iSWS: infraestrutura para publicação, descoberta e composição de Serviços Web Semânticos

Cássio V. S. Prazeres
Universidade Federal da Bahia
prazeres@dcc.ufba.br

Maria da Graça C. Pimentel
Universidade de São Paulo
mgp@icmc.usp.br

RESUMO

Propostas de padrões, tecnologias e infraestruturas para a Web Semântica devem levar em consideração a atual infraestrutura da Web. Nesse contexto, este artigo apresenta uma infraestrutura para implantação de Serviços Web Semânticos que utiliza a infraestrutura da Web atual. A iSWS engloba a implementação de ferramentas e módulos com a finalidade de realizar publicação, descoberta e composição de Serviços Web Semânticos. Alguns experimentos foram realizados e seus resultados são apresentados.

ABSTRACT

Proposals for standards, technologies and infrastructure for the Semantic Web must take into consideration the current Web infrastructure. In this direction, this paper presents an infrastructure for Semantic Web Services deployment that uses the infrastructure of the current Web: iSWS. This infrastructure includes the implementation of tools and modules in order to achieve publication, discovery and composition of Semantic Web Services. Some experiments were performed and results are presented.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: On-line Information Services—*Web-based services*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Semantic Web Services, Publishing, Discovery, Composition

1. INTRODUÇÃO

A simplicidade dos documentos HTML (*HyperText Markup Language*) proporcionou a rápida expansão da Web atual [1, 3]. Porém, essa mesma simplicidade tem limitado a Web em termos de aplicações inteligentes que poderiam explorar a grande quantidade de informações disponíveis. Os documentos HTML possuem um modelo simples de hipertexto com pouca estruturação sintática do conteúdo, não possuem uma separação clara de informações de conteúdo e de apresentação, e quase não contêm informações sobre a semântica do conteúdo. Berners-Lee et al. [1] afirmam que a proposta da Web Semântica inclui estruturar o conteúdo da Web de forma que aplicações possam processar não apenas palavras chaves em documentos na Web. As aplicações vão poder chegar a conclusões e tomar decisões a partir da semântica do conteúdo do documento. Uma nova forma de conteúdo Web que pode ser entendido e manipulado por computadores vai gerar uma revolução de novas possibilidades, serviços e aplicações oferecidos. A Web Semântica permitirá que usuários possam localizar, selecionar, compor e invocar esses Serviços Web automaticamente [10].

Apesar da vasta gama de novas possibilidades e oportunidades, a proposta da Web Semântica não é de substituir a Web atual e sim de estendê-la a partir da adição de estrutura e de semântica explícita aos documentos da Web e à infraestrutura já existentes. Dessa forma, as propostas de novos padrões, arquiteturas, tecnologias, e infraestruturas para a Web Semântica precisam levar em consideração a infraestrutura atual da Web.

Nesse cenário, vários trabalhos investigam a descoberta e composição de Serviços Web Semânticos (p.ex: [2, 4, 5, 8, 9, 16, 18]). Esses trabalhos não apresentam uma infraestrutura que permite, em ordem tempo polinomial, realizar a composição automática de Serviços Web com base em custos, como realizado pela iSWS.

Nesse contexto, este artigo apresenta a iSWS, uma **i**nfraestrutura para publicação, descoberta e composição de **S**erviços **W**eb **S**emânticos, que utiliza tecnologias e padrões existentes na Web atual com adição de semântica para a implementação das funcionalidades de publicação, descoberta e composição de Serviços Web Semânticos.

Neste artigo, a Seção 2 apresenta a ontologia OWL-S; as Seções 3 e 4 a infraestrutura iSWS; a Seção 5 resultados de uma avaliação da infraestrutura; trabalhos relacionados são discutidos na Seção 6; a Seção 7 conclui o trabalho.

2. SERVIÇOS WEB SEMÂNTICOS

Serviços Web Semânticos são baseados em duas das mais importantes tendências da Web: Serviços Web e Web Semântica. A área de Serviços Web Semânticos é definida como o enriquecimento das descrições de Serviços Web através do uso de anotações da Web Semântica [10]. O objetivo

WebMedia'11: Proceedings of the 17th Brazilian Symposium on Multimedia and the Web. Full Papers.
October 3 -6, 2011, Florianópolis, SC, Brazil.
ISSN 2175-9642.
SBC - Brazilian Computer Society

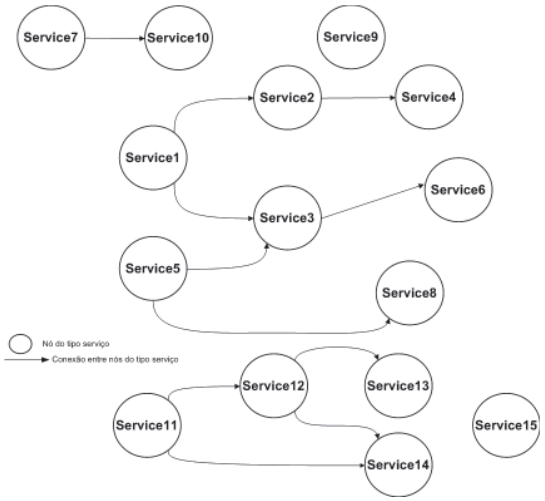

Figura 1: Repositório de serviços como grafo.

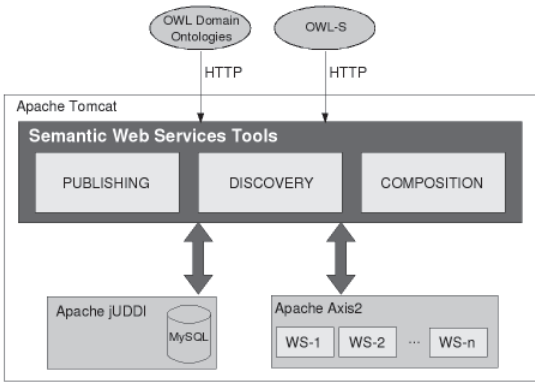

Figura 2: Visão geral da iSWS.

dos Serviços Web Semânticos é o de possibilitar a automação de tarefas como descoberta, composição e invocação de Serviços Web [12].

A ontologia OWL-S tem como objetivo agregar semântica na descrição de Serviços Web com o propósito de possibilitar a automação da descoberta, invocação e composição de Serviços Web através de descrições semânticas [11].

