UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
ESCOLA DE INFORMÁTICA APLICADA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

OWL-S Discovery 3.0: Métricas de Similaridade Baseadas em String para Descoberta de Serviços Web Semânticos

Nome dos autores:
Giulia Cavalcanti de Almeida
Mariana Mendonça Curi

Nome dos Orientadores:
Kate Cerqueira Revoredo
Leonardo Guerreiro Azevedo

Dezembro/2013

OWL-S Discovery 3.0: Métricas de Similaridade Baseadas em String para Descoberta de Serviços Web Semânticos

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação

Nome dos autores:
Giulia Cavalcanti
Mariana Mendonça Curi

Nome dos Orientadores:
Kate Cerqueira Revoredo
Leonardo Guerreiro Azevedo

OWL-S Discovery 3.0: Métricas de Similaridade Baseadas em String para Descoberta de Serviços Web Semânticos

Aprovado em _____/_____/_____

BANCA EXAMINADORA

_____

Prof.ª Kate Cerqueira Revoredo, D.Sc. (UNIRIO)

_____

Prof. Leonardo Guerreiro Azevedo, D.Sc. (IBM Research – Brazil; PPGI-UNIRIO)

_____

Prof. Márcio de Oliveira Barros, D.Sc. (UNIRIO)

_____

Prof. Sean Wolfgand Matsui Siqueira, D.Sc. (UNIRIO)

Os autores deste Projeto autorizam a ESCOLA DE INFORMÁTICA APLICADA da UNIRIO a divulgá-lo, no todo ou em parte, resguardando os direitos autorais conforme legislação vigente.

Rio de Janeiro, _____ de _____ de _____

_____

Giulia Cavalcanti de Almeida

_____

Mariana Mendonça Curi

AGRADECIMENTOS

Agradecemos aos nossos familiares e amigos por todo incentivo e apoio que recebemos durante a elaboração deste projeto. Vocês sempre acreditaram que seríamos capazes, mesmo quando nós duvidávamos disso.

A todo o corpo docente da UNIRIO, pela dedicação e qualidade de ensino que resultou em nossa formação.

E, principalmente, aos nossos orientadores, Leonardo Azevedo e Kate Revoredo, agradecemos os ensinamentos e orientações. Obrigada pela confiança e dedicação. Graças a vocês, este trabalho se fez possível.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS

API - *Application Programming Interface*

DAML – *DARPA Agent Markup Language*

DARPA – *Defense Advanced Research Projects Agency*

HTTP – *Hypertext Transfer Protocol*

OIL – *Ontology Interchange Language*

OWL – *Web Ontology Language*

OWL-S – *Semantic Markup for Web Services*

RDF – *Resource Description Framework*

REST - *Representational State Transfer*

RML – *Rule Markup Language*

SAWDL – *Semantic Annotation For WSDL*

SGML – *Standard Generalized Markup Language*

SOA – *Service-Oriented Architecture*

SOAP – *Simple Object Access Protocol*

SPARQL – *Protocol and RDF Query Language*

TI – *Tecnologia da Informação*

UDDI – *Universal Description, Discovery and Integration*

URI – *Uniform Resource Identifier*

XML – *Extensible Markup Language*

W3C – *World Wide Web Consortium*

WSDL – *Web Services Description Language*

WSMO – *Web Service Modeling Ontology*

RESUMO

Nota-se, atualmente, uma crescente utilização de serviços web em ambientes organizacionais. Entre as etapas de ciclo de vida de um serviço, a tarefa de descoberta possui grande importância. Esse projeto evolui a ferramenta OWL-S Discovery 2.0 para descoberta de serviços web semânticos incluindo funcionalidades para utilização de métricas de similaridade baseadas em *strings*. O resultado deste trabalho é a OWL-S Discovery 3.0, ferramenta que flexibiliza a busca pelo serviço que melhor atende às necessidades do usuário. Através da nova versão da ferramenta, o consumidor do serviço não mais precisa descrever os dados para busca do serviço seguindo uma ontologia (por exemplo, no formato OWL - Web Ontology Language) ou mesmo no formato OWL-S. Estas informações podem ser preenchidas em texto livre. A ferramenta OWL-S Discovery 3.0 encontra nas ontologias que descrevem as operações (dados de entrada e saída do serviço) os conceitos que melhor combinam com as informações preenchidas pelo usuário e a partir daí realiza a busca pelo serviço.

Palavras-chave: SOA, Descoberta de Serviços web, Serviços web Semânticos, OWL-S, OWL-S Discovery, Métricas de Similaridade.

1. INTRODUÇÃO

O objetivo deste capítulo é contextualizar o assunto abordado neste trabalho, bem como apresentar sua motivação, o objetivo e a estrutura dos capítulos.

1.1. Motivação

Inseridas em um ambiente altamente competitivo, organizações modernas precisam estar preparadas para responder rápida e efetivamente às oportunidades que surgem no mercado. Nesse contexto, a arquitetura orientada a serviços (SOA – Service-Oriented Architecture) apresenta-se como uma forma eficaz de atingir a agilidade necessária [Erl, 2005], trazendo como principal solução a decomposição de funcionalidades de sistemas em serviços. Por prover de forma interoperável, serviços padronizados, integrados e de baixo acoplamento, a utilização de Serviços web tem crescido significativamente [Alves *et al*.,2011].

Durante o ciclo de vida de um serviço, a etapa de sua descoberta, que se refere à busca pelos serviços mais semelhantes aos requisitos solicitados, é uma das mais fundamentais, porém, a falta de suporte semântico na linguagem de descrição de serviços (Web Services Description Language – WSDL[1]) dificulta a execução desta tarefa [Forte *et al.*, 2006]. A alternativa mais utilizada para ultrapassar tais limitações é a adoção de ontologias, que atribuem semântica às descrições de serviços.

Ontologias permitem a conceituação de domínios, sendo a OWL (Web Ontology Language[2]) uma das mais conhecidas linguagens de representação de ontologias. Unindo a descrição dos serviços web e a informação semântica da OWL, a linguagem OWL-S[3] (Semantic Markup for Web Services) descreve os serviços web, transformando-os em serviços web semânticos. O OWL-S Discovery, ferramenta que será trabalhada neste projeto, foi desenvolvida para possibilitar a automatização da descoberta de serviços web semânticos.

[1] http://www.w3.org/TR/wsdl
[2] http://www.w3.org/TR/owl2-primer/
[3] http://www.w3.org/Submission/OWL-S/

1.2. Objetivo

A descoberta de um serviço web pode ser uma atividade complexa para o para ser executada manualmente. O objetivo deste trabalho é flexibilizar o processo de descoberta de serviços através da aplicação de conceitos de métricas de similaridade baseadas em string sobre o algoritmo OWL-S Discovery. O resultado deste trabalho é a OWL-S Discovery 3.0, ferramenta que flexibiliza a busca pelo serviço que melhor atende às necessidades do usuário através de *string matching*.

1.3. Estrutura do Trabalho

Os próximos capítulos estão organizados da seguinte forma. No capítulo 2, são apresentados os principais conceitos necessários para este trabalho. O Capítulo 3 apresenta a ferramenta OWL-S Discovery que será evoluída no presente trabalho. As métricas de similaridade utilizadas para tal evolução serão descritas no Capítulo 4. No Capítulo 5 será descrito o algoritmo proposto. O Capítulo 6 abordará os trabalhos relacionados. As análises e validações da proposta são apresentadas no Capítulo 7. Aspectos da implementação deste algoritmo são apresentados no Capítulo 8. No capítulo 9, finalmente, tem-se as conclusões e considerações finais do projeto.

2. CONCEITOS

Este capítulo tem como objetivo apresentar os conceitos de SOA, serviços e web semântica que serão utilizados nesse trabalho.

2.1 Arquitetura Orientada a Serviços (SOA)

A demanda por recursos de sistemas aumentou conforme a relação de apoio entre o mundo dos negócios e a tecnologia da informação (TI) se tornou intrínseca. O desenvolvimento de sistemas cada vez mais complexos, sobrecarregados e com responsabilidades fora do seu escopo para atender às mudanças do negócio, foi se disseminando devido à falta de uma arquitetura que propusesse um padrão e uma solução mais efetiva. A Arquitetura Orientada a Serviços surge então com o grande objetivo de alinhar os dois mundos de forma eficiente.

2.1.1. Definição

O paradigma SOA propõe um modelo em que a lógica de automação é decomposta em pequenas e distintas unidades. Em conjunto, essas unidades compreendem o processo de negócio como um todo, mas individualmente estas unidades podem estar distribuídas [Erl, 2005]. Em outras palavras, SOA traz como principal solução a decomposição de funcionalidades de sistemas em serviços, de modo que sistemas distribuídos mantenham seus papéis e responsabilidades bem definidos dentro de um processo de negócio, e disponibilizem estas funcionalidades em forma de serviços que possam ser reutilizados pelos demais sistemas.

A arquitetura SOA é projetada para propor soluções aos desafios e dificuldades que acompanham a computação distribuída, incluindo integração de aplicações, gerenciamento de transações, políticas de segurança, funcionamento eficaz de múltiplas plataformas, protocolos heterogêneos e sistemas legados. O objetivo que conduz SOA é eliminar barreiras para que aplicações integradas possam funcionar eficientemente. Deste modo, SOA pode entregar a flexibilidade e agilidade

que analistas de negócio necessitam para se adaptar as rápidas mudanças de mercado. [Josuttis, 2007].

2.1.2. Serviços

Soluções de automação de negócio são tipicamente a implementação de um processo de negócio. Esse processo é composto por uma lógica que organiza as ações que devem ser tomadas para se chegar a um objetivo. Esta lógica é decomposta em uma série de passos que são executados em uma ordem pré-definida de acordo com regras de negócio e condições de execução.

Ao implementar uma automação de negócio baseada em SOA, cada serviço é definido como um ou mais passos dessa implementação. Conforme mostrado na Figura 1, a modelagem e implementação de um *workflow* em uma arquitetura SOA pode conter serviços que encapsulem uma simples etapa de um processo de negócio, ou ainda serviços que encapsulem lógicas disponibilizadas por outros serviços [Erl, 2005]. O primeiro caso, pode ser observado com um único elemento do processo de negócio sendo implementado como um serviço (por exemplo, decisões associadas um único serviço). O segundo caso pode ser observado na parte do modelo ressaltadas por um retângulo envolvendo dois serviços que são compostos em um único serviço.

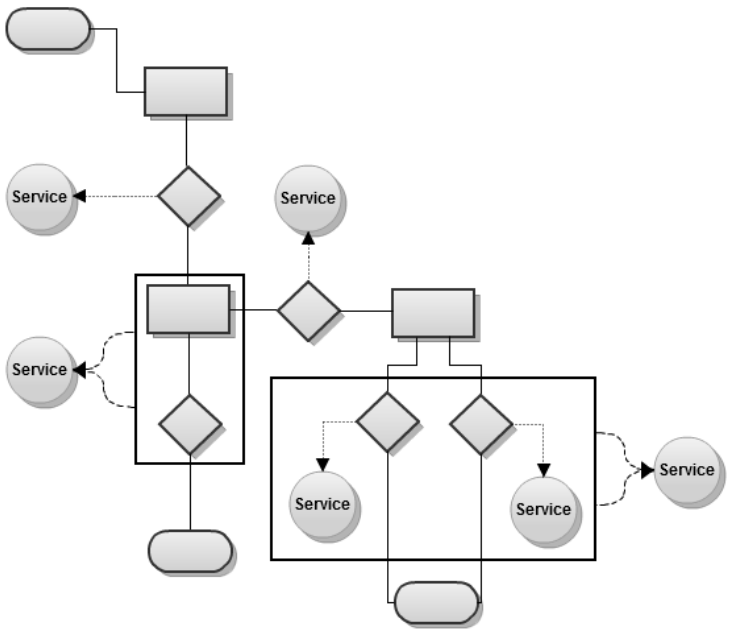

Figura 1 - Processo de Negócio modelado com serviços (adaptado de [ERL, 2005])

Em uma arquitetura SOA, serviços são recursos de software distribuídos e encapsulados, que são módulos auto-contidos, bem definidos e que oferecem funcionalidades de negócio padronizadas e independentes de estado ou contexto de outros serviços. Serviços são descritos por uma linguagem de definição padrão e se comunicam entre si por requisições (mensagens) feitas através de uma interface. [Papazoglou *et al.*, 2007].

Josuttis apresenta uma lista de atributos que serviços devem atender, alguns obrigatórios e outros opcionais:

- *Auto-contidos:* serviços devem atender a um escopo bem definido de funcionalidades, minimizando dependências, para que SOA seja apropriada para sistemas distribuídos e com diferentes proprietários;
- *Localizável:* Serviços e suas descrições devem ser localizáveis, estando disponíveis em algum repositório;
- *Sem estado:* Serviços não devem manter um estado. No entanto, a regra tem exceções dependendo do propósito do serviço e de questões técnicas;
- *Idempotente:* Um serviço é considerado idempotente se o resultado de uma requisição realizada com sucesso, é independente do número de vezes que é executada, ou seja, o serviço retorna a mesma resposta sempre para uma determinada requisição;
- *Reuso:* Serviços encapsulam funcionalidades que podem representar uma necessidade comum a diferentes sistemas, evitando redundâncias e promovendo a reusabilidade;
- *Composto:* Serviços podem usar/chamar outros serviços. Ou seja, funcionalidades de negócio maiores podem ser quebradas em passo menores, que por sua vez também são serviços;
- *Pré e Pós-Condições:* Pré e pós-condições ajudam a especificar o comportamento semântico dos serviços. As pré-condições definem os requisitos específicos que um consumidor deve atender antes de consumir um serviço e as pós-condições garantem as propriedades

específicas do sistema e/ou uma saída quando o serviço for executado com sucesso;

2.1.3. Web Services

O uso de *web services* se tornou a principal opção para a implementação de uma SOA com serviço compartilhado, reuso e interoperabilidade [Erl, 2005]. *Web services* e SOA reduzem a complexidade de sistemas corporativos através do encapsulamento e minimização de requerimentos necessários para se estabelecer uma contrato, feito através da definição da interface do serviço de forma transparente. Baseado em padrões de infraestrutura abertos e universais tais como HTTP, SOAP e XML, *Web Services* parecem ter se consolidado como solução [Papazoglou *et al*. 2007].

Os *web services* podem ser definidos como programas modulares, geralmente independentes e auto-descritivos que podem ser localizados e invocados através da internet ou de uma intranet corporativa [W3C, 2004].

O padrão XML (*Extensible Markup Language*) descreve uma classe de objetos de dados, chamados documentos XML, e parcialmente descreve o comportamento dos programas os processam. XML é um perfil de aplicação ou uma forma restrita do SGML (*Standard Generalized Markup Language*). Os principais objetivos do uso de XML são: (1) facilitar a comunicação na internet (2) suportar a comunicação de uma variedade de aplicações implementadas em diferentes tecnologias [XML, 2008].

O padrão WSDL (*Web Services Description Language*) é uma linguagem baseada em XML para descrever *web services*. Um documento WSDL define uma gramática em XML para descrever serviços como uma coleção de *endpoints* ou portas capazes de trocar mensagens. As definições WSDL de um serviço fornecem documentação para sistemas distribuídos e servem como uma espécie de "guia" para automatizar os detalhes envolvidos na comunicação das aplicações consumidora e provedora [WSDL, 2001].

A especificação UDDI (*Universal Description, Discovery and Integration*) define um serviço de registro para *web services* e outros serviços eletrônicos e não eletrônicos. O serviço de registro UDDI é um *web service* que gerencia informações

sobre provedores de serviço, implementações de serviços e metadados de serviço. Provedores de serviço utilizam a UDDI para anunciar o serviço que oferecem e consumidores para descobrir serviços que atendem seus requerimentos e como obter o metadado do serviço necessário para consumir esses serviços [OASIS, 2002].

O SOAP (*Simple Object Access Protocol*) é independente de plataforma e independente de implementação. Permite baixo acoplamento entre requisitante e provedor e permite comunicação entre serviços de diferentes organizações [SOAP, 2007].

Atualmente, o padrão REST vem ganhando espaço no mercado, como forma alternativa ao padrão SOAP, por ser mais leve e ter a capacidade de transmitir dados via HTTP, aproveitando a infraestrutura web já existente. REST estabelece mecanismos para definir e acessar recursos, que são endereçados através de URIs.

A seguir, são descritos os passos genéricos que um consumidor e um provedor devem seguir para estabelecer uma comunicação utilizando o protocolo SOAP : (1) o consumidor identifica uma necessidade que pode ser atendida por um determinado serviço disponibilizado por um provedor, então provedor e consumidor passam a ser conhecidos um pelo outro; (2) o consumidor e o provedor concordam com o contrato do serviço (WSDL) e semânticas que irão governar a interação entre eles; (3) a descrição e semântica dos serviços são associadas apropriadamente aos agentes em ambos os lados (tanto no consumidor como no provedor do serviços); (4) o consumidor e o provedor trocam mensagens.

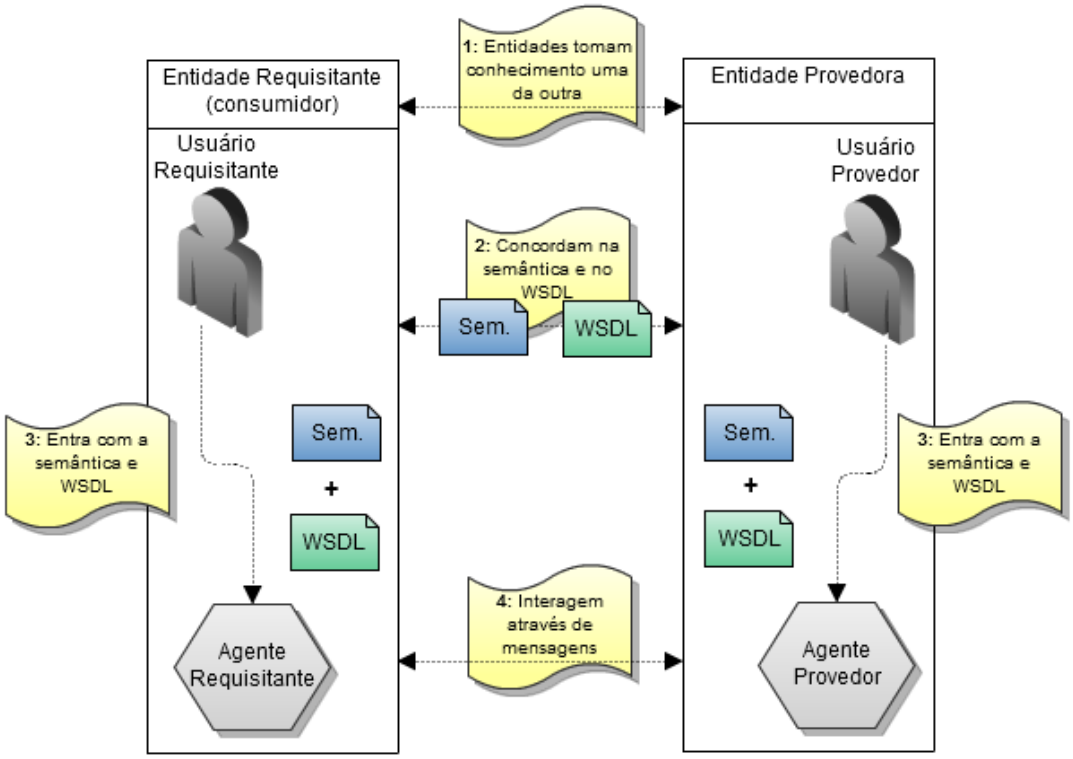

Figura 2 - Processo genérico para se estabelecer comunicação entre *web services* (adaptado de [Azevedo *et al.*, 2009])

2.2. Web Semântica

Idealizada por Tim Bernes-Lee, diretor do W3C e criador da World Wide Web (Rede Mundial de Computadores - Internet), a Web Semântica refere-se a uma visão de Web com dados vinculados, interligados. A proposta dessa visão é possibilitar a execução de tarefas mais complexas pelos agentes de software, como a transformação de dados em informações relevantes para o usuário. Para alcançar tal objetivo um dos passos a serem seguidos é a atividade de atribuir informação aos dados da atual web através de metadados. Metadados descrevem o conteúdo, estrutura, representação e conceito de um dado ou conjunto de dados [Nogueira, 2010]. Dessa forma, a Web Semântica promete tornar o conteúdo da Web legível para máquinas.

