Serviços Web Semânticos: da modelagem à composição

Cássio Vinícius Serafim Prazeres

Serviços Web Semânticos: da modelagem à composição

Cássio Vinícius Serafim Prazeres

Orientadora: *Profa. Dra. Maria da Graça Campos Pimentel*

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional.

USP – São Carlos
Junho/2009

"...meu caminho pelo mundo
eu mesmo traço
a Bahia já me deu
régua e compasso..."
(Aquele Abraço, Gilberto Gil, 1969)

Dedicatória

Aos meus pais Deraldo e Lígia, por terem me dado a base para tudo;
à minha irmã Sheila, por sempre me incentivar;
aos meus sobrinhos João e Letícia, alegrias que Deus colocou na minha vida;
à minha orientadora Graça, por me guiar a chegar neste momento;
e aos meus amigos, pelos momentos de muitas resenhas e descontração.

Agradecimentos

Inicialmente agradeço a Deus por me conceder saúde, força e coragem para buscar mais este objetivo na minha vida.

Em seguida, agradeço aos meus pais e à minha irmã que sempre me apoiaram e me incentivaram em tudo aquilo que eu me propus a fazer durante minha vida. A eles eu devo toda a gratidão do mundo.

Ao meu tio e padrinho Luiz Prazeres, que contribuiu muito para minha formação, sem ele eu não teria chegado até aqui.

Agradeço a toda a minha família – meus avós, tios, tias, primos e primas – que fizeram e fazem parte da minha trajetória de vida.

À professora e orientadora Graça Pimentel, por ter acreditado no meu potencial e pelo apoio e a boa vontade de sempre. Por ter me guiado na realização desse trabalho e mostrado sempre o rumo certo a seguir durante as pesquisas.

Ao Cesar Teixeira, professor da Universidade Federal de São Carlos, amigo e o principal responsável pela minha vinda para São Carlos. Agradeço também por muitas e sempre relevantes contribuições dadas a este trabalho.

Agradeço também a todos os funcionários da Universidade de São Paulo, que sempre se mostraram disponíveis nos momentos em que precisei deles. Em especial às meninas da secretaria do programa de pós-graduação em Ciências de Computação e Matemática Computacional, Beth, Laura, Ana Paula e Lívia, que sempre nos atendem de forma cativante.

A todos meus amigos de longa data dos quais, apesar da distância física, sempre farão parte da minha vida: Percon, Marquinho, Paulino, Gilberto, Renato e meus amigos e primos Beto e Osvaldo.

Aos meus novos e muitos amigos cultivados ao longo dessa jornada em São Carlos, companheiros de república, amigos da Universidade e laboratório, amigos são-carlenses, com os quais dividíamos os momentos de saudade da família e os momentos de muitas e muitas resenhas e descontração: Leonardo (Léo Cabeça), Maycon (no diminutivo), Ricardo (Rios), Gabriel (Ceará), Fernandinho (Beira-mar), Linder, Renatinho, Vinícius (desde os tempos da UNIFACS), Darley (Biriguey), Daniel (Motoqueiro), André, Thalitinha (Conti), Paulo (Madá), Pâmela, Laisa (Turing), Renato (Bulcão), Taciana (Bulcão), Felipe (Filipim), Matheus (Tortim), Dráusio, Flávio (Sorin), Tácito (Xunin), Bruno, Thiago, Rigolin, Daniel (Lobato), Renan, Carlos (Patrão) e Helder.

Por fim, gostaria de agradecer à FAPESP e à CAPES pelo apoio financeiro concedido às minhas pesquisas.

Resumo

A automação de tarefas como descoberta, composição e invocação de Serviços Web é um requisito importante para o sucesso da Web Semântica. Nos casos de insucesso na busca por um serviço, por não existir disponível um serviço completo que atenda plenamente a requisição do usuário, uma possibilidade de contorno é compor o serviço procurado a partir de elementos básicos que atendam parcialmente a requisição inicial e que se completem. A composição de Serviços Web pode ser realizada de forma manual ou de forma automática. Na composição manual, o desenvolvedor de Serviços Web pode tirar proveito da sua expertise sobre os serviços envolvidos na composição e sobre o resultado que se deseja alcançar. Esta tese aborda problemas e apresenta contribuições relacionadas ao processo de composição automática de Serviços Web. A composição automática de Serviços Web requer que os serviços sejam descritos e publicados de forma a modelar o conhecimento (semântica explícita) que o desenvolvedor utiliza para realizar a composição manual. A descoberta automática baseada nas descrições semânticas do serviço é também um passo crucial na direção da composição automática, pois é um estágio anterior necessário para a seleção dos serviços candidatos à composição. Trabalhos da área de pesquisa em Serviços Web Semânticos exploram a utilização dos padrões da Web Semântica para enriquecer, com semântica explícita, a descrição dos Serviços Web. O problema da composição automática de Serviços Web é tratado neste trabalho por meio de três linhas de investigação: modelagem dos Serviços Web Semânticos; descoberta automática de Serviços Web Semânticos; e composição automática de Serviços Web Semânticos. As contribuições desta tese incluem: a plataforma RALOWS para modelagem de aplicações Web como Serviços Web Semânticos, tendo como estudo de caso aplicações para realização de experimentos remotos; um algoritmo para descoberta automática de Serviços Web Semânticos; uma proposta baseada em grafos e caminhos de custo mínimo para prover composição automática de Serviços Web Semânticos; uma infra-estrutura e ferramentas de apoio à descrição, publicação, descoberta e composição de Serviços Web Semânticos.

Abstract

The automation of the discovery, composition and invocation of Web Services is an important step to the success of the Semantic Web. If no single Web Service satisfies the functionality required by one user, an alternative is to combine existing services that solve parts of the problem in order to reach a complete solution. Web Services composition can be achieved manually or automatically. When composing services manually, Web Service developers can take advantage of their expertise and knowledge about the composition services and the target service. This thesis addresses issues and presents contributions related to the process of automating Web Services composition. The automatic composition of Web services requires the description and publication of the services in order to model the necessary knowledge (explicit semantics) that the developer uses to perform the manual composition. The automatic Web Service discovery is a crucial step toward the automatic composition, because it is a previous stage necessary to the selection of composition service candidates. Semantic Web Services researches explore the use of the Semantic Web technologies to enrich the Web Services descriptions with explicit semantics. Three main lines of investigation are adopted in this thesis to explore the process of automatic composition of Web Services. They are the following: Semantic Web Services modeling; automatic discovery of Semantic Web Services; and automatic composition of Semantic Web Services. The main contributions of this thesis include: the RALOWS platform for modeling Web applications as Semantic Web Services; an algorithm for the automatic discovery of Semantic Web Services; a graph-based approach to the automatic composition of Semantic Web Services; and an infrastructure and tools to support the Semantic Web Services description, publishing, discovery and composition.

Publicações

A seguir é apresentada a lista de publicações originadas a partir deste trabalho:

Artigo publicado em periódico internacional (com *referee*)

- **Prazeres, C. V. S.**, Teixeira, C. A. C., Munson, E. V., and Pimentel, M. G. C. (2009b). Toward Semantic Web Services as MVC Applications: from OWL-S via UML. Journal of Web Engineering, 8(3):1–22. Rinton Press.

Capítulo de Livro

- Bulcão Neto, R. F., **Prazeres, C. V. S.**, e Pimentel, M. G. C. (2006). Web Semântica: Teoria e Prática. Cap. 2, pp. 47–86. Tópicos em Sistemas Interativos e Colaborativos. SBC Press. Texto referente a mini-curso ministrado no 12°Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia'06), Natal-RN, Brasil.

Artigos completos publicados em Conferência Internacional (com *referee*)

- **Prazeres, C. V. S.**, Pimentel, M. G. C., and Teixeira, C. A. C. (2007). Remote Experiments as Semantic Web Services. In *Proceedings of the IEEE Conference on Semantic Computing (ICSC'07)*, pp. 791–798, Irvine, CA, USA. IEEE CS Press. `http://dx.doi.org/10.1109/ICOSC.2007.4338424`.

- **Prazeres, C. V. S.**, Teixeira, C. A. C., and Pimentel, M. G. C. (2008). Semantic Web Services Discovery by Matching Temporal Restrictions. In *Proceedings of the 8th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT'08)*, pp. 26–32, Turku, Finland. IEEE CS Press. `http://dx.doi.org/10.1109/SAINT.2008.50`.

- **Prazeres, C. V. S.**, Teixeira, C. A. C., Munson, E. V., and Pimentel, M. G. C. (2009a). Semantic Web Services: from OWL-S via UML to MVC Applications. In *Proceedings of the 24th ACM symposium on Applied computing (ACM SAC'09)*, Vol. I, pp. 675–680, Honolulu, Hawaii, USA. ACM Press. `http://dx.doi.org/10.1145/1529282.1529421`.

- **Prazeres, C. V. S.**, Teixeira, C. A. C., and Pimentel, M. G. C. (2009d). Semantic Web Services discovery and composition: paths along workflows (to appear). In *Submitted to the 7th IEEE European Conference on Web Services (ECOWS'09)*, pp. 1–10, Eindhoven, Netherlands. IEEE CS Press.

Artigos completos publicados em Conferência Nacional (com *referee*)

- Teixeira, C. A. C., Capobianco, D., **Prazeres, C. V. S.**, Santos, F. S., e Barbosa, M. (2005). Processo de Modelagem de Resposta: Refinando Requisitos de Software de Apoio a Laboratórios de Acesso Remoto. In *Anais do Simpósio Brasileiro de Informática na Educação (SBIE'05)*, pp. 518–528, Juiz de Fora, MG, Brasil.

- **Prazeres, C. V. S.** and Teixeira, C. A. C. (2006). A structured document-based approach for Weblab configuration. In *Proceedings of the 12th Brazilian symposium on Multimedia and the web (WebMedia'06)*, pp. 1–10, Natal, RN, Brazil. ACM Press. http://dx.doi.org/10.1145/1186595.1186597.

- Escobedo, E. P., **Prazeres, C. V. S.**, Teixeira, C. A. C., Pimentel, M. G. C., and Kofuji, S. (2007). SeCoAS: An Approach to Develop Semantic and Context-Aware Available Services. In *Proceedings of the 13th Brazilian symposium on Multimedia and the web (WebMedia'07)*, pp. 1–8, Gramado, RS, Brazil. ACM Press.

Artigos completos publicados em Workshop Local (sem *referee*)

- **Prazeres, C. V. S.**, Santos, F. S., Barbosa, M., Capobianco, D., and Teixeira, C. A. C. (2005). Integrating Tools for Accessing Resources Remotely into an e-Learning Environment. In *II Workshop projeto TIDIA FAPESP.*, pp. 1–10, São Paulo, SP, Brazil.

Resumos publicados em Workshop Local (sem *referee*)

- Capobianco, D., Barbosa, M. Q., Santos, F. S., **Prazeres, C. V. S.**, and Teixeira, C. A. C. (2005). The Learning Action Modeling of Responses in Rats Executed in the TIDIA-Ae Environment. In *II Workshop projeto TIDIA FAPESP.*, 2p, São Paulo, SP, Brazil.

- **Prazeres, C. V. S.**, Pimentel, M. G. C., and Teixeira, C. A. C. (2006b). A document-based approach for WebLab configuration. In *III Workshop projeto TIDIA FAPESP.*, pp. 112–114, São Paulo, SP, Brazil.

A seguir é apresentada a lista de publicações indiretamente relacionadas ao contexto deste trabalho:

Artigos completos publicados em Conferência Nacional (com *referee*)

- Pimentel, M. G. C., **Prazeres, C. V. S.**, Ribas, H., Lobato, D., and Teixeira, C. A. C. (2005). Documenting the pen-based interaction. In *Proceedings of the 11th Brazilian symposium on Multimedia and the web (WebMedia'05)*, pp. 1–8, Poços de Caldas, MG, Brazil. ACM Press. `http://dx.doi.org/10.1145/1114223.1114232`.

- **Prazeres, C. V. S.**, Lucredio, D., Fortes, R. P. M., e Teixeira, C. A. C. (2006a). Uma Proposta de Navegação na Web Utilizando Facetas. In *Anais da Escola Regional de Banco de Dados*, pp. 1–6, Passo Fundo, RS, Brazil.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

DAML: *DARPA Agent Markup Language*
DAML-S: *DARPA Agent Markup Language for Services*
DARPA: *Defense Advanced Research Projects Agency*
DL: *Description Logics*
DOI: *Digital Object Identifier*
DTD: *Data Type Definition*
HTML: *Hypertext Markup Language*
HTTP: *Hypertext Transfer Protocol*
IOPR: *Inputs, Outputs, Preconditions and Results*
MVC: *Model View Controller*
OASIS: *Organization for the Advancement of Structured Information Standards*
OIL: *Ontology Inference Layer*
OWL: *Web Ontology Language*
OWL-S: *Web Ontology Language for Services*
RALOWS: *Remote Access Laboratory Ontology and Web Service*
RDFS: *RDF Schema. RDF Vocabulary Description Language*
RuleML: *Rule Markup Language*
SOA: *Service Oriented Architecture*
SOAP: *Simple Object Access Protocol*
SWRL: *Semantic Web Rule Language*
TCP/IP: *Transmission Control Protocol / Internet Protocol*
TIDIA: *Tecnologia da Informação para o Desenvolvimento da Internet Avançada*
UDDI *Universal Description, Discovery, and Integration*
URI: *Universal Resource Identifier*
URL: *Universal Resource Locator*
XML: *Extensible Markup Language*
XSD: *XML Schema Definition*
W3C: *World Wide Web Consortium*
WSDL: *Web Services Description Language*
WWW: *World Wide Web*

CAPÍTULO

1

Introdução

A Web atual[1] disponibiliza conteúdo planejado por pessoas e para pessoas que buscam e utilizam informações, navegam por catálogos de lojas, compram produtos, fazem contatos com outras pessoas, realizam atividades financeiras, dentre outras atividades que geralmente não são passíveis de serem realizadas por ferramentas de software (e.g. Agentes de Software Inteligentes) [Lacy, 2005].

Tim Berners-Lee, pesquisador e criador da *World Wide Web*, idealizou sua evolução com a estruturação e a adição de semântica ao conteúdo da Web atual. Tal evolução não passa pela criação de uma nova Web e sim pela extensão da atual, provendo significado bem definido às informações e permitindo que pessoas e computadores possam trabalhar em cooperação. A esse novo paradigma e conceito de Web em que o conteúdo possui informação estruturada e com semântica associada, cunhou-se o nome de *Web Semântica* [Berners-Lee et al., 2001].

Em seu principal artigo sobre a Web Semântica e a revolução de novas possibilidades que ela deverá promover, Berners-Lee et al. [2001] previram que, em um futuro próximo, a Web Semântica possibilitará o desenvolvimento de novas funcionalidades em que as máquinas se tornarão muito mais capazes de processar e compreender as informações que no momento são passíveis apenas de serem exibidas. Para que essa revolução seja realizada, Berners-Lee et al. [2001] apontam pesquisas e desenvolvimentos em várias áreas e tecnologias que envolvem a estruturação de conteúdo, a representação de conhecimento e a computação autônoma.

Com o conteúdo estruturado e o conhecimento representado por meio de se-

[1] Neste texto, "Web atual" refere-se à Web como amplamente utilizada até 2009.

mântica clara e explícita, documentos e aplicações da Web podem ser utilizados não apenas por pessoas mas também por outras aplicações. Essas aplicações possuem propriedades e características autônomas à medida que passam a processar o conteúdo estruturado e a inferir a partir do conhecimento explicitado.

Um Serviço Web[2] é um tipo de aplicação desenvolvida para ser utilizada por outras aplicações, diferentemente da maioria das aplicações da Web atual, que são produzidas para serem utilizadas por pessoas [Stal, 2002]. O propósito principal de Serviços Web é o de promover a interoperabilidade entre provedores e consumidores de serviços por meio de descrições estruturadas da sua sintaxe de utilização.

As pesquisas em Serviços Web e Web Semântica convergem para a adição de semântica clara e explícita à descrição dos Serviços Web [McIlraith et al., 2001]. Pesquisas recentes têm o propósito de aplicar as tecnologias da Web Semântica na descrição e uso de Serviços Web, resultando nos chamados Serviços Web Semânticos. Enquanto a motivação principal dos "Serviços Web" é promover interoperabilidade baseada na sintaxe dos serviços, a principal motivação dos "Serviços Web Semânticos" é permitir automação do uso da informação e a interoperabilidade dinâmica e automática com base na semântica dos serviços [Martin et al., 2007a].

Serviços Web Semânticos demandam o desenvolvimento de padrões e tecnologias para possibilitar a automação de tarefas que normalmente são realizadas por pessoas (usuários finais ou desenvolvedores de Serviços Web). Tarefas como descoberta, composição e invocação podem ser automatizadas tirando-se proveito da descrição semântica dos Serviços Web.

1.1 Motivação

O desenvolvimento deste trabalho iniciou-se a partir de duas motivações principais: uma motivação inicial referente a um problema específico e uma motivação genérica. A motivação inicial foi a demanda por um conjunto de aplicações com características peculiares denominadas de experimentos remotos. O problema específico era o de encontrar soluções para integrar experimentos remotos em ambientes Web de aprendizado eletrônico. A motivação genérica surgiu quando o autor investigou a utilização de Serviços Web como solução para a integração dos experimentos remotos.

Os experimentos remotos mencionados acima são aplicações, dos mais variados tipos, que estão distribuídas em diversos pontos da Internet com implementações e arquiteturas distintas, e que geralmente necessitam de um agendamento prévio para sua utilização. Devido a essas características, era necessário uma forma padrão de integrar experimentos remotos a ambientes Web de um modo geral e a ambientes Web de aprendizado eletrônico de um modo específico. Após avaliar outras soluções

[2]Neste texto, "Serviço Web" são sempre grafados com as iniciais maiúsculas.

