esse exemplo é muito parecido com o código apresentado na Figura 5.6. As únicas diferenças são o URI e os parâmetros de entrada. É possível notar, então, que o Módulo, da forma como foi desenvolvido, é completamente abstraído pela API para o Cliente, ou seja, o Cliente não precisa de manipular nada no Módulo para que ele funcione, apenas na API.

No segundo teste, as linhas de 5 a 8 no código do primeiro teste foram substituídas pelo apresentado na Figura 6.16. Este exemplo é mais elaborado e carrega mais de um serviço na Base de Conhecimento. Através de uma consulta SPARQL, o serviço é selecionado e convertido para a classe `Service` do Java. A partir dai, o procedimento é o mesmo. A consulta SPARQL realizada, seleciona todos os serviços que contenham saída do tipo `ypc:YNS-Output-Type`, onde `ypc` é um pseudônimo para a ontologia YNS-Process. O retorno dos dois testes foi o mesmo e uma parte dele pode ser vista na Figura 6.17.

[2]Ontologia YNS-Service disponível em http://solutio.in/owlsrestful/YNS-Service.owl
[3]Ontologia YNS-Process disponível em http://solutio.in/owlsrestful/YNS-Process.owl
[4]Ontologia YNS-Grounding disponível em http://solutio.in/owlsrestful/YNS-Grounding.owl
[5]Ontologia YNS-Profine disponível em http://solutio.in/owlsrestful/YNS-Profile.owl
[6]Documento WADL disponível em http://solutio.in/owlsrestful/YNS.wadl

```
1   ProcessExecutionEngine exec = OWLSFactory.createExecutionEngine();
2   exec.addMonitor(new DefaultProcessMonitor());
3   final OWLKnowledgeBase kb = OWLFactory.createKB();
4
5   // Carregando ontologias que descrevem o serviço
6   URI uri = new URI("http://solutio.in/owlsrestful/YNS-Service.owl");
7   Service service = kb.readService(uri);
8   Process process = service.getProcess();
9
10  // Definindo entradas
11  ValueMap<Input, OWLValue> inputs = new ValueMap<Input, OWLValue>();
12  inputs.setValue(process.getInput("YNS-AppID"), kb.createDataValue("YahooDemo"));
13  inputs.setValue(process.getInput("YNS-Query"), kb.createDataValue("Brasil"));
14
15  // Criando conjunto de saídas e executando processo
16  ValueMap<Output, OWLValue> outputs = new ValueMap<Output, OWLValue>();
17  outputs = exec.execute(process, inputs, kb);
18
19  // Exibindo a saída em String
20  final OWLValue outputValue = outputs.getValue(process.getOutput());
21  System.out.println(outputValue.toString());
```

Figura 6.15: *Exemplo de código Java para requisição e execução de um Serviço Semântico através da OWL-S API, utilizando o Módulo RESTful.*

```
1   kb.read(new URI("http://solutio.in/owlsrestful/YNS-Service.owl"));
2   kb.read(new URI("http://www.ai.sri.com/daml/services/owl-s/1.2/BravoAirService.owl"));
3
4   String query =  "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
5                   "PREFIX service: <http://www.daml.org/services/owl-s/1.2/Service.owl#>" +
6                   "PREFIX profile: <http://www.daml.org/services/owl-s/1.2/Profile.owl#>" +
7                   "PREFIX process: <http://www.daml.org/services/owl-s/1.2/Process.owl#>" +
8                   "PREFIX ypc: <http://solutio.in/owlsrestful/YNS-Process.owl#>" +
9                   "SELECT ?service" +
10                  "WHERE {" +
11                  "       ?service service:presents       ?profile." +
12                  "       ?profile rdf:type               profile:Profile." +
13                  "       ?profile profile:hasOutput      ?output." +
14                  "       ?output  process:parameterType  ypc:YNS-Output-Type." +
15                  "}";
16  ClosableIterator<ValueMap<Variable, OWLValue>> results = kb
17                  .makeQuery(query, QueryLanguage.SPARQL).execute(null);
18
19  if(results.hasNext()) {
20          service = results.next().getIndividualValue("service").castTo(Service.class);
21          process = service.getProcess();
22  }
```

Figura 6.16: *Exemplo de código Java para realização da descoberta de Serviços Semânticos RESTful através da OWL-S API, utilizando o Módulo RESTful.*

```
<?xml version="1.0" encoding="UTF-8"?>
<ResultSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="urn:yahoo:yn"
        xsi:schemaLocation="urn:yahoo:yn http://api.search.yahoo.com/NewsSearchService/V1/..."
        totalResultsAvailable="28064" totalResultsReturned="10"
        firstResultPosition="1">
        <Result>
                <Title>Brasil vence no futebol dos Jogos Mundiais Militares</Title>
                <Summary>O Brasil obteve mais uma vitória no futebol masculino nos (...)</Summary>
                <Url>http://br.noticias.yahoo.com/brasil-vence-futebol-dos-jogos-mundiais...</Url>
                <ClickUrl>http://br.noticias.yahoo.com/brasil-vence-futebol-dos-jogos...</ClickUrl>
                <NewsSource>Agência Estado via Yahoo! Brasil Notícias</NewsSource>
                <NewsSourceUrl>http://br.noticias.yahoo.com/</NewsSourceUrl>
                <Language>pt</Language>
                <PublishDate>1311120900</PublishDate>
                <ModificationDate>1311121219</ModificationDate>
        </Result>
        <!-- Outras notícias foram suprimidas. -->
</ResultSet>
```

Figura 6.17: *Fragmento da representação do recurso requisitado nos testes.*

6.4 Discussão

A partir dos conceitos mostrados nos Capítulos 2 e 3, foi desenvolvido o Módulo RESTful para a OWL-S API, baseando-se na implementação realizada no Capítulo 5. Este Capítulo mostrou a análise e projeto de tal Módulo. Além disso, foram descritas as principais partes desenvolvidas, validando os requisitos levantados e seguindo os diagramas criados. Os testes realizados validaram as funcionalidades esperadas para o Módulo. A preocupação em abstrair o Módulo RESTful para o Cliente foi vantajosa, visto que, não há mudanças na implementação já existente da OWL-S API e no uso desta.

O Módulo RESTful da OWL-S API torna possível a descrição semântica de serviços RESTful descritos sintaticamente em WADL. Através dele, é garantida a descoberta, composição e invocação dinâmica e automatizada de tais serviços. Dessa forma, é aberta a possibilidade de descrição semântica de APIs de Redes Sociais *Online*. Através dessa abordagem, os serviços presentes em Redes Sociais *Online* poderão ser invocados automaticamente e os recursos compostos dinamicamente (possivelmente integrando-se dados de diferentes Redes Sociais). O Capítulo a seguir aborda a utilização de Serviços Semânticos RESTful em APIs de Redes Sociais *Online*.

CAPÍTULO 7

Descrevendo Semanticamente APIs de Redes Sociais *Online*

A partir do Módulo RESTful para a OWL-S API, desenvolvido no Capítulo 6, é possível descrever semanticamente APIs de Redes Sociais *Online*. Como visto no Capítulo 4 (Seção 3.6), existem algumas abordagens, ainda pouco maduras, para descrição Semântica de Serviços RESTful. Entretanto, nenhuma abordagem atual utiliza padrões bem definidos e já utilizados para descrição semântica e sintática de Serviços Web. A abordagem proposta nessa pesquisa utiliza dois padrões já bem conhecidos na comunidade Web (OWL-S e WADL).

Este Capítulo destina-se a utilização da OWL-S API, através do Módulo RESTful, na descrição semântica de APIs de Redes Sociais *Online*. Também é dado um enfoque em sistemas multiagentes, que (como visto na Seção 2.5) possibilitam maior escalabilidade da aplicação e introduzem um comportamento inteligente a ela. Para isso, foi utilizado o Airetama [1]. Ele é um arcabouço baseado em sistemas multiagentes que fornece uma infra-estrutura semântica e distribuída para a criação de Comunidades Virtuais de Prática. Para o contexto desta pesquisa, estas comunidades podem ser consideradas como Redes Sociais *Online* que visam a colaboração entre membros que exercem uma atividade em comum.

7.1 Redes Sociais *Online*

Sites de redes sociais podem ser definidos como serviços, baseados na Web, que permitem aos seus usuários [19]:

1. Construir um perfil público ou semi-público em um sistema delimitado;
2. Elaborar uma lista de amigos com quem eles compartilham uma conexão e
3. Ver e navegar pela sua lista de conexões e pelas de outras pessoas.

A natureza e a nomenclatura dessas conexões podem variar de acordo com o site. Apesar de os *sites* de rede social atuais implementarem uma série de funcionalidades, sua

espinha dorsal consiste em perfil público para cada usuário que mostra seus dados e a lista de conexões dele [19]. Após cadastrar-se em um site de rede social, é exibido um formulário ao usuário, com perguntas sobre ele. O perfil é formado a partir das respostas do usuário que podem incluir idade, localização, interesses, uma foto e um texto auto-descritivo do usuário.

Depois que o usuário cria seu perfil, o próximo passo em um site de rede social é identificar os usuários já cadastrados que ele deseja ter como conexões. As nomenclaturas mais comuns para as conexões são "Amigos", "Contatos", "Fãs"ou "Seguidores". Tais conexões podem ser bidirecionais ou unidirecionais. Conexões bidirecionais requerem que os dois indivíduos a aceitem, já as unidirecionais não. As nomenclaturas mais comuns para as conexões unidirecionais são "Seguidores"e "Fãs". É comum os *sites* de rede social possuírem também álbum de fotos e uma área em que os usuários podem se comunicar através de mensagens de texto [19].

Como os *sites* de rede social incentivam os usuários a colocarem dados pessoais e íntimos, pessoas mal intencionadas podem usar tais dados para violar a segurança dos usuários. Para aliviar esse problema, alguns *sites* de rede social adotaram níveis de privacidade. Os usuários têm a opção de escolher, por exemplo, que apenas seus contatos vejam seus dados pessoais.

7.1.1 Histórico

De acordo com a definição acima, o primeiro site de rede social surgiu em 1997. O SixDegrees.com permitia que seus usuários cadastrassem perfis e montassem uma lista de amigos, com perfis de outros usuários. A partir de 1998, os usuários do SixDegrees.com podiam navegar nas listas de amigos de outros usuários. Muitos *sites* já implementavam alguma dessas funcionalidades antes do SixDegrees.com. Entretanto, ele foi o primeiro a combiná-las [19]. O SixDegrees.com promovia-se como sendo uma ferramenta para ajudar pessoas a se conectarem e enviar mensagens para outras. Enquanto ele atraia milhões de usuários, começou a tornar-se um negócio inviável. Em 2000 o serviço foi fechado [19].