2.2.1. Arquitetura da Web Semântica

De forma a permitir a interpretação das informações presentes na web semântica, além de proporcionar a interoperabilidade entre sistemas de software, a W3C definiu um conjunto de padrões para a identificação dos recursos presentes na web, englobando também a representação sintática, estrutural, semântica e lógica de informações referentes a esses recursos [Valadares *et al.*, 2010].

A Figura 3 apresenta como as camadas e respectivos padrões que formam a arquitetura da Web Semântica.

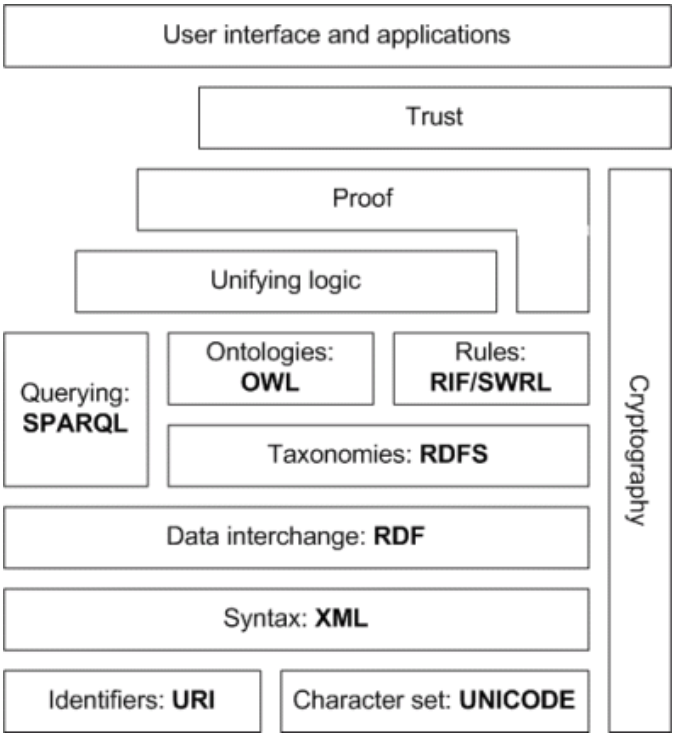

Figura 3 – Arquitetura da Web Semântica

Fonte: http://semanticweb.org/images/3/37/Semantic-web-stack.png

Na arquitetura apresentada acima, cada linguagem se baseia nas linguagens construídas nas camadas inferiores. Os níveis de URI e Unicode são as camadas mais fundamentais, tratando da representação de caracteres e recursos e a camada de XML, por sua vez, define a sintaxe de todas as linguagens de níveis superiores [Nogueira, 2010].

Segue abaixo uma visão geral de cada uma das camadas que compõem a arquitetura da Web Semântica:

- *URI e Unicode:* São responsáveis por fornecer um padrão de caracteres internacionais para a identificação dos recursos da Web Semântica, possibilitando uma representação que não possui dependências de plataformas, linguagens ou programas a serem utilizados. Unicode é um padrão para representação de caracteres e URI, por sua vez, é uma cadeia de caracteres que identifica os recursos na web, tais como páginas HTML, imagens, arquivos ou serviços disponibilizados na web.
- *XML:* Camada composta pelos padrões XML, XML-Schema e XML NameSpace. XML (*Extensible Markup Language*) é uma linguagem de marcação de texto simples e flexível, derivada da SGML (*Standard Generalized Markup Language*), que disponibiliza um padrão para a representação de estruturas que serão utilizadas como base para as linguagens que expressam significado na web. XML é então a base de todas as linguagens da Web Semântica que expressam significado, pois definirá as estruturas destas linguagens. A XML-Schema é uma linguagem de definição de classes e documentos XML, define e descreve então a estrutura do conteúdo de documentos XML, ou seja, a gramática para a validação destes documentos. XML NameSpace, por sua vez, é definido pela W3C como sendo uma coleção de nomes, identificados por uma referência URI, utilizados em documentos XML com o objetivo de obter um vocabulário para os tipos de elementos e nomes de atributos.
- *RDF:* Camada composta pelos padrões RDF e RDF-*Schema*. A linguagem RDF (*Resource Description Framework)* é um modelo padrão para intercâmbio de dados na web. RDF-*Schema*, similar ao XML-Schema, é a linguagem utilizada para definir a estrutura válida para dos documentos RDF.
- *Ontologia:* É a camada que possui as tecnologias de representação de conhecimento com poder de expressividade maior do que o proporcionado pela RDF. As linguagens de ontologias permitem

conceituações de domínios empregando uma sintaxe bem definida, uma semântica formal e alto nível de expressividade.

- *Lógica:* Camada que possui como objetivo especificar regras que facilitem a construção de relações sobre conceitos de uma ontologia de forma que os agentes possam utilizá-las para relacionar e processar informações.
- *Prova:* Responsável por determinar a consistência da informação inferida nas camadas superiores.
- *Confiança:* A confiança avalia se uma prova está correta, utilizando-se de uma assinatura digital que possui o objetivo de garantir a procedência de documentos, de forma a definir se estes são confiáveis.
- *SPARQL:* É a linguagem de consulta em RDF. Como as consultas em SPARQL escondem os detalhes do gerenciamento dos dados, os custos são reduzidos e é aumentada a robustez da integração dos dados na web.

2.2.2. Ontologias

No contexto computacional, ontologias são utilizadas na conceituação de um determinado domínio. Uma ontologia define um vocabulário utilizado para a modelagem de um domínio, assim como especifica as restrições atreladas ao uso do mesmo. Dessa forma, as linguagens utilizadas pelas ontologias representam a semântica das informações na web, possibilitando troca de dados entre diferentes ambientes [Valadares et al., 2010].

Um domínio é modelado através da utilização de quatro tipos de componentes em uma ontologia:

- *Classes:* Componente responsável por representar os conjuntos ou tipos de objetos. Os requisitos para a participação de um objeto na classe deve ser bem definido e apresentado formalmente através de descrições matemáticas. Classes podem ser organizadas em cadeias hierárquicas conhecidas como taxonomias. Uma subclasse é vista como uma especialização de uma superclasse, herdando todas as

características da mesma. De modo geral, pode-se dizer que as classes são representações concretas de conceitos.

- *Instâncias:* São os objetos no domínio de uma ontologia.
- *Propriedades:* São os componentes responsáveis por modelar as características de classes e instâncias. Podem representar atributos inerentes a uma classe, características e parâmetros específicos, ou expressar relações entre classes e instâncias.
- *Restrições:* As restrições, por sua vez, se utilizam das propriedades para descrever conceitos do domínio através de condições.

2.3. OWL

A OWL (*Web Ontology Language)* é um dos padrões propostos como linguagem para a Web Semântica, criada para representar ontologias. Sendo uma linguagem de marcação semântica, a OWL é utilizada para publicação e compartilhamento de ontologias na web. Desenvolvida como uma extensão do vocabulário RDF e derivada da DAML + OIL, a linguagem foi elaborada de tal forma que o conhecimento expresso nela possa ser "raciocinado" por programas de computador de forma a tornar explícito o conhecimento implícito e verificar a consistência do mesmo [Nogueira, 2010]. Ou seja, a OWL foi projetada para o uso de aplicações que possuem a necessidade de não apenas apresentar informações ao usuário, mas processar o conteúdo desta informação. Fornecendo uma semântica formal e um vocabulário adicional, a OWL permite uma facilidade na interpretação do conteúdo web do que o suportado pelos padrões XML, RDF e RDF-Schema (RDF-S) [Valadares *et al.*, 2010].

A OWL possui três sub-linguagens estabelecidas pela W3C com o intuito de serem usadas por diferentes comunidades de implementadores e usuários. São elas: OWL Lite, OWL DL e OWL Full.

2.3.1. As sub-linguagens da OWL

Segue abaixo uma breve explicação de cada uma das três sub-linguagens, com características incrementais, fornecidas pela OWL [Carvalho et al., 2005].

- *OWL Lite:* É sintaticamente a mais simples das sub-linguagens. Indicada a usuários que necessitam de restrições básicas sobre uma hierarquia de classificação simples.
- *OWL DL:* Mais expressiva do que a sub-linguagem OWL Lite, a OWL-DL possibilita computar automaticamente a hierarquia de classes e verificar inconsistências na Ontologia. A sigla "DL" faz alusão a Lógica Descritiva, área de pesquisa que modela conceitos, regras e indivíduos e suas relações através de axiomas lógicos.
- *OWL Full:* Sub-linguagem mais expressiva. Destinada a situações que requerem máxima expressividade e liberdade sintática do RDF. Não há, entretanto, garantias computacionais.

Como citado anteriormente, estas sub-linguagens possuem características incrementais, ou seja, cada uma é uma extensão de sua predecessora. Sendo assim, ontologias válidas em OWL Lite são válidas em OWL DL e, consequentemente, são válidas em OWL Full. Mas esta não é uma relação simétrica, isto é, uma ontologia válida em OWL Full poderá não ser válida em OWL DL [Carvalho et al., 2005].

2.3.2. Estrutura das ontologias em OWL

Levando em conta que os termos descritos em uma ontologia devem ser escritos de forma que possam ser interpretados sem ambiguidades por agentes de software, o primeiro passo durante a escrita de uma ontologia em OWL é a definição de qual vocabulário está sendo empregado. Para tal, deve-se ter um componente inicial na ontologia incluindo um conjunto de *namespaces* contidos na *tag* inicial *rdf:RDF*. *Namespaces* são utilizados na prevenção de colisão de nomes, identificando unicamente os elementos.

```
1  <rdf:RDF
2  xmlns="http://127.0.0.1/ontology/books.owl#"
3  xmlns:base="http://127.0.0.1/ontology/books.owl#"
4  xmlns:book="http://127.0.0.1/ontology/books.owl#"
5  xmlns:owl="http://www.w3.org/2002/07/owl#"
6  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
7  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
8  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
9  >
```

Código 1 – Declaração de namespaces da ontologia *Books*

O exemplo apresentado no Código 1 mostra uma declaração de *namespaces* associados a ontologia *Books*. Na segunda linha é utilizado um *namespace* padrão para a ontologia corrente. Em seguida é identificado um URI base para o documento na terceira linha, e o *namespace* da ontologia *Books* na quarta. Na linha cinco, o *namespace* usado apresenta o vocabulário OWL. Uma vez que a OWL é definida sobre RDF, RDF-Schema e XML-Schema, são definidos *namespaces* para cada uma das linguagens da sexta a oitava linha.

É interessante comentar que a linguagem OWL permite a declaração de entidades XML, como as apresentadas no Código 2 abaixo. Com o uso destas entidades, é possível substituir a declaração do namespace *xmlns:owl= "http://www.w3.org/2002/07/owl#">"* por *xmlns:owl="&owl;"*. De forma similar, caso uma ontologia tenha a necessidade de referenciar uma classe, como *Author*, por exemplo, não é preciso utilizar a forma expandida que consiste no namespace da ontologia *Books* concatenado com o nome classe. Logo, a declaração *http://127.0.0.1/ontology/books.owl#Author* poderia ser compactada da seguinte forma: *&books.owl;Author*.

```
1  <!DOCTYPE rdf:RDF [
2  <!ENTITY books.owl "http://127.0.0.1/ontology/books.owl">
3  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
4  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
5  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
6  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
7  ]>
```

Código 2 – Declaração de entidades XML em ontologia OWL

Uma vez que um documento OWL descreve uma ontologia, é importante fornecer informações sobre a mesma. Isso é feito usando propriedades da OWL e

namespaces XML que são agrupadas dentro da tag *owl:Ontology*, como mostra o código 3.

```
1  <!-- Ontology Information -->
2  <owl:Ontology rdf:about="">
3  <rdfs:label>"Book Ontology"</rdfs:label>
4  <owl:versionInfo>"1.0"</owl:versionInfo>
5  <rdfs:comment>An ontology containing information about books</rdfs:comment>
6  </owl:Ontology>
```

Código 3 – Cabeçalho da ontologia *Books*

O elemento *owl:Ontology* é o responsável por fornecer as informações sobre o documento OWL, ou seja, a ontologia que está sendo descrita. Dessa forma, a segunda linha é a responsável por indicar que o bloco descreve a ontologia do próprio documento em questão, ou seja, indica que a ontologia é a identificada pelo URI base (descrito anteriormente, no Código 1).

Em seguida, o elemento *rdfs:label* permite que seja fornecido um rótulo para a ontologia escrito em linguagem natural e *rdfs:comment* permite a inclusão de comentários sobre a ontologia. Por fim, *owl:versionInfo* indica a versão da ontologia.

Também é possível identificar a versão anterior da ontologia que está sendo criada, como mostra o elemento *owl:priorVersion* no exemplo do Código 4, que apresenta o cabeçalho da ontologia *Wine*.

```
1  <!-- Ontology Information -->
2  <owl:Ontology rdf:about="">
3  <rdfs:comment>An example OWL ontology</rdfs:comment>
4  <owl:versionInfo>2.0</owl:versionInfo>
5  <owl:priorVersion>
6  <owl:Ontology rdf:about="http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine"/>
7  </owl:priorVersion>
8  <owl:imports rdf:resource="http://127.0.0.1/ontology/food.owl"/>
9  <rdfs:comment>
10 Containing information about wine
11 </rdfs:comment>
12 <rdfs:label>Wine Ontology</rdfs:label>
13 </owl:Ontology>
```

Código 4 – Cabeçalho da ontologia *Wine*

Na linha 8 do Código 4, está a declaração do elemento *owl:imports*, que permite a referência de outra ontologia contendo definições que serão usadas como parte do significado da ontologia em questão. Este elemento deve receber uma referência URI, identificado no atributo *rdf:resource*.

Vale ressaltar que as declarações *owl:import* são transitivas, ou seja, se a ontologia A importa B e B importa C, logo a ontologia A importa B e C [Carvalho *et al.*, 2005].

Após a declaração do cabeçalho há então a definição dos elementos da ontologia que são finalmente fechados com a tag de fim *</rdf:RDF>*. Na seção a seguir serão apresentados os elementos básicos de uma ontologia, descritos em OWL.

As ontologias *Books* e *Wine*, utilizadas como exemplos neste trabalho, foram adquiridas do projeto OWLS-MX.

2.3.3. Elementos básicos de uma ontologia em OWL

Como foi apresentado da Seção 2.2.2 sobre ontologias, os elementos básicos para a construção de uma ontologia são as classes, instâncias – também chamadas de indivíduos – e os relacionamentos, ou seja, as propriedades que representam os relacionamentos entre classes e instâncias.

As classes são um mecanismo de abstração utilizado para agrupar os recursos que possuem características similares, isto é, indivíduos que compartilham algumas propriedades. Em OWL, cada indivíduo é membro da classe *owl:Thing*, logo pode-se afirmar que esta é uma superclasse de todas as classes OWL que serão descritas por usuários. Em contrapartida existe também a classe *owl:Nothing*, sendo uma subclasse de todas as classes OWL. Sintaticamente, uma classe em OWL é representada como uma instância nomeada da *owl:Class*, sendo esta uma subclasse da *rdfs:Class* [Carvalho *et al.*, 2005].

Uma classe da ontologia Wine, por exemplo, pode ser definida em OWL da seguinte maneira: *<owl:Class rdf:ID="Region" />*. Logo, a sintaxe *rdf:ID="Region"* é usada para nomear a classe que definirá na ontologia as regiões onde os vinhos são produzidos. Dentro do documento, esta classe pode ser referenciada pela expressão *#Region* e, caso outros ontologias precisem referenciá-la, é possível utilizar a versão estendida composta pela concatenação do namespace da ontologia com o nome da classe ou, caso tenham sido declaradas entidades XML como foi apresentado na seção anterior, é possível utilizar a versão compacta *&wine;Region* ou a tag XML *wine:Region*

Em OWL pode-se representar as hierarquias entre classes através da construção *rdfs:subClassOf*. O Código 5, por exemplo, declara que a classe *Author* na ontologia *Books* é uma subclasse da classe *Person*, definindo assim que os indivíduos da classe *Author* devem ser constituídos por um subconjunto dos indivíduos presentes na classe *Person*.

```
1  <owl:Class rdf:ID="Author">
2  <rdfs:subClassOf rdf:resource="#Person"/>
3  </owl:Class>
4
```

Código 5 – Exemplo de hierarquia de classes na ontologia Books.

Indivíduos são definidos como fatos, ou seja, uma declaração que sempre será verdadeira em um determinado domínio. Para o exemplo da classe *Region*, da ontologia *Wine*, um indivíduo pode ser declarado da seguinte forma: *<Region rdf:ID="Portugal" />*. Pode-se afirmar então que *Portugal* é uma das regiões de produção de vinhos.

As propriedades, por sua vez, são os elementos responsáveis por estabelecer relações binárias em uma ontologia, tanto entre indivíduos quanto entre indivíduos e valores de dados. Tais relacionamentos tornam possível a afirmação de fatos gerais sobre os membros das classes e podem também representar fatos específicos acerca de indivíduos.

Há na OWL, duas principais categorias de propriedades. As *datatype properties* representam as propriedades que relacionam indivíduos e valores enquanto *object properties* são responsáveis por estabelecer as relações entre os próprios indivíduos.

Propriedades de objeto (*object properties*) são definidas como instâncias da classe *owl:ObjectProperty* e as propriedades de dado tipado *(datatype properties)* como instâncias de *owl:DatatypeProperty*. E ambas, por sua vez, são subclasses da classe RDF rdf:Property, herdando portanto as propriedades do RDF Schema, *rdfs:SubPropertyOf*, *rdfs:domain* e *rdfs:range*.

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
<rdfs:subPropertyOf rdf:resource="&food;madeFromFruit" />
<rdfs:domain rdf:resource="#Wine" />
<rdfs:range rdf:resource="#WineGrape" />
</owl:ObjectProperty>
```

Código 6 – Uso de propriedade do tipo *owl:ObjectProperty*

O exemplo descrito no Código 6 apresenta a utilização de uma propriedade de objeto na ontologia *Wine*. A propriedade *madeFromGrape* tem como domínio (*domain*) a classe *Wine*, ou seja, deve ser aplicada somente nos indivíduos que são instâncias dessa classe. Além disso, seu valor (*range*) é a classe *WineGrape*, representando que os valores da propriedade devem pertencer à extensão de classe da classe *WineGrape*.

Por fim, pode-se observar no Código 6 um exemplo de hierarquia entre propriedades. A declaração *rdfs:subPropertyOf* define que a propriedade é uma sub-propriedade de outra. A propriedade *madeFromGrape* é então uma sub-propriedade de *madeFromFruit*, propriedade da ontologia *Food.*

```
<owl:DatatypeProperty rdf:ID="yearValue">
<rdfs:domain rdf:resource="#VintageYear" />
<rdfs:range  rdf:resource="&xsd;positiveInteger" />
</owl:DatatypeProperty>

```

Código 7 – Uso de propriedade do tipo owl:DatatypeProperty

No Código 7 temos o exemplo de uma propriedade do tipo *datatype property* aplicada na ontologia *Wine*. A propriedade *yearValue* relaciona a classe *VintageYear* a um inteiro positivo, representando que uma safra de vinho possui um ano de produção relacionado. O valor do recurso *rdf:resource=”&xsd;positiveInteger”* é um dado tipado definido na XML Schema.

Como mostra o Código 8, também é possível declarar propriedades sobre indivíduos de uma ontologia em OWL. O trecho abaixo declara o indivíduo *StGenevieveTexasWhite* como pertencente a classe *WhiteWine* e também declara, através das propriedades *locatedIn*, *hasMaker*, *hasSugar* e *hasFlavor*, o responsável

pela produção do vinho, sua localização, o nível de açúcar do vinho e o tipo de sabor do mesmo.