A iSWS utiliza a ontologia OWL-S e, para realizar a composição automática por meio algoritmos de cálculo do caminho de custo mínimo, o repositório de Serviços Web Semânticos é modelado como um grafo direcionado e com custos. Conforme ilustrado na Figura 1, cada serviço corresponde a um nó no grafo e os serviços são conectados entre si com base na similaridade semântica de suas entradas e saídas. Ou seja, se um serviço **A** possuir uma saída semanticamente similar (via *subsumption reasoning* [17]) a uma entrada de um outro serviço **B**, é criada uma aresta de **A** para **B**.

3. ISWS

A infraestrutura desenvolvida neste trabalho utiliza tecnologias e padrões existentes na Web atual para a implementação das funcionalidades de publicação, descoberta e composição de Serviços Web Semânticos. A Figura 2 apresenta a infraestrutura iSWS proposta neste trabalho e que fornece as funcionalidades de publicação, descoberta e composição de Serviços Web Semânticos. A infraestrutura provê serviços de repositório UDDI por meio da plataforma *jUDDI*[1]. Os serviços utilizados como base de testes estão implementados na plataforma Axis2[2] e todas as funcionalidades desenvolvidas (publicação, descoberta e composição), bem como as plataformas jUDDI e Axis2, estão implementadas no servidor de aplicações Web *Tomcat*[3]. As ontologias de domínio que descrevem as entradas e saídas dos Serviços Web Semânticos, bem como a ontologia OWL-S, são recuperadas por meio do protocolo *HTTP*.

A plataforma *jUDDI* (Figura 2) é utilizada neste trabalho para oferecer as funcionalidades de um repositório UDDI. O *jUDDI* é utilizado tanto diretamente, para publicação e descoberta de serviços, como indiretamente, por meio da descoberta, para a composição de serviços. Os descritores OWL-S são mapeados para UDDI e armazenados na base de dados relacional utilizada pelo *jUDDI*. Os detalhes da utilização do *jUDDI* nos processos de publicação, descoberta e composição, bem como a forma que os descritores OWL-S são mapeados para UDDI, são apresentados na Seção 4.

A plataforma *Axis2* (Figura 2) é utilizada neste trabalho para desenvolver alguns serviços que serviram como base de testes para algumas das implementações deste trabalho. Descrições em OWL-S para serviços simulados foram geradas automaticamente. Além disso, esqueletos de Serviços Web implementados na linguagem Java são gerados também de forma automática a partir dos descritores OWL-S e implementados na plataforma *Axis2*. Os detalhes da geração automática dos Serviços Web, bem como da sua implantação na plataforma *Axis2*, são descritos na Seção 4.

A infraestrutura está implantada em um servidor Web *Apache Tomcat*. Tanto as plataformas *jUDDI* e *Axis2*, que são aplicações Web, como as funcionalidades para publicação, descoberta e composição, executam nesse servidor.

4. ARQUITETURA DA ISWS

Para validar a proposta deste trabalho foram implementados alguns módulos de software que são utilizados para publicação, descoberta e composição de Serviços Web Semânticos. A Figura 3 apresenta a arquitetura de software utilizada nas implementações das funcionalidades de publicação, descoberta e composição que estão dispostas como *Semantic Web Services Tools* na infraestrutura da Figura 2. Todos os módulos presentes na Figura 3 estão implementados na linguagem *Java* (*J2SE* 1.5 e *J2EE* 5.0).

Os módulos apresentados em tons de cinza médio na Figura 3 são ferramentas ou *APIs* (*Application Programming Interface*) provenientes de outros trabalhos e que são utilizados para prover algumas funcionalidades que não são objetivos diretos deste trabalho. Esses módulos e suas funcionalidades são discutidos na Seção 4.1 por serem necessários ao entendimento dos módulos que foram implementados para a validação deste trabalho (apresentados em tom de cinza claro na Figura 3) e estão descritos nas seções 4.2, 4.3 e 4.4.

4.1 Componentes: reúso

Os módulos apresentados em cinza médio na Figura 3

[1]http://ws.apache.org/juddi/
[2]http://ws.apache.org/axis2/
[3]http://tomcat.apache.org/

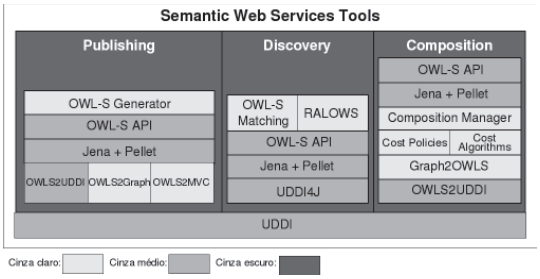

Figura 3: Arquitetura da iSWS.

são componentes ou ferramentas de código fonte aberto, importados de outros projetos e utilizados com finalidades específicas neste trabalho: *Jena*[4], *OWL-S API*[5], Pellet[6], *OWLS2UDDI*[7] e *UDDI4J*[8].

O *Jena* é um *framework* implementado na linguagem *Java* para o desenvolvimento de aplicações voltadas para a Web Semântica. O *Jena* possui *APIs* para manipulação de modelos RDF e também de modelos OWL, e ainda inclui uma máquina de inferência própria. Todo módulo deste trabalho que trata de documentos OWL utiliza o *framework Jena* direta ou indiretamente. Todas as ontologias são manipuladas diretamente pelo *Jena*, exceto a ontologia OWL-S que utiliza o *Jena* por meio do componente *OWL-S API*.

O componente *OWL-S API* é uma *API*, também em *Java* e totalmente desenvolvida a partir do *framework Jena*, para edição, criação e validação de documentos OWL-S. Esse componente provê todos os métodos e construtores necessários para manipulação de documentos OWL-S, e é utilizado na maioria dos módulos implementados. O componente *OWL-S API* está para documentos OWL-S assim como o *framework Jena* OWL *API* está para documentos OWL.

A ferramenta *Pellet* é uma máquina de inferência que, em conjunto com o *framework Jena*, é utilizada neste trabalho nos estágios do algoritmo de descoberta que realizam *matching* baseado em *subsumption*.