De 1997 a 2001, várias ferramentas de comunidade que combinavam perfis e listas públicas de amigos foram lançadas. Uma delas foi o LiveJournal [74], lançado em 1999. Ele permite que os usuários criem "jornais pessoais". Cada pessoa pode montar uma página com as notícias de seu interesse. O LiveJournal permite conexões unidirecionais. Um usuário recebe as notícias dos usuários em sua lista de conexões. Outros *sites*, implementaram características de rede social após seu surgimento. A exemplo o coreano Cyworld, fundado em 1999 e que implementou funcionalidades de redes sociais em 2001 e o sueco LunarStorm, que se tornou um site de rede social em 2000 [19].

A próxima geração de *sites* de rede social veio com o Ryze.com, o Fotolog e o Friendster. Ryze.com, criado em 2001, ajudava as pessoas a montar redes sociais de negócios. O Fotolog surgiu em 2002, com uma ideia semelhante a de um blog, entretanto com fotos como tema central de cada artigo. O Friendster, criado em 2002, surgiu como um complemento ao Ryze. Enquanto todos os *sites* sociais da época focavam em encontros amorosos entre estranhos, o Friendster focava em amigos de amigos (*friends-of-friends*). Seus criadores Jonathan Abrams e Cris Emmanuel assumiram que encontros amorosos em que o casal possuia um amigo em comum, eram mais bem sucedidos [19]. Entretanto, os usuários do Friendster também o usavam para aumentar as suas redes de amigos mais rápido que com encontros presenciais [93]. Com seu sucesso, em apenas um ano o Friendster já possuía mais de 30 milhões de usuários [19]. Contudo, tamanho sucesso acabou acarretando dificuldades técnicas, gerando uma evasão de usuários.

A partir de 2003, redes sociais tornaram-se comuns em *sites* da Web, como pode ser visto na Figura 7.1. *Sites* como LinkedIn, Visible Path e Xing formam redes sociais profissionais. No LinkedIn, por exemplo, os usuários podem cadastrar em seu perfil um currículo acadêmico e profissional. Tal currículo pode ser usado por empresas na seleção de profissionais. No Couchsurfing os usuários compartilham hospedagens. Um usuário pode oferecer hospedagem a outro usuário que deseja viajar para a sua cidade. O MyChurch conecta igrejas cristãs e seus membros. Alguns *sites* usam os conceitos de redes sociais *online* para compartilhamento de conteúdo multimídia. A exemplo, o Flickr para compartilhamento de fotos, o YouTube para vídeos e o Last.FM para músicas [19].

O MySpace, criado em 2003, cresceu rapidamente, como sendo uma alternativa ao então saturado Friendster. Apesar de ter sido criado com propósito similar ao do Friendster, o MySpace ganhou popularidade entre bandas de rock. Até 2004, a maioria dos seus usuários eram músicos. Em 2005, ele tornou-se o maior site de rede social (em número de usuários e visitantes) do mundo [19]. O Orkut, criado em 2004, é a rede social do Google. Desde sua criação, ganhou popularidade entre os brasileiros, tornando-se a rede social mais usada no Brasil [44].

O Facebook, criado em 2004, possuía um foco diferente de seus precursores. Ele foi criado para a formação de redes em faculdades específicas. Inicialmente, estava disponível apenas para os estudantes de Harvard. Em 2005, ele se expandiu para escolas de ensino médio, posteriormente para profissionais e redes corporativas e atualmente para qualquer pessoa. O principal diferencial do Facebook desde sua criação, era a API que possibilitava aos usuários desenvolver suas próprias aplicações embutidas à ele [19].

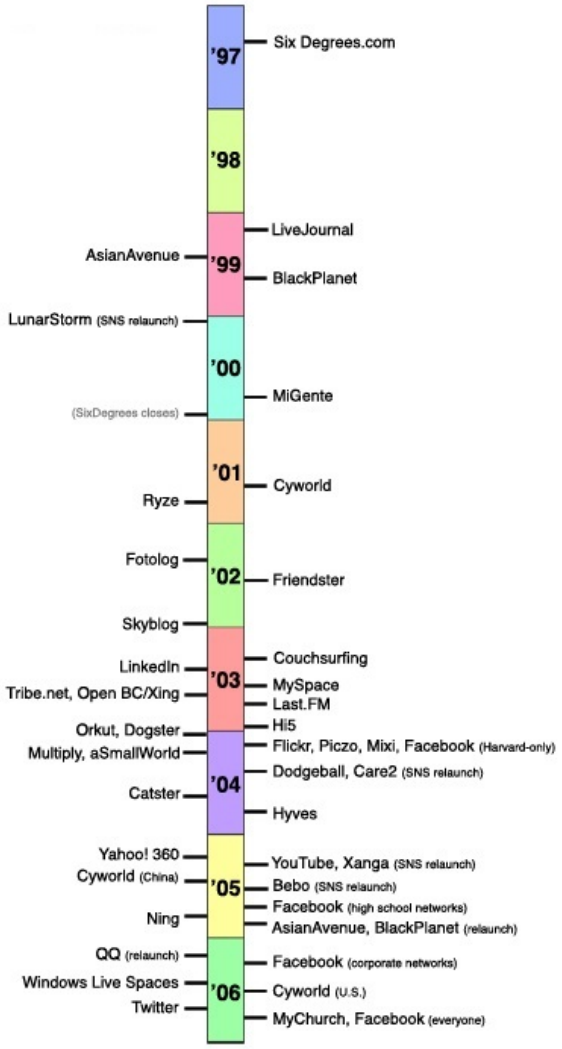


Figura 7.1: *Linha do tempo do surgimento dos principais sites de rede social até 2006. [19]*

Em 2006, o Twitter foi criado, gerando um novo conceito: o *microblogging*. Os usuários escrevem pequenas mensagens (de até 140 caracteres), sobre algo que está sentindo ou acontecendo. As conexões são unidirecionais. Apenas os usuários que estão conectados a uma pessoa, recebem as mensagens dela [69]. Com esse novo conceito, o Twitter cresceu rapidamente, tornando-se um novo meio de comunicação. Atualmente ele é usado para diversos fins como publicação de notícias, sorteios, venda de produtos e reuniões entre outros.

Como o Facebook, o Twitter foi criado com uma vasta API para criação de aplicações pelos próprios usuários. A partir do sucesso das APIs do Facebook e do Twitter, em 2007 o Google lançou o OpenSocial [60]. Um conjunto de APIs para criação de aplicações Web em *sites* de redes sociais. *Sites* como Orkut, MySpace e LinkedIn são compatíveis com o OpenSocial para criação de aplicações. Para facilitar a integração ao OpenSocial, a Apache desenvolveu o Shindig [43], um *container open-source* para

hospedagem de aplicativos em OpenSocial.

7.2 O Arcabouço Airetama

O arcabouço Airetama é de código fonte aberto. Para garantir as características multiagente e semântica, ele utiliza o JADE e o Jena, respectivamente, vistos no Capítulo 2. Sua arquitetura é baseada em três sociedades de agentes, como pode ser visto na Figura 7.2. A sociedade de membros representa os usuários do sistema e suas comunidades. Os agentes da sociedade de controle são responsáveis pelo gerenciamento de todos os agentes. A sociedade de ferramentas possui agentes de terceiros que utilizam as características semântica e multiagentes do arcabouço. Neste Capítulo, será criado um agente na sociedade de ferramentas que utiliza a API de outra Rede Social *Online*, visando integração de dados.

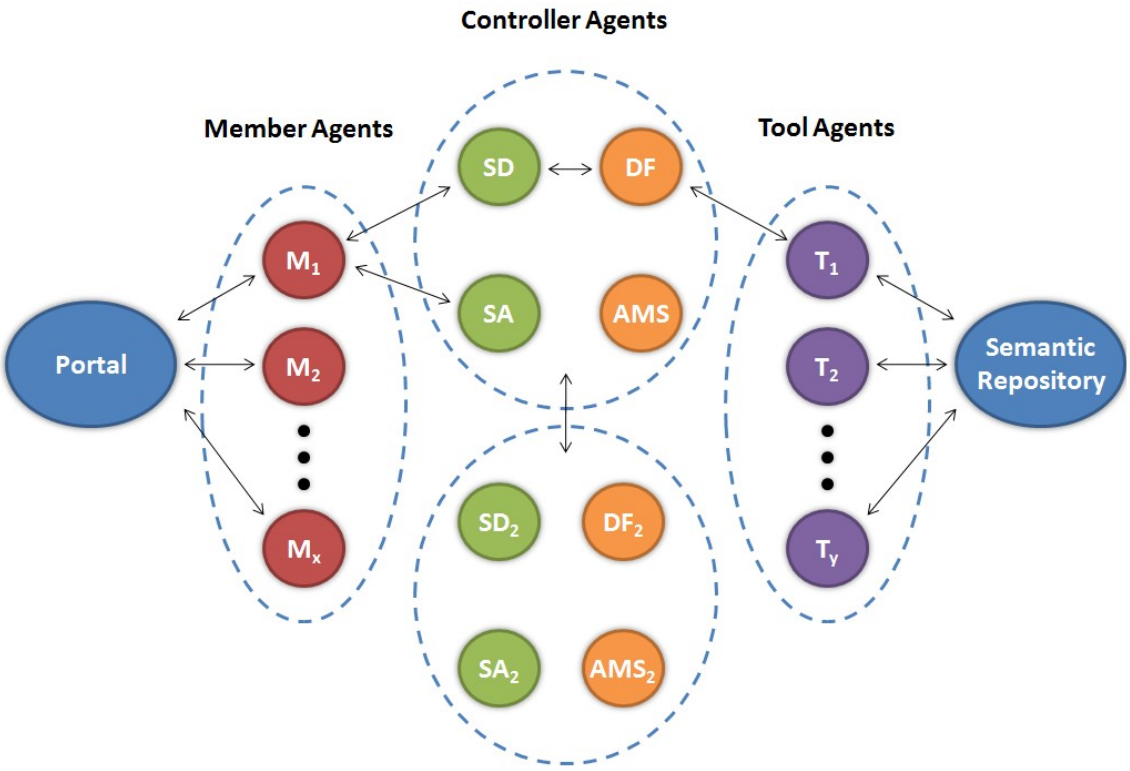


Figura 7.2: *Arquitetura do Arcabouço Airetama [1].*

O Airetama ainda é composto por um portal e um repositório semântico. O portal é utilizado para interação com os usuários do arcabouço. Pelo fato de ser focado em comunidades virtuais de práticas, e ter um portal de fácil utilização, o arcabouço Airetama também pode ser considerado uma Rede Social Online, com características da Web Semântica. O repositório semântico é responsável por armazenar as triplas em RDF e as definições de ontologias em OWL, dos elementos utilizados pelas sociedades

multiagentes. A utilização de raciocinadores e o trabalho com inferências pode ser feito neste repositório. Nota-se então que o arcabouço Airetama ajuda a satisfazer os requisitos não funcionais de Semântica, Paralelismo e Flexibilidade.