([Prazeres et al., 2005; 2006b]), o autor verificou que, dada sua característica principal de promover interoperabilidade entre aplicações, Serviços Web seriam o ponto de partida ideal para a solução do problema de integração de experimentos remotos.

Algumas soluções ([Yan et al., 2005; 2006a;b; Harward et al., 2008]) para disponibilização de experimentos remotos como Serviços Web existentes na literatura foram avaliadas. Essas soluções têm como objetivos descrever, disponibilizar e prover infra-estruturas para os experimentos remotos na forma de Serviços Web. Ao investigar essas soluções, o autor verificou que a falta de semântica nos descritores dos Serviços Web demandava que a integração fosse definida em tempo de programação. Nesse ponto das pesquisas, foi identificada a motivação genérica: investigar a utilização dos padrões e tecnologias da Web Semântica para prover interoperabilidade automática entre Serviços Web. Dessa forma, o trabalho reportado nesta tese identificou a necessidade e a oportunidade de se modelar aplicações na Web, tal como os experimentos remotos, na forma de Serviços Web Semânticos, e investigar abordagens para prover descoberta e composição automáticas de Serviços Web Semânticos.

Serviços Web Semânticos foram investigados em termos de descoberta e composição de serviços. Martin et al. [2007a] observam que a questão central a ser respondida para permitir o desenvolvimento de Serviços Web Semânticos é se realmente é possível utilizar técnicas da Web Semântica para automatizar o relacionamento entre Serviços Web. Ainda segundo Martin et al. [2007a], na busca por resposta à essa questão, as pesquisas têm sido direcionadas para o desenvolvimento de linguagens, ontologias, algoritmos, arquiteturas e infra-estruturas que têm como propósito automatizar tarefas como descoberta, composição e invocação de Serviços Web.

Com relação à descoberta automática de Serviços Web Semânticos, algumas infra-estruturas e algoritmos baseados nas descrições semânticas das funcionalidades do serviço são reportados na literatura (e.g. [Paolucci et al., 2002; Srinivasan et al., 2006; Klusch et al., 2006; Li and Horrocks, 2003; Yao et al., 2006; Jaeger et al., 2005]). Os algoritmos reportados, na sua maioria, fazem a comparação semântica entre as entradas e saídas dos serviços providos e as entradas e saídas requeridas pelo usuário do serviço.

Entretanto, a descoberta de serviços muitas vezes pode não retornar resultados que atendam à requisição inicial do usuário. Nesses casos, surge a oportunidade e o desafio de encontrar serviços que em conjunto realizem a tarefa esperada pelo usuário. A composição automática de Serviços Web é um problema que, na literatura, é tratado com propostas que envolvem síntese de programas, planejamento em inteligência artificial e *workflow* [Martin et al., 2007a].

1.2 Objetivos

O objetivo geral do presente trabalho consiste em investigar uma solução para promover descoberta e composição automática de Serviços Web Semânticos. Para atingir esse objetivo, os serviços devem ser modelados e publicados com as descrições semânticas na forma de Serviços Web Semânticos. Nesse sentido, o objetivo geral pode ser dividido em quatro objetivos específicos:

- modelar e descrever aplicações Web, como por exemplo experimentos remotos, na forma de Serviços Web Semânticos;
- modelar e permitir a descoberta automática de Serviços Web Semânticos;
- promover composição automática de Serviços Web Semânticos nos casos em que a descoberta não retorne resultados satisfatórios;
- modelar, implementar e avaliar uma infra-estrutura que permita a publicação de serviços descritos como Serviços Web Semânticos de forma a possibilitar a descoberta e a composição automáticas dos serviços.

1.3 Contribuições

As contribuições resultantes do trabalho reportado nesta tese são:

- Proposta inicial para modelagem e descrição sintática de experimentos remotos com o objetivo de integração a ambientes Web de aprendizado eletrônico [Prazeres et al., 2005; 2006b; Prazeres and Teixeira, 2006].
- Modelagem e descrição de aplicações Web na forma de Serviços Web Semânticos. Experimentos remotos foram as aplicações inicialmente modeladas e descritas como um Serviço Web Semântico [Prazeres et al., 2007]. Posteriormente, a mesma abordagem foi aplicada em ambientes inteligentes para prover descoberta automática dos serviços disponíveis [Escobedo et al., 2007].
- Descoberta automática de Serviços Web Semânticos baseada nas funcionalidades providas pelo serviço, na categoria do serviço, nas pré-condições necessárias à execução do serviço e na disponibilidade do serviço ou necessidade de agendamento da sua execução [Prazeres et al., 2008].
- Modelo em grafo para composição automática de Serviços Web Semânticos. Políticas para atribuição de custos ao grafo considerando os serviços candidatos a participarem de uma composição. Aplicação de algoritmo para cálculo do caminho de custo mínimo em grafo para composição automática de serviços.

Geração automática do *workflow* da composição final a partir dos caminhos de custo mínimo. Mapeamento do *workflow* da composição final para construtores definidos em uma linguagem de ontologia para modelar *workflow* de processos [Prazeres et al., 2009c;d].

- Infra-estrutura e ferramentas de apoio à descrição, publicação, descoberta e composição de Serviços Web Semânticos [Prazeres et al., 2009b;a;c;d].

1.4 Estrutura da tese

Os demais capítulos desta tese estão estruturados como segue. O Capítulo 2 trata das principais áreas em que este trabalho está inserido: Web Semântica e Serviços Web Semânticos. O capítulo apresenta os principais conceitos da Web Semântica, descreve a sua arquitetura e as ontologias utilizadas neste trabalho. Os conceitos e as tecnologias a respeito dos Serviços Web Semânticos são apresentados e também é detalhada a linguagem de marcação semântica para Serviços Web utilizada neste trabalho.

O Capítulo 3 apresenta a proposta de descrição de experimentos remotos segundo duas abordagens: sintática e semântica. A abordagem sintática é utilizada inicialmente para possibilitar a integração dos experimentos a ambientes Web de aprendizado eletrônico. A seguir, a plataforma RALOWS para descrição de experimentos remotos como Serviços Web Semânticos é apresentada no capítulo.

O Capítulo 4 descreve a proposta para descoberta automática de Serviços Web Semânticos, apresentando um algoritmo para descoberta automática baseada nas funcionalidades providas pelo serviço, na categoria do serviço, nas pré-condições necessárias à execução do serviço e na disponibilidade do serviço ou na necessidade de agendamento da sua execução. A forma como a descoberta é utilizada no processo de publicação e de composição automática de serviços também é apresentada no capítulo.

O Capítulo 5 apresenta a proposta de modelo em grafo utilizado para representar o problema de composição automática de Serviços Web Semânticos. Em seguida é apresentado as políticas de custos adotadas para os Serviços Web Semânticos e os resultados da aplicação de um algoritmo para cálculo do caminho de custo mínimo em composição de serviços. Por fim, é também apresentada a geração automática da composição final a partir dos caminhos de custo mínimo e o mapeamento dessa composição para descritores de fluxo de uma ontologia de processos para Serviços Web Semânticos.

O Capítulo 6 apresenta a proposta da infra-estrutura para implantação de Serviços Web Semânticos. As ferramentas implementadas para oferecer funcionalidades de

publicação, descoberta e composição de Serviços Web Semânticos são descritas no capítulo. São também apresentadas as tecnologias utilizadas para a implementação da referida infra-estrutura. Por fim, a avaliação da publicação, descoberta e composição são discutidas no capítulo.

O Capítulo 7 sumariza o trabalho realizado nesta tese ao apresentar os resultados e contribuições obtidas, ao discutir as limitações das abordagens descritas neste trabalho e ao apresentar as linhas de investigação e de trabalhos futuros visando a continuidade do referido trabalho.

CAPÍTULO 2

Web Semântica e Serviços Web Semânticos

A *World Wide Web*, ou simplesmente Web, tem evoluído rapidamente desde a sua criação em 1989. A Web, inicialmente desenvolvida nos laboratórios do CERN (*European Organization for Nuclear Research*), em Genebra, por Berners-Lee et al. [1992a;b; 1994] é hoje o maior e mais utilizado sistema hipermídia existente.

A maioria do conteúdo da Web atual foi projetada para a leitura por humanos, sem uma estrutura capaz de fornecer mecanismos que possibilitem que programas de computador possam manipular a semântica de todo esse conteúdo [Berners-Lee et al., 2001]. Os *browsers* são capazes de processar uma página Web e identificar instruções, em documentos HTML (*HyperText Markup Language*), sobre como exibir o conteúdo ao usuário. Porém, na maioria dos casos, não há programas capazes de processar a semântica do conteúdo exibido.

A simplicidade dos documentos HTML proporcionou a rápida expansão da Web atual, tal como defendido por Berners-Lee et al. [2001] e Fensel et al. [2003]. Porém, essa mesma simplicidade, segundo Lacy [2005], 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.

Segundo Berners-Lee et al. [2001], a Web Semântica tem a proposta de 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. Essa é a proposta, em termos gerais, da Web Semântica.

Dentre os mais importantes recursos na Web estão aqueles chamados de Serviços Web, que são sites da Web que não contêm apenas informações estáticas, mas permitem que alguém efetue alguma ação ou mudança no mundo. A Web Semântica permitirá que usuários possam localizar, selecionar, compor e invocar Serviços Web automaticamente [Ankolekar et al., 2001; Burstein et al., 2002].

Parte das pesquisas iniciais da Web Semântica tem focado na definição dos conceitos, tecnologias e padrões para possibilitar a proposta da Web Semântica. Em particular, linguagens com semântica explícita têm sido desenvolvidas para adicionar marcação semântica aos recursos da Web.

Entretanto, outro aspecto chave que também deve ser levado em consideração é como será tratado, na Web Semântica, a grande quantidade de aplicações que foram desenvolvidas durante anos na Web atual e que continuarão a serem desenvolvidas na Web Semântica. Uma solução efetiva consiste em realizar acesso programático a essas aplicações e visualizá-las como objetos de primeira classe que podem ser reutilizados e combinados. O conceito de Serviços Web Semânticos é uma tendência recente para materializar essa solução.

Este capítulo descreve duas das principais tendências de evolução da Web que são de interesse particular para o presente trabalho: a Web Semântica na Seção 2.1 e os Serviços Web Semânticos na Seção 2.1. A Seção 2.3 apresenta as considerações finais referentes a este capítulo.

2.1 Web Semântica

Uma das principais tendências de evolução da Web atual é a Web Semântica. A proposta da Web Semântica é a de desenvolver padrões de documentos e tecnologias para possibilitar a descrição estrutural e semântica de todo o conteúdo disponibilizado na Web. Com o intuito de apresentar os principais padrões e tecnologias associados à Web Semântica, esta seção descreve os conceitos gerais e uso de ontologias (Seção 2.1.1) para descrição de informação na Web Semântica, bem como as tecnologias e padrões utilizados e propostos para a estruturação do conteúdo na Web Semântica. Nesse sentido, é apresentada a arquitetura da Web Semântica (Seção 2.1.2) proposta por Berners-Lee [2000], os padrões e as especificações propostos para cada camada da arquitetura e exemplos de ontologias

(Seção 2.1.3), de nível superior e também de domínio, para a Web Semântica que são utilizadas neste trabalho.

2.1.1 Ontologia

Ontologias são especificações formais e explícitas de termos pertencentes a um determinado domínio e as relações entre esses termos. Gruber [1993] define ontologia como "uma especificação formal de uma conceitualização". Uma outra definição bastante utilizada pela comunidade de Inteligência Artificial é a definição de Fensel [2001] que estende a definição inicial de Gruber [1993]: "uma especificação formal e explícita de uma conceitualização compartilhada".

Na computação em geral o termo ontologia tem sido bastante utilizado, englobando um conjunto de definições de conceitos, propriedades, relações, restrições, axiomas, processos e eventos que descrevem certo domínio ou universo de discurso. Essas definições habilitam aplicações e agentes de software a utilizarem semântica formal, precisa e clara para processar a informação descrita pela ontologia e usar essa informação em aplicações inteligentes.

Gómez-Pérez et al. [2004] definem dois tipos de ontologia: ontologia de domínio ou vertical e ontologia de nível superior, independente de domínio, ou ontologia horizontal. Ontologias de domínio definem conceitos para um domínio específico, por exemplo, zoologia, medicina, matemática, vinhos e pizzas. Ontologias de nível superior são aquelas em que os conceitos independem do domínio, ou seja, podem ser utilizadas em diversos domínios. Alguns exemplos de ontologias de nível superior ou horizontal são ontologias para Serviços Web, ontologia de tempo, dentre outras.

A tarefa de se criar uma ontologia para um domínio específico não é uma atividade trivial. Dessa forma, antes de partir para a criação de ontologias deve-se ter em mente a real necessidade de se ter a ontologia. Noy and McGuinness [2001] elicitam algumas razões para se criar uma ontologia: compartilhar entendimento comum da estrutura da informação entre pessoas ou agentes de software; possibilitar reutilização de conhecimento de domínio; tornar explícito especificações sobre o domínio; separar conhecimento de domínio de conhecimento operacional e analisar conhecimento de domínio.

No presente trabalho são utilizadas tanto ontologias de domínio como de nível superior no escopo da Web Semântica e de Serviços Web. As ontologias de domínio da Web Semântica são utilizadas em conjunto com as ontologias de Serviço Web, de tempo e de recursos, combinadas para prover descrição de Serviços Web e para prover a descoberta e composição automáticas desses serviços.

2.1.2 Arquitetura da Web Semântica

A Web Semântica se apóia em um conjunto de especificações e padrões com o objetivo de prover uma infra-estrutura para a criação de aplicações inteligentes na Web. Esse conjunto de especificações e padrões estão distribuídos na arquitetura em camadas proposta para a Web Semântica conforme pode ser visto na Figura 2.1. O desenvolvimento dos padrões para a Web Semântica é controlado e avaliado pelo consórcio W3C (*World Wide Web Consortium*).

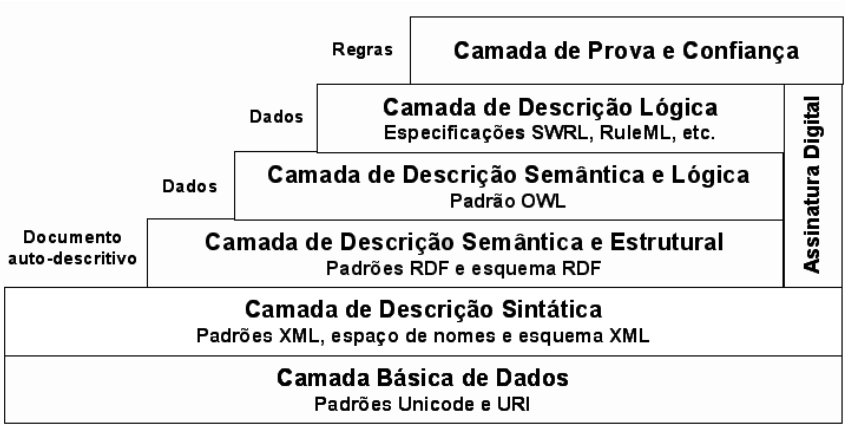

Figura 2.1: Arquitetura da Web Semântica. Adaptado de Berners-Lee [2000] e Bulcão Neto et al. [2006].

O propósito da Web Semântica não é de criar uma nova Web, mas sim de estender a Web atual em que a informação deverá ter semântica explícita e formalmente definida para um melhor entendimento tanto por pessoas como por computadores [Berners-Lee et al., 2001]. Com esse propósito, a arquitetura em camadas apresentada na Figura 2.1 reaproveita padrões e especificações já utilizadas na Web atual, tais como URI, padrões *Unicode*, XML, etc., ao mesmo tempo que adiciona novos padrões e especificações tais como RDF, OWL, SWRL, etc. As seções seguintes apresentam cada uma das camadas da arquitetura da Web Semântica ilustradas na Figura 2.1.

Camada básica de dados

O padrão *Unicode*[1] para codificação de caracteres consiste de um conjunto de cerca de cem mil caracteres que permite manipular e representar de forma universal e não-ambígua texto de qualquer sistema de escrita existente. O *Unicode* tem sido adotado como padrão em diversos sistemas de computação, incluindo os principais sistemas operacionais, editores de texto e navegadores Web [Dürst and Freytag, 2007].

Uma das principais características da Web Semântica é que qualquer pessoa pode nomear e descrever "coisas" na Web. O padrão URI (*Uniform Resource Identifiers*), que

[1] http://www.unicode.org/

também aparece na camada básica de dados da Figura 2.1, é utilizado pelas camadas superiores para identificar recursos na Web.

Para que se possa descrever coisas, é necessário uma maneira de identificá-las de forma única, que tanto na Web atual quanto na Web Semântica são chamadas de recursos e são identificados com o uso de URIs. Uma URI é uma cadeia de caracteres com uma sintaxe particular que serve para identificar um recurso. Recursos podem ser qualquer coisa (abstrata ou física) que tenha uma identidade.

Assim, a camada básica de dados fornece padronização para codificação de caracteres, endereçamento de recursos na Web Semântica e é utilizada pelos padrões de documentos das camadas superiores da arquitetura da Web Semântica.