```
1  <WhiteWine rdf:ID="StGenevieveTexasWhite">
2  <locatedIn rdf:resource="#CentralTexasRegion" />
3  <hasMaker  rdf:resource="#StGenevieve" />
4  <hasSugar  rdf:resource="#Dry" />
5  <hasFlavor rdf:resource="#Moderate" />
6  </WhiteWine>
```

Código 8 – Declaração de propriedades para indivíduos

É possível ainda especificar características para propriedades, definir se são transitivas, simétricas, funcionais, funcionais inversas, ou se representam o inverso de uma outra propriedade. A OWL permite também que sejam impostas restrições sobre as propriedades definidas. Cada restrição é um tipo especial de descrição de classe, descrevendo uma classe anônima de indivíduos que satisfazem tais restrições. Estas podem ser de valores ou de cardinalidade.

2.4. Serviços Web Semânticos

Esta seção apresenta os principais conceitos sobre serviços web semânticos.

2.4.1. Definição

Os serviços web são a base das arquiteturas orientadas a serviços, porém, a falta de suporte semântico em sua linguagem de descrição de serviços, WSDL, dificulta a execução de algumas tarefas, tais como a descoberta e composição de serviços [Forte *et al.*, 2006]. Uma solução para este cenário é a incorporação da semântica na representação de requisitos e capacidades dos serviços web. Tal necessidade por semântica proporcionou o encontro entre serviços web e a web semântica, tendo como fruto dessa convergência de conceitos o aparecimento dos serviços web semânticos. Um serviço web semântico é, portanto, um serviço Web onde as propriedades, capacidades e interfaces são descritas de forma não ambígua e computacionalmente interpretável [Amorim, 2009].

A adoção de ontologias é uma das alternativas mais utilizadas para atribuir semântica a descrição de serviços. A utilização destas enriquece as descrições dos

serviços de forma a uma maior automação nas tarefas de descoberta, seleção e composição de serviços.

2.4.2. Representação de serviços web semânticos

Em geral, um serviço web semântico é composto por seu documento de descrição do serviço, WSDL, e um documento que o descreve semanticamente, escrito em uma linguagem específica de forma a representar as ontologias do serviço. Para tal fim, várias linguagens foram desenvolvidas. Entre as mais conhecidas estão a SAWDL, a WSMO e a OWL-S [Orlandin, 2005]. Segue abaixo uma explicação breve sobre estas linguagens:

- *SAWDL (Semantic Annotation For WSDL):* A SAWDL é um conjunto de extensões para a WSDL. Esta linguagem permite que documentos WSDL possuam ponteiros para descrições semânticas.
- *WSMO (Web Service Modeling Ontology):* A linguagem se baseia nos princípios de compatibilidade com padrões Web, através do uso de URIs, namespaces e ontologias, provendo um framework que suporta a modelagem dos serviços web em descrição semântica [Alves *et al.*,2011].
- *OWL-S (Semantic Markup for Web Services):* A OWL-S é uma linguagem baseada na OWL e utiliza três sub-ontologias para descrever e definir semanticamente os serviços web.

Uma vez que o objetivo deste trabalho é a aplicação de métricas de similaridade para gerar uma melhoria no algoritmo da ferramenta OWL-S Discovery, na próxima seção nos aprofundaremos nos conceitos de OWL-S.

2.4.3. OWL-S

OWL-S surgiu como uma iniciativa de atribuir significado semântico a serviços. Seu formato nada mais é do que uma ontologia que fornece um vocabulário específico para descrição de serviços web semânticos

OWL-S define uma ontologia principal, denominada *Service*, e outras três sub-ontologias, *Profile, Model e Grounding*, conforme representado na Figura 5.

Estas três descrevem, respectivamente, o que o serviço provê ao seu consumidor, como o serviço pode ser usado e como interagir com o serviço.

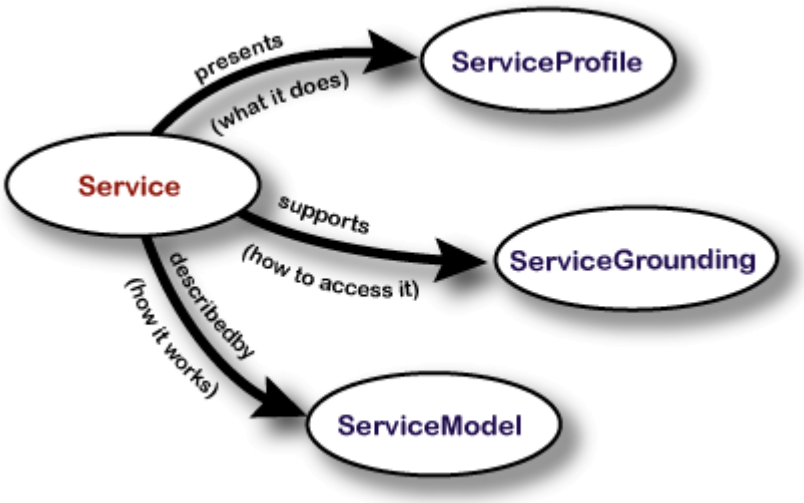

Figura 5 – Modelo da Ontologia Service [Martin *at al.*, 2004]

2.4.3.1. Ontologia *Service*

OWL-S é fundamentada principalmente na ontologia *Services*. Todo serviço descrito neste formato, deve possuir uma instância de *Service*, responsável por referenciar as demais ontologias em que se baseia, conforme demonstrado no exemplo do Código 9, em que se referencia a ontologia Profile por "presents", a ontologia Model, por "describedBy", e a ontologia Grounding, por "supports".