Para a descrição semântica dos usuários e suas comunidades, o arcabouço Airetama utiliza as ontologias FOAF[22] e SIOC[21]. Tais ontologias são bem difundidas na comunidade da Web Semântica, principalmente a FOAF. O agente construído neste Capítulo, chamado de *Facebook Integrator*, utiliza a API do Facebook para requisitar os perfis dos usuários e armazená-los no Airetama. Foi feita a integração entre as duas Redes Sociais e os perfis passaram a ser descritos semanticamente através das características providas pelo Airetama. Foi escolhido o Facebook por este possuir uma API robusta e por ser a Rede Social *Online* mais utilizada e com maior número de usuários, atualmente.

7.3 Descrição Sintática e Semântica de um Recurso da API do Facebook

Como ainda é comum em APIs de Redes Sociais *Online*, a API do Facebook não disponibiliza a descrição sintática dos serviços em WADL. Entretanto, existe uma corporação chamada APIGEE[1] que disponibiliza serviços para usuários e desenvolvedores de APIs. Esta corporação possui um projeto para descrição sintática em WADL da maioria das APIs populares, chamado `wadl-library`[2]. A maioria dos serviços da API do Facebook já foi descrita por este projeto. Com isso, foram utilizados os documentos WADL disponibilizados por este.

Inicialmente, foram realizados alguns testes através da linguagem FQL, provida pelo Facebook (descrita na Seção A.1). Apesar de admitir consultas mais elaboradas, essa linguagem anula alguns princípios da arquitetura REST. Os recursos não podem ser bem especificados e não possuem endereçamento através de URIs, por exemplo. Logo, optou-se por utilizar a *Facebook Graph API* que atende aos princípios REST. A partir dessa API foram criadas as ontologias para descrição semântica dos serviços disponibilizados pelo Facebook. A primeira ontologia criada foi a `FBUserResource`[3]. Esta ontologia descreve semanticamente o serviço que disponibiliza dados de um usuário do Facebook a partir de seu código identificador ou *username*[4].

A ontologia `FBUserResource` descreve a representação do recurso *User* do Facebook através da classe ontológica `FBUser`, como pode ser visto na Figura 7.3.

[1] Site da corporação APIGEE: `http://apigee.com`.
[2] Projeto Apigee wadl-library disponível em `https://github.com/apigee/wadl-library`.
[3] URI para ontologia FBUserResource: `http://solutio.in/owlsrestful/FBUserResource.owl`.
[4] A documentação completa do recurso *User* do Facebook pode ser visualizada em `http://developers.facebook.com/docs/reference/api/user`.

Esta figura exibe a definição da classe ontológica `FBUser` e do processo atômico `FBUserAtomicProcess`. Outros detalhes foram omitidos visando economia de espaço. A classe `FBUser` herda da classe `foaf:Person`, utilizada pelo Airetama. Na elaboração de tal ontologia, foi identificado que a descrição sintática disponibilizada pela APIGEE estava incompleta. Ela não continha as possíveis representações dos recursos e todos os parâmetros das requisições. Logo, foi necessária a criação de um documento WADL[5] adaptado daquele disponibilizado pela APIGEE.

```
<!-- Definição da classe FBUser, filha de foaf:Person -->
<owl:Class rdf:ID="FBUser">
  <rdfs:subClassOf rdf:resource="&foaf;Person"/>
</owl:Class>

<!-- (...) Definição do Processo Atômico para recurso User (...) -->
<process:AtomicProcess rdf:ID="FBUserAtomicProcess">
  <!-- Parametros de Entrada -->
  <process:hasInput rdf:resource="&fb;#FBAccessToken"/>
  <process:hasInput rdf:resource="&fb;#FBUserId"/>
  <!-- Parametros de Saída -->
  <process:hasOutput>
    <process:Output rdf:ID="FBUserOut">
      <process:parameterType rdf:datatype="&xsd;#anyURI">&fb;#FBUser</process:parameterType>
    </process:Output>
  </process:hasOutput>
</process:AtomicProcess>
```

Figura 7.3: *Partes da definição da ontologia `FBUserResource` com classe `FBUser`.*

7.4 Transformação de Representações em JSON para Formatos Semânticos

Os serviços disponibilizados pela API do Facebook retornam representações dos recursos em JSON. É um desafio em aberto a transformação de representações em JSON para formatos semânticos (como por exemplo triplas RDF). O trabalho relacionado SEREDASj[70] apresenta uma forma de descrever representações em JSON possibilitando a transformação destas para formatos semânticos. Para isso, são utilizados documentos JSON que descrevem os elementos de uma determinada representação em JSON. Entretanto, a solução apresentada por este trabalho ainda encontra-se em fase de desenvolvimento e não foi padronizada.

[5] O documento WADL para descrição sintática dos recursos da API do Facebook utilizados está disponível em http://solutio.in/owlsrestful/facebook.wadl.

A presente pesquisa propõe uma nova abordagem para a transformação de representações em JSON para formatos ontológicos. A solução consiste em utilizar o mesmo padrão utilizado para realização desta transformação em representações XML. Entretanto, a linguagem XSL/XSLT só admite a conversão entre arquivos XML. Logo, a proposta é transformar a representação JSON em XML e depois aplicar a transformação XSL. Para esta transformação foi utilizada a JSON API[30]. A Figura 7.4 mostra um exemplo de representação em JSON. Na Figura 7.5, é exibido o documento XML resultante da transformação proposta.

```
{
   "id": "1820391700",
   "name": "Ot\u00e1vio Cala\u00e7a",
   "first_name": "Ot\u00e1vio",
   "last_name": "Cala\u00e7a",
   "link": "http://www.facebook.com/otaviocx",
   "username": "otaviocx",
   "email": "otaviocx@gmail.com",
   "gender": "male",
   "locale": "pt_BR"
}
```

Figura 7.4: *Exemplo de representação de um usuário do Facebook em JSON.*

```
<?xml version="1.0" encoding="UTF-8"?>
<json>
        <id>1820391700</id>
        <name>Otávio Calaça</name>
        <first_name>Otávio</first_name>
        <last_name>Calaça</last_name>
        <link>http://www.facebook.com/otaviocx</link>
        <username>otaviocx</username>
        <email>otaviocx@gmail.com</email>
        <gender>male</gender>
        <locale>pt_BR</locale>
</json>
```

Figura 7.5: *Documento XML correspondente ao exemplo na Figura 7.4, transformado através da JSON API.*

Uma vez que a representação em JSON foi transformada para XML, basta aplicar a transformação XSL. A Figura 7.6 exibe como exemplo partes de um arquivo XSLT para realização de tal transformação. Na Figura 7.7, é possível visualizar a representação no formato semântico. A Figura 7.8 sintetiza o processo de transformação da representação em JSON para formatos ontológicos. A partir desse processo foi possível representar

os usuários do Facebook semanticamente e armazená-los no repositório semântico do Airetama.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

        <xsl:template match="json">
                <rdf:RDF>
                        <foaf:Person>
                                <xsl:attribute name="rdf:about">
                                        <xsl:text>http://graph.facebook.com/</xsl:text>
                                        <xsl:value-of select="id" />
                                </xsl:attribute>
                                <foaf:firstName>
                                        <xsl:value-of select="first_name" />
                                </foaf:firstName>
                                <foaf:lastName>
                                        <xsl:value-of select="last_name" />
                                </foaf:lastName>
                                <!-- Outras propriedades suprimidas -->
                                <rdfs:seeAlso>
                                        <xsl:value-of select="link" />
                                </rdfs:seeAlso>
                        </foaf:Person>
                </rdf:RDF>
        </xsl:template>
</xsl:stylesheet>
```

Figura 7.6: *Parte do documento XSLT para transformação do XML da Figura 7.5 em RDF/XML.*

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

        <foaf:Person rdf:about="http://graph.facebook.com/1820391700">
                <foaf:firstName>Otávio</foaf:firstName>
                <foaf:lastName>Calaça</foaf:lastName>
                <foaf:gender>male</foaf:gender>
                <foaf:account>otaviocx</foaf:account>
                <foaf:mbox>otaviocx@gmail.com</foaf:mbox>
                <rdfs:seeAlso>http://www.facebook.com/otaviocx</rdfs:seeAlso>
        </foaf:Person>
</rdf:RDF>
```

Figura 7.7: *Representação em RDF de um usuário do Facebook requisitado em JSON, conforme Figura 7.4.*

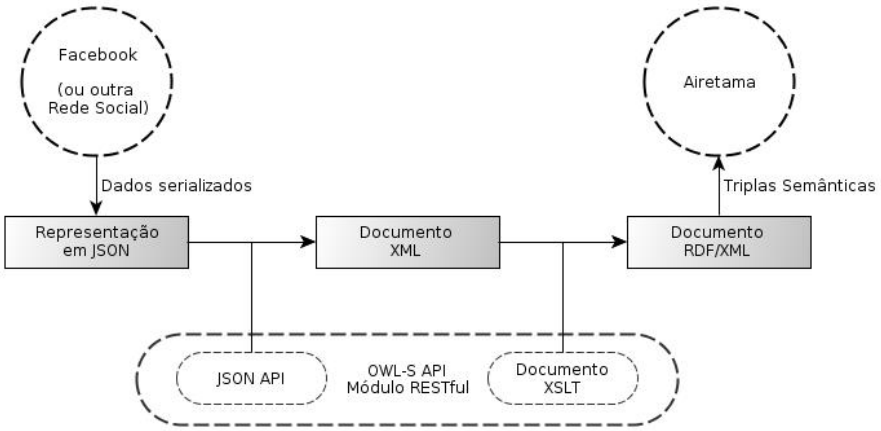


Figura 7.8: *Processo de Transformação de Representação em JSON para Triplas RDF.*

Para tornar possível a transformação automatizada da representação em JSON para formatos ontológicos, foi necessário estender a ontologia *RESTfulGrounding*. Foi criada a classe ontológica `TransformationFileMap` e vinculada a um atributo da classe `WadlMessageParamMap`. Esta nova classe pode ser visualizada na Figura 7.9. A classe possui duas propriedades, `transformationURI` identifica o URI do arquivo para realização da transformação. A propriedade `transformationMediaType` indica qual o formato do arquivo para transformação (no caso `application/xslt+xml`). Caso surja outra abordagem com outro formato (como é o caso da abordagem em [70]), será possível utilizá-la sem alterações na ontologia.

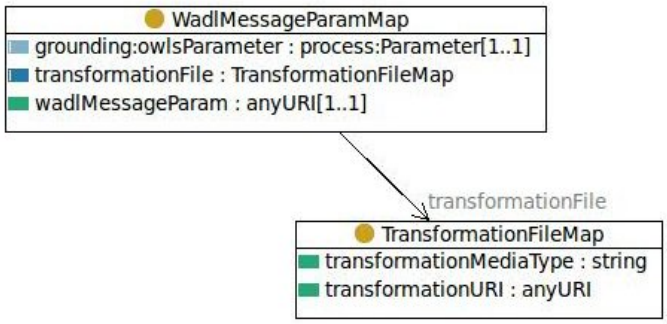


Figura 7.9: *Nova classe `TransformationFileMap` para a ontologia RESTfulGrounding e seu relacionamento com `WadlMessageParamMap`.*

Como visto na Seção 6.3, grande parte das classes especializadas para fundamentação em WADL, vinculam um parâmetro de modelo em sua especialização. Isso é necessário para informar para a API qual é o tipo do transformador para aquela fundamentação. Com a nova classe ontológica `TransformationFileMap`, foi criada uma classe Java de mesmo nome e o vínculo na especialização foi alterado de `String` para `TransformationFileMap`. Esta classe, bem como as representações atualizadas de

`WadlRequestParamMap` e `WadlResponseParamMap`, foram registradas no provedor para realização da conversão.