Camada de descrição sintática

A Web Semântica deve ser representada por uma linguagem que seja consistente, padronizada e estruturada, passível de ser processada por aplicações computacionais. Assim, pode ser vista nas camadas da Web Semântica, conforme ilustrado na Figura 2.1, a camada de descrição sintática, que refere-se aos padrões XML (*Extensible Markup Language*) [W3C, 2004a], espaço de nomes XML (*XML Namespaces*) [W3C, 2001a] e esquema XML (*XML Schema*) [W3C, 2004d].

Qualquer pessoa, organização ou software pode especificar uma URI e essa flexibilidade é importante, porém, pode resultar em duplicações. Um termo chamado TITLE pode estar presente em diferentes contextos e com diferentes semânticas (título do CD, título da música, título de um filme, título de um profissional, etc.). Para evitar colisões de nomes e ambigüidades, utiliza-se o conceito de espaço de nomes XML em conjunto com o endereçamento por URI. A idéia é associar um espaço de nomes a uma URI e ainda que um nome apareça em mais de um espaço ele deve ser único no seu espaço de nomes.

O padrão XML descreve uma classe de objetos chamada documentos XML e o comportamento de aplicações computacionais que processam esses documentos W3C [2004a]. A especificação do XML 1.1 é a versão atual do padrão W3C e corresponde a um conjunto de regras para definir marcações que estruturam um documento em partes e as identificam. XML também pode ser definido como uma meta-linguagem de marcação que determina uma sintaxe usada para definir outras linguagens de marcação para domínios específicos. Os elementos XML adicionam estrutura ao conteúdo dos documentos, não se atendo às questões de formatação e apresentação desses documentos.

Por ser um padrão aberto (não-proprietário) e simples de ler e escrever por aplicações de software, um documento no formato XML torna-se um excelente formato para intercâmbio de dados entre diferentes aplicações.

A recomendação XML admite ainda que sejam especificadas regras de sintaxe

ainda mais rígidas para documentos, através de esquemas XML, que são documentos que atribuem regras de sintaxe para a criação de instâncias de documentos XML em um domínio específico. Um documento XML que segue determinadas regras de formação (ou esquema) é uma instância XML desse esquema.

O XML é utilizado como padrão dos documentos na Web Semântica pois é um padrão de fato para informações estruturadas na Web, pode seguir regras definidas por uma gramática (esquema XML) e possibilita o processamento por computador e intercâmbio de informações. Entretanto, XML é insuficiente para os propósitos da Web Semântica, pois, o foco do XML está na sintaxe e não na semântica das informações. Um problema comum em XML é que existem muitas formas de descrever a mesma coisa e isso o torna ambíguo para um computador. Um outro problema é que o XML também não possibilita a descrição de relacionamentos entre recursos na Web.

É importante notar que a camada de descrição sintática (espaço de nomes XML, XML e esquema XML) utiliza os padrões da camada inferior da arquitetura da Web Semântica, que incluem URI e o padrão Unicode. Todos os recursos contidos em um documento XML são endereçados através de uma URI e os documentos são texto puro com codificação Unicode. Conforme ilustrado na Figura 2.1, em uma camada acima do XML encontra-se o RDF e o esquema RDF, que resolvem alguns dos problemas relacionados ao XML, em relação ao uso na Web Semântica.

Camada de descrição semântica e estrutural

O padrão RDF (*Resource Description Framework*) W3C [2004b], presente na camada de descrição semântica e estrutural da Figura 2.1, e sua forma de concretização em documentos XML chamada RDF/XML [Beckett, 2004], são recomendações do W3C para definição de valores de propriedades associadas com recursos da Web.

A especificação de RDF define um modelo de dados para representação de informação sobre recursos na Web [W3C, 2004b]. Seu propósito é representar metadados de recursos na Web. Em RDF, entende-se como um recurso da Web não apenas objetos que podem ser recuperados por meio da Web, mas qualquer tipo de recurso que pode ser identificado na Web. Ou seja, além de representar objetos recuperáveis pela Web tais como imagens, documentos HTML, áudio, vídeo, etc., RDF também representa recursos como, por exemplo, pessoas e artigos disponíveis em lojas *on-line*.

O modelo de dados RDF está baseado no conceito de declarações, em que uma declaração é composta por um sujeito, um predicado e um objeto. O sujeito é sempre um recurso e, como tal, sempre tem uma URI associada. O predicado, que também é um recurso e sempre tem uma URI associada, é uma propriedade do recurso sujeito. Por fim, o objeto é o valor da propriedade, que pode ser um outro recurso

(e ter uma URI) ou uma literal (que não possui URI). A importância de se identificar com uso de URIs vai além da necessidade de uma identificação, pois, com uma URI, por exemplo, uma vez descrita uma propriedade, ela pode ser utilizada por diversos outros recursos, e um mesmo recurso identificado com uma URI pode ter diversas propriedades associadas. O principal propósito de RDF é identificar recursos usando identificadores Web chamados URI e descrever os recursos em termos de propriedades e seus valores.

O modelo de dados de RDF introduz, na arquitetura em camadas dos padrões para a Web Semântica (ver Figura 2.1), uma especificação para descrição de recursos. Isto já possibilita agregar descrições semânticas aos recursos da Web. Porém, deixa a desejar quando se pensa em relações semânticas mais complexas entre os recursos (ex: classes de recursos, enumeração e tipos de dados).

A linguagem de descrição de vocabulários denominada esquema RDF (*RDF Schema* – RDFS) é uma extensão semântica do RDF, que provê mecanismos para descrição de grupos de recursos relacionados e o relacionamento entre os recursos [W3C, 2004c].

Na Figura 2.1, o esquema RDF aparece na mesma camada que o RDF, posto que esquema RDF é escrito utilizando o próprio RDF. O RDF define um modelo de dados abstrato para fazer declarações sobre recursos. A sintaxe RDF/XML é utilizada para codificar essas declarações em uma sintaxe concreta. Enquanto que o RDFS adiciona características e fornece um vocabulário padronizado para descrever conceitos.

O RDF apenas descreve recursos (objetos individuais) sem se ater a detalhes do domínio da aplicação. Com RDFS é possível descrever classes que definem os tipos desses objetos. O conceito de classes em RDFS é similar ao conceito de classes da orientação a objetos, em que objetos de uma determinada classe são instâncias da classe. Com uso de classes, é possível impor restrições nas declarações em RDF permitir construções do tipo hierárquica com herança de classes.

Todas as primitivas RDF e RDFS na Web Semântica são concretizadas através da sintaxe RDF/XML. Assim, RDF e RDFS utilizam todos os padrões das camadas inferiores da arquitetura da Web Semântica, que incluem Unicode, URI, XML, esquema XML e espaço de nomes XML.

Com RDFS é possível especificar ontologias limitadas através de hierarquia de classes e de propriedades padronizadas. As principais contribuições de RDFS são a adição formal de conceitos semânticos de classes, propriedades, generalizações, domínio e restrições a propriedades. Apesar de RDFS prover diversos aspectos para a especificação de ontologias, ainda existem muitos outros requisitos, não cobertos por RDFS, para uma linguagem de ontologia. Por exemplo, RDFS não provê certos tipos de restrições, tal como restrições em cardinalidades das propriedades.

Em RDFS também existem poucos descritores que permitam inferência. Seriam necessárias mais regras para possibilitar que agentes de software possam inferir

novos fatos a partir das classes e propriedades descritas. Em RDFS não é possível inferir, por exemplo, que um objeto pertence a determinada classe porque ele contém um certo tipo de valor.

O padrão RDFS não é expressivo o suficiente para prover descrições de ontologia necessárias para a realização da Web Semântica. Dessa forma, para prover mais expressividade e para possibilitar inferência, conceitos semânticos mais avançados devem ser adicionados à pilha de padrões (ver Figura 2.1) da Web Semântica, ou seja, é necessário uma linguagem, mais poderosa que RDFS, que permita a definição de ontologias sofisticadas.

Camada de descrição semântica e lógica

O termo ontologia tem se tornado bastante comum na Web, em que é utilizado desde em taxonomias para categorizar Web sites até categorização de produtos para vendas *on-line* [Noy and McGuinness, 2001]. O W3C tem concentrado esforços na criação e padronização de uma linguagem para ontologia na Web Semântica: a OWL [Antoniou and van Harmelen, 2003; Bechhofer et al., 2004; McGuinness and van Harmelen, 2004].

O W3C, através do *Web Ontology Working Group* [W3C, 2001b], identificou uma série de casos de uso da Web Semântica que precisam de mais expressividade que RDF e RDFS podem oferecer. Assim, OWL surgiu da necessidade de uma linguagem de ontologia para a Web Semântica com mais expressividade que RDF e RDFS, pois, RDF limita-se à criação de predicados binários para descrição de recursos e RDFS, embora já tenha conceitos de ontologia, é limitado a hierarquia de classes e propriedades, com especificação de domínio e espaço de valores dessas propriedades. Antoniou and van Harmelen [2003] citam alguns aspectos que, em termos de ontologia, não são cobertos por RDFS, são eles:

- propriedades com escopo local: em RDFS não se pode declarar restrições a espaços de valores para apenas algumas classes;
- classes disjuntas: não existe relação de disjunção entre classes em RDFS. Por exemplo, pessoas do sexo masculino e feminino são classes disjuntas;
- combinações booleanas de classes: com RDFS não é possível criar novas classes a partir de combinações de outras classes, tais como união, intersecção e complemento;
- restrições de cardinalidade: é impossível expressar restrições de cardinalidade em propriedades com RDFS, ou seja, não dá para expressar, por exemplo, que uma pessoa tem exatamente uma mãe biológica;

- características especiais de propriedades: em RDFS, também não é possível expressar que uma propriedade é transitiva, única ou inversa a outra propriedade.

Inicialmente, o DARPA (*Defense Advanced Research Projects Agency*), em conjunto com o W3C, desenvolveram uma linguagem para ontologia chamada DAML+OIL [McGuinness et al., 2002], que surgiu da junção das propostas do DAML-ONT [DAML, 2000] e do OIL [OntoKnowledge, 2000] e que foi o ponto de início para a criação da OWL, uma linguagem com o propósito de ser aceita como a linguagem de ontologia para a Web Semântica.

Na Figura 2.1, a camada de descrição semântica e lógica aparece logo acima da camada do RDF e RDFS, posto que OWL deve ser uma extensão de RDFS, no sentido de que OWL utiliza as definições de classes e propriedades de RDFS e adiciona novas primitivas que devem modelar a riqueza de expressividade requerida pela Web Semântica.

Quanto mais lógica for adicionada para estender RDFS, mais ineficiente será a capacidade de inferência, que pode chegar ao limite da não-computabilidade. Dessa forma, é necessário uma linguagem que consiga aliar um bom poder de expressão a uma eficiente capacidade de inferência computacional. Com esse objetivo, o W3C definiu OWL com três sub-linguagens diferentes, em que, cada uma delas tem maior ou menor poder de expressividade e capacidade de inferência: OWL Full, OWL DL e OWL Lite (ver Figura 2.2).

Conforme pode ser visto na Figura 2.2 a linguagem OWL Full possui todas as primitivas de OWL e permite a combinação de todas essas primitivas de forma arbitrária com RDF e RDFS. Isto inclui a possibilidade de efetuar mudanças em primitivas pré-definidas como, por exemplo, impor restrição de cardinalidade na classe de todas as classes, limitando o número de classes que podem ser descritas em uma ontologia.

Com OWL Full um desenvolvedor de tecnologia pode dizer “qualquer coisa” a respeito de “qualquer coisa”. Porém, a grande desvantagem é que OWL Full é tão poderosa quanto indecidível.

Com o objetivo de atingir eficiência computacional algumas restrições são impostas à linguagem OWL Full gerando duas novas linguagens: a OWL DL (lógica de descrição) e a OWL Lite. Ambas são sub-linguagens de OWL Full que restringem a maneira como os construtores de OWL e RDF podem ser utilizados. OWL DL contém todo o vocabulário de OWL Full e menos restrições que o OWL Lite. Já OWL Lite possui menos construtores e ainda mais restrições em relação à OWL DL. O propósito das restrições é diminuir a complexidade para garantir que seja computável.

As diferenças entre os dialetos de OWL afetam na escolha do desenvolvedor de ontologia devido às restrições impostas. Conforme ilustrado na Figura 2.2 a escolha

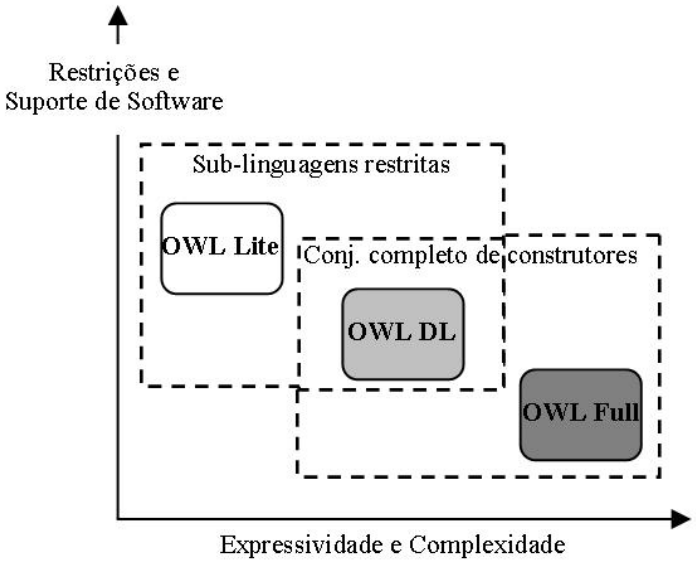

Figura 2.2: Especificação de restrições em OWL. Adaptado de Lacy [2005].

do dialeto reflete nas vantagens e desvantagens de se ter mais expressividade (aliado a complexidade) e a disponibilidade de software para inferência (aliado a restrições). Assim, deve-se analisar os requisitos e, com base neles, tentar escolher o dialeto mais restrito para maximizar o uso de ferramentas de software disponíveis.

Em geral, ontologias OWL são representadas em arquivos com sintaxe RDF/XML disponibilizados na Web e referenciados por uma URI definida pelo autor da ontologia. Para compor uma base de conhecimento OWL, arquivos de ontologias têm associados a si arquivos de instâncias. Arquivos de ontologias contêm descrições dos conceitos do domínio, enquanto que arquivos de instâncias, fatos acerca desses conceitos.

OWL é a linguagem de ontologia proposta para a Web Semântica, que inclui as primitivas RDF e RDFS e adiciona novas primitivas com o propósito de aumentar o poder de expressividade necessário a uma linguagem de ontologia para a Web Semântica. Assim, é mantida a utilização das camadas inferiores da arquitetura da Web Semântica, que que incluem Unicode, URI, XML, esquema XML, espaço de nomes XML, RDF e RDFS.

Camada de descrição lógica

Embora OWL seja expressiva o suficiente para uso na prática, pois possui construtores para modelar conceitos como disjunção, propriedades inversas, cardinalidades, etc., seria ainda mais interessante e poderoso se fosse possível indicar que qualquer princípio lógico pudesse ser entendido (por inferência) pelo computador [Swartz, 2006]. Um exemplo disso seria definir um princípio lógico com a seguinte semântica: “se um cliente sempre compra em uma mesma loja (cativo) e tem mais de 60 anos,

então ele deve receber desconto dessa loja". Isto é facilmente representado com regras, como, por exemplo, em:

```
clienteCativo(X) ^ idade(X) > 60 --> desconto(X)
```

Esse tipo de declaração não é possível de ser representada com OWL. Assim, existem estudos na direção de estender a OWL com regras SWRL (*Semantic Web Rule Language*) [Horrocks et al., 2004]. O SWRL é uma linguagem baseada na combinação de OWL com a linguagem RuleML [Hirtle et al., 2008] para representação de fatos e regras. RuleML utiliza um subconjunto da linguagem Prolog e tem como objetivo principal prover um vocabulário compartilhado para a definição de regras na Web Semântica.

O propósito de SWRL é de combinar os dialetos OWL DL e OWL Lite com o RuleML e utilizar todos os padrões das camadas inferiores da arquitetura (ver Figura 2.1), que incluem URI, XML, esquema XML, espaço de nomes XML, RDF, RDFS e OWL. Atualmente, SWRL é uma proposta submetida ao W3C com o propósito de se tornar a linguagem padrão para definição de regras na Web Semântica.

Camada de prova e confiança

As camadas de prova e confiança são componentes da Web Semântica que ainda não possuem implementações concretas. Swartz [2006] afirma que nem todas as fontes e recursos da Web Semântica deverão ser confiáveis, da mesma forma que não é na Web atual. Uma característica crucial que impulsionou o crescimento da Web atual e da quantidade de documentos disponibilizados é que qualquer pessoa pode criar seus próprios documentos e disponibilizá-lo na Web. Com a Web Semântica não deve ser diferente: qualquer pessoa poderá fazer qualquer declaração a respeito de qualquer recurso na Web.

Assim, é necessário uma forma de garantir que essas informações sejam confiáveis. Segundo Swartz [2006], isto deverá ser realizado com o uso de assinaturas digitais. Todas as declarações RDF, por exemplo, deverão ser assinadas digitalmente e cada pessoa poderá ter certeza de quem criou aquela declaração e ainda informar aos seus programas quais assinaturas são confiáveis e o computador pode decidir, com base nas assinaturas, se a informação que está processando é originada de uma fonte confiável.

2.1.3 Ontologias da Web Semântica

Esta seção apresenta as ontologias da Web Semântica que serviram de apoio para a descrição, publicação, descoberta e composição de Serviços Web conforme apresentado nos capítulos 3, 4, 5 e 6, são elas: OWL-S [Martin et al., 2004a],

OWL-Time [Hobbs and Pan, 2004; Hobbs et al., 2004], *Resource Ontology* [Martin et al., 2004b] e as ontologias de domínio que são combinadas com a OWL-S para prover descrição dos Serviços Web.