```
1  <service:Service rdf:ID="BOOK_AUTHOR_SERVICE">
2  <service:presents rdf:resource="#BOOK_AUTHOR_PROFILE"/>
3  <service:describedBy rdf:resource="#BOOK_AUTHOR_PROCESS_MODEL"/>
4  <service:supports rdf:resource="#BOOK_AUTHOR_GROUNDING"/>
5  </service:Service>
```

Código 9 – Referência à sub-ontologias

2.4.3.2. Ontologia Profile

A ontologia Profile é responsável por especificar características funcionais dos serviços, tais como interfaces de entrada e saída, pré-condições, efeitos, e características não funcionais, como categoria e qualidade do serviço (QoS).

Através das especificações feitas por esta ontologia, é possível realizar a busca automática por serviços [Breitman, 2005]. Esta é a principal ontologia utilizada pelo presente trabalho. A Figura 6 demonstra suas principais classes e propriedades:

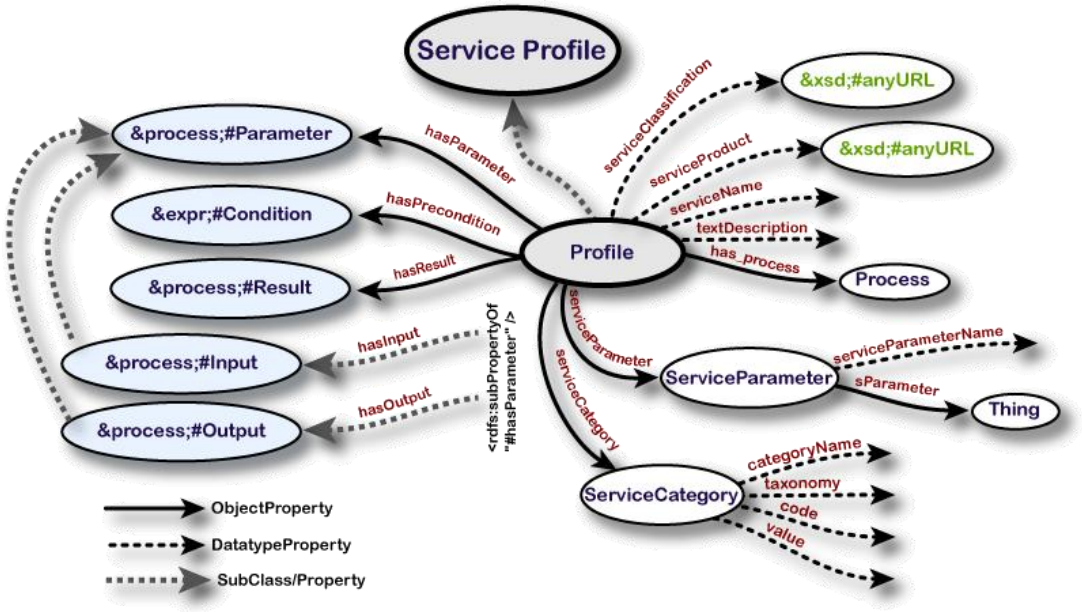

Figura 6 – Classes e propriedades da ontologia Profile [Martin *at al.*, 2004]

2.4.3.3. Ontologia Model

A ontologia *Model* descreve informações sobre a forma de interação com serviços atômicos ou compostos durante a execução de um serviço e a troca de mensagens entre eles. Esta interação é definida por um processo.

Um serviço composto pode ser definido como um serviço que em sua sub-ontologia Model contém informações de invocação de outros serviços, atômicos ou compostos, encapsulados em uma das seguintes estrutura de controle: *Sequence*, *SplitJoin*, *Split*, *Anyorder*, *Choose*, *If-Then-Else* e *Repeat-Until*. Já um serviço atômico não possui invocações de outros serviços nessa sub-ontologia.

3. OWL-S DISCOVERY

Este capítulo tem como objetivo explicar o funcionamento e a evolução da ferramenta de descoberta de serviços web semânticos, OWL-S Discovery.

3.1.1. Definição

No ciclo de vida de um serviço web, a descoberta é um passo importante para a realização das etapas seguintes. Uma falha durante esta tarefa compromete as demais etapas do ciclo e pode gerar erros de execução e efeitos indesejados na invocação do serviço [Erdens, 2010].

A descrição semântica dos serviços web possibilita que a descoberta seja feita de forma automática, através da utilização de um algoritmo de correspondência (*alinhamento algorithm*). Para a execução de tal tarefa, diversos algoritmos foram desenvolvidos [Sena *et al.*, 2010].

O OWL-S Discovery servirá de base para este trabalho, sendo um algoritmo híbrido para descoberta de serviços, contando com uma etapa semântica funcional e uma semântica descritiva.

3.1.2. A Descoberta de Serviços Web Semânticos

A Figura 7 apresenta o ciclo de vida de um serviço web:

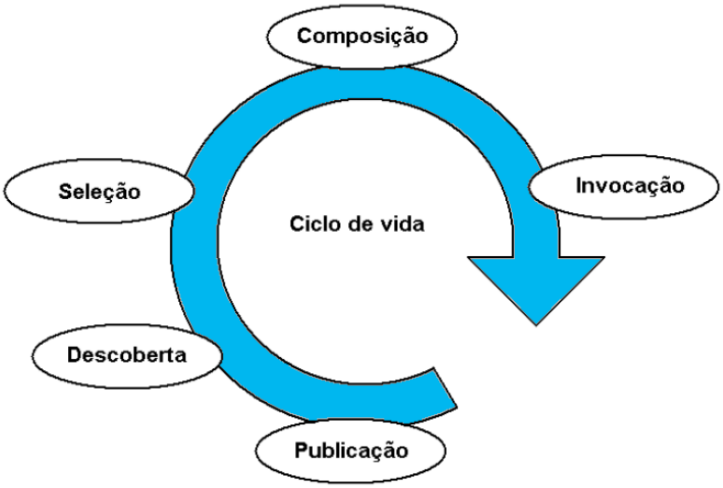

Figura 7 – Ciclo de vida de um serviço web [Amorim *et al.*, 2009]

A publicação de um serviço web se refere à etapa de divulgação do mesmo, ou seja, torná-lo acessível aos usuários. A descoberta é a etapa onde são buscados os serviços com características semelhantes às requisitadas pelo usuário. Uma vez descobertos, esses serviços são selecionados escolhendo-se os que mais se assemelham aos requisitos do usuário. Caso a requisição do usuário possa ser atendida por um único serviço, a etapa de composição não será necessária. Por fim, a há a invocação do serviço, etapa onde os parâmetros são fornecidos pelo usuário e o serviço é finalmente executado [Amorim, 2009].

A descoberta é, portanto, uma tarefa de extrema importância uma vez que todas as etapas subsequentes dependem intrinsecamente dela, ou seja, não é possível executar um serviço web sem antes descobri-lo [Cardoso, 2007].

A maior parte dos algoritmos utiliza as informações da ontologia *Profile* para realizar a descoberta de serviços web semânticos. Dentre todas as informações providas por essa ontologia as mais importantes são o conjunto conhecido como IOPE (*Input, Output, Preconditions e Effects*) [Amorim, 2009].

O Conjunto de Entrada (*Input*) determina quais os dados necessários para a execução de um serviço. O Conjunto de saída (*Output*) corresponde ao retorno do serviço, ou seja, as informações que serão produzidas após a execução do mesmo. As Pré-Condições (*Preconditions*) se referem a todos os requisitos que devem ser satisfeitos para que a execução do serviço seja feita com sucesso. Por fim, os Efeitos (*Effects*) especificam o estado depois da execução de um serviço web, eles são as consequências geradas pela invocação do serviço [Erdens, 2010].

Em [Paolucci *et al.*, 2002], foi proposto um algoritmo de descoberta baseado na capacidade semântica dos termos analisados. Este algoritmo corresponde os atributos de entrada e saída da requisição, com os atributos de entrada e saída dos serviços. O grau de similaridade entre os termos, também chamado de filtro, é baseado no relacionamento que estes possuem em uma determinada ontologia, analisando seus descendentes e ascendentes. A partir dessa análise, Paolucci apresentou quatro possíveis filtros, descritos abaixo:

- *Exact:* Dois outputs são ditos Exact caso sejam exatamente iguais ou se o output do serviço é descendente direto do output da requisição.

- *Plug-in:* Dois outputs são definidos como Plug-in quando o output do serviço possui como descendente o output da requisição (exceto no caso de descendência direta).
- *Subsumes:* Os outputs serão Subsumes caso o output da requisição possua como descendente o output do serviço (exceto o caso de descendência direta).
- *Fail:* Esse filtro representa que não há relacionamento semântico entre os outputs dentro de uma determinada ontologia.

Em relação ao conjunto de entradas, dois inputs serão Plug-in se o input da requisição possuir como descendente o input do serviço, serão Subsumes se o input do serviço possuir como descendente o input da requisição e Exact caso o input da requisição seja igual ou descendente direto ao input do serviço [Erdens, 2010].

Em [Samper *et al.*, 2008] foi proposto ainda o filtro Sibling. Esse filtro determina se dois outputs ou inputs possuem um ascendente direto em comum, ou seja, se são irmãos. Então, em ordem decrescente de similaridade, temos: Exact > Plug-in > Subsumes > Sibling > Fail.

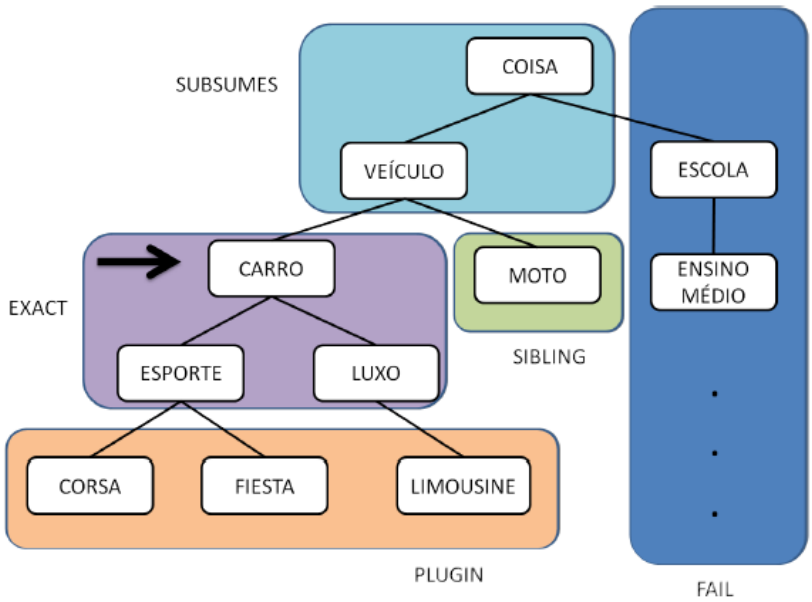

Figura 8 – Representação gráfica da ontologia para o termo CARRO [Erdens et al., 2010]

Analisando a Figura 8, que representa uma ontologia cujo termo raiz é COISA, ao destacar o termo CARRO, temos os seguintes filtros: COISA E

VEÍCULO são Subsumes de CARRO, MOTO é Sibling, ESPORTE E LUXO são Exact, CORSA, FIESTA E LIMOUSINE são Plug-in e, por fim, ESCOLA E ENSINO MÉDIO são Fail.

3.1.3. Algoritmo OWL-S Discovery

O algoritmo da ferramenta OWL-S Discovery [Amorim, 2009] tem como base o uso dos filtros de Paolucci e Samper durante a etapa semântica funcional e utiliza um dicionário de sinônimos [Lopes *et al.*, 2006] em sua fase semântica descritiva. A Figura 9 ilustra as etapas de sua execução.

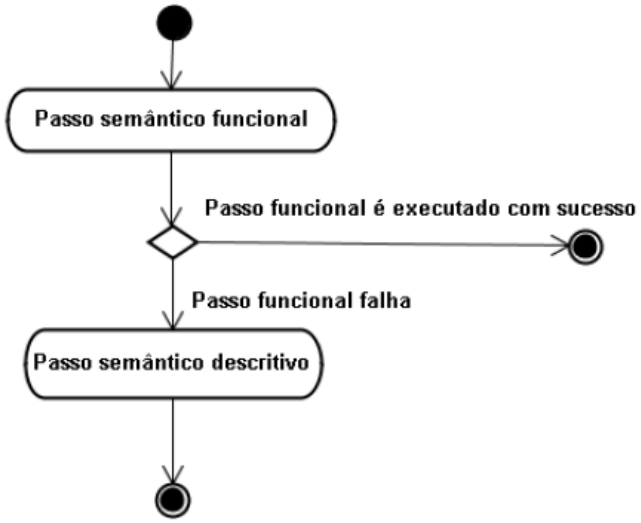

Figura 9 – Execução do OWL-S Discovery [Sena et al., 2010].

É necessário, para o funcionamento do algoritmo, um diretório com as descrições dos serviços web onde a busca será efetuada, um meio de acesso às ontologias que são referenciadas nas descrições e uma requisição de consulta do usuário [Sena *et al.*, 2010].

Na etapa semântica funcional, quando os parâmetros da requisição e do serviço pertencem a mesma ontologia, o algoritmo efetua uma correspondência entre todos os parâmetros dos serviços do repositório com os parâmetros presentes na requisição do usuário. Caso estejam em diferentes ontologias, o algoritmo retorna Fail.

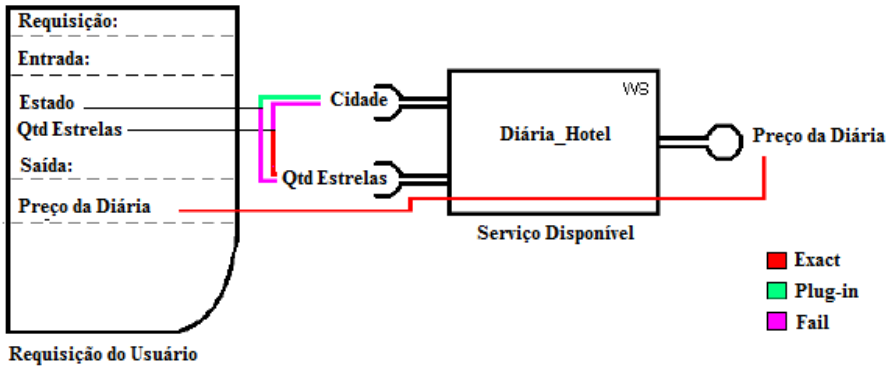

Figura 10 – Processo de correspondência entre parâmetros de uma requisição e de um serviço [Amorim *et al.*, 2009]

Como é possível perceber na Figura 10, o parâmetro *Estado* possui o grau Plug-In com *Cidade* e Fail com *Qtd Estrelas*. O parâmetro de entrada da requisição, *Qtd Estrelas*, se relaciona como Exact com o parâmetro de input do serviço (*Qtd Estrelas*) e Fail com *Cidade*. A saída da requisição, *Preço da Diária*, apresenta, por sua vez, uma relação Exact com o parâmetro de saída do serviço.

No algoritmo OWL-S Discovery, são considerados então os melhores relacionamentos entre os requisitos, ou seja, no final do processo de correspondência, o cenário ficará conforme o apresentado na Figura 11.

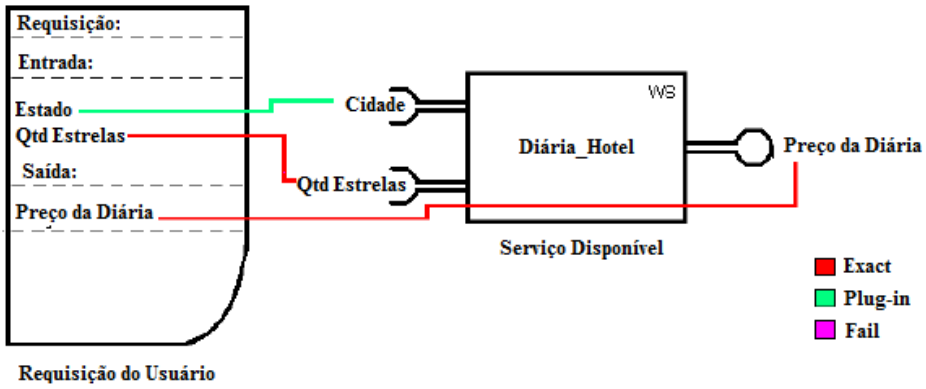

Figura 11 – Resultado do processo de correspondência entre parâmetros de uma requisição e de um serviço [Amorim *et al.*, 2009]

Esse processo de correspondência será efetuado para todos os serviços presentes no repositório. O grau de similaridade de um serviço é representado pelo

menor filtro presente nos relacionamentos entre parâmetros. No exemplo acima, o grau de similaridade do serviço em relação à requisição utilizada, será Plug-in.

A etapa semântica descritiva, por sua vez, se baseia no trabalho de [Lopes *et al.*, 2006] fazendo uma comparação entre as classes (parâmetros da requisição e do serviço) e uma consulta ao dicionário de sinônimos fornecido pelo usuário.

O algoritmo utiliza uma função que calcula a similaridade entre duas classes, retornando 1.0 (um) caso as classes sejam idênticas, 0.0 (zero) se forem diferentes e 0.5 (meio) caso tenham o mesmo significado de acordo com o dicionário consultado. Aliado a este cálculo de similaridade, foi implementado o uso de uma comparação estrutural entre os vizinhos das classes analisadas [Lopes *et al.*, 2006].

3.1.4. OWL-S Discovery 2.0

O algoritmo da ferramenta OWL-S Discovery 1.0 baseia-se somente nos conjuntos de entrada e saída dos serviços. Para muitos serviços, porém, existe um conjunto de condições que devem ser atendidas para que o este seja executado corretamente. Esse conjunto é denominado pré-condições e uma análise focada apenas nas entradas e saídas não é capaz de determinar se as pré-condições do serviço em questão serão atendidas. Nesse caso, o serviço descoberto estará propenso a falhas de execução.

Outro conjunto de informações relevantes no momento da descoberta é o conjunto de efeitos. Tais informações definem quais as consequências da execução do serviço. Uma busca sem considerar os efeitos, pode gerar consequências indesejadas ao usuário ao deixar de produzir algum efeito esperado.

De forma a aprimorar a tarefa de descoberta realizada pelo OWL-S Discovery, um algoritmo de pré-condições e efeitos foi implementado por [Erdens, 2010] sobre as bases do OWL-S Discovery [Amorim, 2009] provendo a versão 2.0 da ferramenta.

Para possibilitar essa adaptação, foi usada a versão OWL-S 1.2 na descrição dos serviços, pois esta tem suporte à SWRL, linguagem que permite a representação de pré-condições e efeitos. A SWRL é uma combinação das sublinguagens OWL-DL e OWL Lite com a DataLog RuleML, sublinguagem da RML (Rule Markup

Language), que permite que regras sejam combinadas com uma base de conhecimento OWL.

Uma regra SWRL é composta por um predicado e dois atributos. Por exemplo, caso se queira estabelecer uma regra para validar se um usuário possui cartão de crédito, é necessário verificar se existe um usuário com uma conta do tipo cartão. Dessa forma, esta regra seria dividida em Predicado: temCartão e Atributos: Usuário e ContaCartão [Erdens, 2010]. No apêndice A é exibido o exemplo de um serviço escrito em OWL-S 1.2 cujas pré-condições e efeitos estão escritos em SWRL.

Após a comparação das entradas e saídas realizada pelo algoritmo original, a versão 2.0 realiza uma comparação entre as pré-condições e efeitos do serviço e da requisição utilizando-se de uma análise semântica do predicado e de seus respectivos atributos cujos termos estão descritos em uma ontologia.

3.2. OWL-S Discovery 3.0

Na versão apresentada pelo presente trabalho, acrescentou-se a forma de entrada de uma requisição em OWL-S para termos em texto livre, para assim facilitar a busca do usuário. Nas versões anteriores, não há tratamento para termos presentes em ontologias distintas, mesmo que estes sejam idênticos. De forma a flexibilizar a tarefa de descoberta de serviços executada pela ferramenta, este trabalho também propõe, como solução desta limitação, a aplicação de métricas de similaridade baseadas em string. Os conceitos destas métricas, assim como a explicação das métricas aplicadas neste trabalho, serão apresentados no próximo capítulo.

4. MÉTRICAS DE SIMILARIDADE

Este capítulo tem por objetivo apresentar o conceito de métricas de similaridade, suas classificações e apresentar as métricas escolhidas para serem aplicadas na evolução da ferramenta OWL-S Discovery 2.0.

4.1. Definição

Com a crescente utilização de ontologias na área de tecnologia da informação, a heterogeneidade entre os domínios representados por cada ontologia é algo inevitável. Para que as informações disponíveis na Web sejam utilizadas por diferentes sistemas, estas devem ser localizadas, compreendidas e processadas pelos mesmos, logo é preciso tratar a heterogeneidade dessas informações, compatibilizar seus conteúdos. Várias métricas foram então desenvolvidas para tratar desse problema, efetuando cálculos de similaridade entre as entidades das ontologias com o objetivo de encontrar correspondências entre as mesmas [Revoredo *et al.*, 2013].

A combinação e integração das informações nas ontologias disponíveis e o acesso a essas informações para consultas e raciocínio é um dos pilares da Web Semântica [Padilha *et al.*, 2012].

4.2. Classificação das métricas de similaridade

Segundo [Shvaiko *et al.*, 2004] as métricas de similaridade se dividem em duas classificações: métricas no nível de elemento e técnicas no nível de estruturas.

As métricas de similaridade no nível de elementos consideram as entidades de uma ontologia e suas instâncias de forma isolada, ou seja, ignorando possíveis relacionamentos com outras entidades ou instâncias. Essas técnicas podem ser subdivididas em:

- *Métricas baseadas em string:* Essas métricas interpretam uma string como sendo uma sequência de letras pertencentes a determinado alfabeto e são frequentemente utilizadas para casar nomes de entidades e suas descrições, considerando que quanto mais similares

são as cadeias de caracteres, maior a probabilidade das entidades corresponderem ao mesmo conceito.

- *Métricas baseadas em linguagens:* As métricas baseadas em linguagens são baseadas em técnicas de processamento que exploram propriedades morfológicas das palavras, considerando nomes como sendo palavras em alguma linguagem natural.
- *Métricas baseadas em restrições:* Grupo de métricas composto por algoritmos que lidam com restrições internas aplicadas às definições das entidades, como seus tipos de dados, cardinalidade de atributos e instâncias.
- *Recursos Linguísticos:* Recursos como dicionários de domínio são utilizados na tentativa de combinar palavras com base nos relacionamentos linguísticos entre elas, tais como sinônimos e hipônimos.

Métricas de similaridade no nível estrutural, por sua vez, consideram as entidades de uma ontologia e suas instâncias de forma a comparar suas relações com outras entidades ou instâncias. Essas métricas podem ser subdivididas em: baseadas em grafos, em taxonomias, repositório de estruturas, entre outras subdivisões.

Este trabalho se utilizará apenas de métricas no nível de elemento, focando nas baseadas em string.

4.3. Métricas baseadas em string

As métricas baseadas em string trabalham na estrutura de uma string, visualizando-a como uma sequência de letras. Por exemplo, tais métricas analisarão as palavras *Book* e *Textbook* identificando-as como similares, porém, não encontrarão similaridade entre *Book* e *Volume*.

As métricas com base em string possuem ainda subdivisões, sendo as mais conhecidas:

i) *Métricas de normalização*: Utilizadas para reduzir strings de forma que possam ser comparadas em um mesmo formato. Essas métricas são usualmente aplicadas antes da comparação entre strings, sendo um conjunto de procedimentos que podem ser utilizados de forma a

otimizar a aplicação de outras métricas. Entre as métricas de normalização, temos a normalização de *case* (onde todas as letras são convertidas para um único estilo, por exemplo, todas maiúsculas ou todas minúsculas), retirada de acentuação, normalização de brancos (todos os caracteres em branco são substituídos por um único caracter em branco), supressão de dígitos, eliminação da pontuação e *link stripping* (eliminação de links, como hífen ou underline, entre as palavras).

ii) *Métricas de substring*: Baseiam-se na similaridade entre as strings de acordo com os caracteres que possuem em comum.

iii) *Distância de edição*: Avaliam como uma string pode ser uma versão errônea de outra string.

Nas subseções a seguir é possível encontrar uma breve descrição sobre as métricas citadas neste trabalho.

4.3.1. Jaro

A métrica Jaro foi definida para a identificação de palavras contendo algum erro de escrita. Sendo *n* o número de caracteres comuns entre as duas strings, *t* o número de transposições necessárias e *x* e *y* sendo o número de caracteres das strings *X* e *Y*, Jaro calcula a similaridade entre as strings *X* e *Y* da seguinte forma:

$$Jaro(X,Y) = 1/3 \left((n/x) + (n/y) + (n-t/n) \right).$$

O número de caracteres em comum, que ocupam diferentes posições nas duas strings, dividido por dois define o número de transposições (*t*) utilizado no cálculo acima.

Como exemplo, considere x a *string Martha* e y a *string Marhta*, por exemplo, teríamos *n*, *x* e *y* igual a 6 e o número de transposições (T/H e H/T) seria 2/2, logo, *t*=1. O grau de similaridade encontrado pela métrica Jaro seria então de aproximadamente 0.94.

4.3.2. Jaro Winkler

Jaro Winkler é uma evolução da métrica Jaro, formulada de forma a favorecer comparação entre strings com prefixos em comum. Sendo *p* o número de caracteres do prefixo e *c* uma constante, o grau de similaridade é calculado pela fórmula: $JaroWinkler(x,y) = Jaro(x,y) + p \times c\ (1 - Jaro(x,y))$.

Assumindo *c=0.1*, valor padrão da constante, o grau de similaridade entre *Martha* e *Marhta*, sendo *p=3*, seria então de aproximadamente 0.96.

4.3.3. Levenshtein

A métrica da distância de Levenshtein, por sua vez, representa o número mínimo de inserções, deleções e substituições de caracteres necessários para transformar uma *string* em outra. As palavras *Kitten* e *Sitting*, por exemplo, teriam a distância de Levenshtein igual a 3:

i) Kitten

ii) Sitten (substituição de *k* por *s*)

iii) Sittin (substituição de *e* por *i*)

iv) Sitting (inserção de *g*)

A Figura 12 representa a matriz de substituição resultante da aplicação da métrica Levenshtein para as palavras *Kitten* e *Sitting*. As matrizes de substituição surgem da necessidade de atribuir um valor ao alinhamento de cada par de caracteres. Na métrica Levenshtein, as operações de inserção, remoção e substituição de caracteres tem valor um.

		K	I	T	T	E	N
	0	1	2	3	4	5	6
S	1	1	2	3	4	5	6
I	2	2	1	2	3	4	5
T	3	3	2	1	2	3	4
T	4	4	3	2	1	2	3
I	5	5	4	3	2	2	3
N	6	6	5	4	3	3	2
G	7	7	6	5	4	4	3

Figura 12 – Matriz de substituição para a métrica Levenshtein

4.3.4. Needleman-Wunch

A métrica de distância de Needleman-Wunch calcula a similaridade entre strings de forma similar a distância de Levenshtein. Ela atribui pesos para igualdade (*match*), desigualdade (*mismatch*) e inserção (*gap*). Considerando a igualdade como peso 1 e desigualdade e espaçamento como -1, a Figura 13 ilustra a aplicação desta métrica para as *strings* “AACGTTAC” e “CGATAAC”.

		A	A	C	G	T	T	A	C
	0	-1	-2	-3	-4	-5	-6	-7	-8
C	-1	-1	-2	-1	-2	-3	-4	-5	-4
G	-2	-2	-2	-2	0	-1	-2	-3	-4
A	-3	-1	-1	-2	-1	-1	-2	-1	-2
T	-4	-2	-2	-2	-2	0	0	-1	-2
A	-5	-3	-1	-2	-3	-1	-1	1	0
A	-6	-4	-2	-2	-3	-2	-2	0	0
C	-7	-5	-3	-1	-2	-3	-3	-1	1

Figura 13 – Matriz de substituição para a métrica Needleman-Wunch

4.3.5. Smith Waterman

Esta métrica é uma variação da distância de Needleman-Wunch. Sua principal diferença é definir como zero as células da matriz de substituição com valores negativos. A Figura 14 apresenta a aplicação desta métrica para as *strings* “ACACACTA” e “AGCACACA”, considerando a igualdade como peso 2, espaçamento como 0 e desigualdade como -1.

		A	C	A	C	A	C	T	A
	0	0	0	0	0	0	0	0	0
A	0	2	1	2	1	2	1	0	2
G	0	1	1	1	1	1	1	0	1
C	0	0	3	2	3	2	3	2	1
A	0	2	2	5	4	5	4	3	4
C	0	1	4	4	7	6	7	6	5
A	0	2	3	6	6	9	8	7	8
C	0	1	4	5	8	8	11	10	9
A	0	2	3	6	7	10	10	10	12

Figura 14 – Matriz de substituição para a métrica Smith Waterman

4.3.6. Gotoh

O algoritmo Goth é uma variação da métrica Smith Waterman, se diferindo desta por possuir a penalidade *affined gap*, onde há uma penalização específica para a criação de *gaps* (processo de inserção). Esta penalidade atribui um custo ao se abrir uma nova sequência de *gaps* e outro custo para cada *gap* acrescentado.

Considere o custo de uma nova sequência de *gaps* como 2, cada *gap* acrescentado como 0.5, igualdade 0 e desigualdade 1. A matriz de substituição referente à aplicação da métrica Gotoh para as *strings* "AAG" e "AGTAC" está ilustrada na Figura 15.

		GAP	A	A	G
		0	1	2	3
GAP	0	0.0	2.5	3.0	3.5
A	1	2.5	0.0	2.5	3.0
G	2	3.0	2.5	1.0	2.5
T	3	3.5	3.0	3.5	2.0
A	4	4.0	3.5	3.0	4.5
C	5	4.5	4.0	4.5	4.0

Figura 15 – Matriz de substituição para a métrica Gotoh

4.3.7. Euclidean

A métrica de distância euclidiana foi desenvolvida para calcular a distância, D, entre dois pontos, p e q, através da fórmula:

$$Dx(p,q) = \sqrt{\sum_{j=1}^{J} [x_p(j) - x_q(j)]^2}$$

Caso tenha-se, por exemplo, as strings P="ACGT" e Q="ACGTA", a distância euclidiana entre elas pode ser obtida calculando primeiramente a quantidade de vezes que cada caracter aparece, obtendo-se os conjuntos p=[1,1,1,1] e q=[2,1,1,1]. Uma vez calculados os conjuntos, aplica-se os valores na fórmula:

$$d(p,q) = \sqrt{[(1-2)^2 + (1-1)^2 + (1-1)^2 + (1-1)^2]} = \sqrt{1} = 1$$

4.3.8. Matching Coefficient

Este algoritmo calcula a similaridade entre duas strings calculando a diferença entre o comprimento da cadeia de caracteres mais curta e a distância de edição e dividindo-a pelo comprimento da cadeia mais curta.

4.3.9. Overlap

A métrica Overlap, por sua vez, realiza seu cálculo de similaridade contabilizando o número de palavras que duas strings possuem em comum dividido pelo número de palavras na menor string, como demonstrado na fórmula abaixo:

$$\mathrm{sim}_{\mathrm{overlap}}(s_1, s_2) = \frac{c_{\mathrm{common}}}{\min(c_1, c_2)}.$$

4.3.10. Dice Coefficient

O valor obtido a partir do dobro da interseção dos termos comuns sobre a soma dos termos das duas strings comparadas representa o grau de similaridade obtido através da aplicação desta métrica, como demonstrado pela fórmula abaixo, onde c1 e c2 representam as strings que terão seu grau de similaridade calculado:

$$\mathrm{sim}_{\mathrm{dice}}(s_1, s_2) = \frac{2 \times c_{\mathrm{common}}}{c_1 + c_2}$$

4.3.11. City Blocks

Também conhecida como Block Distance, esta métrica calcula a sua similaridade através do somatório da diferença entre as duas strings comparadas.

$$Dx(p, q) = \sum_{j=1}^{J} [x_p(j) - x_q(j)]$$

Aplicando-se a métrica City Blocks às strings P=”ACGT” e Q=”ACGTA”, o grau de similaridade entre elas pode ser obtido calculando-se primeiramente a quantidade de vezes que cada caracter aparece, obtendo-se os conjuntos p=[1,1,1,1] e q=[2,1,1,1]. Uma vez calculados os conjuntos, aplica-se os valores na fórmula:

$$d(p,q) = [(1-2) + (1-1) + (1-1) + (1-1)] = -1$$

4.3.12. Cosine

O algoritmo Cosine calcula a similaridade entre duas strings dividindo-as em um vetor contendo suas cadeias de caracteres e outro contendo tokens destas cadeias, associa-se então um peso a cada token de acordo com a frequência que estes aparecem no vetor de cadeia de caracteres.

4.3.13. N-Grams

A métrica N-Grams converte as strings em conjuntos de n-grams e os compara usando algumas métricas de similaridade como Cosine e Dice. Vale comentar que um n-gram é formado por uma sequência de n itens de um determinado texto. Tais itens podem ser sílabas, letras ou palavras de acordo com a aplicação.

4.3.14. TFIDF

Similar à métrica Cosine, o algoritmo TFIDF também retorna seu grau de similaridade com base em uma análise estatística que reflete a importância de uma palavra dentro de um conjunto de palavras.

5. ALGORITMO PROPOSTO

O objetivo deste capítulo é explicar o funcionamento do algoritmo proposto que implementa uma etapa de análise sintática ao algoritmo OWL-S Discovery 2.0, de forma a flexibilizar a tarefa de busca por serviços web semânticos.

5.1. Funcionamento do Algoritmo Proposto

A execução do OWL-S Discovery 3.0 está condicionado à existência de arquivos OWL-S dos serviços – feitos através de *upload* pela aplicação –, uma requisição, seja ela um arquivo OWL-S ou argumentos livres, e um meio de acesso às ontologias referenciadas pelos arquivos.

O início da execução do algoritmo semântico funcional pode ser feito de duas formas. A primeira delas mantém a forma original do OWL-S Discovery 2.0, utilizando arquivos OWL-S como forma de requisição e um diretório de arquivos OWL-S de serviços a serem analisados, exceto pelo fato de que na atual versão será feito o *upload* dos arquivos, ao invés de apontamento de diretórios, por se tratar de uma aplicação web. A segunda forma apresenta uma requisição constituída por argumentos em texto livre, ou seja, o usuário apenas escreve as entradas e saídas desejadas para iniciar sua busca.

O algoritmo aqui proposto entra em ação na segunda forma apresentada, aplicando uma análise sintática para o reconhecimento dos parâmetros imputados pelo usuário. Dessa forma, para cada combinação de entradas e saídas com os serviços analisados é feita uma busca sintática na ontologia a qual o parâmetro do serviço em questão pertence, de modo a localizar dentro desta qual é a classe mais similar ao argumento livre. Então é dada continuidade ao fluxo original de alinhamento, considerando tal classe como o parâmetro da requisição. A figura abaixo apresenta, de forma simplificada, o funcionamento do OWL-S Discovery 3.0 no caso de uma requisição composta por argumentos livres.

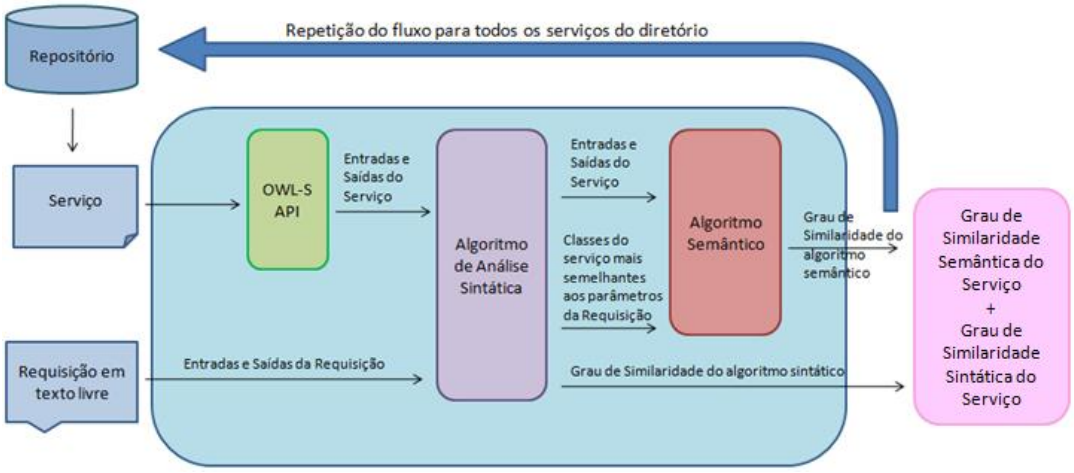

Figura 16 - Visão geral do funcionamento do OWL-S Discovery 3.0

Assim como nas versões anteriores, o primeiro passo do algoritmo é a leitura da requisição e dos arquivos OWL-S dos serviços, realizando o mapeamento de suas respectivas entradas e saídas. Estes dados são armazenados em listas encadeadas que serão usadas posteriormente para o alinhamento.

Vale ressaltar que antes da aplicação de métricas o algoritmo aqui proposto realiza normalizações nos parâmetros comparados, tais como normalização de *case* e *link stripping.*

A análise sintática consiste então na aplicação de métricas de similaridade (definidas no capítulo 2) para se chegar a um denominador comum sobre qual classe, dentro de uma dada ontologia, um argumento livre mais se assemelha. Aplicando-se tais métricas, calcula-se uma média simples usando os valores de similaridade fornecidos por cada uma, onde esses valores são normalizados a fim de se manter uma escala entre 0 e 1.

Após uma análise das métricas de similaridade – análise esta apresentada no próximo capítulo – chegou-se a conclusão que a melhor abordagem seria uma separação entre o conjunto de métricas baseadas em *token* e baseadas em *caracter*. Dessa forma, o algoritmo aqui proposto aplica estes dois conjuntos separadamente, optando pelo grupo de métricas que apresentar a maior média como resultado.

A Figura 17 apresenta o algoritmo do OWL-S Discovery 3.0.