7.5 Descrição e Requisição de Outros Recursos

O recurso *User* do Facebook, requisitado através da ontologia `FBUserResource`, representa apenas um usuário do Facebook. Entretanto, a partir de um usuário é possível requisitar seus amigos. Isso é feito através do recurso *Friends* do Facebook, que retorna os nomes e códigos de todos os amigos de um usuário. Para descrição do serviço que disponibiliza esse recurso foi criada a ontologia `FBFriendsResource`[6]. Como os dois serviços possuem coisas em comum (como os parâmetros de requisição), foi criada a ontologia `FBCommon`[7] que descreve as características em comum dos dois serviços e seus recursos. Para relacionar os usuários como amigos, foi utilizada a propriedade `foaf:knows`, como pode ser visto na Figura 7.10.

Com base nas ontologias `FBUserResource` e `FBFriendsResource` foi criada uma nova ontologia, a `FBUserFriendsResource`[8]. Essa ontologia descreve um serviço semântico que possui dois processos compostos. Um baseado no construtor de controle `Sequence`, que requisita a lista de amigos de um usuário e em seguida envia-a para o segundo processo composto. Este, por sua vez, é baseado no construtor de controle `For-Each`, uma implementação de `Iterate`. Ele recebe a lista e chama o processo atômico para requisição do recurso *User* do Facebook, para cada amigo na lista. Logo, a partir de um único serviço semântico foi possível requisitar os dados de um usuário e de todos os seus amigos.

```
1 <foaf:Person rdf:about="http://graph.facebook.com/503516337">
2         <foaf:account>503516337</foaf:account>
3         <foaf:knows>
4                 <foaf:Person rdf:about="http://graph.facebook.com/otaviocx" />
5         </foaf:knows>
6 </foaf:Person>
```

Figura 7.10: *Trecho de código RDF/XML que representa o relacionamento entre amigos com FOAF.*

O próximo serviço descrito semanticamente foi o responsável por fornecer o recurso `Likes` do Facebook. Trata-se de uma lista de coisas que o usuário marcou como

[6]URI para FBFriendsResource: http://solutio.in/owlsrestful/FBFriendsResource.owl.
[7]URI para ontologia FBCommon: http://solutio.in/owlsrestful/FBCommon.owl.
[8]FBUserFriendsResource: http://solutio.in/owlsrestful/FBUserFriendsResource.owl.

sendo de seu interesse. A lista é composta pelo nome, categoria, código identificador do item de interesse, bem como a data em que o usuário marcou o item como sendo de seu interesse. A ontologia que descreve semanticamente o serviço que requisita a lista de interesses foi chamada de `FBLikesResource`[9]. Cada item de interesse foi transformado em uma instância da classe ontológica `foaf:Document`. Esta classe relaciona-se com a classe `foaf:Person` a partir da propriedade `foaf:interest`. A Figura 7.11 exibe um exemplo de tal relacionamento.

```
<foaf:Person rdf:about="http://graph.facebook.com/otaviocx">
        <foaf:interest>
                <foaf:Document rdf:about="http://graph.facebook.com/19691681472">
                        <foaf:topic>Michael Jackson</foaf:topic>
                        <foaf:primaryTopic>Musician/Band</foaf:primaryTopic>
                </foaf:Document>
        </foaf:interest>
</foaf:Person>
```

Figura 7.11: *Trecho de código RDF/XML que representa um tópico de interesse de um usuário.*

É importante lembrar que, para utilização dos serviços mostrados acima, é necessária a realização de um processo de autorização pelo usuário. No Facebook, este processo é realizado a partir do protocolo OAuth, como visto no Anexo A. Este processo não pode ser completamente automatizado pois depende da interação com o usuário. Essa etapa, foi realizada manualmente até a obtenção do *token* de acesso. A Figura 7.12 exibe um exemplo da tela que o Facebook mostra ao usuário para requisitar autorização de acesso aos seus dados. Esta tela é exibida apenas na primeira vez em que a aplicação requisita autorização.

[9]URI para FBLikesResource: `http://solutio.in/owlsrestful/FBLikesResource.owl`.

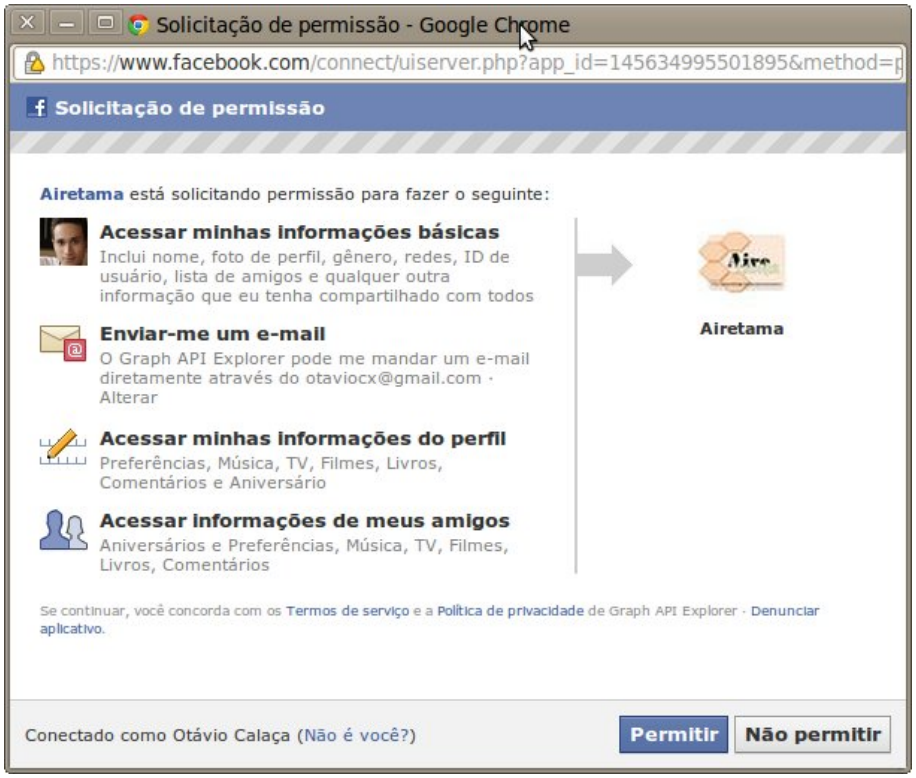


Figura 7.12: *Exemplo de Tela do Facebook para Requisição de Autorização de Acesso.*

7.6 Realizando Consultas Semânticas em Dados Extraídos do Facebook

Através dos serviços semânticos descritos acima, os dados de usuários, seus relacionamentos de amizade, e os tópicos de interesse foram armazenados semanticamente no repositório do Airetama. O conteúdo semântico armazenado no Airetama, admite que consultas SPARQL elaboradas sejam realizadas. Como exemplo, foi construída uma consulta semântica que retorna o número de tópicos de interesse em comum entre diferentes usuários. A Figura 7.13 exibe uma consulta SPARQL que retorna o nome de dois usuários e o número de interesses que eles tem em comum. O resultado é ordenado pelo número de interesses em comum entre os dois usuários, em ordem decrescente.