OWL-S

OWL-S é uma linguagem de marcação semântica submetida ao W3C [Martin et al., 2004a] como proposta de uma linguagem de ontologia para Serviços Web de propósito genérico, ou seja, uma ontologia de nível superior capaz de descrever quaisquer tipos de Serviços Web independentemente do domínio ao qual pertença o serviço.

O objetivo da ontologia OWL-S é de prover descrição semântica aos Serviços Web com o intuito de possibilitar descoberta e composição automáticas dos serviços. No presente trabalho, OWL-S é combinada com outras ontologias para prover descrição semântica aos Serviços Web Semânticos. Assim, por ser a principal ontologia utilizada neste trabalho, OWL-S é descrita com mais detalhes na Seção 2.2.

OWL-Time

Hobbs and Pan [2004] definiram uma ontologia de nível superior para conceitos temporais denominada *OWL-Time*. Essa ontologia contém uma especificação da teoria de tempo requerida para aplicações na Web Semântica e é utilizada para descrever conteúdo temporal de páginas Web, bem como para descrever propriedades e restrições temporais de Serviços Web.

A Figura 2.3 apresenta a hierarquia de classes da ontologia *OWL-Time*. *TemporalThing* é a classe mais genérica da ontologia e representa qualquer entidade com características temporais e todas as outras classes da ontologia, exceto *TemporalEntity* e *Event*, são sub-classes, direta ou indiretamente, dessa classe.

Na parte superior da Figura 2.3, tem-se as classes que representam instantes de tempo. *InstantThing* é a classe genérica para representar instantes e possui duas subclasses: a classe *Instant*, que representa um ponto preciso no tempo; e a classe *InstantEvent* que representa um evento instantâneo, tal como a ocorrência de um acidente de carro ou a chegada de um vôo.

A parte inferior da Figura 2.3 apresenta as classes que representam intervalos de tempo. *IntervalThing* é a classe genérica para representar intervalos e possui três sub-classes: *Interval* representa um período de tempo limitado por dois instantes; a classe *IntervalEvent* representa um evento que ocorre em um determinado intervalo de tempo, como exemplo, uma reunião; e as classes *ProperInterval* e *ProperIntervalThing* representam um período de tempo limitado por dois instantes, em que os instantes não podem ser os mesmos.

OWL-Time permite ainda descrever relações temporais entre os conceitos apresen-

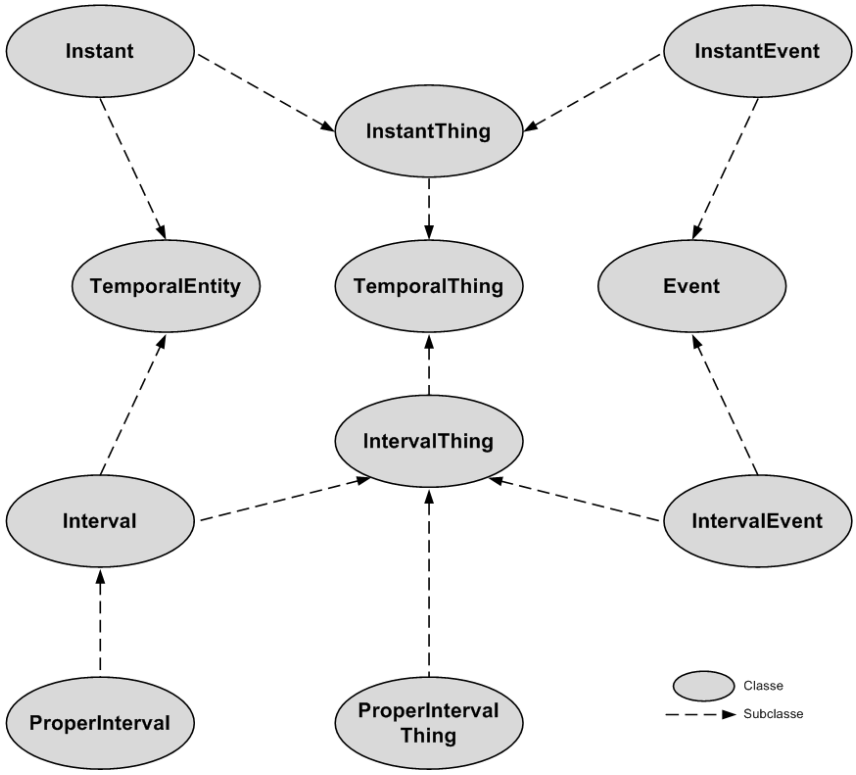

Figura 2.3: Hierarquia de classes da ontologia *OWL-Time*.

tados na Figura 2.3. As principais relações temporais são *before*, *after*, *begins*, *ends* e *inside*.

Hobbs and Pan [2006] submeteram a ontologia OWL-Time para apreciação no consórcio W3C com o propósito de tornar a ontologia um padrão para descrição de conceitos e relações temporais na Web Semântica. Neste trabalho, a ontologia *OWL-Time* é utilizada para prover restrições temporais aos Serviços Web Semânticos [Prazeres et al., 2007; Escobedo et al., 2007; Prazeres et al., 2008].

Ontologia de recursos

A ontologia *Resource Ontology* (ontologia de recursos) [Lassila, 2001] é uma proposta para descrição de recursos para Serviços Web e sua intenção é de ser utilizada em conjunto com a OWL-S. Por ser uma ontologia de nível superior, tem o propósito de descrever recursos com nível de abstração suficiente para cobrir recursos físicos, temporais e computacionais, bem como outros tipos de recursos. Tipos específicos de recursos possuem propriedades específicas e a ontologia de recursos apresenta as principais propriedades e tipos que um recurso pode assumir. Ou seja, a ontologia foi criada para ser bastante genérica, assim como fácil de ser estendida para tipos de recursos específicos.

A Figura 2.4 apresenta a ontologia de recursos em que destacam-se três aspectos: tipo de alocação do recurso, o tipo da capacidade e a composição (se é atômico ou uma agragação de recursos).

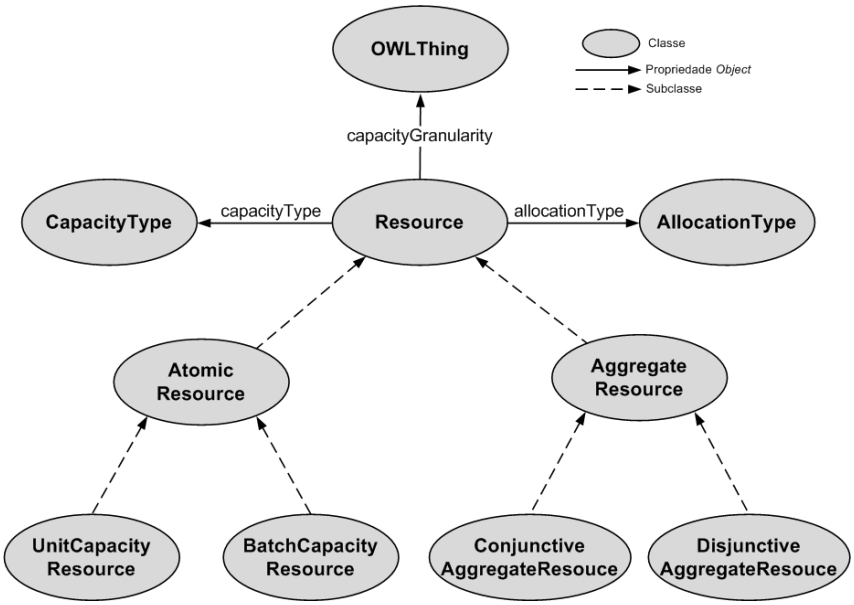

Figura 2.4: Hierarquia de classes da ontologia de recursos.

Na ontologia, os recursos são alocados para atividades ou processos. No caso de Serviços Web descritos por OWL-S, que é a forma como a ontologia de recursos é utilizada neste trabalho, os recursos são alocados para processos do Serviço Web. A principal distinção no tipo de alocação de um recurso é se o recurso é consumido (*ConsumableAllocation*) ou não (*ReusableAllocation*) após ser alocado.

O tipo da capacidade de um recurso pode ser uma medida contínua, tal como a quantidade de um líquido, ou discreta, tal como o número de assentos disponíveis em um vôo. Assim, os tipos de capacidade de um recurso em *Resource Ontology* são *DiscreteCapacity* e *ContinuousCapacity*.

Ontologias de domínio

No presente trabalho a OWL-S é utilizada na descrição dos Serviços Web de forma a prover a semântica necessária para possibilitar a descoberta e composição automáticas descritas nos capítulos 4 e 5. Essa descrição é realizada através da combinação de OWL-S, *OWL-Time* e da ontologia de recursos, tal como descrito nos capítulos 3 e 4. Para descrever serviços em domínios específicos, OWL-S é combinada com ontologias de domínio. Algumas dessas ontologias de domínio são utilizadas ao logo deste trabalho, tal como ontologia para classificação de experimentos remotos.

2.2 Serviços Web Semânticos (SWS)

Como o próprio nome sugere, os Serviços Web Semânticos são baseados em duas das mais importantes tendências na evolução da *World Wide Web*: Serviços Web e Web Semântica. Os Serviços Web Semânticos são uma área recente e vigorosa de pesquisa tecnológica, que geralmente é definida como o enriquecimento das descrições de Serviços Web através do uso de anotações da Web Semântica [Martin et al., 2007b;c]. O objetivo dos Serviços Web Semânticos é o de possibilitar a automação de tarefas como descoberta, composição e invocação dos Serviços Web [Payne and Lassila, 2004].

Os Serviços Web Semânticos representam um papel importante na direção da realização da Web Semântica da forma como foi apresentada por Berners-Lee et al. [2001]. Payne and Lassila [2004] afirmam que a relação entre a Web Semântica e a atual arquitetura dos Serviços Web pode ser vista de duas formas. A primeira forma, em um curto prazo, é a visão de que o desenvolvimento dos Serviços Web é crucial e as técnicas da Web Semântica podem enriquecer a atual arquitetura dos serviços. Segundo, em um prazo mais longo, a visão da Web Semântica, por si só, irá tornar-se mais interessante à medida que os Serviços Web ofereçam uma infra-estrutura ubíqua, em que a próxima geração de sistemas multi-agentes seja implementada.

As pesquisas em Serviços Web Semânticos começaram a tomar força no ano de 2001. Nesse ano, McIlraith et al. [2001] publicaram suas pesquisas e discussões a respeito do potencial e importância de se adicionar os padrões estabelecidos para a Web Semântica à arquitetura dos Serviços Web. Ainda em 2001, a primeira versão do OWL-S (inicialmente denominado DAML-S) foi disponibilizada. Dai por diante outras iniciativas de uso das técnicas da Web Semântica em conjunto com os Serviços Web tiveram início, das quais pode-se destacar o WSMO [Bruijn et al., 2005; Fensel et al., 2006], SWSF [Battle et al., 2005] e WSDL-S [Akkiraju et al., 2003; 2005].

Esta seção aborda os conceitos e tecnologias dos Serviços Web. A primeira geração de Serviços Web é apresentada na Seção 2.2.1 e tem o foco na descrição sintática das operações e das mensagens que podem ser trocadas entre os Serviços Web ou entre Serviços Web e outras aplicações Web. A próxima geração de Serviços Web, os Serviços Web Semânticos são alvo de muitas pesquisas na comunidade acadêmica e é objeto de estudo deste trabalho. O foco dos Serviços Web Semânticos é na descrição semântica dos serviços, das suas funcionalidades, das suas operações e mensagens e da execução do serviço. A linguagem para descrição de Serviços Web Semânticos utilizada neste trabalho é apresentada em detalhes na Seção 2.2.2. A Seção 2.2.3 apresenta outras linguagens e iniciativas também com foco na descrição semântica de Serviços Web.

2.2.1 Serviços Web

O W3C define Serviços Web como sistemas de *software* identificados por uma URI e especificados por um documento de descrição baseado na linguagem XML. A descrição em XML permite que outros sistemas identifiquem o Serviço Web, possibilitando interação com o mesmo. Além da descrição, o XML também é usado na comunicação, com troca de mensagens e utilizando protocolos da Internet [W3C, 2002].

Segundo Stal [2002] um Serviço Web é uma unidade de aplicação interoperável que independe de linguagens de programação, sistemas operacionais, protocolos de comunicação e formato de representação dos dados. É também uma infra-estrutura para desenvolver e servir aplicações distribuídas. Os Serviços Web são produzidos para serem utilizados por aplicações, em contraste com a maioria das aplicações da Web atual que são produzidas para serem utilizadas por pessoas.

A interação entre uma aplicação baseada em Serviços Web e o próprio Serviço Web utiliza a abordagem da Arquitetura Orientada a Serviços (*Service Oriented Architecture* – SOA). Em SOA, quando um serviço é criado as informações sobre sua interface e sua localização são armazenadas em um registro (Figura 2.5 - a). O consumidor do serviço (aplicação) pode recuperar essas informações (Figura 2.5 - b) e usá-las para chamar o serviço (Figura 2.5 - c).

Figura 2.5: Arquitetura Orientada a Serviços (SOA). Adaptado de Bond et al. [2004].

O fraco acoplamento, característico da abordagem SOA, garante que os clientes sejam independentes do serviço, pois, o cliente não precisa conhecer a plataforma de desenvolvimento e nem mesmo a linguagem utilizada para codificar o serviço. Os serviços tem interfaces bem definidas, garantindo que mudanças na aplicação não

afetem o cliente. A implementação do serviço (regras de negócio) pode mudar, porém, a interface provida deve continuar a mesma.

Serviços Web trazem vantagens similares ao uso de componentes de *software*. O uso de serviços permite tirar proveito, em áreas que não sejam o foco de uma organização, da expertise de outras organizações, sem precisar tornar-se um especialista na área em questão.

Os Serviços Web são baseados em algumas tecnologias padrões da indústria: XML, *Simple Object Access Protocol* (SOAP) [W3C, 2003], *Web Service Description Language* (WSDL) [W3C, 2006] e *Universal Description, Discovery, and Integration* (UDDI) [OASIS, 2004].

A linguagem XML já foi abordada neste trabalho e não será detalhada aqui. Porém, é importante notar que o XML tem um papel fundamental na realização de Serviços Web, pois, como se pode verificar a seguir, a linguagem de descrição de Serviços Web (WSDL), o protocolo de comunicação (SOAP) e as informações sobre serviços registrados (UDDI) são todos baseados em documentos XML.

A linguagem de descrição de Serviços Web (WSDL) é usada para descrever um Serviço Web, especificando sua localização e descrevendo as operações providas pelo serviço. Os documentos WSDL são documentos XML que fornecem informações suficientes sobre como interagir com o Serviço Web em questão e contém cinco elementos [Nandigam et al., 2005]:

- três elementos abstratos que definem a interface do serviço:
  - tipos: são esquemas XML em que os tipos de dados são definidos;
  - mensagens: descrevem detalhes dos métodos e seus parâmetros;
  - tipo de porta: definem operações em termos de mensagens de entrada/saída.
- dois elementos concretos que definem propriedades físicas:
  - *binding*: provê informação de protocolo para as operações. Permitem a concretização das mensagens de entrada/saída;
  - endereços de serviço: especifica a URI para localização de um serviço.

Os objetos do lado do cliente que utiliza o serviço o fazem por meio de uma classe *proxy* que "imita" as chamadas de métodos do Serviço Web. Assim, os desenvolvedores de aplicações trabalham com o *proxy* em vez de escrever mensagens SOAP diretamente. A classe *proxy* gerencia a construção, envio e recebimento de mensagens SOAP.

O SOAP é um protocolo de comunicação e formato de codificação baseado em XML que tem o propósito de realizar comunicação entre aplicações. A especificação de mensagens SOAP tem três partes [Nandigam et al., 2005]:

- envelope para encapsulamento de dados – é o nó raiz de uma mensagem SOAP e tem duas partes:
  - cabeçalho: especifica os requisitos de aplicação, tais como assinatura digital para serviços protegidos por senha, número de conta para serviços pagos e gerenciamento de transação;
  - corpo: descreve o conteúdo da mensagem e instruções de processamento, tais como nome do método, parâmetros e valores de retorno. Também pode conter tratamento de erros, como código da mensagem de erro, descrição do erro e qual objeto causou o erro;
- regras para codificação dos dados – usadas para expressar tipos de dados, baseados nos tipos de dados do esquema XML;
- convenções para chamadas remotas a procedimentos (RPC) – especifica uma convenção para representação de chamadas e respostas a procedimentos remotos.

As informações sobre serviços registrados (UDDI) funcionam como uma coleção de informação de todos os Serviços Web registrados, possibilitando a descoberta de provedores de serviço. O UDDI é um registro de domínio público – provedores de serviço registram seus Serviços Web e clientes buscam pelo serviço apropriado. O repositório UDDI é representado na formato XML que é formado por três componentes:

- páginas brancas: contém endereço, detalhes de contato e identificadores conhecidos para provedores de Serviços Web.
- páginas amarelas: categorização industrial de Serviços Web, baseada em taxonomias padrões.
- páginas verdes: contém informações técnicas sobre serviços.

Segundo Arroyo et al. [2005], o papel dos Serviços Web pode mudar, de forma significante, de um mero provedor de informações sobre serviços para algo mais elaborado com verdadeiro impacto sobre o mundo real. Isto requer localização e composição de "pedaços de *software*" (Serviços Web) sob demanda, acompanhando as necessidades da tarefa que o usuário está executando. Nesse sentido, conforme Sirin et al. [2004a], o uso de ontologias de domínio da Web Semântica e regras de inferência

tem um papel fundamental. A idéia é poder minimizar a intervenção humana na utilização de Serviços Web, agregando marcações semânticas a esses serviços.