O componente *OWLS2UDDI* é uma ferramenta desenvolvida por Srinivasan et al. [17] que mapeia os descritores de serviço em OWL-S para o padrão UDDI. Essa ferramenta permite a adição de semântica a repositórios UDDI e, utilizada em conjunto com a infraestrutura *jUDDI*, tal como neste trabalho, provê uma infraestrutura para publicação de Serviços Web Semânticos descritos em OWL-S.

O componente *UDDI4J* é uma *API* para manipulação de repositórios UDDI. Essa *API* é utilizada diretamente pela ferramenta *OWLS2UDDI* e também é utilizada, neste trabalho, para realizar consultas ao repositório UDDI implementado na infraestrutura do *jUDDI*.

4.2 Publicação

São três os módulos implementados com o propósito de prover e validar a publicação de serviços.

4.2.1 Módulo *OWL-S Generator*

Esse módulo tem a função de gerar, automaticamente, descrições de serviços em OWL-S para servir de base de testes na descoberta e na composição de serviços. Esse módulo cria serviços artificias e os publica no repositório UDDI.

[4]http://jena.sourceforge.net/
[5]http://www.daml.ri.cmu.edu/owlsapi/
[6]http://clarkparsia.com/pellet/
[7]http://owl-s2uddi.projects.semwebcentral.org/
[8]http://uddi4j.sourceforge.net/

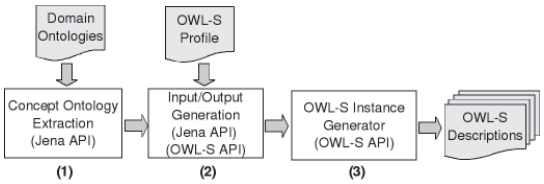

Figura 4: Visão geral do módulo *OWL-S Generator*.

A Figura 4 apresenta uma visão geral do *OWL-S Generator*, que tem como entrada ontologias de domínio e a ontologia de perfil do serviço em OWL-S. O perfil do serviço em OWL-S possui a descrição das entradas e saídas de um serviço com propósito de descoberta. Cada entrada e saída é descrita com uma ontologia de domínio referente ao domínio do serviço em questão. Na Figura 4, o módulo *OWL-S Generator*: (1) extrai conceitos de ontologias de domínio; (2) cria entradas e saídas descritas com esses conceitos; e (3) gera instâncias de serviços descritos em OWL-S.

4.2.2 Módulo *OWLS2Graph*

O módulo *OWLS2Graph* é responsável por adicionar serviços descritos em OWL-S ao grafo (ver Figura 1) de todos os serviços cada vez que um novo serviço é registrado no repositório UDDI. O Documento 1 apresenta o algoritmo para publicação de um novo serviço no repositório. O objetivo do algoritmo é descobrir todos os serviços registrados no repositório que têm conexão semântica com o serviço que está sendo publicado e efetuar as conexões no grafo.

Documento 1 Algoritmo para inserir um novo serviço

```
1  procedimento insereServico (owlsServico, ci): booleano
2  inicio
3    grafo.adicionaVertice();
4    para i=1 até Nservico faça
5    //para cada serviço no repositório UDDI
6    inicio
7      se existe(saida em servico[i]
       similar a entrada em owlsServico) entao
8        //existe ao menos uma saida em servico[i]
9        //que conecta com entrada em owlsServico
10       grafo.insereAresta(servico[i], owlsServico, ci);
11
12     se existe(saida em owlsServico
       similar a entrada em servico[i]) entao
13       //existe ao menos uma saida em owlsServico
14       //que conecta com entrada em servico[i]
15       grafo.insereAresta(owlsServico, servico[i],
         servico[i].ci);
16   fim para
17   //registra o serviço no UDDI
18   retorna registraUDDI(owlsServico);
19 fim insereServico
```

O procedimento *insereServico* (na linha 1 do Documento 1) recebe como parâmetro a descrição em OWL-S do serviço a ser publicado e o custo inicial do serviço. Esse custo inicial, que é calculado segundo a política de custos escolhida pelo módulo *Cost Policies* (Seção 4.4.2), é o que vai determinar o comportamento do algoritmo de composição automática.

Inicialmente um novo nó é adicionado ao grafo na linha 3 do Documento 1. Da linha 4 até a linha 16 todos os serviços do repositório UDDI são percorridos em busca de serviços que tenham conexão semântica com o serviço que está sendo publicado. As linhas 7 a 15 verificam se existe conectividade e, caso exista, novas arestas são inseridas no grafo.

As novas arestas criadas podem ser: conectando um ser-

Tabela 1: Mapeamento do OWL-S para a Java.

Type	OWL-S	Java Code
Process	Composite	Servlet Class
	Atomic	Method
Control Construct	Sequence	sequence of code
	If-Then-Else	if-then-else
	Choice	switch
	Repeat-While	while
	Rapeat-Until	do-while
Parameter	Input	arguments of Method
	Output	return of Method
	parameterType	type of attributes

viço existente no grafo ao novo serviço (novo nó no grafo) como o contrário. O primeiro caso pode ser visto na linha 10 do Documento 1: ao *servico[i]* (do repositório) adicionou-se uma aresta direcionada ao novo serviço (*owlsServico*) com o custo igual a *ci* (custo inicial do novo serviço). O segundo caso é mostrado na linha 15 do Documento 1: ao *owlsServico* (novo serviço) é adicionada uma aresta direcionada a um serviço do repositório (*servico[i]*) utilizando o custo igual a *servico[i].ci* (custo inicial do serviço do repositório).

Toda vez que um serviço é publicado ele é adicionado ao grafo utilizado para realizar a composição automática de serviços. O módulo *OWLS2Graph* utiliza a descoberta de serviços (módulo *OWL-S Matching*) para buscar todos os serviços que podem ser conectados ao serviço que está sendo publicado e realiza as conexões.

4.2.3 Módulo *OWLS2MVC*

A ontologia OWL-S possui descritores para entrada e saída, processos, construtores de controle de fluxo e de fluxo de dados, dentre outros. Em função disso, observou-se a oportunidade de propor o mapeamento entre os descritores e construtores do OWL-S e linguagens de programação. A Tabela 1 apresenta o mapeamento proposto para a linguagem Java utilizando uma implementação da arquitetura MVC (*Model-View-Controller*) [7] para a Web.