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
select ?mName ?aName (count( ?interest ) as ?num)
where {
        ?member foaf:name ?mName.
        ?member foaf:interest ?interest.
        ?anotherMember foaf:interest ?interest.
        ?anotherMember foaf:name ?aName.
        FILTER(?member != anotherMember)
}
group by ?mName ?aName
order by desc(?num)
```

Figura 7.13: *Consulta SPARQL que retorna o número de coisas em comum entre dois usuários.*

A Figura 7.14 mostra um grafo com um exemplo de relacionamentos semânticos entre os usuários e seus tópicos de interesse. Ao lado, é exibida uma tabela com os resultados da consulta SPARQL aplicada no grafo. Em uma simulação prática, o agente *Facebook Integrator* foi executado em quatro perfis de usuários do Facebook, cada um com aproximadamente 250 amigos. Foram encontrados casos de usuários com mais de 40 interesses em comum e que não se conheciam. Com isso, torna-se possível inferir possíveis relacionamentos de afinidade entre os usuários, propiciando a colaboração em uma atividade em comum.

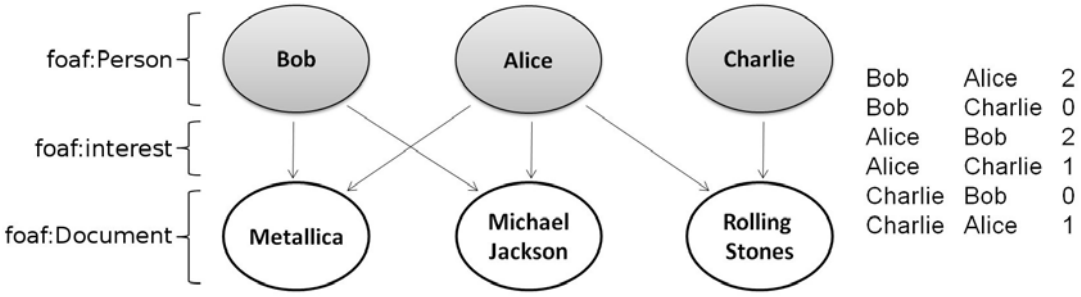


Figura 7.14: *À esqueda, um grafo com os relacionamentos entre pessoas e seus interesses. À direita, uma tabela com o resultado da SPARQL (Figura 7.13) aplicada no grafo.*

CAPÍTULO 8

Considerações Finais

Em 1999, Tim Berners-Lee disse: “Eu tenho um sonho para a Web, em que os computadores irão tornar-se capazes de analisar todos os dados na rede - o conteúdo, links e transações entre pessoas e computadores. ‘A Web Semântica’, que tornará isto possível, ainda não surgiu, mas quando isso acontecer, o dia a dia dos mecanismos de comércio, a burocracia e as nossas vidas diárias serão manipulados por máquinas falando com outras máquinas.”[16]. A realização completa da Web da qual Berners-Lee falava, 13 anos atrás, ainda depende de um considerável esforço da comunidade acadêmica e instituições comerciais.

A Web Semântica, como proposta em 2001 por Tim Berners-Lee, James Hendler e Ora Lassila[99], ainda encontra-se em fase de desenvolvimento e implantação. Como ocorreu com a primeira geração da Web, é comum a pesquisa estar a frente da implementação em alguns anos. Entretanto, no caso da Web Semântica, houve um advento adicional, a Web Social. Esta geração da Web, disponibilizou aos usuários um meio de comunicação mais interativo que qualquer outro até então utilizado. Então, o enfoque para a Web Semântica, comercialmente, foi sendo substituído pela Web Social. Logo, a Web Semântica, permaneceu por muito tempo apenas no meio acadêmico.

Muitas são as pesquisas em torno da Web Semântica. Entretanto, muito poucas chegam a ser implementadas comercialmente. Isso se dá, porque o enfoque principal ainda é a Web Social. Porém, com o crescimento de Redes Sociais *Online*, de *Wikis*, e todas as aplicações sociais que surgiram na Web 2.0, tem sido cada vez mais necessária a utilização dos recursos da Web Semântica. Os Serviços Web Semânticos ganham enfoque neste contexto. A partir deles, é possível vincular as duas gerações da Web em uma Web Semântica Social. Entretanto, a arquitetura predominante de Serviços Web na Web Social (RESTful), não possui ainda mecanismos bem estabelecidos de descrição semântica.

O estudo acerca de Serviços Web Semânticos, especificamente em arquitetura REST, apresentado por este trabalho, conduz a alguns resultados interessantes. A seguir serão abordadas algumas contribuições realizadas por esta pesquisa. Em seguida serão apresentadas algumas limitações da ferramenta desenvolvida neste trabalho. Por fim, são levantados possíveis melhoramentos e desafios ainda em aberto que poderão ser

realizados em trabalhos futuros.

8.1 Contribuições

A principal contribuição deste trabalho para a sociedade está na implementação de descrição semântica de Serviços RESTful, possibilitando a constatação da realização do objetivo principal, e dos secundários, desta pesquisa. A construção de um Módulo RESTful para a OWL-S API foi a principal contribuição tecnológica deste trabalho. O código fonte deste projeto é livre[1] e disponibilizado para alterações por qualquer pessoa. A integração do Módulo desenvolvido com a OWL-S API, para ser disponibilizado em conjunto com ela nas próximas versões, está sendo discutida com os criadores da API.

Este trabalho também possui como contribuição, soluções para alguns trabalhos futuros propostos em [1]. Dentre eles, destaca-se a implementação de Serviços Web Semânticos no Airetama, a partir do desenvolvimento de agentes que consomem serviços de outras redes sociais. Também são abordados alguns trabalhos futuros propostos em [41]. Dentre eles, destaca-se o desenvolvimento de agentes de software que utilizam a ontologia *RESTfulGrounding*, validando as funcionalidades propostas teoricamente por aquele trabalho. Ainda como continuação daquele trabalho, foi colocado em prática um caso de uso para utilização de tal ontologia.

Adicionalmente, foi desenvolvida uma abordagem para transformação de representações em JSON para formatos semânticos. A solução proposta levou em conta padrões já estabelecidos para conversão entre formatos, tornando mais fácil a implantação de tal abordagem. Também foram propostas melhorias na ontologia `RESTfulGrounding` para atender a esta abordagem de transformação de dados sintáticos em ontológicos. Estas mudanças foram apresentadas ao autor da ontologia e serão introduzidas em sua próxima versão.

Por fim, este trabalho também gerou a contribuição de descrever alguns serviços disponibilizados pelo Facebook, sintaticamente e semanticamente. As ontologias construídas foram testadas com o Módulo desenvolvido e validadas. Com isso, validou-se também a descrição sintática no padrão WADL. A descrição clara de como estas ontologias são criadas também é uma contribuição. Ela permite que outras pessoas criem novas ontologias sem muitas dificuldades.

[1] Endereço para o repositório em que o Módulo RESTful da OWL-S API encontra-se: `http://code.google.com/p/ocx-mestrado/`

8.2 Limitações e Trabalhos Futuros

Duas limitações podem ser constatadas no projeto desenvolvido. A primeira é o fato de que a ferramenta desenvolvida utiliza uma tecnologia específica de descrição semântica de serviços Web. Trata-se da abordagem OWL-S. Logo, outras abordagens, como WSMO, não são contempladas. Entretanto, estas abordagens têm sido muito pouco utilizadas e a comunidade parece estar convergindo para o padrão OWL-S. Outra limitação é acerca da fundamentação oferecida pela ontologia. Tal fundamentação necessita de descrição sintática no padrão WADL. Outros padrões, como hRESTS, não são contemplados. Entretanto, a conversão entre padrões é possível.

Como trabalhos futuros são propostas, inicialmente, a elaboração de soluções para as limitações identificadas. Através de mecanismos de transformação entre formatos, como o XSLT, é possível transformar as descrições sintáticas criadas em outros formatos para WADL. Outro trabalho futuro a ser realizado trata-se da elaboração de um mecanismo para automatização da autenticação e autorização de acesso em APIs de Redes Sociais *Online*. Para isso, pode ser utilizada a abordagem proposta em [78].

Outro trabalho que pode ser realizado futuramente é a implementação do padrão SAWSDL para melhor descrição semântica das entradas e saídas dos serviços. Apesar deste padrão ser focado em WSDL ele pode ser utilizado com outros formatos como o WADL. A utilização deste padrão é interessante visto que ele é uma recomendação do W3C. Adicionalmente, também há a possibilidade de realizar melhorias para que um recurso ontológico corresponda a mais de um recurso sintático. Isso significaria utilizar mais de um WADL para um único processo atômico OWL-S.

Também poderão ser construídos, no futuro, processos compostos que envolvam recursos de diferentes Redes Sociais *Online*. A partir disso, será possível validar a utilização de Serviços Web Semânticos para integração de Redes Sociais *Online*. Por fim, indexadores semânticos (também chamados de *Crawlers*, *Robots* ou *Spiders*) poderiam ser utilizados com Serviços Web Semânticos. Tais indexadores poderiam processar as páginas que descrevem os serviços RESTful para seres humanos, gerando automaticamente a descrição sintática de tais serviços.

Referências Bibliográficas

[1] ALARCÓN, J. A. B. L. **Airetama - Um Arcabouço Baseado em Sistemas Multiagentes para a Implantação de Comunidades Virtuais de Prática na Web**. Master's thesis, Universidade Federal de Goiás, 2010.

[2] ALARCÓN, R.; WILDE, E. **Linking Data from RESTful Services**. *Linked Data on the Web (LDOW) 2010*, 2010.

[3] ALEXANDER, K. **RDF/JSON: A Specification for serialising RDF in JSON**. In: *Scripting for the Semantic Web (SFSW)*, 2008.

[4] ALLSOPP, J. **Microformats - Empowering Your Markup for Web 2.0**. Springer, 2007.

[5] ALOWISHEQ, A.; MILLARD, D. E. **EXPRESS: EXPressing REstful Semantic Services**. In: *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on*, volume 3, p. 453 –456, sept. 2009.

[6] ANDREESSEN, M. **NCSA Mosaic Technical Summary**. *National Center for Supercomputing Applications*, 1993.

[7] ANTONIOU, G.; HARMELEN, F. V. **Web Ontology Language: OWL**. In: *Handbook on Ontologies in Information Systems*, p. 67–92. Springer, 2003.

[8] ANTONIOU, G.; VAN HARMELEN, F. **A Semantic Web Primer**. The MIT Press, Cambridge, Massachusetts, 2 edition, 2008.

[9] BELL, G. **Building Social Web Applications**. O'Reilly, Beijing, 2009.

[10] BELLIFEMINE, F. L.; CAIRE, G.; GREENWOOD, D. **Developing Multi-Agent Systems with JADE**. Wiley, 2007.

[11] BELLWOOD, T.; CAPELL, S.; CLEMENT, L.; COLGRAVE, J.; DOVEY, M. J.; FEYGIN, D.; HATELY, A.; KOCHMAN, R.; MACIAS, P.; NOVOTNY, M.; PAOLUCCI, M.; VON RIEGEN, C.; ROGERS, T.; SYCARA, K.; WENZEL, P.; WU, Z. **UDDI Version 3.0.2**. OASIS, Oct. 2004.

[12] BERNERS-LEE, T. **Information Management: A Proposal**. *CERN*, 1989.

[13] BERNERS-LEE, T. **The WorldWideWeb browser**. http://www.w3.org/People/Berners-Lee/WorldWideWeb, último acesso em Maio de 2011, 2004.

[14] BERNERS-LEE, T.; CAILLIAU, R.; LUOTONEN, A.; NIELSEN, H. F.; SECRET, A. **The World-Wide Web**. *Communications of the ACM*, 37(8):76–82, 1994.

[15] BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. **Uniform Resource Identifier (URI): Generic Syntax**. The Internet Society, 2005.

[16] BERNERS-LEE, T.; FISCHETTI, M. **Weaving The Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor**. Harper, San Francisco, 1999.

[17] BÖRGER, E.; RICCOBENE, E.; SCHMID, J. **Capturing Requirements by Abstract State Machines: The Light Control Case Study**. *Journal of Universal Computer Science*, 6(7), 2000.

[18] BOUROS, P. **Semantic Web Services: A conceptual comparison of OWL-S, WSMO and METEOR-S approaches**. Technical report, Department of Informatics and Telecommunications - National and Kapodistrian University of Athens (NKUA), Panepistimiopolis, T.Y.P.A. Buildings, GR-157 84 Ilisia, Athens, Greece, 2005.

[19] BOYD, D. M.; ELLISON, N. B. **Social Network Sites: Definition, History, and Scholarship**. *Journal of Computer-Mediated Communication*, 13(11), 2007.

[20] BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M.; MALER, E. **Extensible Markup Language (XML) 1.0 (Fifth Edition)**. http://www.w3.org/TR/REC-xml, último acesso em Março de 2011, 2008.

[21] BRESLIN, J.; BOJARS, U.; PASSANT, A.; FERNÁNDEZ, S.; DECKER, S. **SIOC: Content Exchange and Semantic Interoperability Between Social Networks**. In: *W3C Workshop on the Future of Social Networking*, 2009.

[22] BRICKLEY, D.; MILLER, L. **FOAF Vocabulary Specification 0.98**. Namespace document, FOAF Project, 2010.

[23] BURKE, B. **RESTful Java with Jax-RS**. O'Reilly Media, Inc., 2009.

[24] BURSTEIN, M.; HOBBS, J.; LASSILA, O.; MCDERMOTT, D.; MCILRAITH, S.; NARAYANAN, S.; PAOLUCCI, M.; PARSIA, B.; PAYNE, T.; SIRIN, E.; SRINIVASAN, N.; SYCARA, K. **OWL-S: Semantic Markup for Web Services**. *W3C Member Submission*, 2004.

[25] CAILLIAU, R. **A Short History of the Web**. *International WWW Conference Committee*, 1995.

[26] CAILLIAU, R.; GILLIES, J. **How the Web Was Born: The Story of the World Wide Web** . Oxford University Press, 2000.

[27] CLARK, K. G.; FEIGENBAUM, L.; TORRES, E. **SPARQL protocol for RDF**. W3C Recommendation, 2008. http://www.w3.org/TR/rdf-sparql-protocol/.

[28] COLE, C.; RUSSELL, C.; WHYTE, J. **Building OpenSocial Apps: A Field Guide to Working with the MySpace Platform**. Addison-Wesley Professional, 2009.

[29] CROCKFORD, D. **JavaScript Object Notation (JSON)**. http://www.ietf.org/rfc/rfc4627.txt?number=4627, ultimo acesso em Julho de 2011, 2006.

[30] CROCKFORD, D. **JSON in Java**. Disponível em https://github.com/douglascrockford/JSON-java, último acesso em Julho de 2011., 2011.

[31] DE BRUIJN, J.; BUSSLER, C.; DOMINGUE, J.; FENSEL, D.; KIFER, M.; KOPECKY, J.; LARA, R.; OREN, E.; POLLERES, A.; STOLLBERG, M. **Web Service Modeling Ontology (WSMO)**. *W3C Member Submission*, 2005.

[32] DIETZOLD, S.; HELLMANN, S.; PEKLO, M. **Using JavaScript RDFa Widgets for Model/View Separation inside Read/Write Websites**. In: *Proceedings of the 4th Workshop on Scripting for the Semantic Web*, 2008.

[33] ERL, T. **Service-Oriented Architecture: Concepts, Technology, and Design**. Prentice Hall, 2005.

[34] FACEBOOK. **Facebook Developers - Facebook Markup Language (FBML)**. http://developers.facebook.com/docs/reference/fbml/, último acesso em Janeiro de 2011, 2011.

[35] FACEBOOK. **Facebook Developers - Facebook Query Language (FQL)**. Disponível em http://developers.facebook.com/docs/reference/fql/, último acesso em Janeiro de 2011, 2011.

[36] FACEBOOK. **Facebook Developers - Graph API**. Disponível em http://developers.facebook.com/docs/reference/api/, último acesso em Janeiro de 2011, 2011.

[37] FARRELL, J.; AKKIRAJU, R.; J.MILLER.; NAGARAJAN, M.; SCHMIDT, M.; SHETH, A.; VERMA, K. **Web Service Semantics - WSDL-S**. *W3C Member Submission*, 2005.

[38] FENSEL, D.; BUSSLER, C. **The Web Service Modeling Framework WSMF**. Technical report, Web Service Modeling Framework Initiative, 2003. White paper.

[39] FENSEL, D.; FISCHER, F.; KOPECKÝ, J.; KRUMMENACHER, R.; LAMBERT, D.; VITVAR, T. **WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web**. *W3C Member Submission*, 2010.

[40] FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. PhD thesis, University of California, 2000.

[41] FILHO, O. F. F. **Serviços Semânticos: Uma abordagem RESTful**. Master's thesis, Universidade de São Paulo, 2010.

[42] FLICKR. **App Garden - API Documentation - flickr.photos.search**. http://www.flickr.com/services/api/flickr.photos.search.html, último acesso em Janeiro de 2011, 2011.

[43] FOUNDATION, T. A. S. **Apache Shindig**. Apache Incubator, 2010.

[44] FRAGOSO, S. **WTF a Crazy Brazilian Invasion**. *Fifth International Conference on Cultural Attitudes Towards Technology and Communication*, 1:255–274, 2006.

[45] FRANKLIN, S.; GRAESSER, A. **Is It an agent, or just a program?: A taxonomy for autonomous agents**. In: Wooldridge, M.; Jennings, N., editors, *Intelligent Agents III Agent Theories, Architectures, and Languages*, volume 1193 de **Lecture Notes in Computer Science**, chapter 2, p. 21–35. Springer Berlin / Heidelberg, Berlin/Heidelberg, 1997.

[46] GOMADAM, K.; RANABAHU, A.; SHETH, A. **SA-REST: Semantic Annotation of Web Resources**. *W3C Member Submission*, 2010.

[47] GOMADAM, K.; RANABAHU, A.; SHETH, A. **Towards a RESTful Service Ecosystem: Perspectives and Challenges**. *4th IEEE International Conference on Digital Ecosystems and Technologies*, 2010.

[48] GRUBER, T. R. **Towards Principles for the Design of Ontologies Used for Knowledge Sharing**. In: Guarino, N.; Poli, R., editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[49] GRUBER, T. **Encyclopedia of Database Systems**, chapter Ontology, p. 3749. Springer reference. Springer-Verlag, 2009.

[50] GUDGIN, M.; HADLEY, M.; MENDELSOHN, N.; MOREAU, J.-J.; NIELSEN, H. F.; KARMARKAR, A.; LAFON, Y. **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)**. http://www.w3.org/TR/soap12-part1/, último acesso em Abril de 2011, 2007.

[51] HADLEY, M. **Web Application Description Language**. W3C: http://www.w3.org/Submission/wadl/, último acesso em Maio de 2011, 2009.

[52] HALPIN, H.; SUDA, B.; OGBUJI, C.; BOOTH, D.; GANDON, F.; DAVIS, I.; RECK, R. Y. R. P.; CLARK, J.; AYERS, D.; ONOFRI, S. **Gleaning Resource Descriptions from Dialects of Languages (GRDDL)**. *W3C Recommendation*, 2007.

[53] HAMMER-LAHAV, E. **The Authoritative Guide to OAuth 1.0**. *Hueniverse*, 2009.

[54] HAMMER-LAHAV, E. **Introducing OAuth 2.0**. *Hueniverse*, 2010.

[55] HAMMER-LAHAV, E. **The OAuth 1.0 Protocol**. *Internet Engineering Task Force (IETF)*, 2010.