A principal limitação da primeira geração de Serviços Web é a pouca, ou até inexistente, descrição semântica dos serviços. Uma tendência recente em se tratar esse aspecto consiste na combinação de Serviços Web com os conceitos fundamentais da Web Semântica. Isto tem resultado no surgimento de uma nova geração de Serviços Web chamados de "Serviços Web Semânticos", que é abordado a seguir nas Seção 2.2.2.

2.2.2 OWL-S: marcação semântica para Serviços Web

A linguagem DAML-S (*DARPA Agent Markup Language for Services*) [Ankolekar et al., 2001; Burstein et al., 2002] foi a primeira tentativa de definir uma linguagem com o objetivo de agregar semântica na descrição de Serviços Web. A partir dai foi definida, através de um esforço combinado entre as comunidades da Web Semântica e de Serviços Web, a linguagem OWL-S, cujo propósito é possibilitar a automação da descoberta, invocação e composição de Serviços Web através de descrições semânticas. No momento da escrita desta tese o OWL-S está na versão 1.2 [Martin et al., 2006].

OWL-S é uma ontologia para serviços que é completamente escrita utilizando a linguagem de ontologia para Web OWL, isto é, os axiomas de OWL-S são definidos em OWL e o OWL-S possui descrições de classes, hierarquia de sub-classes e definições de tipos e relações entre as classes. A expressividade de OWL-S é dada em lógica de descrição pois é baseada na OWL-DL.

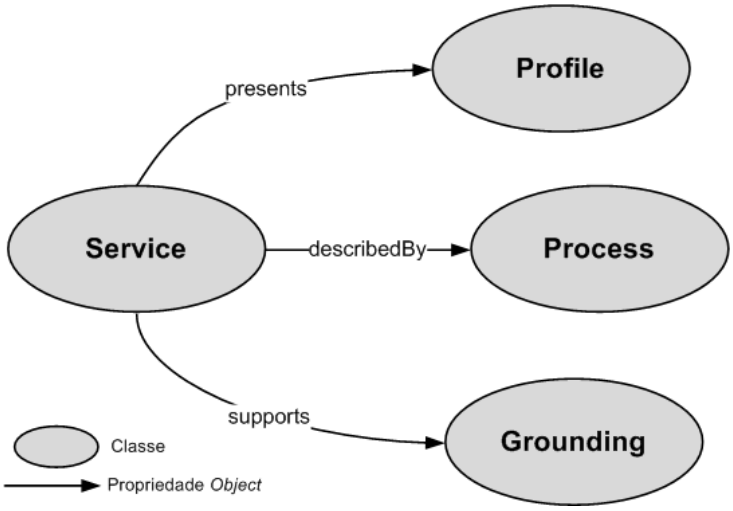

Figura 2.6: Visão geral da ontologia OWL-S. Adaptado de Martin et al. [2004b].

A estrutura da ontologia OWL-S é composta de três sub-ontologias como pode ser

visto na Figura 2.6: um perfil do serviço (classe *Profile*) para anúncio e descoberta de serviços; um modelo de processo (classe *Process*) que fornece a descrição detalhada das operações do serviço; e os fundamentos para concretização dos serviços (classe *Grounding*), que fornece detalhes em como interoperar com um serviço através de troca de mensagens.

Entradas, saídas pré-condições e resultados

As três sub-ontologias de OWL-S utilizam quatro conceitos também definidos em OWL-S que são entradas, saídas, pré-condições e resultados, comumente chamados de IOPR na literatura sobre OWL-S, devido ao nome desses conceitos em inglês na ontologia: *Inputs*, *Outputs*, *Preconditons* e *Results*.

Entradas e saídas são, respectivamente, as descrições dos objetos em que o serviço opera e as descrições dos objetos produzidos pelo serviço. As entradas e saídas representam a transformação da informação processada pelo serviço descrito por OWL-S.

As pré-condições e resultados representam a mudança de estado produzida pela execução do serviço. A pré-condição representa uma proposição que deve ser verdadeira para que o serviço execute corretamente, ou mesmo, mais restritivo ainda, para que o serviço execute. Os resultados são efeitos, que são preposições que irão se tornar verdadeiras logo após a execução do serviço.

O propósito de OWL-S é permitir que agentes de software executem Serviços Web para realizarem tarefas do mundo real. Segundo Martin et al. [2007a], as descrições de Serviços Web com o uso da ontologia OWL-S devem permitir que os agentes de software realizem tarefas cotidianas como, por exemplo, fazer reservas, comprar itens, agendar reuniões, dentre outras. As tarefas devem estar especificadas na forma de preposições e restrições em recursos e ações em entidades do mundo real, através das IOPRs. Tal entidades do mundo real, que são definições abstratas na ontologia OWL-S, devem ser mapeadas para mensagens concretas para que o serviço se realize.

Sub-ontologia de perfil do serviço

A sub-ontologia do perfil do serviço em OWL-S define um conjunto de conceitos para especificar as funcionalidades e capacidades do serviço. O objetivo do perfil de um serviço em OWL-S é fornecer semântica clara e explícita para agentes de software realizarem descoberta baseada nas funcionalidades ou capacidades do serviço [Martin et al., 2007a].

Para promover a descoberta automática dos serviços, o perfil em OWL-S permite que provedores de serviços anunciem o que seus serviços fazem, enquanto que permite também que clientes de serviços especifiquem o que eles esperam do serviço.

O perfil de um serviço fornece uma visão alto nível do serviço e é análogo à representação que o UDDI fornece para os Serviços Web atuais. Porém, o perfil em OWL-S provê descrição explícita e formal, baseada em ontologias, das capacidades dos serviços de forma que essas capacidades não são extraídas de propriedades fortuitas, tais como nome do serviço ou da empresa que provê o serviço, assim como o é em UDDI. Para descrever as funcionalidades de um Serviço Web, a ontologia de perfil de OWL-S faz referências externas a conceitos definidos em OWL na ontologia de domínio ao qual pertence o Serviço Web em questão.

Um perfil é uma instância da classe *Profile* apresentada na Figura 2.7. Essa classe especifica o serviço baseando-se em seus parâmetros funcionais e não funcionais. Os parâmetros funcionais apresentados no perfil são as entradas que um serviço requer, as saídas produzidas pelo serviço, as pré-condições necessárias à execução do serviço e os resultados gerados pela execução do serviço.

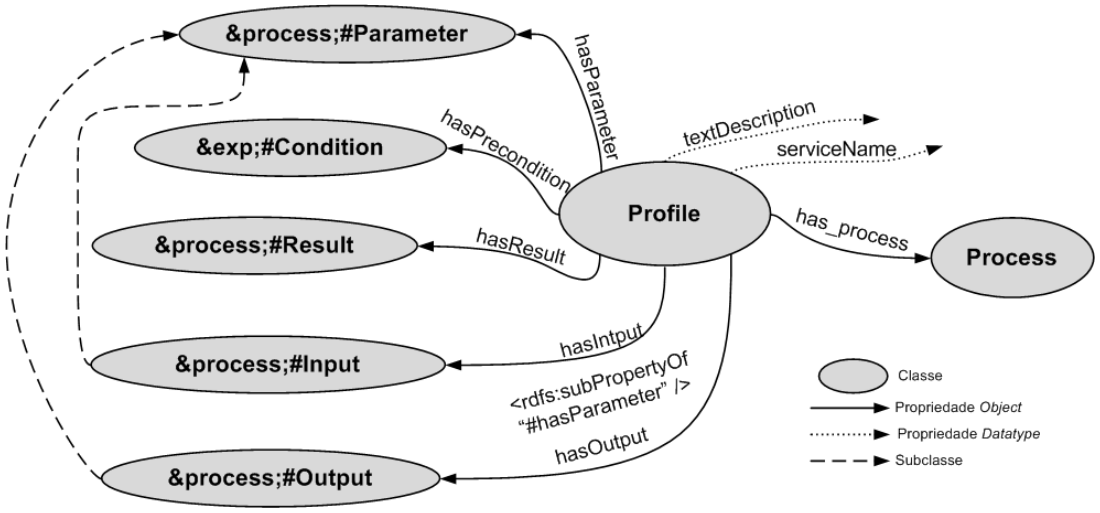

Figura 2.7: A ontologia de perfil do serviço em OWL-S. Adaptado de Martin et al. [2007a].

Conforme pode ser observado na Figura 2.7, os IOPRs não são definidos na classe *Profile* de OWL-S. As entradas, saídas, pré-condições e resultados de um serviço são definidos na ontologia de processos e na ontologia de perfil são referenciados como forma de expor as funcionalidades do serviço. Com isto, a proposta de OWL-S é que a descoberta seja realizada com base na transformação executada pelo serviço (entradas e saídas) e na mudança de estado no mundo real proporcionada pelo serviço (pré-condições e efeitos).

Os parâmetros não funcionais da classe *Profile* são divididos em duas seções. Primeiro, são informações semi-estruturadas e legíveis apenas para humanos, tais como, nome do serviço (*serviceName* na Figura 2.7) e uma descrição textual do serviço (*textDescription* na Figura 2.7). Segundo, são parâmetros não funcionais que também

podem ser utilizados na descoberta de serviços como, por exemplo, raio geográfico, parâmetros para definição de QoS do serviço ou tipo do serviço (que pode definir subclasses de *Profile* para domínios específicos).

Diversos algoritmos de busca têm sido propostos para OWL-S e a maioria é baseada na comparação do perfil (funcionalidade) do serviço. O algoritmo para busca de Serviços Web apresentado no Capítulo 4, como resultado deste trabalho, também é baseado na comparação do perfil do serviço (funcionalidades), porém difere dos demais na literatura por apresentar uma extensão que permite levar em consideração restrições temporais de acesso ao serviço.

Sub-ontologia de processo do serviço

A ontologia de processo (classe *Process* na Figura 2.6) define modelos de execução do Serviço Web sob a forma de processos que detalham o fluxo de dados o controle de fluxo entre os métodos do Serviço Web. Uma vez que o perfil do serviço parece relevante para atender aos objetivos definidos pelo agente de software que o solicitou, é necessário um modelo detalhado do serviço para determinar se o serviço pode atingir as necessidades do agente de software.

Para alcançar os resultados definidos no perfil do serviço um agente de software deve executar o modelo de processo considerando quais as restrições podem ser satisfeitas e quais os padrões de interações são requeridos para fazer uso do serviço.

Um processo em OWL-S especifica os padrões de interação na execução do serviço através da definição de dois tipos de processos: atômico e composto. A Figura 2.8 apresenta os processos atômicos e compostos e ainda a visualização dos principais elementos da ontologia de processos do OWL-S.

Os processos atômicos (*AtomicProcess* na Figura 2.8) podem ser diretamente executados e não contém sub-processos ou qualquer outra forma de estrutura interna. Na visão do cliente do serviço, processos atômicos são executados em um único passo que consiste na invocação de um método do serviço. O processo atômico corresponde a uma simples troca de entradas e saídas entre o consumidor e o provedor do serviço.

Os processos compostos (*CompositeProcess* na Figura 2.8) são especificados através da composição de processos atômicos ou de outros processos compostos. Em uma composição, os processos são conectados através de estruturas de controle de fluxo e de fluxo de dados (*ControlConstruct* na Figura 2.8).

O controle de fluxo é descrito usando construtores típicos de linguagens de programação ou construtores de *workflow* como, por exemplo, seqüência e estruturas de repetição. O fluxo de dados é dado pela descrição de como a informação é adquirida e utilizada nos passos subseqüentes do processo.

Ainda existe um terceiro tipo de processo definido em OWL-S que são os processos

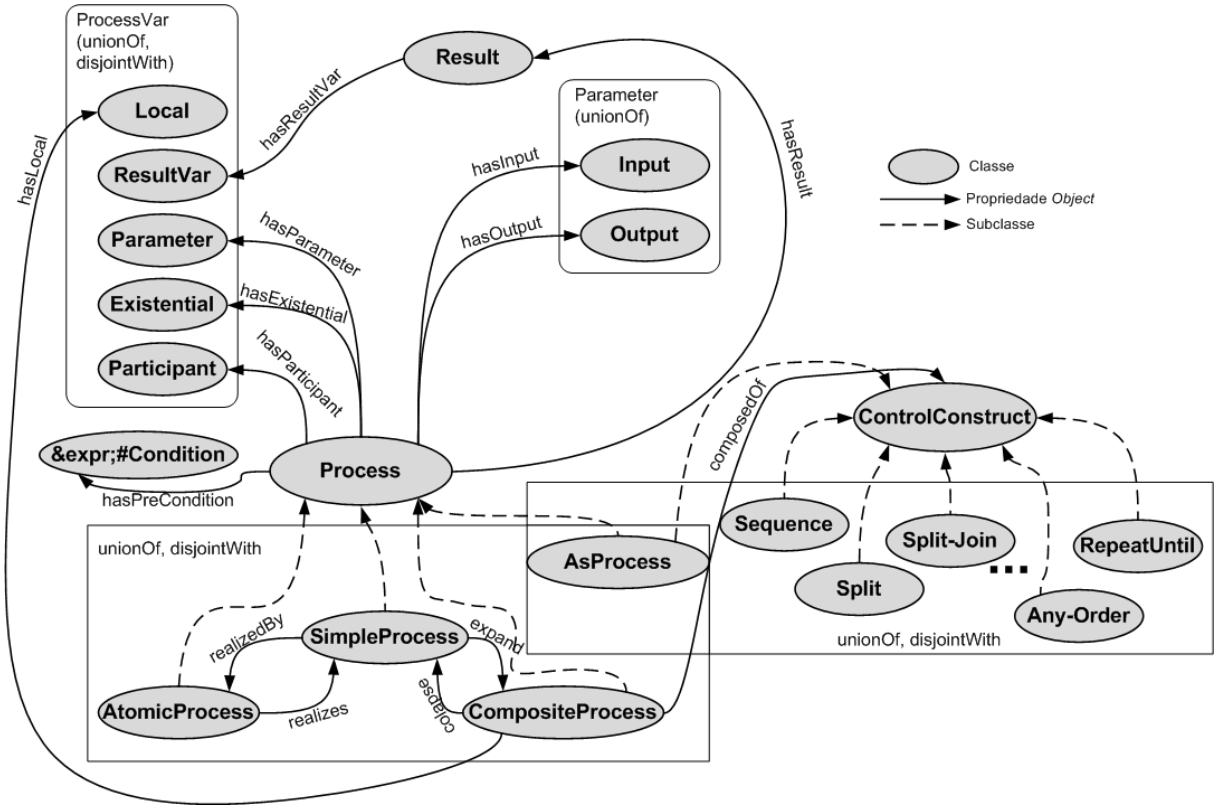

Figura 2.8: A ontologia de processo de OWL-S. Adaptado de Martin et al. [2007a].

simples (*SimpleProcess* na Figura 2.8). Tais processos não são executáveis e são utilizados para especificar visões abstratas de processos concretos (atômicos ou compostos) [Martin et al., 2007a].

Todos os conceitos utilizados para descrever um processo são termos definidos na ontologia OWL-S. A definição de um processo possui um conjunto de termos (*Input*, *Output*, *Preconditon*, *Result*, etc. – na Figura 2.8) que são conectados ao processo por meio de propriedades OWL (*hasInput*, *hasOutput*, *hasResult*, etc. – na Figura 2.8). Os processos compostos são relacionados à uma estrutura de controle por meio da propriedade OWL *composedOf* ilustrada na Figura 2.8. Por fim, ainda podem existir propriedades que relacionam as especificações abstratas da ontologia de processos com mensagens concretas para a realização do serviço, conforme especificado na sub-ontologia de concretização do serviço.

Sub-ontologia de concretização do serviço

O perfil e o modelo de processo dos serviços são especificações abstratas de características do Serviço Web que podem ser utilizadas por agentes semânticos para descoberta, invocação e composição de serviços. A especificação de como concretizar

o serviço permite a comunicação com um Serviço Web concreto realizando a ligação (*binding*) das entradas e saídas (abstratas) e dos processos atômicos, para formatos concretos de mensagens.

A classe *Grounding* (ver Figura 2.6) de OWL-S, neste trabalho chamada de sub-ontologia de concretização do serviço, define a forma como o serviço vai ser mapeado das definições abstratas do perfil e do processo para informações concretas que deverão ser realizadas por meio de mensagens trocadas entre o consumidor e o provedor do serviço escolhido.

A maioria das pesquisas em OWL-S, assim como este trabalho, utilizam a tradicional arquitetura para Serviços Web SOA (apresentada na Seção 2.2.1). Porém, OWL-S não impõe o uso de arquiteturas orientadas a serviço e existem diversos trabalhos reportados na literatura que utilizam OWL-S e não são baseados na arquitetura SOA. Paolucci et al. [2003b] e Noia et al. [2003] utilizam a ontologia OWL-S em sistemas com arquitetura ponto-a-ponto (P2P). Masuoka et al. [2003] utilizam OWL-S em sistemas para dispositivos sem fio com o conjunto de protocolos UPnP (*Universal Plug and Play*). Ainda existem trabalhos que utilizam OWL-S em sistemas com mediadores centralizados [Martin et al., 1999; Uszok et al., 2004; Paolucci et al., 2004b;a].

A especificação de OWL-S não obriga a utilização de uma única forma de concretização do serviço. Entretanto, Martin et al. [2004a; 2007a] propõem a reutilização da infra-estrutura já existente para Serviços Web ao mapear os elementos abstratos de OWL-S para operações e mensagens concretas em WSDL da forma que é apresentada da Figura 2.9.