Na Tabela 1: os processos compostos do OWL-S são mapeados para um *Servlet* em Java e os atômicos para métodos; os construtores de controle de fluxo do OWL-S são mapeados para as estruturas de controle da linguagem Java; e os parâmetros do OWL-S (entradas e saídas) são mapeados para argumentos e retornos de métodos em Java.

O módulo *OWLS2MVC* implementa o mapeamento apresentado na Tabela 1 tal como ilustrado na Figura 5. A partir do mapeamento é gerado um esqueleto de uma aplicação Web na arquitetura MVC. Essa aplicação é então incorporada, na forma de um Serviço Web, à infraestrutura da Figura 2 e o OWL-S correspondente é inserido no repositório UDDI por meio do módulo *OWLS2UDDI*.

A Figura 5 apresenta o módulo *OWLS2MVC*, que tem como entrada instâncias OWL-S de um Serviço Web e documentos XSLT. Esse módulo executa: (1) extração dos dados apresentados na Tabela 1 a partir dos documentos OWL-S de entrada; (2) extração dos esquemas referentes aos parâmetros de entrada e saída do OWL-S; (3) transformação XSLT que gera interfaces gráficas para o Serviço Web.

A extração dos dados provenientes do documento OWL-S de entrada (Fig. 5(1)) gera os *Servlets* (Fig. 5(a)) que são os controladores da aplicação Web em MVC e também gera como sub-produto os esquemas referentes às entradas e saídas do serviço. Os esquemas são processados (Fig. 5(1)) e

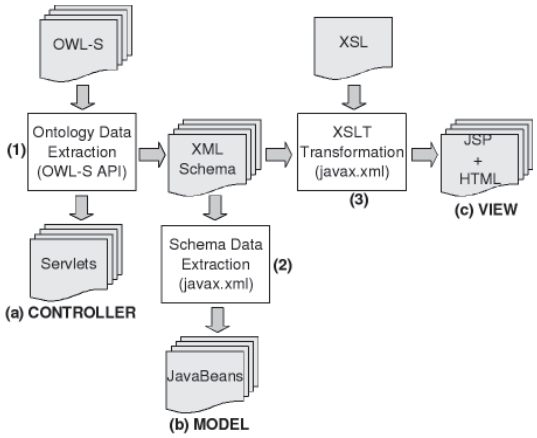

Figura 5: Visão geral do módulo *OWLS2MVC*.

então são gerados os *JavaBeans* (Fig. 5(b)), que são os modelos da aplicação Web em MVC. Finalmente, uma transformação XSLT é aplicada (Fig. 5(3)) aos esquemas para gerar algumas interfaces gráficas em *HTML* e *JSP* (Fig. 5(c)), que são as visões da aplicação Web em MVC.

O módulo *OWLS2MVC* é utilizado para, a partir de documentos OWL-S, gerar o esqueleto de uma aplicação para Serviço Web Semântico na arquitetura MVC. Esse esqueleto de aplicação pode ser utilizado pelo desenvolvedor para gerar uma aplicação completa. Neste trabalho, o módulo *OWLS2MVC*, em conjunto com o módulo *OWL-S Generator*, foi utilizado para gerar Serviços Web para uma base de testes utilizada na avaliação do trabalho. Todos os Serviços Web gerados pelos módulos *OWL-S Generator* e *OWLS2MVC* foram registrados no repositório *jUDDI* e implantados na infraestrutura *Axis2* apresentados na Seção 3.

4.3 Descoberta

São dois os módulos implementados com o propósito de validar a descoberta de Serviços Web Semânticos.

4.3.1 Módulo *RALOWS*

O módulo *RALOWS* refere-se a uma extensão da *OWL-S API* feita com o intuito de prover informações temporais em Serviços Web que possuem restrições de tempo para sua utilização efetiva. Esse módulo, específico para uma classe de Serviços Web, é detalhado em outro trabalho [13].

4.3.2 Módulo *OWL-S Matching*

O módulo *OWL-S Matching* utiliza o *framework* Jena, a *OWL-S API* e a máquina de inferência *Pellet*, combinados, para prover a descoberta de Serviços Web Semânticos.

Na implementação atual da iSWS, o *matching* é realizado pelo algoritmo de descoberta proposto por Prazeres et al. [15]: esse algoritmo pode ser substituído por qualquer algoritmo de descoberta que retorne um conjunto de serviços similares ao serviço solicitado.

O *Jena* é utilizado pelo OWL-S API e também para leitura das ontologias de domínio externas que definem as entradas e saídas do perfil do serviço em OWL-S. Por meio do *Jena* e da *OWL-S API*, cria-se o perfil do serviço requerido a partir das entradas e saídas informadas pelo usuário. Esse perfil é comparado com os perfis de todos os serviços que estão armazenados no *jUDDI* por meio do *matching* baseado em

subsumption realizado pela máquina de inferência *Pellet*.

Para acessar o *jUDDI*, o módulo *OWL-S Matching* utiliza a *API UDDI4J*. Com essa *API* o módulo *OWL-S Matching* recupera os perfis dos serviços existentes no repositório.

O módulo *OWL-S Matching* também é utilizado, na publicação e na composição de serviços, para descobrir as similaridades entre entradas e saídas de serviços do repositório *jUDDI* com o propósito de estabelecer as conexões semânticas no grafo de serviços.

4.4 Composição

Quatro módulos foram implementados para prover a composição automática de Serviços Web Semânticos.

4.4.1 Módulo *Composition Manager*

O módulo *Composition Manager* tem a função de gerenciar os outros módulos da composição de serviços. Esse módulo possui funções tais como decidir qual custo vai ser aplicado, escolher qual algoritmo para cálculo de caminho de custo mínimo vai ser aplicado, efetuar a ligação entre os outros módulos da composição e integrar a geração da composição final com a ferramenta *OWLS2UDDI* com o propósito de armazenar a composição final no repositório *jUDDI* para utilização posterior.

4.4.2 Módulo *Cost Policies*

A política de custos utilizada no grafo do repositório de serviços (Figura 1) é definina por meio da utilização do módulo *Cost Policies*, que está implementado para receber custos referentes aos atributos funcionais (baseados na funcionalidade do serviço) ou não-funcionais. Dessa forma, é esse módulo que vai determinar o comportamento do algoritmo de composição. Na implementação da iSWS, a política de custos implantada foi a definida por Prazeres et al. [14], que utiliza custos funcionais. Entretanto, o módulo é extensível, ou seja, outras políticas de custo podem ser inseridas.