```
01 //Tratamento para requisições feitas com parâmetros em texto livre

02 List listaEntradaReq;
03 List listaSaidaReq;
04 List listaEntradaServ;
05 List listaSaidaServ;

11 for(int i=0; i< listaEntradaReq.size(); i++) {
12          for(int j = 0; j < listaEntradaServ.size(); j++) {
13                    double grauDeSimilaridade = calculoDaSimSintatica(listaEntradaReq(i),
listaEntradaServ(j));
14                    if(grauDeSimilaridade > limiarDeAceitação) {
15                              URI classeMaisSimilar =
classeMaiorSimilaridade(grauDeSimilaridade);
16                              GrauSimSemantica resultado =
analisePorFiltrosSemanticos(classeMaisSimilar, listaEntradaServ(j));
17                    }
18          }
19 }

20 public double calculoDaSimSintatica(argRequisicao, argServico){
21          List listaDeClasses = listarClassesDaOntologia(argServico);
22          List metricasDeToken = aplicaçãoDeMétricasToken(argRequisicao, listaDeClasses);
23          List metricasDeCaracteres = aplicaçãoDeMétricasCaractere(argRequisicao,
listaDeClasses);

24          double mediaParaToken = calculoDaMediaDeMetricasDeToken(metricasDeToken);
25          double mediaParaCaracteres =
calculoDaMediaDeMetricasDeCaracteres(metricasDeCaractere);

26          if(mediaParaToken > mediaParaCaracteres) {
27                    return mediaParaToken;
28          } else {
29                    return mediaParaCaracteres;
30          }
31 }