[56] HAMMER-LAHAV, E.; RECORDON, D.; HARDT, D. **The OAuth 2.0 Protocol**. *Network Working Group*, 2010.

[57] HENDERSON, C.; COPE, A. S.; COSTELLO, E.; MOURACHOV, S.; BUTTERFIELD, S. **Flickr Authentication API**. *Flickr App Garden*, 2010.

[58] HENDLER, J. **Agents and the Semantic Web**. *IEEE Intelligent Systems*, 16:30–37, 2001.

[59] HORROCKS, I.; PATEL-SCHNEIDER, P. F.; BOLEY, H.; TABET, S.; GROSOF, B.; DEAN, M. **SWRL: A Semantic Web Rule Language Combining OWL and RuleML**. W3c member submission, World Wide Web Consortium, 2004.

[60] HORWITZ, R. **Google Launches OpenSocial to Spread Social Applications Across the Web**. *Google Press Center*, November 2007.

[61] HUNT, A.; THOMAS, D. **The Pragmatic Programmer: From Journeyman to Master**, chapter The Evils of Duplication, p. 26–33. Addison-Wesley Prof., 1999.

[62] ISO/IEC. **ISO/IEC 9126. Software engineering - Product quality**. ISO/IEC, 2001.

[63] JOHANSSON, R. **posh - Microformats Wiki**. Disponível em http://microformats.org/wiki/poshformats, último acesso em Abril de 2011.

[64] KAYAL, D. **Pro Java EE Spring Patterns: Best Practices and Design Strategies Implementing Java EE Patterns with the Spring Framework**. Apress, Berkely, CA, USA, 2008.

[65] KLUSCH, M.; FRIES, B.; SYCARA, K. **OWLS-MX: A hybrid semantic web service matchmaker for OWL-S services**. *Web Semantics*, 7(2):121–133, 2009.

[66] KOPECKÝ, J.; GOMADAM, K.; VITVAR, T. **hRESTS: an HTML Microformat for Describing RESTful Web Services**. *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2008.

[67] KOPECKÝ, J.; MORAN, M.; VITVAR, T.; ROMAN, D.; MOCAN, A. **WSMO Grounding**. *WSMO Working Draft*, 2007.

[68] KOPECKÝ, J.; VITVAR, T.; FENSEL, D. **MicroWSMO and hRESTS**. *Service Oriented Architectures for All*, 2009.

[69] KWAK, H.; LEE, C.; PARK, H.; MOON, S. **What is Twitter, a social network or a news media?** In: *WWW '10: Proceedings of the 19th international conference on World wide web*, p. 591–600, New York, NY, USA, 2010. ACM.

[70] LANTHALER, M.; GUTL, C. **A semantic description language for RESTful Data Services to combat Semaphobia**. In: *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference on*, p. 47 –53, 31 2011-june 3 2011.

[71] LEENHEER, P. D. **Open Standards for the Semantic Web: XML / RDF(S) / OWL / SOAP**. Vrije Universiteit Brussel, 2009.

[72] LIPKIN, D.; MARSH, J.; THOMPSON, H.; WALSH, N.; ZILLES, S. **eXtensible Stylesheet Language Transformations (XSLT)**. *W3C Recommendation*, 1999.

[73] LOUVEL, J. **Restlet**. Disponível em http://www.restlet.org/, último acesso em Abril de 2011, 2010.

[74] MACKINNON, I.; WARREN, R. **Age and geographic inferences of the LiveJournal social network**. In: *ICML'06: Proceedings of the 2006 conference on Statistical network analysis*, p. 176–178, Berlin, Heidelberg, 2007. Springer-Verlag.

[75] MAES, P. **Modeling Adaptive Autonomous Agents**. *Artificial Life*, 1:135–162, 1994.

[76] MAGKANARAKI, A.; ALEXAKI, S.; CHRISTOPHIDES, V.; PLEXOUSAKIS, D. **Benchmarking RDF Schemas for the Semantic Web**. In: Horrocks, I.; Hendler, J. A.,

editors, *International Semantic Web Conference*, volume 2342 de **Lecture Notes in Computer Science**, p. 132–146. Springer, 2002.

[77] MAHMOUD, H. **Web 3.0 The Semantic Web**. Expression Lab, 2009.

[78] MALESHKOVA, M.; PEDRINACI, C.; DOMINGUE, J.; ALVARO, G.; MARTINEZ, I. **Using semantics for automating the authentication of Web APIs**. In: *The 9th International Semantic Web Conference (ISWC 2010)*, 2010.

[79] MANDEL, L. **Describe REST Web services with WSDL 2.0**. Rational Software Developer, IBM, 2008.

[80] MANOLA, F.; MILLER, E. **RDF Primer**. W3C Recommendation, 2004.

[81] MARTIN, D.; PAOLUCCI, M.; WAGNER, M. **Bringing Semantic Annotations to Web Services: OWL-S from the SAWSDL Perspective**. *ISWC/ASWC*, p. 340 – 352, 2007.

[82] MCBRIDE, B. **Jena: a semantic Web toolkit**. *Internet Computing, IEEE*, 6(6):55 – 59, 2002.

[83] MCLAUGHLIN, B.; EDELSON, J. **Java & XML**. O'Reilly Media, Inc., 2006.

[84] MICROFORMATS.ORG. **Microformats Principles**. Disponível em http://microformats.org/wiki/principles, último acesso em Abril de 2011.

[85] MICROFORMATS.ORG. **Microformats Process**. Disponível em http://microformats.org/wiki/process, último acesso em Abril de 2011.

[86] MOORE, R.; HOBBS, J.; MOORE, R. **A Formal Theory of Knowledge and Action**. In: *Formal Theories of the Commonsense World*, p. 319–358. Abrex Publishing Corporation, 1985.

[87] MÖLLER, T. **OWL-S API Documentation**. Disponível em http://on.cs.unibas.ch/owls-api/documentation.html, último acesso em Julho de 2011., 2011.

[88] O'REILLY, T. **What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software**. *International Journal of Digital Economics*, 65:17–37, 2007.

[89] PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. **RESTful Web Services vs. "Big"Web Services: Making the Right Architectural Decision**. 17th International World Wide Web Conference, 2008.

[90] PEMBERTON, S.; ET AL. **XHTML 1.0: The Extensible HyperText Markup Language**. W3C, 2000.

[91] PERRY, B. W. **Java Servlet & JSP Cookbook**. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.

[92] PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL Query Language for RDF**. W3C Recommendation, 2008. http://www.w3.org/TR/rdf-sparql-query/.

[93] RIVLIN, G. **Wallflower at the Web Party** . *The New York Times*, Outubro 2006.

[94] SEABORNE, A.; MANJUNATH, G.; BIZER, C.; BRESLIN, J.; DAS, S.; DAVIS, I.; HARRIS, S.; IDEHEN, K.; CORBY, O.; KJERNSMO, K.; NOWACK, B. **SPARQL/Update A language for updating RDF graphs**. W3C Member Submission, 2008.

[95] SHAFIQ, O.; MORAN, M.; CIMPIAN, E.; MOCAN1, A.; ZAREMBA1, M.; FENSEL, D. **Investigating Semantic Web Service Execution environments: A comparison between WSMX and OWL-S tools**. *Second International Conference on Internet and Web Applications and Services (ICIW'07)*, 2007.

[96] SHETH, A. P.; GOMADAM, K.; LATHEM, J. **SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups**. *IEEE Computer Society*, 2007.

[97] SIRIN, E.; PARSIA, B.; GRAU, B.; KALYANPUR, A.; KATZ, Y. **Pellet: A practical OWL-DL reasoner**. *Journal of Web Semantics*, 5(2):51–53, June 2007.

[98] TEAM, T. H. S. W. **Jena: A Semantic Web Framework for Java**. Disponível em http://jena.sourceforge.net, último acesso em Julho de 2011.

[99] TIM BERNERS-LEE, J. H.; LASSILA, O. **The Semantic Web**. *Scientific American*, 284(5):28–37, 2001.

[100] TYAGI, S. **RESTful Web Services**. http://java.sun.com/developer/technicalArticles/WebServices/restful/, último acesso em Maio de 2011, 2006.

[101] WALLS, C.; BREIDENBACH, R. **Spring in Action**. Manning, 2nd edition, 2007.

[102] WILKINS, G. **Jetty://**. http://jetty.codehaus.org/jetty/, último acesso em Maio de 2011, 2009.

[103] WOOLDRIDGE, M.; JENNINGS, N. R. **Intelligent Agents: Theory and Practice**. *Knowledge Engineering Review*, 10:2:115–152, 1995.

APÊNDICE A

Aplicações Sociais

Aplicações Sociais são aplicações que utilizam as informações disponibilizadas em redes sociais, para promover alguma funcionalidade desejada por um conjunto de pessoas. Tais aplicações podem ser categorizadas em três focos principais: atividades, produtos ou conteúdo. A principal característica que difere uma aplicação social das convencionais é a possibilidade que o usuário tem de interação com os outros usuários. Entretanto, para que uma aplicação social prospere, ela deve fazer sentido para que o usuário possa utilizá-la sozinho [9].

As aplicações sociais utilizam APIs Web de Redes Sociais *Online* para manipular as suas informações. Tais APIs são *Web Services* que permitem acesso e manipulação de informações sociais. A maioria das APIs são implementadas através dos princípios REST e utilizam JSON ou XML para a representação dos dados. Para realizar a autenticação e a autorização de uso das informações, presentes nas redes sociais, pelo usuário, é utilizado o protocolo OAuth.

OAuth ou *Open Authorization* é um padrão aberto que permite aos usuários de um site garantirem acesso, por uma aplicação externa, aos seus recursos privados sem ter que compartilhar suas senhas e logins [55]. O foco principal do OAuth está na autorização e não na autenticação. Sendo assim, ele prevê níveis de autorização, que o usuário pode aceitar ou não. O OAuth foi construído baseado nos padrões proprietários Google AuthSub, Yahoo BBAuth e Flickr API Auth. Também pode ser caracterizado como um protocolo. Sua especificação consiste em duas partes. A primeira parte define um processo no navegador, para redirecionamento do usuário final para autorização aos seus recursos, pelo cliente[1]. A segunda parte define um método para realização de requisições HTTP autenticadas usando dois conjuntos de credenciais. Um conjunto destinado à identificação do cliente e outro à identificação do dono do recurso a ser requisitado [53]. O protocolo segue o seguinte fluxo:

[1] No OAuth, o cliente ou consumidor é a aplicação que está requisitando os recursos privados. O site que os fornece é chamado de servidor ou provedor do serviço. O usuário é chamado de dono do recurso.

1. ***Token* de Requisição**. Quando o usuário entra na aplicação consumidora, esta requisita ao servidor um *token* de requisição. Quando a aplicação consumidora recebe o *token*, ela redireciona o usuário para a tela de autenticação do servidor. O *token* de requisição é enviado, bem como um link para redirecionamento, assim que o usuário autenticar-se.
2. **Autenticação e Autorização**. Ao ser redirecionado para a tela de autenticação do servidor, o usuário é requisitado a autenticar-se. Uma vez autenticado, o usuário recebe um questionamento acerca da autorização para a aplicação consumidora.
3. **Redirecionamento à aplicação consumidora**. Caso o usuário autorize o acesso, o servidor marca o *token* de requisição, enviado no passo 1, como autorizado. O usuário então é redirecionado para o URI previamente informado pela aplicação consumidora.
4. ***Token* de acesso**. Quando o fluxo retorna à aplicação consumidora, ela encarrega-se de fazer um intercâmbio do *token* de requisição por um *token* de acesso. O *token* de requisição tem como único objetivo a aprovação, pelo usuário. Entretanto, o *token* de acesso é destinado às requisições dos recursos privados.
5. **Acesso aos recursos privados**. Com o *token* de acesso, a aplicação consumidora pode consultar todos os recursos privados, aos quais o usuário concedeu permissão.

OAuth foi desenvolvido em 2006. Com mais de 3 anos de experiência trabalhando com esse protocolo, em 2009 a comunidade começou a repensá-lo. A versão 2.0 do OAuth [56] é um protocolo completamente novo. Sendo assim, ela não garante compatibilidade com as versões anteriores. Essa versão ganhou melhorias em 3 grandes áreas em que a versão 1.0 era limitada ou confusa [54]:

1. **Autenticação e Assinaturas**. A principal falha das tentativas de implementação do OAuth 1.0 foi causada pela complexidade da criptografia requerida na especificação. Mesmo depois de uma revisão, para a versão 1.0a, OAuth continuou não trivial para uso no lado do cliente[2]. Para usar a especificação, os desenvolvedores eram obrigados a buscar, instalar e configurar bibliotecas externas.
2. **Experiência do Usuário e Opções Alternativas de Emissão de *Tokens***. OAuth inclui duas partes principais: obtenção do *token*, através da autorização de acesso pelo usuário e uso do *token* para obtenção de recursos protegidos. O método para obtenção do *token* é chamado *flow*. A versão 1.0 do OAuth possuia 3 *flows* (*web, desktop e mobile*). Entretanto, a especificação evoluiu posteriormente para um único *flow*, que atendia os três tipos de clientes. Porém, com o aumento do número de *sites* que usam o OAuth, o *flow* disponível acabou se tornando cada vez mais limitado.

[2]Lado do cliente, nesse contexto, refere-se às linguagens interpretadas pelo navegador, como JavaScript e HTML.

3. **Performance em Escala**. Com grandes empresas aderindo ao OAuth, a comunidade chegou à conclusão de que o protocolo não possuía um bom escalonamento. Ele necessitava de gerenciamento de estados através de diferentes etapas e requeria a emissão de credenciais de longa duração, que são menos seguras e mais difíceis de gerir.

Com a versão 2.0 do OAuth foram criados 6 novos *flows*:

- ***User-Agent Flow***. Para clientes em navegadores Web.
- ***WebServer Flow***. Para clientes que são parte de uma aplicação em servidor Web.
- ***Device Flow***. Adequados para clientes de execução em dispositivos limitados.
- ***Username and Password Flow***. Usado quando há um elevado grau de confiança entre o usuário e o cliente, para guardar suas credenciais.
- ***Client Credentials Flow***. O Cliente usa suas credenciais para obter o *token* de acesso.
- ***Assertion Flow***. O cliente apresenta uma afirmação como uma SAML[3] para o servidor de autorização, em troca de um *token* de acesso.

A versão 2.0 do OAuth ainda provê uma opção para autenticação sem criptografia, utilizando HTTPS. As assinaturas foram simplificadas, não sendo mais necessária a análise, codificação e a ordenação de parâmetros. Nesta versão, é necessária apenas uma chave secreta. Também são implementados *tokens* de acesso de curta duração, que são renovados a partir de *tokens* de atualização. Isso permite ao cliente requisitar um novo *token* de acesso, de forma transparente ao usuário. O OAuth 2.0 separa o papel de autenticação e autorização pelo servidor, da requisição de recursos através da API. Isso facilita o uso de servidores distribuídos [54].

A.1 APIs das Principais Redes Sociais *Online*

Como toda aplicação Web, *sites* de redes sociais são hospedados em servidores Web. Para que outras aplicações consigam interagir com aplicações Web, são usados *Web Services*. Então, as APIs de redes sociais *online* são *Web Services* destinados a oferecer recursos e funcionalidades, presentes na rede social, para aplicações externas. Essas APIs transformam os *sites* de rede social em plataformas para aplicações sociais. Nesta seção são descritas as APIs das principais redes sociais *online*.

O **Facebook**[4] possui uma API, para leitura e escrita de dados, em seu núcleo. Ela é chamada de *Graph API*. Com ela é possível, de maneira simples, requisitar informações

[3] *Security Assertion Markup Language*. Mais detalhes em http://saml.xml.org/
[4] URL para o Facebook: http://www.facebook.com

como o relacionamento entre usuários, suas fotos, gostos, eventos, páginas entre outras coisas. Para que uma aplicação consiga acessar as funcionalidades da *Graph API* é necessário passar por um processo de autorização. A aplicação só poderá consultar informações de um usuário que a autorizou. Para isso é usado o protocolo OAuth, em sua versão 2.0. Na API, existe um conjunto de *Web Services* REST que podem ser requisitados a partir de URIs. Tais *Web Services* podem ser chamados de métodos, em APIs de Redes Sociais *Online*, como o Facebook. Uma lista completa dos métodos da API do Facebook pode ser vista em [36].

No Facebook, cada aplicação registrada possui um ID e um código secreto, necessários para autenticação[5]. Esses dados são utilizados para identificação da aplicação quando é requisitado o acesso às informações dos usuários. Para facilitar o acesso à API, o Facebook disponibiliza alguns SDKs[6]. A Figura A.1 exibe um exemplo de utilização do JavaScript SDK[7] para o Facebook. O Facebook ainda possui SDKs para PHP, Python, iOS e Android. Tais SDKs abstraem a utilização do protocolo OAuth e as requisições para os *Web Services* REST, da API. Entretanto, caso o desenvolvedor prefira, poderá fazer as requisições manualmente.

A API do Facebook também possui uma linguagem para consulta de dados. Parecida com o SQL, ela é chamada de FQL (*Facebook Query Language*). O exemplo da Figura A.2 retorna o nome, a foto e o gênero de um usuário. A FQL pode retornar os dados no formato XML ou JSON. O padrão é XML. Para especificar o formato, basta enviar o parâmetro `format` com valor igual a `xml` ou `json`. Uma lista completa de tabelas e campos pode ser obtida em [35]. Também há uma linguagem de marcação, chamada de FBML (*Facebook Markup Language*). Para usá-la é necessário criar uma página interna ao Facebook ou usar o JavaScript SDK. Na linha 7 da Figura A.1, é ativado o interpretador de FBML. Uma lista completa com todas as tags da FBML pode ser consultada em [34].

[5] Link para registro de aplicações no Facebook: http://developers.facebook.com/setup/

[6] *Software Development Kit* (SDK) é um conjunto de ferramentas para facilitar o desenvolvimento ligado à algum *software*. Neste caso, a API do Facebook.

[7] Foi utilizado o SDK para JavaScript, visto que não há nenhum SDK oficial para Java. O presente trabalho foca no desenvolvimento em Java, para Web.