O módulo *Cost Policies* recebe como parâmetro o grafo com os custos uniformes (custos inicias - *ci* - definidos no módulo *OWLS2Graph*) e tem a finalidade de alterar os custos com base nos descontos estabelecidos para cada serviço. *Cost Policies* utiliza o módulo *OWL-S Matching* que vai descobrir todos os serviços no repositório *jUDDI* que possuem entradas disponíveis e saídas requeridas.

Após descobertos os serviços com entradas disponíveis e saídas requeridas, o módulo *Cost Policies* realiza operações sobre o grafo com o objetivo de calcular e aplicar os descontos para cada serviço descoberto. Os serviço recebem descontos com base nas similaridades entre entradas e saídas com o serviço requerido.

O desconto, definido pelo algoritmo de Prazeres et al. [14], para cada entrada do serviço é: total (igual a 3) se a entrada do serviço (*InServico*) for equivalente a uma entrada disponível (*InR*); parcial e igual a 2 se a entrada do serviço (*InServico*) for um sub-conceito de uma entrada disponível (*InR*); parcial e igual a 1 se a entrada do serviço (*InServico*) for um super-conceito de uma entrada disponível (*InR*); nulo nos casos em que não existe similaridade.

E o desconto, também definido pelo algoritmo de Prazeres et al. [14], para cada saída do serviço é: total (igual a 3) se a saída do serviço (*OutServico*) for equivalente a uma saída requerida (*OutR*); parcial e igual a 2 se a saída do serviço (*OutServico*) for um super-conceito de uma saída requerida (*OutR*); parcial e igual a 1 se a saída do serviço (*OutServico*) for um sub-conceito de uma saída requerida (*OutR*); nulo nos casos em que não existe similaridade.

Tabela 2: Mapeamento do grafo para a OWL-S.

Grafo	Construtor OWL-S
Caminho seqüencial	*Sequence*
Caminho paralelo com barreira de sincronização apenas no início (*AND-Split*)	*Split*
Caminho paralelo com barreiras de sincronização no início e no final (*AND-Split+AND-Join*)	*Split+Join*
Caminho paralelo sem barreiras de sincronização	Dois construtores *Sequence*
OR-Split com exatamente duas opções	*If-Then-Else*
OR-Split com mais de duas opções	*Choice*
Iteration	*Repeat-While* ou *Repeat-Until*

O desconto total do serviço é obtido com a soma dos dois descontos parciais, desconto das entradas e desconto das saídas, da seguinte forma:

$DT_s = DT_{in} + DT_{out}$, em que:
DT_s: desconto total do serviço;
DT_{in}: desconto total em relação às entradas do serviço;
DT_{out}: desconto total em relação às saídas do serviço.

4.4.3 Módulo *Cost Algorithms*

O grafo, já com os custos alterados pelo módulo *Cost Policies*, é enviado ao módulo *Cost Algorithms* que calcula os caminhos de custo mínimo. Esse módulo possui uma implementação do algoritmo de Dijkstra para cálculo dos caminhos de custo mínimo e pode ser modificado para a inclusão de outros algoritmos, entre outros, algoritmos que aceitam custos negativos para representar, por exemplo, promoções em preços de serviços pagos).

O módulo *Cost Algorithms* calcula um caminho de custo mínimo para cada saída requerida, a partir das entradas disponíveis, utilizando o algoritmo de composição proposto por Prazeres et al. [14]. A avaliação da aplicação desse algoritmo para cálculo dos caminhos de custo mínimo para as saídas requeridas da requisição de um usuário são apresentados na Seção 5. Esses caminhos são utilizados pelo módulo *Graph2OWLS* para a geração da composição final.

4.4.4 Módulo *Graph2OWLS*

O módulo *Graph2OWLS* implementa o mapeamento da composição final para o OWL-S tal como descrito na Tabela 2. Esse módulo recebe como entrada os caminhos para cada saída requerida gerados pelo módulo *Cost Algorithms* e compõe um grafo único e conectado, eliminando os caminhos duplicados por meio da análise das intersecções entre os caminhos encontrados.

Dessa forma, o módulo *Graph2OWLS* realiza o mapeamento do grafo para uma representação utilizando a subontologia de processos em OWL-S. Com isto, o novo serviço composto pode ser tratado como um serviço único com seu fluxo interno baseado na lógica de processos do OWL-S. Além disso, o módulo *Graph2OWLS* registra a composição em OWL-S no repositório de serviços *jUDDI* , que poderá ser utilizada da próxima vez que um usuário buscar por serviços com as mesmas funcionalidades.

Tabela 3: Configuração dos experimentos executados.

Varriável	Variação
Número de serviços	10 – 1000
Número de parâmetros por serviços	10 – 20

Tabela 4: Tempo de resposta para a descoberta de serviço no repositório UDDI.

Número de serviços	Tempo de descoberta (ms)
10	879
100	947
500	1076
1000	1126

5. AVALIAÇÃO DA ISWS

Conforme descrito na Seção 4, este trabalho incorporou ferramentas e *APIs* desenvolvidas por terceiros. Dessa forma, o desempenho das funcionalidades de publicação, descoberta e composição depende também do desempenho dessas ferramentas e APIs. Na prática, a ferramenta externa que realmente tem influência no desempenho da iSWS é a máquina de inferência *Pellet* que, em conjunto com a API *Jena*, possibilita realizar a inferência para o *matching* baseado em *subsumption* utilizado para a descoberta de serviços.

A descoberta de serviços é utilizada também nas funcionalidades de publicação e composição e, por esse motivo, é necessário primeiro avaliar o desempenho da descoberta para se obter o desempenho das outras duas funcionalidades.

O ambiente de hardware e software utilizado para as medições inclui CPU Intel Pentium 4 3.0 GHz HT, 1 GiB de RAM, sistema operacional Linux Ubuntu 8.04, plataforma Java J2SE 1.5 e Eclipse 3.3.2. Os experimentos executados com propósito de avaliação têm o objetivo de avaliar se o desempenho do sistema como um todo é aceitável para as funcionalidades de publicação, descoberta e composição. As configurações dos experimentos estão apresentadas na Tabela 3: ao repositório UDDI, implementado com o *jUDDI* e o *OWLS2UDDI*, foram inseridos de dez a mil serviços, sendo que cada serviço tem entre 10 e 20 entradas e saídas no total.