32 public URI classeMaiorSimilaridade(melhorGrauSimilaridadeSintatica) {
33          URI classeMaisSimilar = mapeamentoClassePorMetrica
                                                  .buscar(melhorGrauSimilaridadeSintatica);
34          return classeMaisSimilar;
35 }
```

Figura 17 – Algoritmo do OWL-S Discovery 3.0

Após aplicação do algoritmo apresentado, inicia-se a análise dos serviços considerando como o parâmetro da requisição a classe mais similar encontrada na ontologia do serviço. Esta análise segue o fluxo original do OWL-S Dicovery 2.0, consistindo na classificação dos serviços através dos filtros de Paolucci, durante a etapa semântico funcional, e no cálculo do coeficiente básico e estrutural durante a fase semântico descritiva.

Ao fim da execução do algoritmo, o usuário terá então visibilidade dos graus de similaridade de cada serviço de acordo com a etapa semântica do algoritmo e também poderá visualizar o grau de similaridade sintática e a classe mais similar encontrada na ontologia de cada serviço.

Por fim, caso o usuário deseje iniciar a execução do algoritmo através de uma requisição em OWL-S, o OWL-S Discovery 3.0 seguirá o fluxo original, apresentado no capítulo 2.

5.2. Cenário de uso do Algoritmo

Um dos cenários onde o algoritmo proposto será de grande serventia é no caso de palavras escritas incorretamente. Para ilustrar o funcionamento do OWL-S Discovery 3.0, suponha que o usuário não possua uma requisição descrita formalmente em OWL-S e decida entrar com seus parâmetros de busca na forma de texto livre. O usuário deseja encontrar serviços web semânticos que, dado determinado *livro*, seja retornado seu *preço*. Para tal, ele deve imputar o parâmetro de entrada *Book* e o parâmetro de saída *Price*.

Suponha agora que este usuário acabou por escrever o termo *Bok* como entrada. O algoritmo analisará então as ontologias de cada serviço no repositório encontrando, para cada ontologia analisada, a classe que mais se assemelha com o termo presente na requisição do usuário.

O serviço *BookPrice*, por exemplo, possui como entrada o parâmetro *Book*, presente na ontologia *Books*, e seu parâmetro de saída é *Price*, presente em *Concept*. Durante a execução do algoritmo, cada uma das classes das ontologias *Books* e *Concept* serão analisadas a procura dos termos da requisição.

Métricas de similaridade serão então aplicadas para cada uma das classes das ontologias. Na busca pelo termo *Bok* todas as classes da ontologia *Books*, ilustradas na Figura 12, serão estudadas de forma a encontrar aquela que apresenta maior similaridade com o termo.

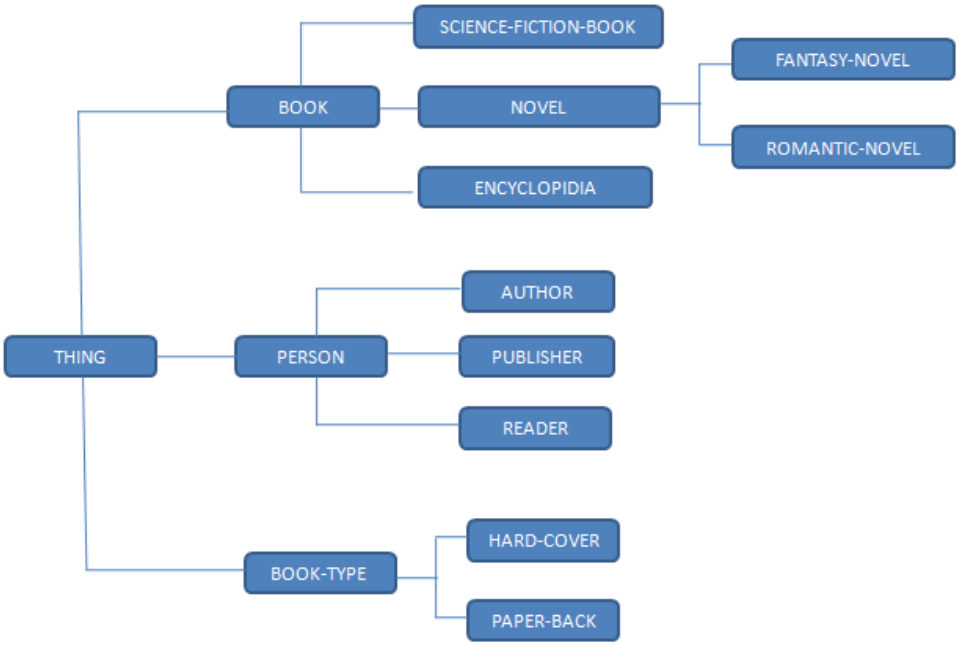

Figura 18 - Grafo da ontologia *Books*

Abaixo, segue a tabela resultante da aplicação das métricas Levensthein, Jaro, Jaro Winkler e Smith Waterman utilizadas para comparar o termo *Bok* com cada uma das classes da ontologia *Books*.

O resultado é obtido então através do cálculo de uma média simples entre os graus de similaridade observados. Como é possível perceber, o algoritmo identificará a classe *Book* como sendo o parâmetro imputado pelo usuário durante a sua busca.

	Levenshtein	Jaro	Jaro Winkler	Smith Waterman	Média
book	1,00	0,92	0,93	0,63	0,87
person	0,75	0,00	0,00	0,28	0,26
book-type	0,69	0,78	0,82	0,20	0,62
science-fiction-book	0,00	0,00	0,00	0,17	0,04
novel	0,81	0,51	0,51	0,17	0,50
encyclopidia	0,38	0,00	0,00	0,13	0,13
author	0,75	0,00	0,00	0,11	0,22
publisher	0,56	0,00	0,00	0,10	0,17
reader	0,69	0,00	0,00	0,10	0,20
hard-cover	0,50	0,00	0,00	0,08	0,15
paper-back	0,56	0,00	0,00	0,08	0,16
fantasy-model	0,31	0,00	0,00	0,07	0,10
romantic-novel	0,25	0,47	0,47	0,00	0,30

Tabela 1 – Aplicação de métricas de similaridade entre termo *Bok* e classes da ontologia *Books*

No caso do input *Price*, uma vez que este termo é idêntico ao termo de saída do serviço, a classe *Price* será encontrada com grau de similaridade 1.0. Desta forma, o algoritmo seguirá seu fluxo original, encontrando através de sua análise descritiva que o serviço em questão apresenta o filtro EXACT.

O usuário visualizará que o serviço *BookPrice* é um serviço EXACT em relação a sua requisição, com grau de similaridade 1.0 para o parâmetro *Price* e 0.87 para *Book*. Dessa forma, o usuário poderá avaliar se o parâmetro entendido pelo algoritmo condiz, ou ao menos se assemelha, com o termo que este intentava procurar.

6. TRABALHOS RELACIONADOS

Devido ao aumento da utilização dos serviços web e ao crescente número de serviços publicados na Internet nos últimos anos, muitos algoritmos de descoberta foram desenvolvidos. O objetivo deste capítulo é apresentar alguns dos algoritmos criados que serviram de base para o OWL-S Discovery, ferramenta cuja evolução está sendo trabalhada nesse projeto.

6.1. OWL-S/UDDI MATCHMAKER

A ferramenta OWL-S/UDDI Matchmaker [Paolucci *et al*, 2003] foi uma das primeiras ferramentas de descoberta de serviços web desenvolvida. Ela analisa os parâmetros de entrada e saída do serviço web e o grau de semelhança dos parâmetros fica atrelado ao relacionamento entre estes e a ontologia de domínio utilizada. Em sua execução, o algoritmo considera as informações semânticas extraídas dos arquivos OWL-S e a categorização dos relacionamentos encontrados entre parâmetros e classes da ontologia segue o modelo de [Paolucci *et al*, 2002].

A proposta de Paolucci *et al.* defende o uso da semântica para determinar o quanto um serviço é "suficientemente similar" a certa requisição. O conceito de "suficientemente similar é o ponto exato trabalhado em [Paolucci *et al*, 2002]. Para atingir tal objetivo, o algoritmo realiza comparações mais flexíveis do que simplesmente determinar o termo da requisição e do serviço como iguais ou diferentes. Esta semelhança leva em consideração a distância entre os termos na árvore taxonômica da ontologia utilizada, conforme explicado no capítulo anterior.

Em termos de tempo de descoberta, a ferramenta OWL-S/UDDI Matchmaker apresenta um desempenho ótimo em comparação com os algoritmos desenvolvidos com o mesmo propósito, porém, não traz um número de resultados significativos para uma determinada requisição [Cardoso, 2007].

A abordagem do OWL-S Discovery, algoritmo utilizado como base desse trabalho, tem como vantagem em relação ao OWL-S/UDDI Matchmaker a possibilidade de explorar outros meios de descoberta de serviços, como a análise estrutural dos vizinhos das classes analisadas e a incorporação do novo filtro

semântico proposto por [Samper *et al.*, 2008]. A melhoria proposta neste projeto visa ainda a flexibilização da descoberta através da incorporação de uma análise sintática dos termos buscados.

6.2. OWLS-MX

Assim como o OWL-S Discovery, a ferramenta OWLS-MX [Klush *et al*, 2006] faz uso de uma técnica híbrida para a realização da busca por serviços web semânticos. Além de realizar um raciocínio baseado em ontologias, especificamente em OWL-DL, o algoritmo também possui uma técnica de similaridade sintática entre palavras. Utilizando-se dos filtros de Paolucci, a ferramenta remodela os filtros semânticos encapsulando os filtros híbridos que levam em consideração a semelhança semântica e a similaridade sintática dos termos analisados. A similaridade sintática se apoia em trabalhos que visam a extração de informações em texto, utilizando métricas como, por exemplo, o número de vezes que determinados termos são repetidos em uma sentença.

O OWL-S Discovery, por sua vez, também é um algoritmo híbrido que visa capturar fontes de informações sobre dados analisados de duas vertentes. Seu diferencial está na aquisição de informações baseadas nos vizinhos estruturais das classes envolvidas, presentes na ontologia analisada. Essa técnica apresentou resultados mais significativos aos obtidos pelo OWLS-MX, como mostra o trabalho de [Amorim, 2009].

O algoritmo proposto nesse projeto visa lidar com as limitações que OWL-S Discovery possui em relação ao OWLS-MX por não possuir uma análise sintática dos termos buscados.

6.3. SAMT4MDE

A SAMT4MDE [Lopes *et al.*, 2006], dentre outros objetivos, visa a geração semi-automática de correspondências entre metamodelos. O algoritmo utilizou como base o trabalho desenvolvido em [Lopes e*t al.*, 2005] que efetua uma análise dos termos entre metamodelos usando um dicionário de sinônimos como base. Em

[Lopes *et al.*, 2006] uma melhoria a esta abordagem propõe que os termos de um metamodelo são tão semelhantes quanto seus vizinhos estruturais.

O algoritmo OWL-S Discovery adaptou e incorporou as ideias presentes nesta ferramenta, trabalhando com ontologias de domínio ao invés de metamodelos. Como diferencial, o OWL-S Dicovery apresenta a análise baseada nos filtros semânticos de Paolucci, executada antes da abordagem baseada no SAMT4MDE.

6.4. SAM+

O algoritmo SAM+ [Bener; Ozadali; Ilhan, 2009] foi utilizado como base para ao trabalho de [Erdens, 2010] que resultou na evolução do OWL-S Discovery para sua versão 2.0. O SAM+ utiliza análise de pré-condições e efeitos escritos em SWRL para realizar a descoberta de serviços web semânticos. Depois que os parâmetros de entrada e saída são analisados, o algoritmo lista todas as pré-condições e efeitos da requisição e dos serviços e os compara. Três diferentes métodos são então utilizados, atribuindo pesos de maneira independente aos pares comparados, e depois combinados para o cálculo do peso final.

No primeiro método, aplica-se o filtro de Paolucci nas classes dos argumentos, uma vez que as pré-condições e efeitos são comparados segundo os conceitos dos argumentos. A cada grau de compatibilidade de Paolucci atribui-se determinado peso. Ao Exact, por exemplo, é atribuído o peso 1 enquanto ao Fail atribui-se 0. O segundo método se baseia no conceito de distância semântica, calculando a distância entre dois termos dentro de uma ontologia. O peso dessa etapa é calculado de acordo com a quantidade de subclasses que envolvem os termos buscados. O último método, por sua vez, utiliza o WordNet, um dicionário de sinônimos, atribuindo um peso no valor de [0,1] de acordo com a similaridade entre as classes.

Uma das principais diferenças entre o OWL-S Discovery 2.0 e o SAM+ está no fato de que o segundo algoritmo ao analisar um serviço que tem uma quantidade diferente de pré-condições e efeitos em relação à requisição, já o descarta como serviço em potencial. O trabalho de [Erdens, 2010], por sua vez, prossegue a análise dos termos mesmo quando a quantidade de pré-condições e efeitos são diferentes entre o serviço e a requisição. Isso se dá devido ao fato de que, caso uma requisição

forneça mais pré-condições do que o necessário pelo serviço é possível descartar esse excesso de informações e utilizar apenas as pré-condições necessárias para a execução do mesmo. No caso dos efeitos, se um serviço possui mais efeitos do que os desejados, não há restrição, uma vez que o objetivo é o atendimento de todas as necessidades do usuário e não do serviço. Além disso, se há uma compatibilidade entre as condições, uma pré-condição ou efeito pode ser usado mais de uma vez para satisfazer uma pré-condição do serviço ou para corresponder a um efeito da requisição.

O algoritmo proposto no presente trabalho tem como foco a aplicação da análise sintática apenas nos parâmetros de entrada e saída. Um dos trabalhos futuros aqui apontados é a evolução de tal análise na busca de pré-condições e efeitos.

7. ANÁLISE E VALIDAÇÃO

Este capítulo tem por objetivo a apresentação de algumas análises realizadas durante o desenvolvimento do algoritmo que auxiliaram em algumas tomadas de decisão. Também serão apresentados testes que visam validar a execução do algoritmo proposto.

7.1. Análise de Métricas de Similaridade

Visando encontrar as métricas que mais se adequassem ao cenário da ferramenta trabalhada, uma análise comparativa foi realizada entre as métricas de similaridade baseadas em string mais comuns na literatura.

A adição da possibilidade de entrada da requisição como texto livre flexibiliza o uso da ferramenta OWL-S Discovery uma vez que o usuário não necessita possuir uma requisição descrita formalmente em OWL-S, porém, como já citado no capítulo anterior, tal modificação abre precedentes para a ocorrência de termos escritos incorretamente. As métricas baseadas em caracteres são então a forma de resolução deste problema.

De modo a testar quais métricas fornecem os melhores resultados, testes foram realizados imputando-se diferentes variações da palavra *Book*, tais como: *Bok*, *Boook*, *Boks*, *Bbook*, entre outras. As métricas disponíveis nas bibliotecas Java SimMetrics e Secondstring foram então aplicadas de forma a comparar estes termos com alguns dos termos presentes na ontologia *Book*. A Tabela 2 apresenta a média dos resultados de cada uma das métricas aplicadas para alguns exemplos de classes analisadas.

Classes Analisadas	Jaro	Jaro Winkler	Levenshtein	Smith Waterman	Needleman Wunch	Gotoh	Euclidean	Média
book	0,93	0,94	0,98	0,58	0,83	0,89	0,00	0,74
book type	0,76	0,81	0,69	0,67	0,00	0,76	0,23	0,56
science fiction book	0,00	0,00	0,00	0,92	0,17	0,76	0,37	0,32
romantic novel	0,44	0,44	0,25	0,23	0,04	0,29	0,23	0,27
romantic	0,46	0,46	0,58	0,13	0,00	0,24	0,00	0,27

Tabela 2 – Médias de graus de similaridade para variações da palavra *Book*

Seguindo a lógica de comparação apresentada no trabalho de [Gondim, 2006], gerou-se então um gráfico comparativo, Figura 19, para o auxílio na identificação das métricas de melhor resultado.

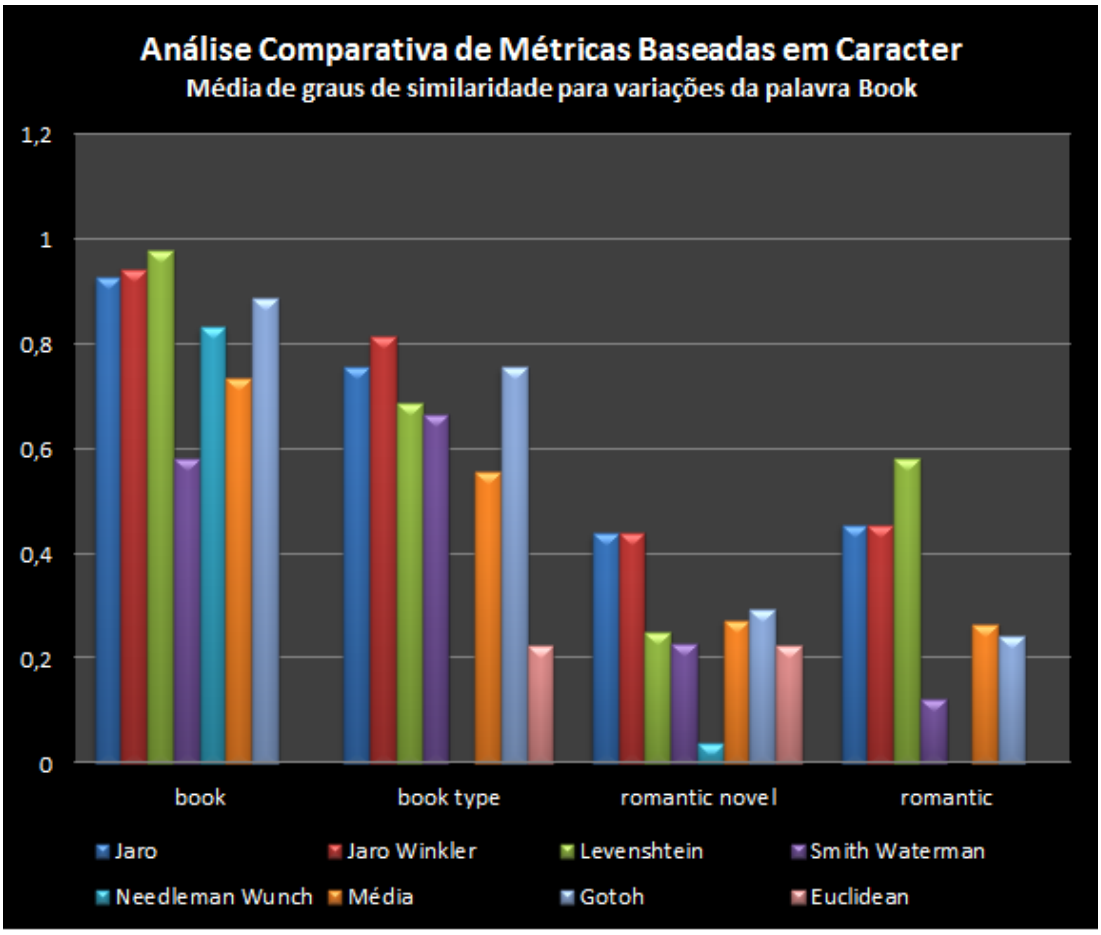

Figura 19 – Gráfico de comparação de métricas baseadas em caracter

Observa-se então que, em comparação a média geral dos algoritmos aplicados, as métricas Euclidean, Smith Waterman e Needleman Wunch apresentaram graus de similaridade abaixo do esperado.

O algoritmo Euclidean não encontrou similaridade com a classe *Book* para a maioria dos parâmetros de requisição testados. Já a métrica Smith Waterman, na maioria dos casos testados, encontrou uma similaridade maior com a classe *Book-Type* do que com *Book*, apesar dos parâmetros comparados se tratarem de variações errôneas de *Book*. Needleman Wunch, por sua vez, encontrou um grau de

similaridade alto com *Book*, mas não foi capaz de encontrar a similaridade presente em entre os termos buscados e a classe *Book-Type*.

Uma vez que as demais métricas demonstraram um comportamento satisfatório, estas foram selecionadas para serem aplicadas ao algoritmo proposto nesse trabalho.

Outro cenário com grande probabilidade de ocorrência no caso de uma requisição composta por argumentos livres consiste na possibilidade do usuário entrar com parâmetros com alto nível de especificidade. Por exemplo, ainda no contexto dos serviços com parâmetros de entrada mapeados pela ontologia *Book*, o usuário pode desejar buscar algo mais específico do que o preço de um livro qualquer e entrar com o argumento *Romantic Book*.

Para esse caso, as métricas apresentadas acima acabariam por encontrar baixos graus de similaridade uma vez que, em termos de número de caracteres, os argumentos *Book* e *Romantic Book* são consideravelmente distantes. Levando em consideração tal cenário, foi feita então uma análise referente às métricas baseadas em token, a Figura 20 apresenta o resultado encontrado.

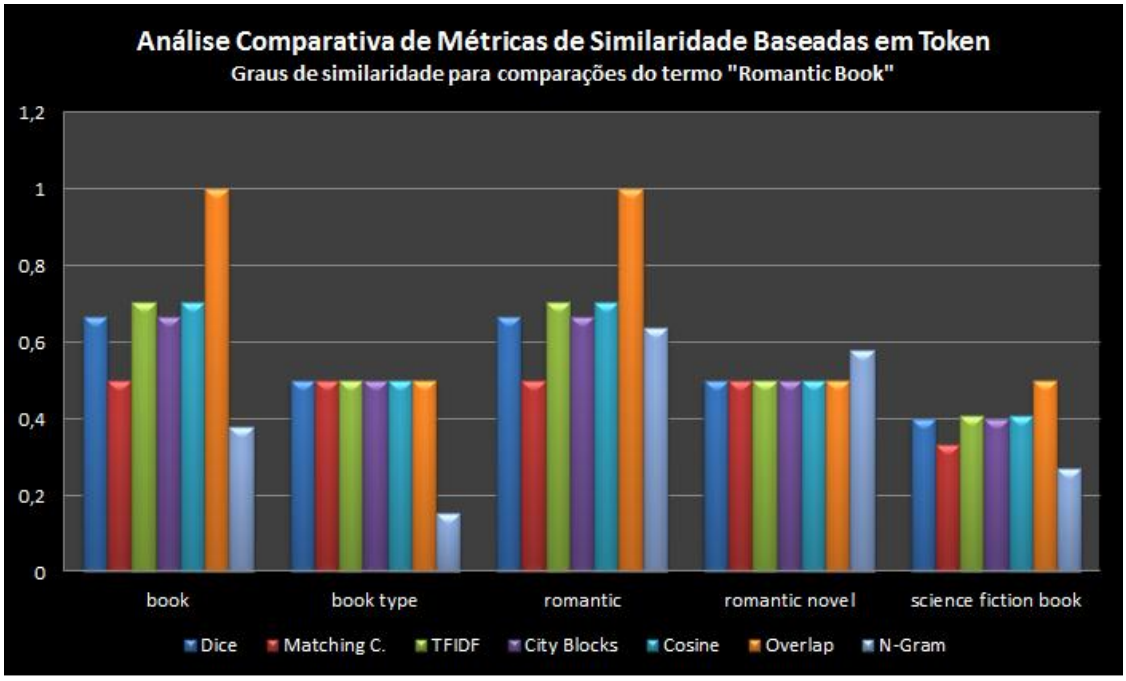

Figura 20 – Gráfico de comparação de métricas baseadas em token

O resultado ideal para a busca do termo *Romantic Book* entre as classes apresentadas seria um nível mais alto de confiança para as classes *Book* e *Romantic*, e níveis mais baixos para as demais classes analisadas. Como é possível perceber pelo gráfico, o algoritmo N-Gram apresenta melhor grau de similaridade para *Romantic Novel* do que para *Book*, o que não é um comportamento desejado uma vez que *Book* se aproxima mais do conceito imputado pelo usuário. A métrica Alinhamento Coefficient também não apresentou um resultado satisfatório, fornecendo o mesmo nível de confiança para maioria das classes comparadas.

As demais métricas baseadas em token forneceram resultados muito semelhantes entre si e optou-se por implementar todas elas no algoritmo proposto.

Durante os testes, foi percebido que as métricas baseadas em token apresentavam resultados insatisfatórios quando tratavam erros de digitação do usuário, além disso, percebeu-se também que quando aplicava-se métricas de distância de edição em casos de strings compostas os resultados também não eram os esperados. Decidiu-se então aplicar os dois conjuntos de métricas separadamente, escolhendo o melhor resultado apresentado.

7.2. Testes de Validação do Algoritmo

Após a implementação do algoritmo proposto, uma bateria de testes foi realizada de forma a validar os resultados obtidos. Abaixo, na tabela 3, são apresentados três exemplos de serviços presentes no repositório.

Serviços			
Nome	**getAuthorBook**	**getPriceAndAuthor**	**getMonographReview**
Inputs	Book-Type	Book	Monograph Author
Outputs	Author	Price Author	Review Size

Tabela 3 – Exemplos de serviços presentes no repositório

Foram testados então os resultados das buscas por argumentos livres, comparando os resultados obtidos a partir da versão anterior da ferramenta, através de uma requisição OWL-S.

Testes para o Serviço: getAuthorBook	Teste 1			Teste 2		
	Requisição	Encontrado	Similaridade	Requisição	Encontrado	Similaridade
Entradas	book type	Book-Type	1.0/EXACT	type	Book-Type	0.9217/EXACT
Saídas	autor boks	Author Book	0.9764/EXACT 0.925/EXACT	reader enciclopedia	Reader Encyclopedia	1.0/SIBLING 0.9764/EXACT
Resultado	EXACT			SIBLING		

Tabela 4 – Exemplos de testes para o serviço *getAuthorBook*

Testes para o Serviço: getPriceAndAuthor	Teste 1			Teste 2		
	Requisição	Encontrado	Similaridade	Requisição	Encontrado	Similaridade
Entradas	publicacao	Publication	0.937/PLUGIN	books	Book	0.9733/EXACT
Saídas	preco autor	Price Author	0.9764/EXACT 0.9025/EXACT	max price publishr	MaxPrice Publisher	1.0/EXACT 0.9764/SIBLING
Resultado	PLUGIN			SIBLING		

Tabela 5 – Exemplos de testes para o serviço *getPriceAndAuthor*

Testes para o Serviço: getMonographReview	Teste 1			Teste 2		
	Requisição	Encontrado	Similaridade	Requisição	Encontrado	Similaridade
Entradas	monografia person	Monograph Person	0.9304/EXACT 1.0/EXACT	publicafdo autor	Publication Author	0.937/EXACT 0.9764/EXACT
Saídas	user review	UserReview	1.0/EXACT	grade small	Grade Small	1.0/SIBLING 1,0/EXACT
Resultado	EXACT			SIBLING		

Tabela 6 – Exemplos de testes para o serviço *getMonographReview*

Como é possível perceber pelos exemplos de testes apresentados acima, ainda que os parâmetros passados pelo usuário estejam escritos incorretamente, a nova versão da ferramenta é capaz de identificar a intenção de busca do usuário. As requisições, escritas corretamente, apresentaram os mesmos graus de similaridade ao serem testadas no OWL-S Discovery 2.0.

A interface desta nova versão, como será apresentado no próximo capítulo, também foi adaptada de forma que o próprio usuário possa validar o resultado de busca, através da exibição da classe mais similar ao parâmetro imputado e seu grau de confiança.

7.3. Tempo de Execução

Após os testes de validação do algoritmo foi realizada então uma análise referente ao tempo de execução do mesmo, cujo resultado se encontra na tabela 6:

Tempos de Execução Registrado (em ms):		
Testes	OWL-S Discovery 3.0	OWL-S Discovery 2.0
1	1.910	882
2	1.935	760
3	1.987	554
4	2.235	499
5	2.253	409
6	2.476	406
7	2.419	358
8	2.671	338
9	2.655	344
10	2.821	352

Tabela 7 – Análise Comparativa do Tempo de Execução do OWL-S Discovery 3.0

A requisição utilizada para os testes contava com um único parâmetro de *input* e dois *outputs* e o repositório utilizado como base possuía um total de 14 serviços.

Nota-se que o tempo de execução da busca na atual versão, aumentou consideravelmente em relação à versão anterior. Isso se deve principalmente ao fato de que a lógica de *alinhamento* sintático aumenta bastante o custo computacional, uma vez que, para cada métrica aplicada, faz-se um *loop* percorrendo todas as classes da ontologia, atribuindo a elas valores de grau de confiança em relação ao parâmetro da requisição e retornando mapas com essas consolidações, de forma a se montar uma consolidação final da média de todos os graus.

Outra alteração que pode ter relevância neste aumento do tempo de execução tem relação com a alteração do OWL-S Discovery 2.0 para uma versão web. Apesar do cliente e servidor estarem rodando na mesma máquina, existe ainda o tempo de *request response* entre eles.

8. IMPLEMENTAÇÃO

Este capítulo tem como objetivo a apresentação do OWL-S Discovery 3.0, resultado das adaptações realizadas no aplicativo já existente OWL-S Discovery [Amorim, 2009]. As mudanças que foram efetuadas sobre o OWL-S Discovery 2.0 [Erdens, 2010] tem como objetivo a implementação de uma análise sintática, de forma a flexibilizar o uso da ferramenta e aprimorar a tarefa de descoberta de serviços web semânticos.

8.1. Evolução da Ferramenta

De forma a auxiliar a ilustrar as evoluções realizadas na ferramenta, foi construído um diagrama de classes, apresentado na Figura 21.

Conforme o diagrama de classes demonstra, a aplicação está modularizada e pronta para acoplar novas *engines* com variações de métodos de busca. As atuais *engines* estão representadas pelas classes *FunctionalEngine*, *DescriptiveEngine* (implementadas no OWL-S Discovery 1.0) e *PEEngine* (implementada no OWL-S Discovery 2.0), que estendem a interface *IEngine*. Todas as *engines* estão associadas as suas respectivas classes responsáveis pelo seu algoritmo de match. Estas classes são *FunctionalMatcher*, *DescriptiveMatcher* e *PEMatcher*, que estendem a interface *IMatcher*. Acoplada às classes *FunctionalMatcher* e *DescriptiveMatcher*, implementou-se a classe *SyntacticMatcher* na versão do OWL-S Discovery 3.0, a fim de possibilitar uma busca sintática através de parâmetros em texto livre. Esta classe possui como elementos auxliares a classe *Normalization*, responsável por aplicar métodos de normalização aos parâmetros entrados e a interface *Metrics*, estendida pelas diversas métricas de similaridade a serem aplicadas na busca sintática. Isto possibilita também a extensão da aplicação para o uso de métricas adicionais.

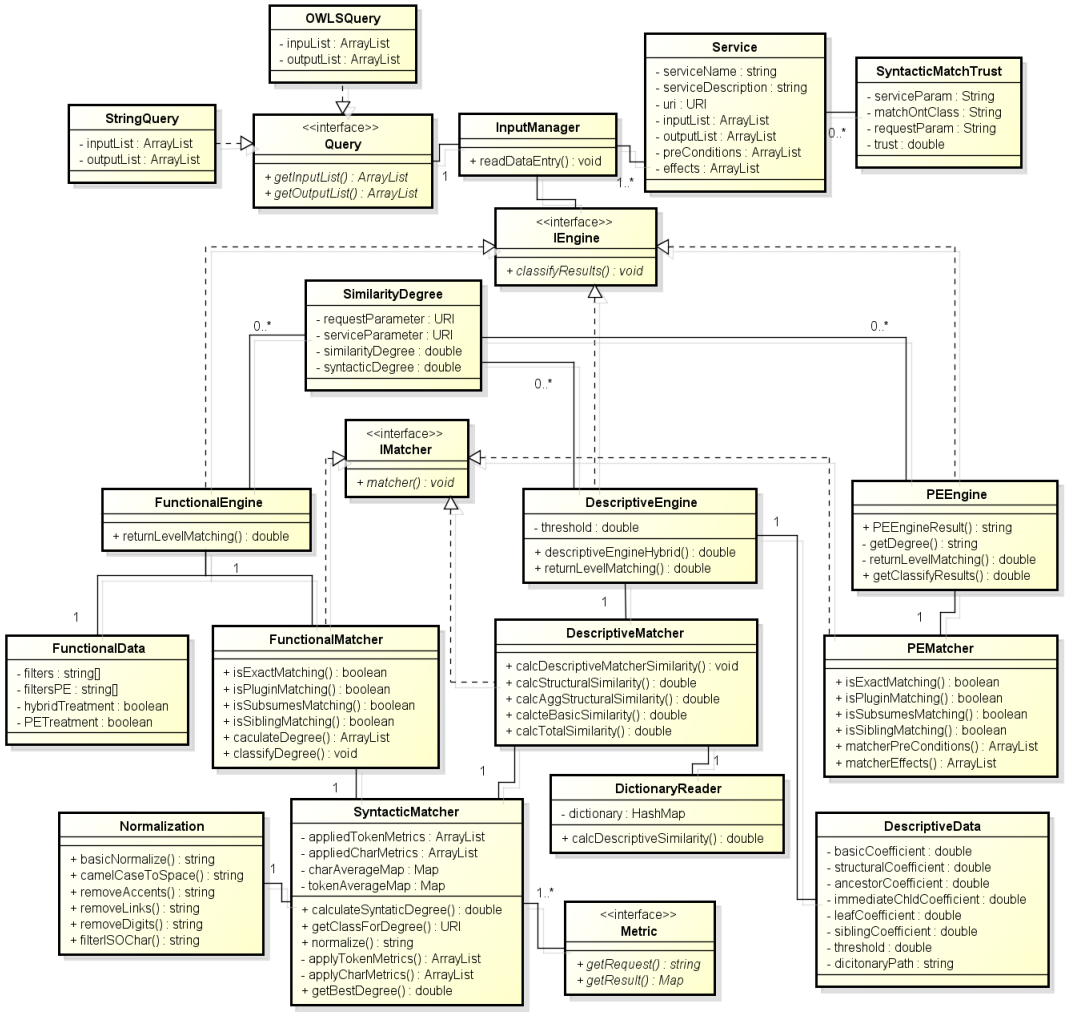

Figura 21 – Diagrama de Classes

Cada *engine*, após a execução de suas lógicas de match, monta uma lista de objetos da classe *SimilarityDegree* e a partir desta lista é feita a classificação da relevância no match feito.

As *engines FunctionalEngine* e *DescriptiveEngine* possuem ainda classes responsáveis por dados referentes a especificidades sobre a forma como serão executadas, estas são respectivamente *FunctionalData* e *DescriptiveData*. *DescriptiveMatcher* possui ainda uma classe auxiliar, *DictionaryReader*, responsável pela leitura de um dicionário de sinônimos necessário para sua execução.

Os dados de input da aplicação, são de responsabilidade da classe *InputManager*, que tem relacionamento com todas as *engines* e gerencia os dados necessários para dar início as suas execuções. Esta classe faz a leitura dos dados de entrada e é

responsável por montar os objetos *Query* e uma lista de objetos *Service*. A classe *Query*, original do OWL-S Discovery 1.0 e 2.0, foi alterada para a interface *Query*, de modo a abstrair o formato da busca, nas versões anteriores feita apenas através de arquivos OWL-S. Na versão atual, foi implementada a classe *StringQuery*, representando a entrada com parâmetros livres, e a classe *OWLSQuery*, representando a entrada da busca usando OWL-S.

Por fim, na versão atual, foi adicionada à classe *Service*, uma lista de objetos *SyntacticMatchTrust*, responsável por registrar quais inputs e outputs do objeto *Service* obtiveram match, qual a classe mais similar encontrada, e com qual grau de similaridade.

8.2. Arquitetura

Como suas versões anteriores, o OWL-S Discovery 3.0 foi desenvolvido em Java e fornece interfaces de comunicação de modo a permitir que novos módulos sejam integrados ao aplicativo. Também foi mantido o alto grau de escalabilidade do aplicativo, o que possibilita a inclusão de novos filtros e métodos de busca que podem ser desenvolvidos futuramente e acoplados ao aplicativo.

A Figura 22 apresenta a estrutura da ferramenta, dividida em quatro módulos.

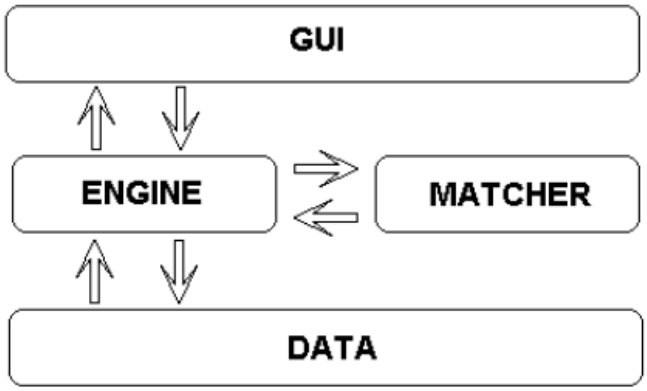

Figura 22 – Arquitetura do OWL-S Discovery 3.0 [Amorim, 2009]

O módulo DATA é a parte da aplicação que encapsula a leitura dos serviços OWL-S dos serviços, assim como o dicionário necessário para a execução do fluxo descritivo do algoritmo. Uma abstração de todos os arquivos necessários para a

execução do cálculo da similaridade é fornecido à ENGINE através de *Business Objects*.

Trafegam para a ENGINE dados em listas encadeadas com interfaces de entrada, saída, pré-condições e efeitos de cada serviço disponível para a execução da descoberta, assim como os requisitos dos usuários que também são encapsulados em listas encadeadas.

Uma vez que tenha posse dos dados, a ENGINE os envia para o módulo MATCHER, onde é efetuado o cálculo da similaridade entre as classes que recebidas. Pensando na escalabilidade do problema, uma interface denominada IMatcher foi criada por [Amorim, 2009] de forma a padronizar o comportamento de todas as instâncias, tanto na entrada quanto na saída dos dados.

Uma vez tratados pelo MATCHER, os resultados dos dados são mais uma vez encapsulados em *Business Objects* e transferidos novamente para a ENGINE. Seguindo os mesmos mecanismos do MATCHER, uma interface também foi criada para a ENGINE, cujo papel é classificar os resultados de acordo com os parâmetros solicitados pelo usuário da ferramenta. Com a classificação dos dados já realizada, os mesmos são enfim apresentados para o usuário e encapsulados em outros *Business Objects* de modo que outras aplicações possam utilizá-los em sua computação.

8.3. Ferramentas Utilizadas

Durante o desenvolvimento do OWL-S Discovery 3.0, a utilização de algumas ferramentas para a leitura e análise dos documentos em OWL-S e OWL, assim como para a aplicação de métricas de similaridade, se fez necessária. Foi-se utilizado então o Jena 2.1, PELLET 2.3.1 e o OWL-S API 3.0, versão estável mais atual e as bibliotecas Java SimMetrics e SecondString.

Em linguagem Java, Jena é um framework para o desenvolvimento de aplicações que se utilizem do paradigma da Web semântica. Ele provê um ambiente para a manipulação de RDF, RDFS, OWL, SPARQL, incluindo também um motor de inferência com base em regras.

No escopo deste projeto, utilizou-se a API de manipulação de OWL, módulo do framework Jena. Esta API foi utilizada na leitura dos descritores de serviço, OWL-S, e nas ontologias associadas aos serviços descritos em OWL. Esta API

permite que o OWL-S Discovery abstraia as extrações da informações em OWL. Diante de um documento em OWL, o Jena é capaz de transformá-lo em objetos Java facilitando seu uso e análise.

O motor de inferência Pellet, também incluso no Jena, foi o software escolhido para a realização de inferências lógicas referentes a um conjunto de axiomas presente em uma ontologia. O sistema abordado nesse projeto também faz uso da OWL-S API 3.0, responsável por prover suporta a criação, leitura, escrita e execução de arquivos OWL-S.

Por fim, foram utilizadas as bibliotecas Java SecondString e SimMetrics durante a implementação da análise sintática, auxiliando na aplicação das métricas de similaridade.

8.4. Limitações da Ferramenta OWL-S Discovery 3.0

Algumas das limitações observadas durante a evolução da ferramenta OWL-S Discovery merecem uma atenção especial, principalmente por possibilitarem trabalhos futuros. Segue abaixo as limitações encontradas.

8.4.1. Composição de Serviços

Um ponto a ser mais explorado refere-se à descoberta de serviços levando em consideração não somente serviços atômicos, mas composições destes. Seria interessante retornar ao usuário um conjunto dos serviços que, compostos, poderiam atender a sua requisição. Uma vez que a ferramenta, desde sua primeira versão, já possui a base para o tratamento dos dados de entrada, bastaria a codificação deste novo método de busca.

8.4.2. Repositório de Serviços

Nas versões 1.0 e 2.0 da ferramenta, por se tratarem de aplicações desktop, trabalhavam com arquivos locais. Sendo assim, o usuário referenciava o diretório local onde estariam os serviços a serem analisados. Como a ferramenta utiliza arquivos localmente para fazer a leitura dos serviços, uma vez convertida para aplicação web, na versão 3.0 essa funcionalidade foi mantida através de um upload

do repositório desejado. O ideal, porém, seria a existência de um repositório web de forma que o só fosse preciso uma referência a sua URL.

8.4.3. Utilização da Linguagem SWRL

Outra limitação encontrada se refere ao fato da linguagem SWRL, responsável pela descrição de pré-condições e efeitos, não estar sendo usada em sua plenitude, com todas as suas possibilidades.

8.4.4. Processos em Paralelo

O processo de alinhamento entre serviços e requisições é realizado atualmente de forma sequencial. Caso o algoritmo trabalhasse com processos paralelos, seu tempo de execução poderia ser altamente otimizado.

8.5. Exemplo de Utilização

Nesta seção será apresentado um exemplo de utilização da ferramenta desenvolvida neste trabalho. Um dos focos de evolução da ferramenta foi a adaptação da interface do OWL-S Discovery de forma a torná-la mais amigável ao usuário, apresentando um conjunto de informações relevantes que não eram exibidas anteriormente.

Para iniciar sua busca por serviços, o usuário tem de, primeiramente, realizar o *upload* do repositório de serviços no qual deseja realizar sua busca. Depois, ele deverá escolher se utilizará uma requisição em OWL-S ou se realizará sua busca por argumentos livres. Ao escolher os argumentos livres, ele deverá preencher os campos de *inputs* e *outputs* conforme demonstra a Figura 23, definir o limiar de aceite para a análise sintática e, caso decida por uma busca funcional ou híbrida, deverá também escolher quais filtros serão levados em conta.

No resultado, como é possível observar na Figura 24, serão exibidos os nomes dos serviços encontrados, suas descrições e graus de similaridade semântica. Em relação à análise sintática, o usuário terá disponíveis as informações referentes aos parâmetros imputados por ele, classes similares encontradas com seus devidos graus de confiança e parâmetros do serviço.

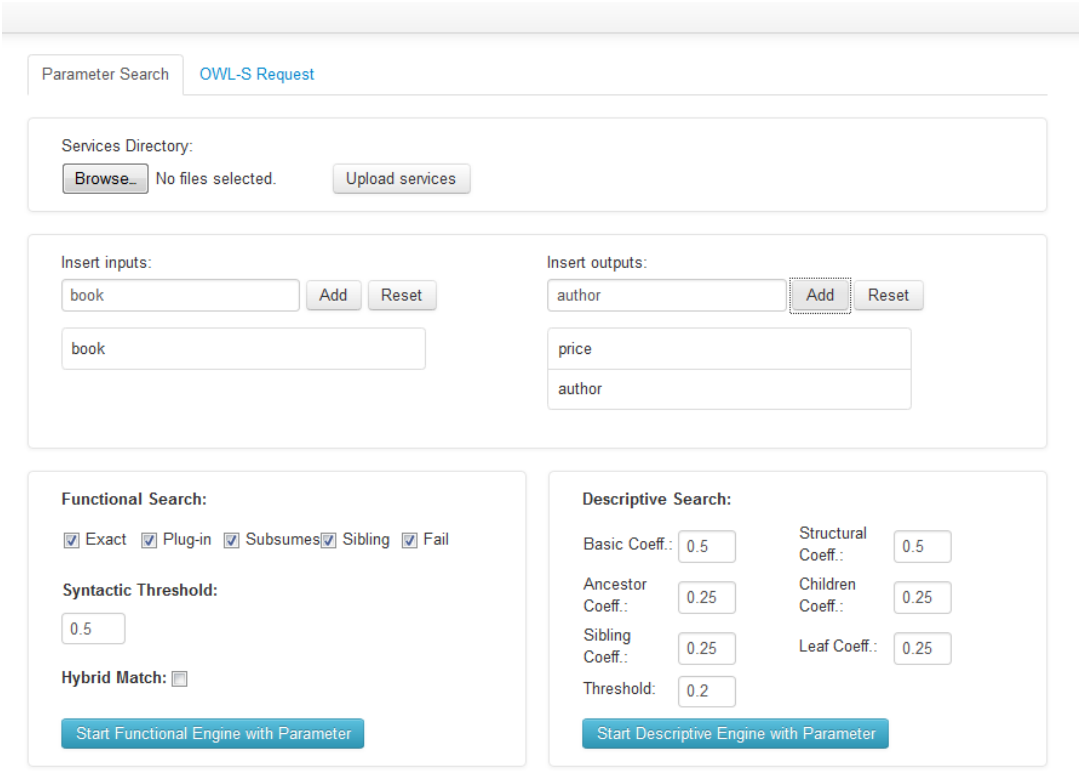

Figura 23 – Exemplo de busca no OWL-S Discovery 3.0

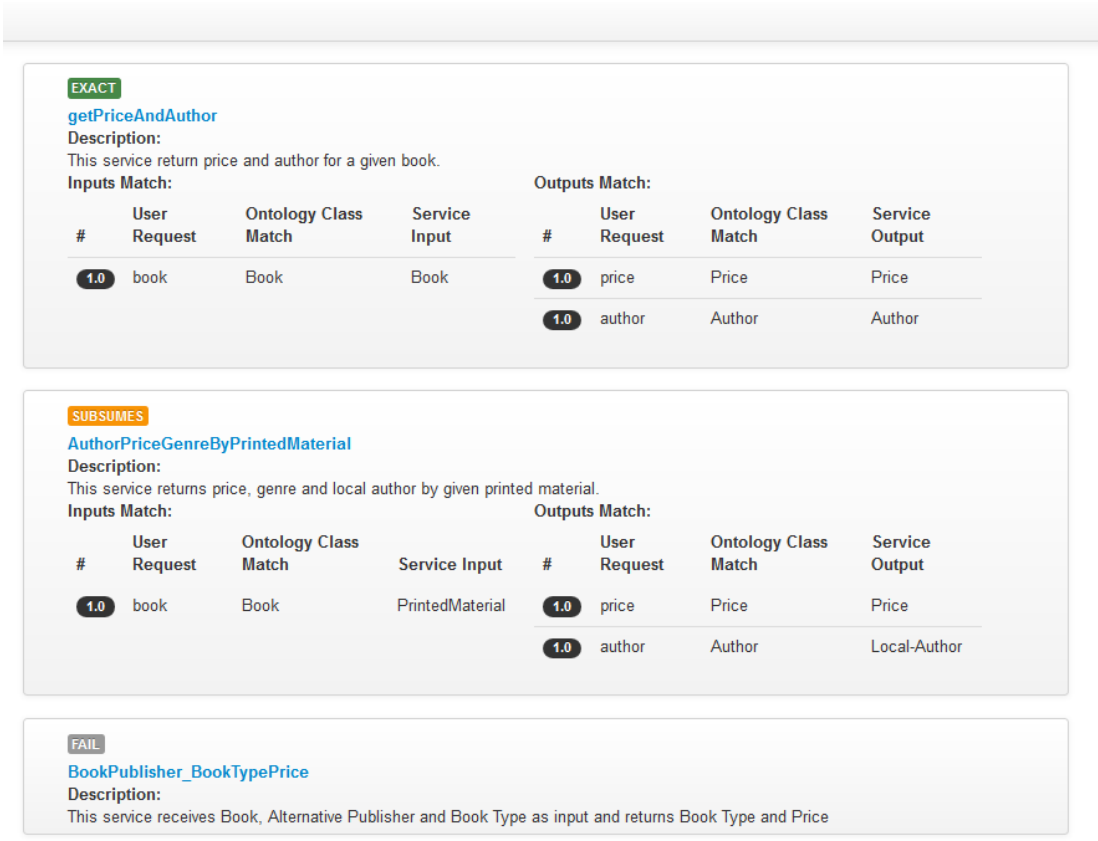

Figura 24 – Exemplo de resultado de busca no OWL-S Discovery

Anteriormente, o usuário só tinha acesso ao nome do arquivo OWL-S do serviço e seu respectivo grau de similaridade, como demonstra a Figura 25:

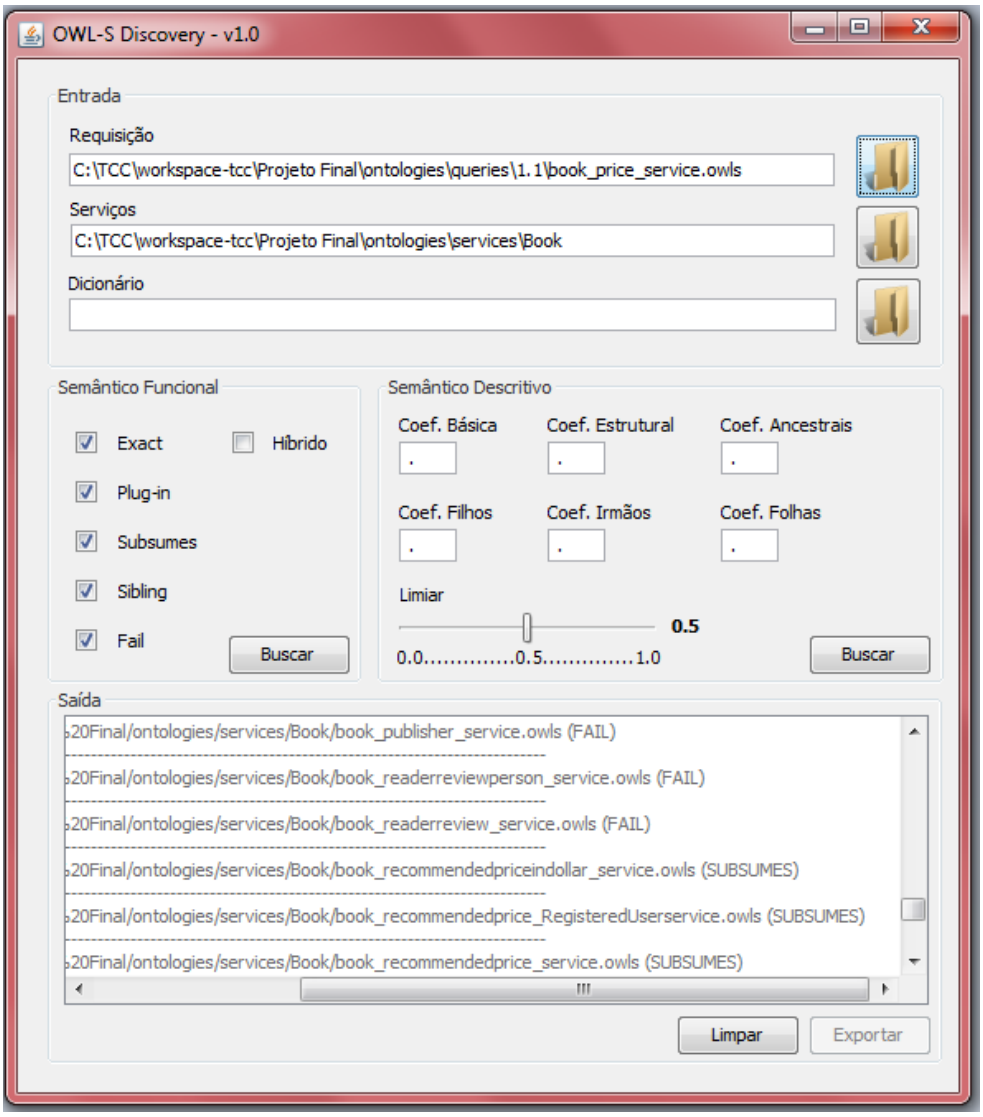

Figura 25 – Interface do OWL-S Discovery 1.0

Por fim, caso o usuário não deseje entrar com argumentos livres, mas sim com uma requisição em OWL-S, basta que ele opte pela aba *OWL-S Request* e realize o *upload* de sua requisição. Novamente, se forem ser realizadas as buscas funcional ou híbrida, os filtros desejados deverão ser escolhidos. Note que, caso se queira realizar a busca levando em consideração pré-condições e efeitos, a requisição deverá estar no formato OWL-S. A Figura 26 apresenta a tela referente à busca utilizando uma requisição OWL-S.

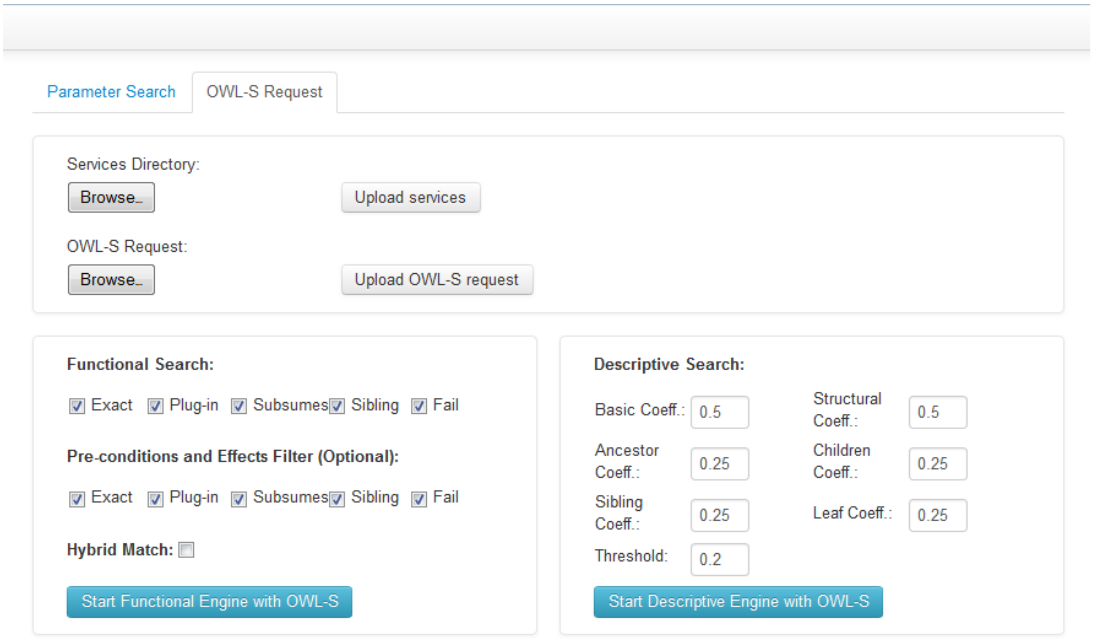

Figura 26 – Tela de busca por requisição OWL-S no OWL-S Discovery

9. CONCLUSÃO

Dentro do ciclo de vida de execução de um serviço web, a tarefa de descoberta é de grande relevância, pois caso um serviço não seja encontrado, a sua execução nunca poderá ser efetuada. A descoberta automática de serviços web semânticos é um campo de estudo relativamente novo e necessita que mais estudos e ferramentas sejam desenvolvidos sobre ele. Uma das maiores necessidades deste campo é o desenvolvimento de técnicas que sejam eficientes, demandando pouco tempo computacional na geração dos resultados e apresentando um resultado eficaz ao trazer a maior quantidade de serviços correspondentes a requisição do usuário.

O atual projeto propõem um algoritmo que adiciona uma análise sintática de modo a flexibilizar a tarefa de descoberta dos serviços. Dessa forma, um dos trabalhos futuros apontados por [Erdens, 2010], que chama a atenção para a necessidade de desenvolvimento de métodos de análise de termos presentes em ontologias diferentes, está sendo tratado por este algoritmo proposto. Com essas adaptações foi gerada então a nova versão da ferramenta, o OWL-S Discovery 3.0, responsável por realizar uma tarefa de busca semi-automática de serviços web.

Além de aprimorar a eficácia da ferramenta, o presente trabalho converteu as versões anteriores, aplicações desktop, para uma aplicação web e desenvolveu uma interface mais amigável preocupando-se em apresentar dados que possibilitem que o usuário valide o resultado de sua busca.

Como suas versões anteriores, este trabalho deixa como legado uma ferramenta que possibilita o acoplamento de novos métodos de descoberta. Em relação a trabalhos futuros, pode-se destacar:

- Estudos sobre novas técnicas de descobertas de forma a complementar aquelas já implementadas.
- Evolução das presentes técnicas de modo a tratar de composição de serviços web.
- Utilização de outras formas de representação do conhecimento, tais como a WSMO, com o objetivo de testar possíveis ganhos de informação.

- Aplicação do algoritmo proposto à etapa referente à análise de pré-condições e efeitos.
- Encapsulamento do algoritmo em um ambiente de execução, de forma que o serviço poderá ser invocado após sua descoberta.

APÊNDICE A

A Listagem A apresenta o código de um serviço Web.

Listagem A: Serviço Web semântico escrito em OWL-S 1.2 com pré-condições e efeitos escritos em SWRL