```
<script src="http://connect.facebook.net/pt_BR/all.js"></script>
<script>
 FB.init({
  appId  : '138213456214689', // ID da aplicação
  status : true, // verificar se está logado
  cookie : true, // ativar cookies (sessões)
  xfbml  : true  // interpretar XFBML
 });
 function listarAmigos(response) {
  if (response.session) {
   // Usuário está logado, lista os nomes de seus amigos
   FB.api('/me/friends', function(response) {
    var amigos = response.data;
    for(i in amigos) {
     document.getElementById('saida')
      .innerHTML += amigos[i].name+"<br/>";
    }
   });
  } else {
   // Usuário não está logado, pede para ele logar-se
   alert('Desculpe, é necessário fazer login.');
   FB.login(function(response) {
    listarAmigos(response);
   });
  }
 }
 // Verificar se usuário já está logado
 FB.getLoginStatus(function(response) {
  listarAmigos(response);
 });
</script>
```

Figura A.1: *Requisitando Lista de Amigos do Facebook com JavaScript SDK*

```
        https://api.facebook.com/method/fql.query?
                query=SELECT name, pic, sex FROM user WHERE uid = 1820391700
```

Figura A.2: *Exemplo de uso da FQL na API do Facebook.*

O **Twitter**[8] possui uma API que pode ser dividida em três partes. A parte principal, chamada de REST API possui recursos para manipulação dos dados dos usuários e das mensagens trocadas por eles. Também há duas APIs destinadas à requisição de mensagens enviadas pelos usuários do Twitter, chamadas *Search* API e *Streaming* API. As duas diferem-se pelo fato de a *Streaming* API ser destinada à conexões pertinentes, para o envio síncrono de mensagens. Essa funcionalidade é útil para sistemas Desktop.

[8]URL para o Twitter: http://www.twitter.com

Para facilitar o acesso e a autorização (que também é feita através do protocolo OAuth), o Twitter dispõe de várias bibliotecas terceirizadas. Para Java, a mais comum é o Twitter4J[9]. A Figura A.3 exibe um exemplo de utilização da biblioteca Twitter4J para requisição de mensagens que contenham a expressão "*Semantic Web*".