Mapear os elementos abstratos de OWL-S para os elementos concretos em WSDL, além de reutilizar toda a infra-estrutura já existente para Serviços Web, mantendo a máxima de Berners-Lee et al. [2001] de que a Web Semântica deve ser uma extensão da Web Atual, acarreta uma outra vantagem: situar os Serviços Web nas camadas da Web Semântica. Ao mapear elementos de ontologia, que corresponde à camada de descrição semântica e lógica da arquitetura da Web Semântica (Figura 2.1), para elementos XML e esquema XML do WSDL mantém-se a arquitetura em camadas da Web Semântica.

A ontologia de concretização de serviços de OWL-S apresentada na Figura 2.9 define um mapeamento para WSDL baseado em três elementos descritivos de OWL-S e WSDL. Primeiro, um processo atômico em OWL-S é mapeado para uma operação em WSDL (*wsdl:operation*), sendo que os processos compostos são sub-divididos em processos atômicos. Segundo, as entradas e saídas de cada processo atômico do serviço descrito em OWL-S são referenciadas pelas definições de mensagens de entrada e saída em WSDL (*wsdl:parts*). Terceiro, os tipos das entradas e saídas (conceitos em OWL-DL, por exemplo) em OWL-S correspondem ao conceito de tipos abstratos em WSDL. Por fim, as operações e mensagens em WSDL são concretizadas

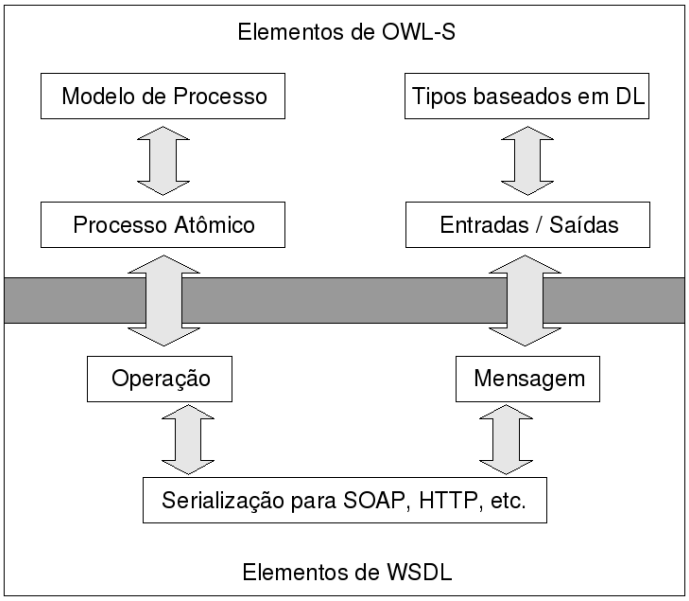

Figura 2.9: Visão geral do mapeamento entre OWL-S e WSDL. Adaptado de Martin et al. [2007a].

utilizando o protocolo SOAP que realiza a comunicação entre o consumidor e o provedor do serviço.

Uma característica importante dessa abordagem da Figura 2.9 é que o mapeamento entre OWL-S e WSDL pode ser realizado em tempo de execução do serviço. Isto permite que o provedor do serviço o realize de forma dinâmica, acabando com a necessidade de que o desenvolvedor (ou usuário) do código do serviço cliente ter que escolher o provedor do serviço em tempo de codificação do serviço. Ou seja, possibilitando que a composição de serviços seja automática, dinâmica e em tempo de execução.

Existem alguns trabalhos na direção da geração automática dos descritores OWL-S, incluindo geração automática das três sub-ontologias de OWL-S, a geração do WSDL correspondente ou o mapeamento de OWL-S para UDDI. A maioria deles utilizam transformações de documentos e foram desenvolvidos pelos proponentes [Martin et al., 2007a] do OWL-S. Outros trabalhos, desenvolvidos por outros grupos, são trabalhos na direção da modelagem de Serviços Web Semânticos por meio de diagramas UML e da transformação desses diagramas em descrições OWL-S. O Capítulo 6 apresenta, como uma das contribuições deste trabalho, uma forma de gerar um esqueleto de um Serviço Web na arquitetura MVC a partir da transformação de documentos descritores em OWL-S.

2.2.3 Outras linguagens semânticas para Serviços Web

Conforme apresentado na Seção 2.2.2, este trabalho utiliza a ontologia OWL-S para a descrição semântica de Serviços Web. Porém, existem outras abordagens com o objetivo de unir Serviços Web e Web Semântica. Nas seções seguintes são apresentados os principais trabalhos relacionados ao OWL-S que envolvem a combinação de tecnologias e padrões da Web Semântica para a descrição de Serviços Web.

SWSF

Battle et al. [2005] apresentam uma proposta de um framework para Serviços Web Semânticos chamado de SWSF (*Semantic Web Services Framework*), que é baseado no OWL-S e na linguagem de modelagem de processos PSL (*Process Specification Language*) [Grüninger and Menzel, 2003].

O SWSF especifica uma linguagem denominada SWSL (*Semantic Web Services Language*) que define a ontologia SWSO (*Semantic Web Services Ontology*). SWSL é uma linguagem orientada para a Web que é composta por uma camada com uma linguagem de programação lógica e uma camada com lógica de primeira ordem. A ontologia SWSO é uma ontologia de conceitos de serviços que utiliza a linguagem SWSL para a definição dos conceitos e todo os conceitos da ontologia são definidos baseando-se apenas em lógica de primeira ordem, diferente de OWL-S que é baseado em lógica de descrição e lógica de primeira ordem.

WSMO

O WSMO (*Web Services Modeling Ontology*) [Bruijn et al., 2005; Fensel et al., 2006] compartilha a mesma visão do OWL-S e do SWSF de que ontologias são essenciais para a automatização de tarefas como descoberta, composição e interoperabilidade de Serviços Web. Assim como o SWSF, o WSMO define uma linguagem (WSML – *Web Service Modeling Language*) [Bruijn et al., 2006] expressiva e orientada para a Web que é baseada em lógica de descrição e lógica de primeira ordem.

Da mesma forma que o OWL-S, o WSMO declara entradas, saídas, pré-condições e resultados (embora com terminologias diferentes) associados aos serviços. Uma diferença com o OWL-S é que o WSMO não provê uma notação para construção de processos compostos (*composite process* em OWL-S), visto que WSMO usa uma abordagem baseada em máquinas de estado.

WSDL-S e SAWSDL

Akkiraju et al. [2003; 2005] e Farrell and Lausen [2007] propõem a adição de

semântica como extensão a documentos WSDL. A primeira proposta de adição de anotações semânticas ao WSDL submetida ao consórcio W3C foi o WSDL-S (*Web Service Semantics*) [Akkiraju et al., 2003; 2005]. Posteriormente, uma outra proposta, denominada SAWSDL (*Semantic Annotations for WSDL and XML Schema*), com os mesmos propósitos que o WSDL-S e, inclusive com alguns autores em comum, foi submetida ao consórcio W3C[Farrell and Lausen, 2007].

Ambas as propostas permitem que as anotações semânticas sejam associadas a elementos WSDL (operações, entradas, saídas, tipos de dados e interfaces). A forma com que o WSDL-S e SAWSDL permitem especificar a correspondência entre elementos WSDL e conceitos semânticos é muito similar à proposta de concretização do serviço na sub-ontologia de concretização do serviço (ver Figura 2.9) de OWL-S.

A principal diferença entre as duas propostas e a sub-ontologia de concretização do serviço do OWL-S é que nessas propostas o mapeamento entre os elementos WSDL e os conceitos semânticos deve ser executado na própria especificação WSDL, enquanto que em OWL-S isto é realizado em um documento OWL à parte.

2.3 Considerações finais

A pesquisa abordada no presente trabalho está situada na relação entre Web Semântica e Serviços Web: os chamados Serviços Web Semânticos. Por esse motivo, este capítulo teve o objetivo de fornecer os conceitos, padrões e tecnologias relativos aos Serviços Web Semânticos e necessários para o entendimento deste trabalho.

A descrição de Serviços Web utilizando os padrões e ontologias da Web Semântica, descritos na Seção 2.1, e utilizando a ontologia OWL-S para descrição de Serviços Web, descrita na Seção 2.2, está apresentada no Capítulo 3 sob a forma de descrição de experimentos remotos (motivação inicial deste trabalho) como Serviços Web Semânticos.

O OWL-S provê as descrições semânticas necessárias para a automação de tarefas tais como descoberta, composição e invocação de Serviços Web. Este trabalho apresenta contribuições que utilizam as descrições semânticas de OWL-S nas áreas de descoberta e composição automáticas da forma que são apresentadas nos capítulos 4, 5 e 6.

Em relação à descoberta automática, este trabalho apresenta e propõe, no Capítulo 4, um algoritmo para descoberta de Serviços Web Semânticos. Já em relação à composição automática, o Capítulo 5 apresenta: um modelo que mapeia o problema de composição automática para um grafo direcionado e com custos; políticas de custos adotadas para a realização da composição automática; e um algoritmo baseado em cálculo de caminhos de custo mínimo que é usado para composição automática de Serviços Web Semânticos.

No presente trabalho foi desenvolvida e implementada uma infra-estrutura para implantação de Serviços Web Semânticos capaz de realizar publicação de serviços a partir de suas descrições semânticas e efetuar descoberta e composição automáticas, infra-estrutura essa apresentada no Capítulo 6.

CAPÍTULO

3

Modelagem de Experimentos Remotos

Durante a fase inicial das pesquisas relacionadas ao desenvolvimento deste trabalho, o autor buscava soluções para integrar experimentos remotos em ambientes de aprendizado eletrônico. O trabalho estava inserido em um projeto[1] mais amplo de desenvolvimento de um ambiente para aprendizado eletrônico (LMS[2] - *Learning Management System*) que incluía outras ferramentas já integradas ao ambiente como, por exemplo, ferramentas de comunicação síncrona baseadas em troca de mensagens de texto (ferramenta *chat*), de áudio e vídeo (ferramenta *instant messenger*) e de tinta eletrônica [FAPESP, 2005].

Naquela fase foi identificada a oportunidade de realizar a integração de experimentos com atividades de aprendizagem num contexto mais específico, tal como uma disciplina, ou um curso, ou como uma atividade independente de experimentação remota que deveria ser realizada por um grupo de alunos. A proposta inicial da integração tinha como característica principal o uso de esquemas XML para a descrição da sintaxe das configurações envolvidas na experimentação remota e do uso de transformações de documentos para prover a configuração de forma dinâmica e extensível [Teixeira et al., 2005; Prazeres et al., 2005; Prazeres and Teixeira, 2006].

Entretanto, a integração de aplicações de experimentos remotos ao ambiente

[1] Projeto TIDIA-Ae (Tecnologia da Informação no Desenvolvimento da Internet Avançada – Aprendizado Eletrônico). O objetivo do projeto é o de auxiliar as atividades de aprendizado eletrônico e oferecer apoio ao ensino presencial.

[2] LMS são sistemas de gerenciamento de aprendizado e corresponde a um conjunto de ferramentas integradas com finalidades educacionais.

Web (LMS) apresentava características peculiares em comparação com as outras ferramentas, pois os recursos disponibilizados para acesso remoto poderiam possuir restrições de acesso no tocante à disponibilidade e à utilização desses recursos necessários aos experimentos.

Muitos experimentos necessitavam de tratamentos pré-execução e pós-execução, tais como reserva prévia de equipamentos e disponibilização dos resultados ao final da execução. Diante desse cenário, o autor propôs investigar o uso das tecnologias e padrões dos Serviços Web na descrição, integração e disponibilização dos experimentos remotos, de modo a explorar a característica de Serviços Web de prover interoperabilidade entre sistemas.

Nesse sentido, foram levantadas as tendências [Martin et al., 2004b] relativas ao o uso de tecnologias da Web Semântica na descrição dos Serviços Web, os Serviços Web Semânticos apresentados no Capítulo 2 deste trabalho.

Ao pesquisar sobre Serviços Web Semânticos, o autor verificou que era possível modelar Serviços Web descrevendo suas entradas e saídas, e suas pré-condições e resultados, com a utilização da ontologia de Serviços Web OWL-S. Assim, Prazeres et al. [2007] propuseram descrever experimentos remotos utilizando marcações semânticas para Serviços Web (OWL-S). A proposta permite descrever equipamentos, insumos, restrições e resultados, envolvidos no uso dos experimentos remotos, na forma de entradas, saídas, pré-condições e resultados em OWL-S e em ontologias para descrição de conceitos temporais e de recursos. Disponibilizar experimentos remotos na forma de Serviços Web Semânticos possibilita a integração com outros experimentos remotos, bem como com ambientes Web, tal como LMS, por meio da descoberta e composição automáticas dos serviços.

Este capítulo apresenta, na Seção 3.1, a definição de experimentos remotos e outros termos envolvidos nesse domínio. A Seção 3.2 apresenta a proposta inicial de integração de experimentos remotos em ambientes de aprendizagem eletrônica por meio da descrição sintática dos experimentos remotos e suas configurações. A descrição e modelagem semântica de experimentos remotos como Serviços Web Semânticos são apresentados na Seção 3.3. Por fim, as seções 3.5 e 3.6 apresentam, respectivamente, os trabalhos relacionados aos resultados discutidos neste capítulo e as considerações finais referentes a este capítulo.

3.1 Experimentos remotos

Redes com largura de banda da ordem de dezenas de *gigabits/s* e baixa latência possibilitam interação com boa qualidade entre usuários e aplicações remotas. Assim, muitos laboratórios têm disponibilizado seus experimentos para serem acessados por usuários remotos. Esses laboratórios são, na literatura da área, chamados de

WebLabs [Ross et al., 1997] se os experimentos remotos e seus recursos associados podem ser acessados e controlados via Web.

Ferreira and Muller [2004] definem experimento remoto como uma atividade na qual uma pessoa (ou um grupo de pessoas) utiliza uma rede de comunicação para realizar algum tipo de atividade em um laboratório. Experimentos remotos, em muitos casos, fornecem acesso remoto a recursos raros ou de custo elevado. Recursos, neste trabalho, são coleções de um ou mais recursos que podem ser disponibilizados para acesso remoto no contexto de experimentos remotos: equipamentos, software, produtos químicos, animais vivos, etc. Um recurso ainda pode ser uma pessoa, pertencente a um determinado WebLab, que fornece apoio à preparação e à execução de um experimento remoto.

Segundo Harward et al. [2008], experimentos remotos podem ser classificados com base na arquitetura de software necessária para tornar um experimento acessível remotamente, na configuração dos parâmetros e no tipo de interação com o usuário remoto:

- Experimentos do tipo ***Batch***: todos os parâmetros que norteiam a execução do experimento são especificados pelo usuário antes do experimento ser iniciado. A sessão de experimentação consiste apenas em submeter os parâmetros, executar o experimento e retornar e os resultados para análise;
- Experimentos do tipo **Sensor**: geralmente nenhum parâmetro é especificado. O usuário apenas pode obter fluxo de dados de um determinado sensor;
- Experimentos do tipo **Interativo**: o usuário configura uma série de parâmetros iniciais, inicia o experimento e monitora o curso de sua execução, modificando parâmetros se necessário.

Harward et al. [2008] destacam que experimento interativo é uma superclasse que inclui as outras duas e envolvem todos os problemas relacionados à disponibilização remota dos experimentos. Assim, no presente trabalho, os experimentos são tratados como pertencente ao tipo interativo. Isto significa que o tratamento dado à superclasse experimento interativo vai abranger também as outras duas classes.

No domínio educacional, uma diversificada gama de experimentos remotos têm sido propostos para complementar as experiências presenciais bem como para aprimorar a educação (e.g. [Toderick et al., 2005], [Zin and Harun, 2007], [Karadimas and Efstathiou, 2007], [Paladini et al., 2008]).

Alguns autores ([Fjeldly, 2003], [Ferreira and Muller, 2004]) defendem ainda que um experimento remoto não deve ser simplesmente uma réplica de um experimento presencial. De fato, uma abordagem não substitui a outra, mas ambas devem ter o mesmo objetivo: facilitar o aprendizado.

Teixeira et al. [2005] argumentam que a experimentação remota agrega novos objetivos educacionais àqueles estabelecidos para experimentação presencial. De uma maneira geral, é aceito que a observação e o controle remoto dos eventos que ocorrem durante a realização de um experimento exercem múltiplas funções, dentre elas:

- a capacitação do estudante para operar o controle de fenômenos de interesse através de sistemas informatizados e redes;
- a familiarização do estudante aos recursos comumente oferecidos por esses sistemas como, por exemplo, o tratamento dos dados em tempo real, que fornece informações relevantes para a tomada de decisões durante a execução do experimento;
- a minimização do efeito de variáveis intervenientes, possivelmente introduzidas inadvertidamente pelo estudante no experimento, se realizado presencialmente;
- a explicitação das relações entre as variáveis envolvidas no fenômeno de interesse, dado que variáveis não planejadas para o experimento não podem ser deliberadamente manipuladas pelo estudante como meio alternativo para produzir os resultados esperados.

O processo de disponibilizar experimentos e seus recursos agregados por meio da Web deve levar em consideração que cada ator envolvido na experimentação remota precisa ter acesso, de acordo com o seu papel no processo, às informações relativas ao WebLab e seus experimentos. Diversos atores podem ser elencados e isso depende do proponente do experimento remoto. Todavia, os principais papéis envolvidos no processo podem ser atribuídos a três atores: o “administrador de WebLab”, o “proponente de experimento” e o “executor de experimento”.