5.1 Descoberta

Nos testes realizados, o algoritmo para descoberta, implementado no módulo *OWL-S Matching* da Figura 3, apresentou o tempo de resposta para a busca por um serviço que possua as entradas e saídas requeridas pelo usuário tal como apresentado na Tabela 4. Do total de mil serviços inseridos no repositório, foram definidas quatro classes (10, 100, 500 e 1000) referentes à quantidades de serviços presentes no repositório no momento da busca pelo serviço.

Cada valor de tempo apresentado na Tabela 4 é referente ao maior valor de tempo medido nos testes para cada número de serviços. Esses valores máximos ocorreram nos casos em que os serviços que possuíam o número máximo de parâmetros (20 – ver Tabela 3) foi utilizado na descoberta. O tempo da descoberta é influenciado pela inferência realizada pela máquina *Pellet* nos estágios iniciais do algoritmo de descoberta.

A inferência neste trabalho foi realizada sempre em ontologias do dialeto OWL DL e, por esse motivo, os tempos para a descoberta não aumentam exponencialmente com o número de serviços (Tabela 4).

O tempo maior de resposta obtido nos testes foi de *1126ms*.

Tabela 5: Tempo de resposta para a publicação de serviço. Onde: N_{Out} = número de saídas do serviço. N_{In} = número de entradas do serviço.

Função	Complexidade	Tempo (ms)
Registro no UDDI	–	5480
Descoberta $Out \rightarrow In$	–	1126
Descoberta $In \rightarrow Out$	–	1126
Conexões $Out \rightarrow In$	$O(N_{Out})$	< 10
Conexões $In \rightarrow Out$	$O(N_{In})$	< 10
Tempo total	–	< 7752

Esse tempo máximo é utilizado (Seções 5.2 e 5.3) para avaliar as funcionalidades de publicação e composição.

5.2 Publicação

A publicação de um serviço executa duas funções distintas: o registro do serviço no repositório e a inclusão do serviço no grafo utilizado para a composição. Essas funções são executadas em cinco procedimentos listados na Tabela 5.

Dessa forma, o desempenho da publicação do serviço depende dos cinco procedimentos (Tabela 5): tempo de resposta do registro no repositório UDDI; tempo de resposta para a descoberta das conexões (dois procedimentos: $Out \rightarrow In$ e $In \rightarrow Out$) com o grafo; e tempo de resposta para a criação das conexões (dois procedimentos: $Out \rightarrow In$ e $In \rightarrow Out$).

O registro no repositório UDDI é efetuado utilizando a ferramenta *OLWS2UDDI* descrita na Seção 4. A transformação do documento OWL-S para descritores UDDI demanda o maior tempo da publicação (Tabela 5).

A Tabela 5 ilustra que o tempo da descoberta de conexões é o tempo demandado para a realização de duas descobertas no repositório UDDI: a descoberta de todos os serviços que podem se conectar ao serviço que está sendo publicado ($Out \rightarrow In$); e a descoberta de todos os serviços para os quais o serviço que está sendo publicado pode se conectar ($In \rightarrow Out$). O tempo para cada uma das descobertas é dado pelo tempo máximo apresentado na Tabela 4.

A Tabela 5 também apresenta os tempos para realizar as conexões a partir das descobertas realizadas. Essas conexões são efetuadas no módulo *OWLS2Graph*. A complexidade para efetuar as conexões é de $O(N_{Out})$ e $O(N_{In})$, ou seja $O(N)$. Cada uma das duas descobertas retorna uma lista de serviços que podem se conectar ao serviço que está sendo publicado ou ser conectado por ele. Os tempos de conexões ($Out \rightarrow In$ e $In \rightarrow Out$) é o tempo demandado para se percorrer cada uma das duas listas de serviços e criar as arestas (com o custo inicial do serviço – C_{is}) entre os serviços que se conectam.

A Tabela 5 informa que o tempo de publicação de um serviço é menor que *7752ms*, que não influencia no tempo de descoberta e composição do serviço. A decisão de montar o grafo no momento da publicação, inserindo um serviço por vez ao grafo, reduz consideravelmente o tempo da composição. Se o grafo fosse criado no momento da composição, o tempo de criação do grafo seria, no mínimo, a quantidade de serviços no repositório multiplicada por duas vezes o tempo máximo da descoberta ($|S| * 2 * 1126$), sendo $|S|$ a quantidade de serviços. Considerando a amostra de mil serviços utilizada, o tempo de *38 minutos* inviabilizaria a composição automática.

Tabela 6: Tempo de resposta para a composição de serviço. Onde: N_s = número de serviços no repositório; N_{OutR} = número de saídas requeridas. N_{InR} = número de entradas requeridas. N_{adj} = número de serviços ajacentes.

Função	Complexidade	Tempo (ms)
Descoberta OutR	–	1126
Descoberta InR	–	1126
Conectar *Node0*	$O(N_s)$	< 10
Conectar saídas requeridas	$O(N_s * N_{OutR})$	< 10
Cálculo descontos das entradas	$O(N_s * N_{InR})$	< 10
Cálculo descontos das saídas	$O(N_s * N_{OutR})$	< 10
Gerar grafo transposto	$O(N_s * N_{adj})$	< 60
Aplicar descontos	$O(N_s * N_{adj})$	< 60
Gerar grafo transposto (volta)	$O(N_s * N_{adj})$	< 60
Dijkstra por saída	$O(N^2)$	< 10
Tempo total	–	**< 2482**

5.3 Composição

O algoritmo de Prazeres et al. [14], utilizado neste trabalho para a realização da composição automática de serviços, executa cinco funções distintas: inserção do nó *Node0* – nó artificial de onde parte a execução do algoritmo de Dijkstra – no grafo; inserção dos nós das saídas requeridas no grafo; cálculo dos descontos; aplicação dos descontos ao grafo; e a execução do algoritmo de Dijkstra para cada saída requerida. Essas funções são executadas em dez procedimentos listados na Tabela 6[9].