```
1       Twitter twitter = new TwitterFactory().getInstance();
2       Query query = new Query("Semantic Web");
3       QueryResult result = twitter.search(query);
4       System.out.println("hits:" + result.getTotal());
5       for (Tweet tweet : result.getTweets()) {
6               System.out.println(tweet.getFromUser() + ":" + tweet.getText());
7       }
```

Figura A.3: *Exemplo de uso da biblioteca Twitter4J para acesso à API do Twitter.*

A API do **Flickr**[10] possui, como adicional em relação à maioria das APIs de Redes Sociais *Online*, além da arquitetura REST, o SOAP e o XML-RPC. Entretanto, neste trabalho, será utilizada a arquitetura REST, em busca de maior semelhança e integração com as demais APIs. O Flickr trabalha com um protocolo de autenticação/autorização proprietário, semelhante ao OAuth. Tal protocolo foi um dos utilizados como inspiração para a elaboração do OAuth [54]. Como nas demais, na API do Flickr é necessário cadastrar uma aplicação[11]. Com o registro da aplicação, são criados uma chave e um segredo. Caso a aplicação seja Web, é necessário configurar um URL de retorno (*callback*). A autenticação segue o seguinte fluxo [57]:

1. **Código frob**. Inicialmente é feita uma requisição para o URI `http://flickr.com/services/auth/`. São enviados 3 parâmetros. O parâmetro `api_key` envia a chave da aplicação. O `api_sig`, deve ter como valor a assinatura desta requisição (gerada a partir do procedimento presente no passo 2). O `perms`, com o nível de permissões desejado para a aplicação. Esse parâmetro pode assumir três valores: `read` , para leitura de dados; `write` para inserção/atualização de dados e `delete` para exclusão de dados. Cada nível possui a permissão de seus antecessores. Logo, o nível `delete` possui `write` e `read`. Caso o usuário não esteja logado é exibido um formulário de *login*. Em seguida, o usuário é requisitado a aceitar ou negar autorização à aplicação. Caso o usuário aceite, a requisição redireciona o fluxo para o URL de retorno, enviando o parâmetro `frob`.

[9] Mais informações sobre Twiiter4J em: http://twitter4j.org
[10] URL para o Flickr: `http://www.flickr.com`
[11] O registro de aplicações para uso da API do Flickr pode ser feito em: `http://www.flickr.com/services/apps/`. É necessária uma conta de e-mail no Yahoo.

2. **Assinatura**. Para gerar a assinatura necessária no passo anterior, primeiramente ordena-se alfabeticamente os parâmetros a serem enviados. Em seguida, o segredo da aplicação é concatenado com os parâmetros e seus respectivos valores. A partir da expressão resultante, é calculado o *hash* MD5[12]. Esse *hash* será a assinatura.
3. **Obtendo *token* de acesso**. O aplicativo deverá então chamar o método `flickr.auth.getToken` da API. Os métodos da API do Flickr são requisitados a partir do URI `http://flickr.com/services/rest/`, para a arquitetura REST. O método desejado é informado pelo parâmetro `method`. Este método requer 3 parâmetros. A chave da aplicação, o código frob e uma assinatura, seguindo o mesmo procedimento do passo 2. A resposta terá um *token* para futuras chamadas de métodos.

Existem alguns métodos na API do Flickr que não necessitam de autorização. Entretanto, mesmo para estes é necessário enviar a chave da aplicação. O método `flickr.photos.search` é um exemplo. Ele tem como objetivo a busca por fotos em toda a base do Flickr. Este método possui uma série de parâmetros para filtros opcionais[13]. Na Figura A.4 pode ser visto um exemplo de URI para requisitá-lo.

```
1  http://www.flickr.com/services/rest/?
2      method=flickr.photos.search&
3      api_key=...&
4      tags=Goiania,Praça&
5      format=json
```

Figura A.4: *Exemplo de URI, no Flickr, para o método* `flickr.photos.search`.

O URI acima retorna uma lista, em formato JSON, de dados de fotos que possuem as tags "Goiânia"e "Praça". A paginação é configurada pelos parâmetros `page` e `per_page`, com valores padrões de 1 e 100 respectivamente. O parâmetro `format` é opcional, sendo que o padrão é XML. Este parâmetro está presente em qualquer método da API. Para requisitar uma foto (e não apenas seus dados) é utilizada a sintaxe de URI exibida na Figura A.5. Os valores entre chaves são parâmetros obtidos através do retorno do método de busca, conforme pode ser visto na Figura A.6.

```
1  http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}.jpg
```

Figura A.5: *Sintaxe de URI para requisição de fotos do Flickr.*

[12] MD5, *Message-Digest algorithm 5*, é um algoritmo de *hash* (criptografia sem volta) de 128 bits, desenvolvido pela RSA Data Security, Inc. e descrito na RFC 1321.

[13] A lista completa de parâmetros do método `flickr.photos.search` está disponível em [42]

```
1        // Exemplo de tag do XML de retorno:
2        <photo id="1318815545" owner="10298760@N03"
3               secret="0a1a634be8" server="1439" farm="2" ... />
4        // URI para a foto:
5        http://farm2.static.flickr.com/1439/1318815545_0a1a634be8.jpg
```

Figura A.6: *Exemplo de tag do XML de retorno e URI para foto.*

Diferentemente das APIs descritas anteriormente, o **OpenSocial**[14] é um conjunto de APIs que possui como principal objetivo a interoperabilidade de aplicações em *sites* de rede social. Tais APIs descrevem um "modelo"de plataforma para desenvolvimento de aplicativos em redes sociais *online*. Os *sites* de redes sociais que implementam as APIs descritas pelo OpenSocial são chamados de recipientes. Os aplicativos desenvolvidos em OpenSocial funcionam em qualquer recipiente que aceite a versão em que este foi desenvolvido. Atualmente, mais de 40 *sites* de redes sociais são recipientes para o OpenSocial[15] [28]. Estão entre estes, *sites* como o Orkut, o MySpace, o LinkedIn, o Yahoo! e o Friendster. Entretanto, cada site implementa o OpenSocial em uma versão diferente. As versões mais atuais dão compatibilidade com as mais antigas, sendo que a atual é a 0.9. A API é completamente baseada em XML, HTML e JavaScript [28].

As aplicações desenvolvidas em OpenSocial consistem em um arquivo XML com chamadas à arquivos JavaScript e HTML, caso necessário. Tal arquivo XML centraliza as informações gerais sobre a aplicação. No cadastro de uma aplicação em OpenSocial, em um recipiente, deve ser informado o caminho para este arquivo XML. A Figura A.7 exibe um exemplo simples de aplicação OpenSocial que apenas exibe a mensagem "Olá Mundo!"em uma aplicação de nome "Teste - INF/UFG"e descrição "Teste de OpenSocial".

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <Module>
3      <ModulePrefs title="Teste - INF/UFG"
4                   description="Teste de OpenSocial">
5          <Locale lang="pt" country="br" />
6      </ModulePrefs>
7      <Content type="html">
8          <![CDATA[
9              Olá, Mundo!
10         ]]>
11     </Content>
12 </Module>
```

Figura A.7: *Exemplo de aplicação básica em OpenSocial.*

[14] URL para o OpenSocial: http://code.google.com/intl/pt-BR/apis/opensocial
[15] A lista completa de recipientes para o OpenSocial pode ser vista em: http://wiki.opensocial.org/