```
1 […]
2 <owl:ObjectProperty rdf:ID="DELIVERY">
3       <rdfs:comment>hasAccount</rdfs:comment>
4       <rdfs:domain rdf:resource="#AirTransportation"/>
5       <rdfs:range rdf:resource="#Address"/>
6 </owl:ObjectProperty>
7
8 <owl:ObjectProperty rdf:ID="HASACCOUNT">
9       <rdfs:comment>hasAccount</rdfs:comment>
10      <rdfs:domain rdf:resource="#User"/>
11      <rdfs:range rdf:resource="#Journals"/>
12 </owl:ObjectProperty>
13
14 <owl:ObjectProperty rdf:ID="HASCARD">
15      <rdfs:comment>hasCard</rdfs:comment>
16      <rdfs:domain rdf:resource="#User"/>
17      <rdfs:range rdf:resource="#CreditCardAccount"/>
18 </owl:ObjectProperty>
19
20 <service:Service rdf:ID="BOOK_PRICE_SERVICE">
21 <service:presents rdf:resource="#BOOK_PRICE_PROFILE"/>
22 <service:describedBy rdf:resource="#BOOK_PRICE_PROCESS_MODEL"/>
23 <service:supports rdf:resource="#BOOK_PRICE_GROUNDING"/>
24 </service:Service>
25
26 <profile:Profile rdf:ID="BOOK_PRICE_PROFILE">
27 <service:isPresentedBy rdf:resource="#BOOK_PRICE_SERVICE"/>
28 <profile:serviceName xml:lang="en">
29 BookPriceService
30 </profile:serviceName>
31 <profile:textDescription xml:lang="en">
32 return price of a book
33 </profile:textDescription>
34 <profile:hasInput  rdf:resource="#_ORDINARYPUBLISHER"/>
35 <profile:hasInput  rdf:resource="#_NOVEL"/>
36 <profile:hasInput  rdf:resource="#_PAPERBACK"/>
37 <profile:hasOutput rdf:resource="#_LOCALAUTHOR"/>
```