Assim, o desempenho da composição automática depende desses dez procedimentos (Tabela 6): descoberta dos serviços que possuem entradas disponíveis; descoberta dos serviços que possuem saídas requeridas; conectar o nó virtual *Node0* a todos os serviços descobertos que tenham ao menos uma entrada disponível; conectar todos os serviços que possuem uma saída requerida ao nó virtual correspondente à referida saída; calcular os descontos para os serviços que possuem entradas disponíveis; calcular os descontos para os serviços que possuem saídas requeridas; gerar o grafo transposto para a aplicação dos descontos; aplicar os descontos; gerar o grafo transposto para retornar ao grafo original; aplicar o algoritmo de Dijkstra para cada saída requerida.

Os procedimentos para inserção dos nós virtuais e para a aplicação dos descontos demandam a descoberta de todos os serviços que possuem entradas disponíveis e de todos os serviços que possuem saídas requeridas. Dessa forma, é necessária a execução de duas descobertas (uma para as entradas e outra para as saídas) como apresentado na Tabela 6.

A Tabela 6 ilustra que os procedimentos de conexão dos nós virtuais *Node0* e saídas requeridas demandam complexidade de $O(N_s)$ e $O(N_s * N_{OutR})$ respectivamente. Isto porque para a inserção do nó *Node0* percorre-se a lista dos serviços que possuem entradas disponíveis, encontrados com a descoberta, e insere-se as arestas do nó *Node0* para cada serviço. Ainda para a inserção dos nós virtuais referentes às saídas requeridas é preciso percorrer a lista dos serviços que possuem saídas requeridas, encontrados com a descoberta, verificar qual das saídas é uma saída requerida, e inserir uma aresta do serviço para o nó virtual correspondente.

A função cálculo dos descontos também utiliza as listas dos serviços com entradas disponíveis e saídas requeridas. As complexidades de $O(N_s * N_{InR})$ e $O(N_s * N_{OutR})$ ilustradas na Tabela 6 ocorrem porque é preciso percorrer a lista dos serviços, e para cada serviço verificar quais entradas são disponíveis e quais saídas são requeridas e, com esses dados, efetuar o cálculo dos descontos.

A aplicação dos descontos ao grafo é realizada por meio do grafo transposto que é utilizado devido ao fato de que, na decisão da atribuição dos custos ao grafo os custos são aplicados para cada serviço, ou seja, toda aresta que aponta para um dado serviço possui o custo daquele serviço. Dessa forma, com o grafo transposto pode-se modificar todos os custos de um determinado serviço, pois é possível obter a lista de todos as arestas que apontam para o serviço. A Tabela 6 indica que a criação do grafo transposto é efetuada antes e depois da aplicação dos descontos. Isso ocorre porque, após a aplicação dos descontos, o grafo deve voltar à sua configuração original para o cálculo dos caminhos de custo mínimo. A criação do grafo transposto possui complexidade de $O(N_s * N_{adj})$ assim como na aplicação dos descontos, pois ambos precisam percorrer todos os serviços das listas de serviços descobertos e para cada serviço percorrer sua lista de nós adjacentes efetuando a alteração dos custos.

A Tabela 6 ainda apresenta o tempo de resposta do algoritmo de Dijkstra para encontrar o caminho de custo mínimo para uma das saídas requeridas e a sua complexidade que é de $O(N^2)$ no pior caso. É importante notar que o algoritmo é executado uma vez para cada saída requerida. Entretanto, isto não é problema pois o tempo de execução do referido algoritmo nos testes realizados foi menor que *10ms* para cada saída requerida. Além disso, como os caminhos para as saídas requeridas são, nesse momento, independentes entre si, a implementação pode ser otimizada para que a descoberta dos caminhos seja feita de forma paralela.

Por fim, a Tabela 6 apresenta o tempo total da composição automática de serviços que é menor que *2482ms*. Esse tempo é referente a uma saída requerida e para cada saída adicional basta acrescentar o tempo referente à execução do algoritmo de Dijkstra. Esse tempo de resposta é resultante do fato de que a composição está associada a algoritmos de ordem polinomial.

6. TRABALHOS RELACIONADOS

A literatura reporta vários trabalhos que investigam a descoberta e composição de Serviços Web Semânticos. Vários autores [4, 6, 8, 9, 18] apresentam propostas para descoberta e/ou composição de Serviços Web baseadas em QoS (*Quality of Service*) descrito semanticamente. Dessa forma, os autores adicionam requisitos não funcionais na descoberta e composição dos serviços. A implementação da descoberta e composição na iSWS utiliza apenas requisitos funcionais. Entretanto, a arquitetura da iSWS permite a inclusão de novas políticas de custo que podem utilizar custos não funcionais, tal como o QoS.

Brogi et al. [2] apresentam um algoritmo para seleção e composição de serviços, tal que a flexibilidade na composição e o fato do processo poder ser realimentado pelo usuário são suas principais características. Os autores, tal como proposto no algoritmo de composição utilizado na iSWS, priorizam encontrar uma solução que atenda completamente a requisição do usuário e, se essa solução não existir, o algoritmo é flexível para retornar composições parciais.

Gekas e Faslil [5] apresentam heurísticas para reduzir a

[9]A complexidade $O(N^2)$ para o algoritmo de Dijkstra é para o pior caso. As complexidades $O(N_s * N_{OutR})$, $O(N_s * N_{InR})$ e $O(N_s * N_{adj})$ serão $O(N^2)$ no pior caso.

quantidade de serviços no grafo em que o algoritmo de composição vai ser executado. Os autores justificam suas heurísticas afirmando que o processo de criar o grafo em tempo de composição limita as abordagens baseadas em grafo. Como a infraestrutura iSWS apresenta uma abordagem para a criação do grafo no momento da publicação, a criação do grafo não é um limitador da abordagem.

Silva et al. [16] empregam grafos para realizar a composição automática de Serviços Web, e utilizam similaridade semântica para efetuar a conexão do grafo de serviços, assim como na iSWS. Os autores não aplicam custos ao grafo: o algoritmo implementa um caminho inverso no grafo, partindo das saídas em direção às entradas. Como resultado, não existe seleção do melhor serviço para a composição, e o algoritmo pode gerar diversas composições. O algoritmo testa todas as possibilidades e, se não houver caminho para chegar às entradas, nenhuma composição é gerada.