O administrador de WebLab possui funções, como o próprio nome sugere, administrativas no gerenciamento do WebLab, experimentos e recursos. Esse ator tem acesso ao gerenciamento de informações relativas à descrição do WebLab e do cadastro, edição e exclusão de experimentos e recursos.

O proponente de experimento é o ator que deve conhecer a fundo cada experimento que será disponibilizado. Ele deve prover informações cruciais para a execução do experimento, tais como configurações iniciais, roteiro de execução, tempo máximo de cada sessão de experimentação, dentre outros. Esse ator pode ainda, a depender do contexto em que se insere o experimento, escolher as pessoas que devem executar o experimento, definir quando o experimento deve ser executado, etc.

O executor de experimento é o usuário final do experimento. É quem vai buscar o acesso aos WebLabs, seus experimentos e recursos, com a finalidade de executar um experimento previamente cadastrado e configurado no ambiente.

Considerando que existem muitas tarefas que devem ser associadas a cada um desses atores, a abordagem de integração de experimentos remotos a ambientes de aprendizagem, apresentada na Seção 3.2, possibilita a automatização das tarefas e permite a reutilização das especificações resultantes.

Para disponibilizar os experimentos, os WebLabs devem produzir informações a respeito do próprio WebLab (descrição dos laboratórios), bem como informações sobre configuração e agendamento de experimentos. Importantes pesquisas em experimentação remota têm dedicado esforços para resolver problemas técnicos no tocante a oferecer acesso remoto aos experimentos (e.g. [Toderick et al., 2005], [Zin and Harun, 2007], [Karadimas and Efstathiou, 2007], [Paladini et al., 2008], dentre outros).

Entretanto, muitas das informações necessárias para oferecer um experimento de forma remota são comuns aos experimentos, ou a grande parte deles: são informações sobre configuração e agendamento. Cada WebLab produz uma solução *ad hoc* para prover as informações de configuração e agendamento e isso dificulta a integração dos experimentos a outros ambientes que não seja o ambiente para o qual foi projetado.

Com o intuito de prover informações de configuração e agendamento de experimentos remotos de uma forma padrão e extensível, a Seção 3.2 apresenta uma abordagem baseada em fluxo e transformações de documentos cujo objetivo final é a geração de documentos de configuração e de agendamento de experimentos remotos.

3.2 Integração com LMS: descrição sintática de experimentos

A abordagem para integração de experimentos remotos em LMS apresentada nesta seção é extensível e independente do domínio, embora tenha sido criada e implementada com o objetivo de integrar experimentos remotos a ambientes de aprendizagem [Prazeres and Teixeira, 2006]. Essa abordagem tem como característica principal a utilização de fluxo de documentos que realizam transformações XSLT[3] em esquemas XML para prover descrição sintática para configuração de experimentos remotos.

Considerando a variedade de WebLabs e experimentos que podem ser definidos, a abordagem proposta foi implementada para facilitar a extensão e customização de acordo com novos requisitos. Isto foi possível por meio do uso intensivo de esquemas XML para estruturar a informação não apenas *a priori*, mas também *on-the-fly*, para acomodar as características únicas de cada WebLab e de seus experimentos.

A especificação *on-the-fly* baseada em documentos foi alcançada com o uso siste-

[3]Do inglês *Extensible Stylesheet Language Transformations*, XSLT é uma linguagem baseada em XML utilizada para realizar transformações de documentos XML em outros documentos XML ou em documentos para leitura por pessoas.

mático de transformações XSLT que permitem a geração dos documentos requeridos para a apresentação. O processamento *on-the-fly* de documentos foi implementado por meio de fluxos de processamento de documentos que envolveram especificações em esquemas XML, transformações XSLT e documentos de apresentação baseados em formulários (os fluxos são ilustrados nas Figuras 3.1, 3.3, 3.5 e 3.7).

Todos os documentos gerados devem ser enviados para o WebLab correspondente, pois algumas atividades devem ser executadas por membros do WebLab, garantindo, por exemplo, que o agendamento do experimento seja executado conforme o planejado.

Baseada em esquemas XML e transformações XSLT, a abordagem descrita nas seções 3.2.1 a 3.2.4 apresenta exemplos das possíveis especificações demandadas para configuração de experimentos, com o objetivo de ilustrar a utilização da abordagem proposta. Em outras palavras, a definição de uma recomendação formal e completa com respeito à configuração de WebLabs e experimentos requer a discussão de especialistas da área.

A abordagem assume que as interações ocorrem em um ambiente Web que provê autenticação de usuários e definição de permissão e papéis. Assim, cada fluxo de documentos apresentados nas Seções 3.2.1 a 3.2.4 deve ser associado ao usuário adequado.

3.2.1 Descrição de WebLabs

Uma característica importante da abordagem apresentada nesta seção é que, embora tenha sido designada para, de forma interativa, facilitar o uso do sistema pelo usuário final (administrador de WebLab, proponente de experimentos, executor de experimentos, etc.), tanto a abordagem quanto a implementação permitem que documentos de especificação sejam adicionados manualmente e diretamente no fluxo de documentos e, uma vez validados, seguem o fluxo normal do sistema. Isto significa que é possível a reutilização de especificações criadas por outros WebLabs e para outros experimentos. Essa possibilidade é explicitada no fluxo de documentos no ponto em que permite a inclusão de um documento de especificação como uma alternativa ao preenchimento dos formulários de configuração (essa opção é indicada pelo multiplexador presente no fluxo de documentos – ver Figuras 3.1, 3.3, 3.5 e 3.7).

A Figura 3.1 apresenta o primeiro fluxo de documentos o qual pode ser associado, por exemplo, a um usuário autenticado com papel de Administrador de WebLab.

O fluxo inicia com o processamento de uma transformação de documento que recebe como entrada um esquema XML correspondente às informações comuns na descrição de WebLabs. A transformação gera um formulário que é apresentado ao administrador do WebLab, o qual deve preencher com informações referentes ao seu WebLab. Uma vez que o formulário tenha sido preenchido, as informações são

validadas para garantir a conformidade com os tipos de dados e a estrutura do esquema XML de entrada do fluxo. O administrador usa o mesmo formulário, de forma interativa, até que todos os dados sejam validados. Na Figura 3.1:

- o esquema XML indicado por *WLC* corresponde à definição do conceito de WebLab;
- a transformação de documentos indicada por *WLC2FORM* gera um formulário de entrada de dados baseado no esquema *WLC*;
- o formulário apresentado ao administrador do WebLab é indicado por *WLDF* (formulário para descrição de WebLab);
- o preenchimento do formulário pelo usuário é indicado por *MEP* (processo de edição manual);
- a instância do documento produzido como resultado da edição é indicado por *WLD* (descrição de WebLab);
- a validação da informação provida pelo usuário é indicada por *VC* (verificador de validação);
- a provisão de uma instância de documento alternativa, possibilitando reutilização, é indicada por *IWLD* (importação do *WLD*);
- a seleção entre o documento IWLD ou a informação provida manualmente (via o preenchimento dos formulários MEP) é indicada por MUX (multiplexador).

O esquema XML apresentado no Documento 1 exemplifica o tipo de informação demandada para o registro de cada WebLab (*WLC* na Figura 3.1). O processamento desse documento com uma transformação apropriada (*WLC2FORM* na Figura 3.1) gera o formulário apresentado na Figura 3.2 (*WLDF* na Figura 3.1) e esse formulário é preenchido pelo adminitrador do WebLab (*MEP* na Figura 3.1).

A linhas 15, 17 e 19 do esquema WLC apresentado no Documento 1 correspondem, respectivamente, aos campos do tipo texto *Nome do WebLab*, *URL* e *Descrição* que foram gerados na transformação *WLC2FORM*.

3.2.2 Especificação de experimentos

Diversos experimentos podem ser associados a um WebLab. Além disso, um único experimento, com seus recursos associados, pode possibilitar diversas configurações de experimentos. Esta seção apresenta os fluxos de documentos (ilustrado na Figura 3.3) para especificação de experimentos, enquanto que a Seção 3.2.3 apresenta as possíveis configurações que podem ser efetuadas em um experimento.

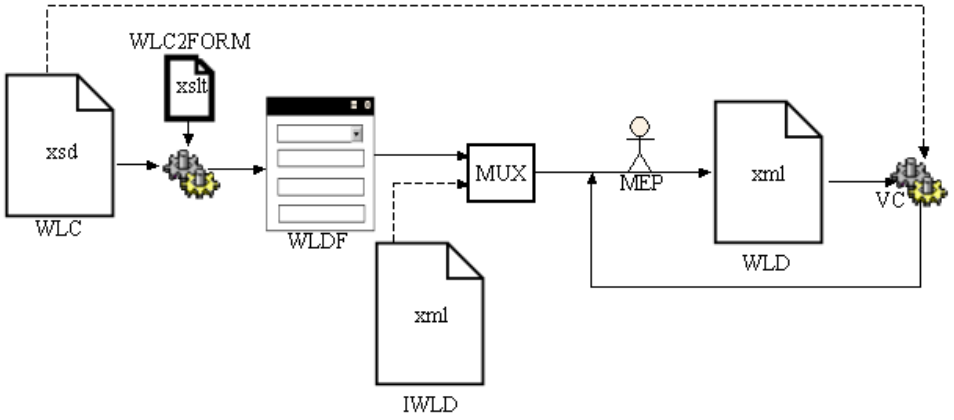

Figura 3.1: Fluxo de documentos para descrição de WebLabs. *Esquema WLC*: definição do conceito WebLab. *WLC2FORM*: especificação da transformação de WLC para um formulário WLDF. *WLDF*: Formulário para descrição de WebLab. *MUX (multiplexador)*: seletor de múltiplas entradas. *WLD*: Descrição XML do WebLab. *MEP*: Processo de Edição Manual executado por usuário autenticado. *IWLD*: WLD Importado. *VC*: verificador de validação.

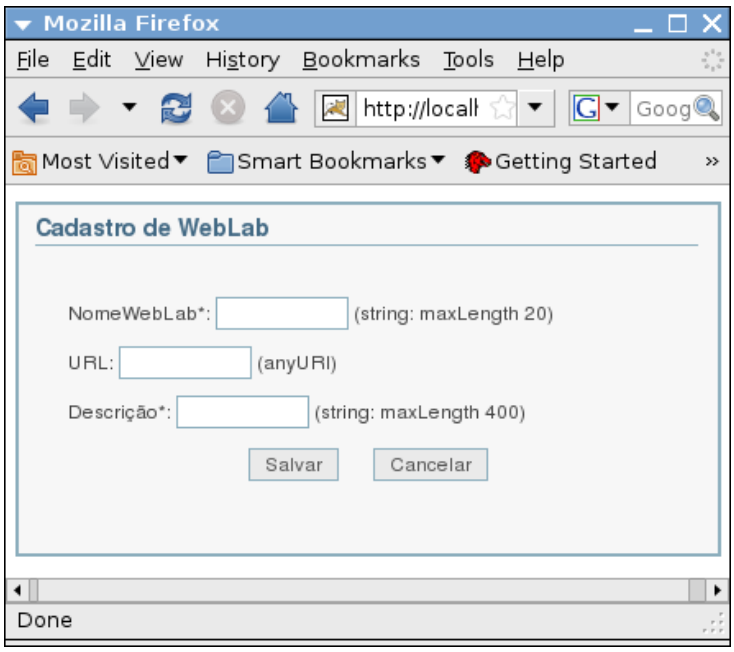

Figura 3.2: Exemplo de formulário para descrição de WebLab (WLDF).

Uma vez que o projetista de um experimento tenha definido todos os detalhes e informações relativos aos recursos que serão disponibilizados para tornar um determinado experimento remotamente acessível, essas informações devem ser registradas e validadas, e corresponderá à especificação do experimento. A especificação é utilizada para gerar documentos que serão apresentados para os proponentes e para

Documento 1 Segmento do esquema XML WLC.