```
38 <profile:hasOutput rdf:resource="#_GENRE"/>
39 <profile:hasPrecondition rdf:resource="#_HASACCOUNT"/>
40 <profile:hasPrecondition rdf:resource="#_HASCARD"/>
41 <profile:hasResult rdf:resource="#_DELIVERY"/>
42
43 <profile:has_process rdf:resource="BOOK_PRICE_PROCESS" />
44 </profile:Profile>
45 <process:ProcessModel rdf:ID="BOOK_PRICE_PROCESS_MODEL">
46 <service:describes rdf:resource="#BOOK_PRICE_SERVICE"/>
47 <process:hasProcess rdf:resource="#BOOK_PRICE_PROCESS"/>
48 </process:ProcessModel>
49
50 <process:AtomicProcess rdf:ID="BOOK_PRICE_PROCESS">
51 <process:hasInput  rdf:resource="#_ORDINARYPUBLISHER"/>
52 <process:hasInput  rdf:resource="#_NOVEL"/>
53 <process:hasInput  rdf:resource="#_PAPERBACK"/>
54 <process:hasOutput rdf:resource="#_LOCALAUTHOR"/>
55 <process:hasOutput rdf:resource="#_GENRE"/>
56 <process:hasPrecondition rdf:resource="#_HASACCOUNT"/>
57 <process:hasPrecondition rdf:resource="#_HASCARD"/>
58 <process:hasResult rdf:resource="#_DELIVERY"/>
59 </process:AtomicProcess>
60
61 <process:Input rdf:ID="_ORDINARYPUBLISHER">
62 <process:parameterType
64 rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
65 http://127.0.0.1/ontology/modified_books.owl#Ordinary-Publisher
66 </process:parameterType>
67 </process:Input>
68
69 <process:Input rdf:ID="_NOVEL">
70 <process:parameterType
71 rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
72 http://127.0.0.1/ontology/modified_books.owl#Novel
73 </process:parameterType>
74 </process:Input>
75
76 <process:Input rdf:ID="_PAPERBACK">
77 <process:parameterType
78 rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
79 http://127.0.0.1/ontology/modified_books.owl#Paper-Back
80 </process:parameterType>
81 </process:Input>
82
83 <process:Output  rdf:ID="_LOCALAUTHOR">
84 <process:parameterType
85 rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
86 http://127.0.0.1/ontology/modified_books.owl#Local-Author
```

```
87 </process:parameterType>
88 </process:Output>
89
90 <process:Output  rdf:ID="_GENRE"
91 <process:parameterType
92 rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
93 http://127.0.0.1/ontology/modified_books.owl#Genre
94 </process:parameterType>
95 </process:Output>
96
97 <expr:SWRL-Condition rdf:ID="_HASACCOUNT">
98                 <rdfs:label>hasAccount(_User, _Journals)</rdfs:label>
99                 <expr:expressionLanguage rdf:resource="&expr;#SWRL"/>
100                <expr:expressionBody rdf:parseType="Literal">
101         <swrl:AtomList>
102           <rdf:first>
103                         <swrl:IndividualPropertyAtom>
104             <swrl:propertyPredicate
105 rdf:resource="http://127.0.0.1/queries/1.1/r1.owls#HASACCOUNT"/>
106             <swrl:argument1
107 rdf:resource="http://127.0.0.1/ontology/modified_books.owl#User"/>
108              <swrl:argument2
109 rdf:resource="http://127.0.0.1/ontology/modified_books.owl#Journals"/>
110            </swrl:IndividualPropertyAtom>
111           </rdf:first>
112           <rdf:rest rdf:resource="&rdf;#nil"/>
113                 </swrl:AtomList>
114                </expr:expressionBody>
115 </expr:SWRL-Condition>
116
117 <expr:SWRL-Condition rdf:ID="_HASCARD">
118                 <rdfs:label>hasCard(_User, _CreditCardAccount)</rdfs:label>
119                 <expr:expressionLanguage rdf:resource="&expr;#SWRL"/>
120                 <expr:expressionBody rdf:parseType="Literal">
121         <swrl:AtomList>
122           <rdf:first>
123                         <swrl:IndividualPropertyAtom>
124             <swrl:propertyPredicate
125 rdf:resource="http://127.0.0.1/queries/1.1/r1.owls#HASCARD"/>
126              <swrl:argument1
107 rdf:resource="http://127.0.0.1/ontology/modified_books.owl#User"/>
128              <swrl:argument2 rdf:resource="http://127.0.0.1/ontology/Mid-level-
129 ontology.owl#CreditCardAccount"/>
130            </swrl:IndividualPropertyAtom>
131           </rdf:first>
132           <rdf:rest rdf:resource="&rdf;#nil"/>
133                 </swrl:AtomList>
```

```
134                 </expr:expressionBody>
135 </expr:SWRL-Condition>
136
137 <process:Result rdf:ID="_DELIVERY">
138                 <process:hasEffect>
139    <expr:SWRL-Expression>
140         <rdfs:comment>
141           DELIVERY
142         </rdfs:comment>
143         <expr:expressionBody rdf:parseType="Literal">
144          <swrl:AtomList>
145           <rdf:first>
146                           <swrl:IndividualPropertyAtom>
147            <swrl:propertyPredicate
148 rdf:resource="http://127.0.0.1/queries/1.1/r1.owls#DELIVERY"/>
149              <swrl:argument1 rdf:resource="http://127.0.0.1/ontology/Mid-level-
150 ontology.owl#AirTransportation"/>
151              <swrl:argument2 rdf:resource="http://127.0.0.1/ontology/Mid-level-
ontology.owl#Address"/>
152             </swrl:IndividualPropertyAtom>
153           </rdf:first>
154           <rdf:rest rdf:resource="&rdf;#nil"/>
155                 </swrl:AtomList>
156         </expr:expressionBody>
157       </expr:SWRL-Expression>
158     </process:hasEffect>
159   </process:Result>
[...]
```

REFERÊNCIAS

ALVES, F. O. ; CLARO, D. B., **Serviços Web e a Computação em Nuvem: Definições e Desafios,** Apresentação de trabalho/Seminário, 2011.

AMORIM, R., **Um algoritmo híbrido para descoberta de serviços Web semânticos,** Trabalho de Conclusão de Curso (Graduação), 2009.

AZEVEDO, L. G. ; FURTADO, C. ; PEREIRA ; BAIAO, F. ; SANTORO, F., **Arquitetura Orientada a Serviço: Conceituação,** Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT-0012/2009, 2009.

BREITMAN, K., **Web Semântica a Internet do futuro**, LTC, 2005.

CARDOSO, J., **Semantic Web Services Theory, Tools and Applications,** IGI Global, 2007.

CARVALHO, C.; LIMA, J., **Ontologias – OWL (*Web Ontology Language*),** Relatórios Técnicos do INF/UFG, RT-INF_004-05, 2005.

ERDENS, L., **Um algoritmo de análise de pré-condições e efeitos para descoberta de serviços Web semânticos,** Trabalho de Conclusão de Curso (Graduação), 2010.

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

FORTE, M.; SOUZA, W.L.; PRADO, A.F, **Utilizando Ontologias e Serviços Web na Computação Ubíqua,** XX Simpósio Brasileiro de Engenharia de Software, 2006.

GONDIM, F., **Algoritmo de Comparação de Strings para Integração de Esquemas de Dados,** Trabalho de Conclusão de Curso (Graduação), 2006.

GUARINO, N., **Formal Ontology in Information Systems,** International Conference on Formal Ontology in Information Systems (FOIS'98), 1998.

JOSUTTIS, N. M., **SOA in Practice – The Art of Distributed System Design,** O'Reilly, 2007.

LOPES, D. C. P.; HAMMOUDI, S; DE SOUZA, J.; BONTEMPO, A., **Metamodel matching: Experiments and Comparison,** International Conference on Software Engineering, 2006.

LUNA, A.; SANTOS, N., **Anotações Semânticas em Serviços Web para Aprendizagem Colaborativa,** Cadernos do IME: Série Informática, 2007.

MARTIN, D.; BURSTEIN, M.; HOBBS, J.; LASSILA, O.; MCDERMOTT, D.; MCILRAITH, S.; NARAYANAN, S.; PAOLUCCI, M.; PARSIA, B.; PAYNE, T. *et al.*, **OWL-S: Semantic Markup for Web Services,** Internet Computing, 2004.

NOGUEIRA, V., **Modelagem de Base de Conhecimentos Baseada em Ontologia: Estudo de Caso em Recuperação de Informação Bibliográfica,** Trabalho de Conclusão de Curso (Graduação), 2010.

OASIS, BROWN, P.; ESTEFAN, J.; LASKEY, K.; McCABE, F.; THORNTON, D., **Reference Architecture Foundation for Service Oriented Architecture Version 1.0,** 2012. Disponível em: <http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.html> Acessado em: Novembro de 2013.

ORLANDIN, P., **Serviços Web Semânticos,** 2005. Disponível em: < http://pt.slideshare.net/paulovitor_e/servicos-web-semanticos> Acessado em: Novembro de 2013.

PADILHA, N.; BAIÃO, F.; REVOREDO, K., **Ontologias de fundamentação apoiando o alinhamento de ontologias de domínio,** VIII Simpósio Brasileiro de Sistemas de Informação, 2012.

PAOLUCCI, M. *et al.*, **Semantic matching of web services capabilities,** International Semantic Web Conference (ISWC'02), 2002.

PAPAZOGLOU, M. P.; HEUVEL, W., **Service Oriented Architectures: approaches, technologies and research issues,** VLDB Journal, 2007.

REVOREDO, K.; SILVA, A.; BAIÃO, F.; GUEDES, A., **Classificador de Alinhamento de Ontologias Utilizando Técnicas de Aprendizado de Máquina,** IX Simpósio Brasileiro de Sistemas de Informação, 2013.

SAMPER, J. J. *et al.*, **Improving semantic web service discovery,** Journal of networks, 2008.

SENA, V. A. ; CLARO, D. B. ; AMORIM, R. ; LOPES D., **Similaridade semântica na composição de sistemas de informação através dos serviços web,** VI Simpósio Brasileiro de Sistemas de Informação, 2010.

SHVAIKO, P.; YATSKEVICH, M.; GIUNCHIGLIA, F., **S-match: an algorithm and an implementation of semantic alinhamento,** European Semantic Web Symposium (ESWS'04), 2004.

SOAP, GUDGIN, M; HADLEY, M.; MOREAU, J.; NIELSEN, H., **SOAP Version 1.2,** 2001. Disponível em: < http://www.w3.org/TR/2001/WD-soap12-20010709/ > Acessado em: Novembro de 2013.

VALADARES, J.; MELO, I., **Recuperação da Informação de Serviços Odontológicos Baseada em Ontologias de Tarefa/Aplicação,** Trabalho de Conclusão de Curso (Graduação), 2010.

W3C, BOOTH, D.; HAAS, H.; McCABE, F.; NEWCOMER, E.;CHAMPION, M.; FERRIS, C.; ORCHARD, D., **Web Services Architecture, W3C Working Group Note,** 2004 Disponível em <http://www.w3.org/TR/ws-arch/#engaging> Acessado em: Novembro de 2013.

XML, BRAY, T.; PAOLI, J.; SPERBERG-McQUEEN, C.; MALER, E.; YEARGEAU F., **Extensible Markup Language (XML) 1.0 (Fifth Edition),** 2008. Disponível em: <http://www.w3.org/TR/2008/REC-xml-20081126/ > Acessado em: Novembro de 2013.

WSDL, CHRISTENSEN, E.; CURBERA, F.; MEREDITH, G.; WEEWARANA, S., **Web Services Description Language (WSDL) 1.1,** 2001. Disponível em: < http://www.w3.org/TR/2001/NOTE-wsdl-20010315 > Acessado em: Novembro de 2013.