A proposta de composição utilizada na iSWS [14] procura garantir todas as saídas na composição final (caso seja possível): isso explica a opção por realizar um caminho para cada saída, partindo sempre do nó único que representa todas as entradas. Nos casos em que não existe uma composição com todas as entradas e saídas requeridas, o algoritmo encontra uma solução que pode agregar o máximo dos parâmetros requeridos pelo usuário.

A infraestrutura apresentada neste artigo tem o objetivo de agrupar funcionalidades de publicação, descoberta e composição de Serviços Web Semânticos. Para isso, os autores fazem reúso de tecnologias, padrões e da infraestrtura da Web atual e adicionam semântica, por meio da ontologia OWL-S, com finalidade de automatizar as tarefas de descoberta e composição. Além disso, a iSWS é extensível o suficiente para incorporar novos algoritmos de descoberta e composição e novas políticas de custo (funcionais ou não funcionais). Assim, os trabalhos aqui relacionados, em sua maioria, podem ser incorporados na iSWS de forma a oferecer mais funcionalidades.

7. CONSIDERAÇÕES FINAIS

A arquitetura da iSWS e todos os seus módulos implementados são descritos com foco em cada uma das três funcionalidades principais e na forma em que essas funcionalidades interagem entre si. Dessa forma, o artigo apresenta resultados em publicação, descoberta e composição de Serviços Web Semânticos. Entretanto, o foco principal do artigo é demonstrar a viabilidade de uma infraestrutura baseada em padrões e tecnologias da Web Semântica, e que funciona utilizando a infraestrutura da Web Atual.

Para demosntrar tal viabilidade foram realizados alguns experimentos em publicação, descoberta e composição de Serviços Web Semânticos. Os resultados desses experimentos são apresentados no artigo por meio de testes de escalabilidade versus tempo de execução. Ou seja, verificou-se o comportamento das três funcionalidades implementadas à medida que o número de serviços registrados no repositório e no grafo de composição aumentava. Ainda com propósito de validação, o artigo apresenta a complexidade dos algoritmos utilizados nos módulos da infraestrutura iSWS. A análise da complexidade dos algoritmos mostra que não existem comportamentos exponenciais, visto que o pior caso, que ocorre na aplicação do algoritmo de Dijkstra no algoritmo de composição utilizado, possui complexidade quadrática ($O(N^2)$). Como consequência disso, nenhuma das três funcionalidades implementadas apresentou comportamento exponencial em relação ao aumento do tempo de execução face ao aumento da quantidade de serviços no repositório.

Pesquisas que podem ser realizadas a partir do trabalho reportado incluem trabalhos: i) no intuito de inserir novas políticas de custos para a composição de serviços (i.e. custos não funcionais baseados em QoS); ii) para avaliação de outros algoritmos para composição e descoberta existentes na literatura; iii) na direção de tornar a iSWS mais genérica para suportar, por exemplo, algoritmos de composição que não sejam baseados na utilização de grafos.

Acknowledgment: We thank CAPES, CNPq and FAPESP.

8. REFERÊNCIAS

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):35–43, 2001.

[2] A. Brogi, S. Corfini, and R. Popescu. Semantics-based composition-oriented discovery of Web Services. *ACM Trans. Internet Technol.*, 8:19:1–19:39, 2008.

[3] D. Fensel, J. A. Hendler, H. Lieberman, and W. Wahlster, editors. *Spinning the Semantic Web: bringing the World Wide Web to Its Full Potential*. MIT Press, 2003.

[4] D. Z. Garcia and M. B. F. Toledo. Semantics-enriched QoS policies for Web Service interactions. In *WebMedia '06*, pages 35–44, 2006.

[5] J. Gekas and M. Faslil. Employing Graph Network Analysis for Web Service Composition. *Int. J. Inf. Techn. and Web Engineering*, 2(4):21–40, 2007.

[6] S. Jean, F. Losavioy, A. Matteoy, and N. Levyz. An extension of OWL-S with quality standards. In *RCIS'10*, pages 483–494, 2010.

[7] G. Krasner and S. Pope. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *J. Obj.Or. Program.*, 1(3):26–49, 1988.

[8] S. Li and J. Zhou. The WSMO-QoS Semantic Web Service Discovery Framework. In *CiSE'09*, pages 1–5, 2009.

[9] Q. Li-li and C. Yan. QoS ontology based efficient Web Services selection. In *ICMSE 2009*, pages 45–50, 2009.

[10] D. Martin, J. Domingue, M. L. Brodie, and F. Leymann. Semantic Web Services, Part 1. *IEEE Intelligent Systems*, 22(5):12–17, 2007.

[11] D. Martin, J. Domingue, A. Sheth, S. Battle, K. Sycara, and D. Fensel. Semantic Web Services, Part 2. *IEEE Intelligent Systems*, 22(6):8–15, 2007.

[12] T. Payne and O. Lassila. Guest Editors' Introduction: Semantic Web Services. *IEEE Intelligent Systems*, 19(4):14–15, 2004.

[13] C. V. S. Prazeres, M. G. C. Pimentel, and C. A. C. Teixeira. Remote Experiments as Semantic Web Services. In *IEEE ICSC´07*, pages 791–798, 2007.

[14] C. V. S. Prazeres, C. A. C. Teixeira, and M. d. G. C. Pimentel. Semantic Web Services Discovery and Composition: Paths Along Workflows. In *IEEE ECOWS'09*, pages 58–65, 2009.

[15] C. V. S. Prazeres, C. A. C. Teixeira, and M. G. C. Pimentel. Semantic Web Services Discovery by Matching Temporal Restrictions. In *SAINT'08*, pages 26–32, 2008.

[16] E. M. G. Silva, L. F. Pires, and M. J. Sinderen. An Algorithm for Automatic Service Composition. In *ICSOFT'07*, pages 65–74, 2007.

[17] N. Srinivasan, M. Paolucci, and K. Sycara. Semantic Web Service Discovery in the OWL-S IDE. In *HICSS'06*, page 109.2, 2006.

[18] Z. Ying-Hui, F. Feng-Rui, and Y. Chao. Semantic Web Service selection based on QoS Ontology. In *ICINA´10*, volume 2, pages V2–492 –V2–496, 2010.