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:simpleType name="nameType">
4      <xsd:restriction base="xsd:string">
5        <xsd:maxLength value="20" />
6      </xsd:restriction>
7    </xsd:simpleType>
8    <xsd:simpleType name="descriptionType">
9      <xsd:restriction base="xsd:string">
10       <xsd:maxLength value="400" />
11     </xsd:restriction>
12   </xsd:simpleType>
13   <xsd:complexType name="webladType">
14     <xsd:sequence>
15       <xsd:element name="name" type="nameType"
16                    minOccurs="1" maxOccurs="1"/>
17       <xsd:element name="url" type="urlType"
18                    minOccurs="1" maxOccurs="1"/>
19       <xsd:element name="description"
20                    type="descriptionType" minOccurs="1" maxOccurs="1"/>
21     </xsd:sequence>
22     <xsd:attribute name="WebLabId" type="xsd:ID" use="required" />
23   </xsd:complexType>
24   <xsd:element name="WebLab" type="webladType" />
25 </xsd:schema>
```

os executores de experimentos.

As informações contidas na especificação de um experimento podem ser divididas em duas partes: informações comuns a maioria dos experimentos (metadados como nome do experimento, tipo, autor, etc.) e informações específicas para cada experimento. As informações específicas incluem a lista dos recursos associados ao experimento, os parâmetros (e seus possíveis valores) associados a cada recurso e, ainda, valores padrões ou valores iniciais para cada parâmetro.

A Figura 3.3 apresenta um fluxo de documentos para cada tipo de informação de especificação de experimentos. Os atores que aparecem na Figura 3.3 são administradores do WebLab que especificam tanto as informações comuns quanto as informações específicas ao experimento.

As informações comuns aos experimentos são tratadas no fluxo de documentos apresentado na parte superior da Figura 3.3. O formulário *TGCF* é utilizado para definir as propriedades comuns de um experimento. Esse formulário é gerado automaticamente por meio do processamento do esquema *TGC* com a transformação *TGC2FORM*. Após o preenchimento desse formulário, dois documentos são gerados automaticamente. O primeiro é o documento *TD*, que contém informações relativas ao experimento (nome do experimento, tipo, autor, etc.). O segundo é o *TGCP*, que

corresponde a um esquema XML com informações que serão propagadas para um outro fluxo de documentos, e que é utilizado pelo proponente e pelo executor do experimento. O *TGCP* inclui informações de utilização do experimento, tal como informações de agendamento (e.g. tempo de disponibilidade do experimento na rede).

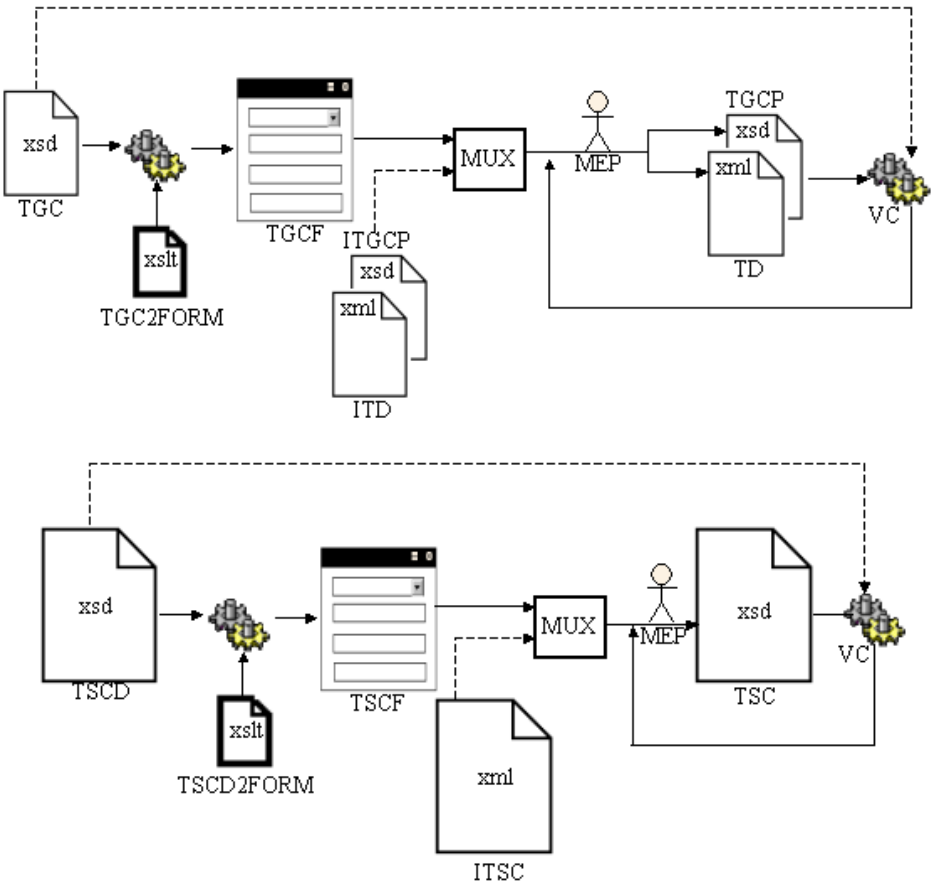

Figura 3.3: Fluxo de documentos para especificação de experimentos: informações comuns (parte superior) e informações específicas (parte inferior).
(parte superior) *Esquema TGC*: configurações comuns de experimentos. *TGCF*: formulário para configurações comuns de experimentos. *TGC2FORM*: especificação da transformação de TGC para formulário TGCF. *Esquema TGCP*: propagação dos conceitos comuns de experimentos. *ITD*: descritor de experimento importado. *ITGCP*: TGCP importado. *Instância TD*: descritor de experimento. *MEP*: processo de edição manual. *MUX (multiplexador)*: seletor de múltiplas entradas. *VC*: verificador de validação.
(parte inferior) *Esquema TSCD*: definição de conceitos específico de experimentos. *TSCF*: formulário para configuração específica de experimentos. *TSCD2FORM*: especificação da transformação de TSCD para formulário TSCF. *Esquema TSC*: conceitos específicos de experimentos. *ITSC*: TSC importado. *MEP*: processo de edição manual. *MUX (multiplexador)*: seletor de múltiplas entradas. *VC*: verificador de validação.

As informações específicas aos experimentos são tratadas no fluxo de documentos

apresentado na parte inferior da Figura 3.3. O formulário *TSCF* é gerado automaticamente após o processamento do esquema *TSCD* por meio da transformação *TSCD2FORM*. Após o preenchimento desse formulário, dois documentos são gerados automaticamente. O primeiro é o documento *TSC* que, combinado com o *TGCP* gerado no fluxo das configurações comuns (parte superior da Figura 3.3), especifica a gramática do documento que será utilizado pelo proponente do experimento para especificar um experimento em particular (Seção 3.2.3). O *TSC*, combinado com o *TGCP*, é também utilizado para a definição do documento que o executor do experimento utilizará para especificar as informações sobre agendamento e outras informações específicas (Seção 3.2.4).

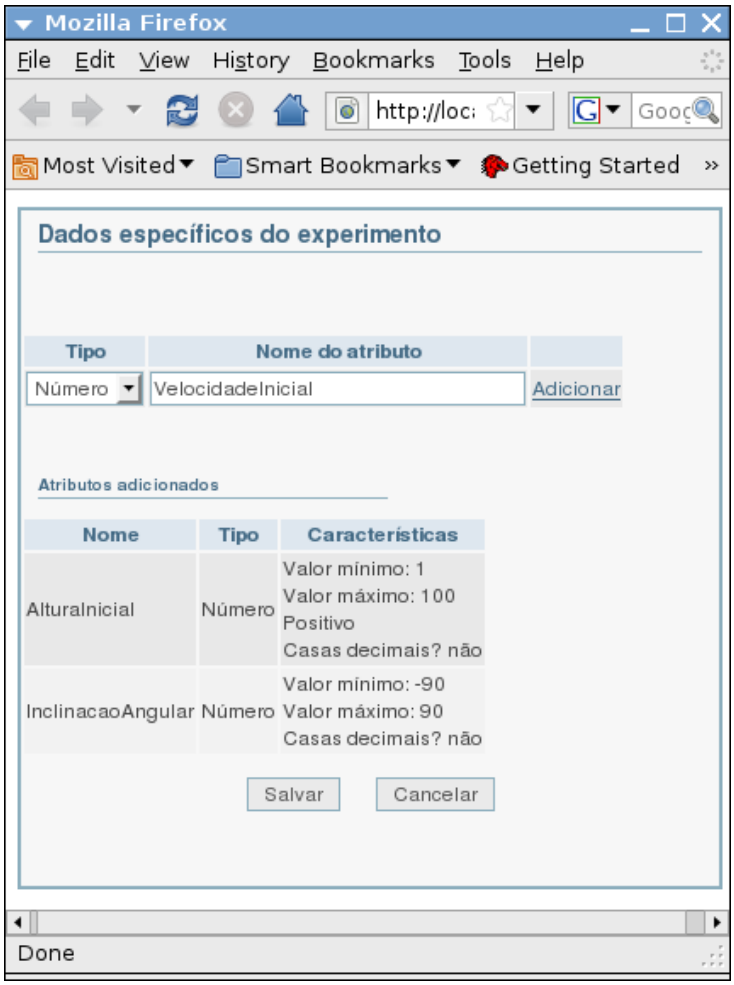

Figura 3.4: Exemplo de formulário de configuração específica para um experimento de lançamento de projétil (TSCF).

A Figura 3.4 ilustra uma instância do formulário *TSCF* em que um administrador de WebLab especifica os atributos e possíveis valores relativos a um experimento de lançamento de projéteis. Na figura, os atributos *AlturaInicial* e *InclinacaoAngular*

foram definidos, junto com o tipo e valores que os atributos podem assumir, e o usuário está definindo mais um atributo denominado *VelocidadeInicial*. A principal idéia ilustrada na Figura 3.4 é a possibilidade de esconder do usuário a complexidade de se definir um esquema XML.

3.2.3 Configuração de experimentos

Um proponente de experimento pode especificar quais parâmetros serão configurados em um dado experimento e, ainda, quais valores eles devem assumir inicialmente, gerando uma configuração daquele experimento que pode ser proposta para execução por uma pessoa ou grupo de pessoas (executores de experimento). Essa configuração está ilustrada no fluxo de documentos da Figura 3.5 por meio do formulário *ECPF*. Os esquemas gerados como resultado do fluxo de documentos da Figura 3.3 (*TGCP* e *TSC*) são utilizados: para gerar o formulário *ECPF* (Figura 3.5); e na verificação dos documentos gerados como resultado dos dados providos pelo proponente do experimento, o esquema EECC e o documento PEC (ambos no fluxo de documentos da Figura 3.5).

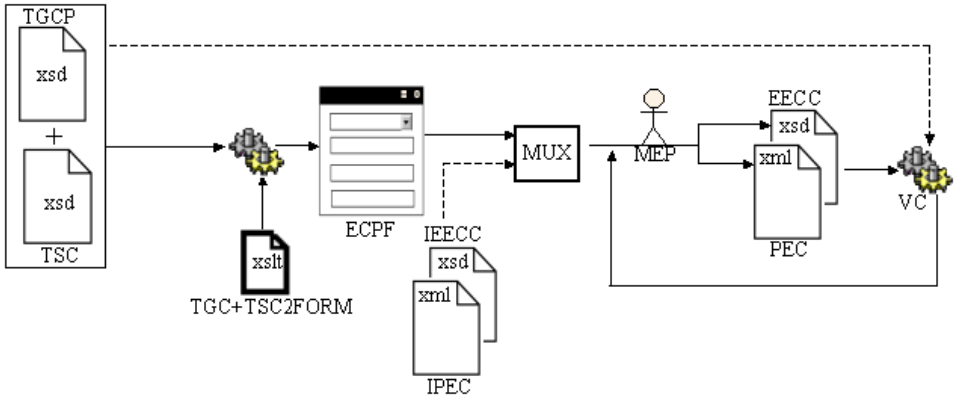

Figura 3.5: Fluxo de documentos para definir valores dos parâmetros de experimentos. *Esquema TGCP*: propagação dos conceitos comuns de experimentos. *Esquema TSC*: conceitos específicos de experimentos. *ECPF*: formulário para configuração de experimento pelo proponente. *TGC+TSC2FORM*: especificação da transformação de TGF para formulário TGCF. *Esquema IEECC*: importação de conceito de configuração de experimento pelo executor. *IPEC*: importação da configuração de experimento pelo proponente. *EECC*: conceito de configuração de experimento pelo executor. *Instância PEC*: configuração de experimento pelo proponente. *MEP*: processo de edição manual. *MUX (multiplexador)*: seletor de múltiplas entradas. *VC*: verificador de validação.

O documento PEC registra a configuração efetuada pelo proponente do experimento. O esquema EECC é utilizado para a geração do formulário a ser apresentado ao executor do experimento, conforme detalhado na Seção 3.2.4.

O Documento 2 apresenta uma parte do esquema XML (*TSC*) correspondente às configurações do experimento de lançamento de projétil, e a Figura 3.6 ilustra o documento de apresentação (formulário *ECPF*) gerado por meio de transformação XSLT.

Documento 2 Segmento do esquema XML para TSC.

```
1  <xsd:complexType name="InitialHeightType">
2    <xsd:sequence>
3      <xsd:element name="InitialHeightValue"
4                   minOccurs="1" maxOccurs="1">
5        <xsd:simpleType>
6          <xsd:restriction base="xsd:positiveInteger">
7            <xsd:minInclusive value="1" />
8            <xsd:maxInclusive value="100" />
9          </xsd:restriction>
10       </xsd:simpleType>
11     </xsd:element>
12   </xsd:sequence>
13 </xsd:complexType>
14 <xsd:complexType name="AngleInclinationType">
15   <xsd:sequence>
16     <xsd:element name="AngleInclinationValue"
17                  minOccurs="1" maxOccurs="1">
18       <xsd:simpleType>
19         <xsd:restriction base="xsd:integer">
20           <xsd:minInclusive value="-90" />
21           <xsd:maxInclusive value="90" />
22         </xsd:restriction>
23       </xsd:simpleType>
24     </xsd:element>
25   </xsd:sequence>
26 </xsd:complexType>
27 <xsd:complexType name="attributesType">
28   <xsd:sequence>
29     <xsd:element name="InitialHeight" type="InitialHeightType"
30                  minOccurs="1" maxOccurs="1" />
31     <xsd:element name="AngleInclination" type="AngleInclinationType"
32                  inOccurs="1" maxOccurs="1" />
33   </xsd:sequence>
34 </xsd:complexType>
```

3.2.4 Configuração de reservas de experimentos

Um executor do experimento pode executar um experimento proposto ou efetuar o agendamento necessário à execução do experimento, definindo um *slot* de tempo em que ele quer executar o referido experimento. O executor pode ainda ter a oportunidade de reconfigurar alguns parâmetros propostos na configuração do experimento, desde que não viole a configuração inicial definida pelo proponente do

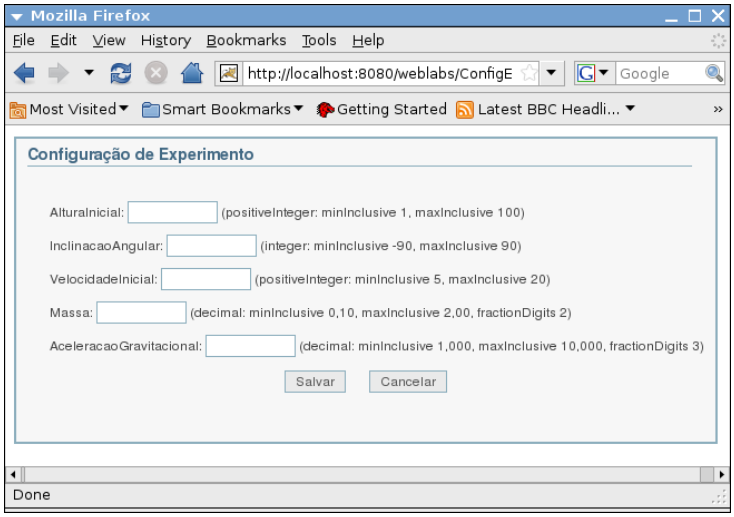

Figura 3.6: Exemplo de formulário para configuração de experimento pelo proponente (ECPF): gerado automaticamente para a configuração de um experimento.

experimento.

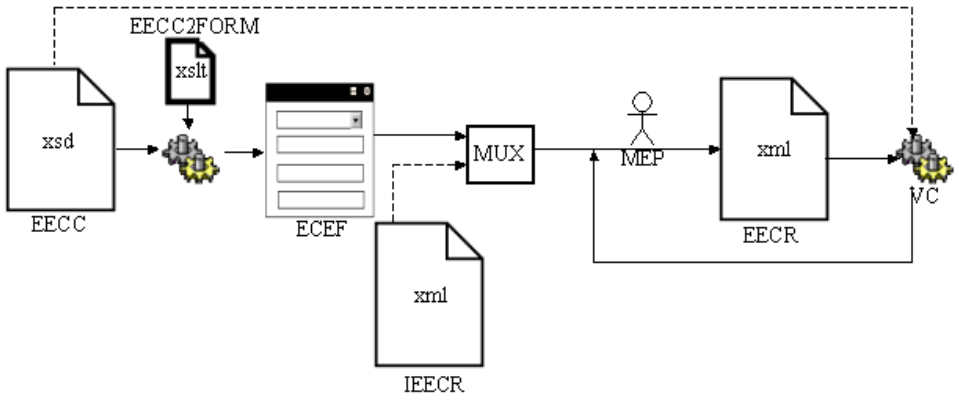

Figura 3.7: Fluxo de documentos para reserva e complementação de configuração de experimento. *Esquema EECC*: conceito de configuração de experimento pelo executor. *ECEF*: formulário para configuração de experimento pelo executor. *EECC2FORM*: especificação da transformação de EECC para formulário ECEF. *EECR*: configuração e reserva do experimento pelo executor. *IEECR*: importação da configuração e reserva do experimento pelo executor. *MEP*: processo de edição manual. *MUX (multiplexador)*: seletor de múltiplas entradas. *VC*: verificador de validação.

A Figura 3.7 ilustra que o esquema *EECC*, gerado como resultado da configuração de um experimento por um proponente de experimento, é utilizado para gerar um formulário *ECEF*, para configuração do experimento pelo executor de forma automática por meio da transformação *EECC2FORM*. Uma vez que o executor do experimento tenha escolhido todas as possíveis alternativas de configuração, um

documento EECR válido, para configuração e reserva, é produzido.

3.2.5 Implementação no ambiente TIDIA-Ae

A abordagem para descrição e integração de experimentos em LMS apresentada nesta seção foi utilizada para integrar e disponibilizar alguns WebLabs no contexto do projeto TIDIA-Ae. Um exemplo da integração [Prazeres et al., 2006b] é o WebLab LaDABio[4] (*Laboratory for the Development and Automation of Bioprocesses*).

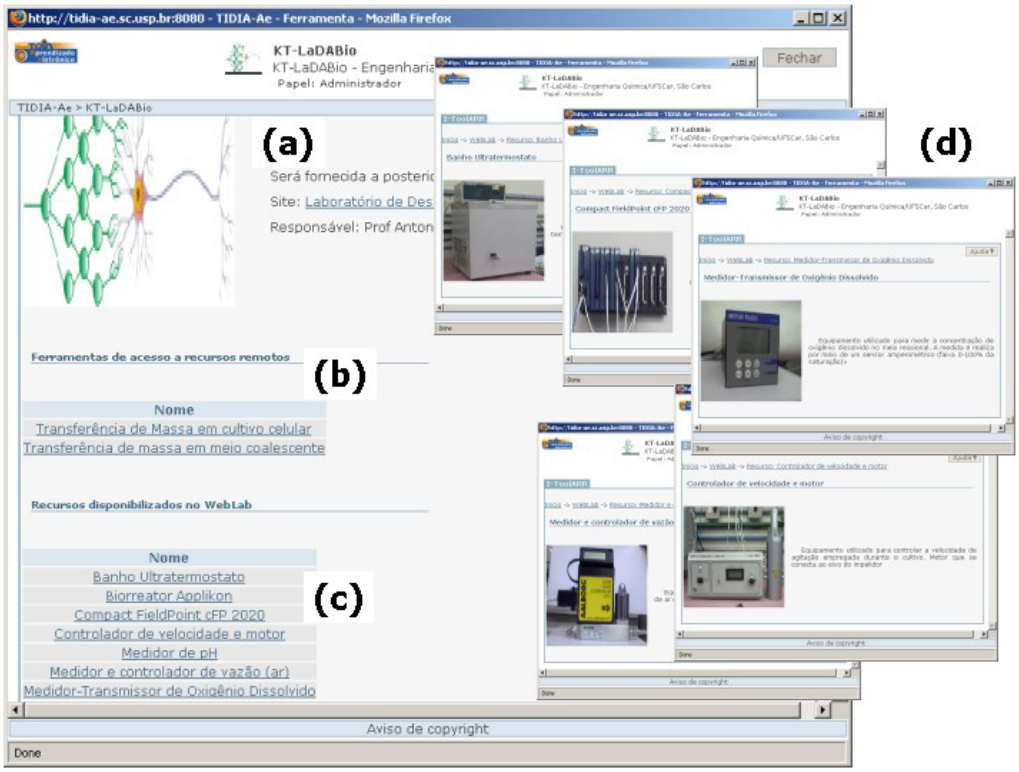

Figura 3.8: WebLab LaDABio integrado ao ambiente TIDIA-Ae. Obtido de Prazeres et al. [2006b]

A Figura 3.8 ilustra a integração do WebLab LaDABio ao ambiente TIDIA-Ae. Na figura, são apresentadas informações: (a) sobre registro do WebLab; (b) sobre cada experimento disponibilizado pelo WebLab junto com suas especificações de propriedades, parâmetros e configurações; (c) sobre cada recurso disponibilizado pelo WebLab e que compõe os experimentos; (d) sobre os detalhes de cada recurso disponibilizado.

A partir do uso dos fluxos de documentos apresentados nesta seção foi possível a integração do WebLab LaDABio, bem como de seus experimentos, recursos e também a configuração de períodos de disponibilidade para cada recurso, possibilitando

[4]http://ladabio.deq.ufscar.br/kyatera.html

o agendamento de experimentos. O gerenciamento do acesso e dos usuários foi executado pelo ambiente TIDIA-Ae, que já fornecia ferramentas para essa finalidade. Todavia, o gerenciamento de toda a parte de configuração do experimento bem como o agendamento foi realizado por meio do uso dos fluxos de documentos aqui descritos.

3.3 RALOWS: descrição semântica de experimentos remotos

Apesar de a descrição sintática dos experimentos remotos e de a integração com um ambiente LMS terem sido satisfatórias, algumas limitações podem ser observadas e a principal delas é que a integração é manual e fixa. Embora o fluxo de documentos apresentado na Seção 3.2 possibilite a integração dos experimentos, essa integração demanda uma série de configurações pré-fixadas para todo e qualquer experimento que venha a ser integrado ao ambiente. Os experimentos devem ser conhecidos *a priori*, inseridos no ambiente e só então podem ser utilizados.

Uma solução natural para a integração de aplicações Web são os Serviços Web, cujo principal objetivo é o de prover interoperabilidade. Entretanto, os Serviços Web também precisam de configurações prévias para cada serviço que for integrado à uma aplicação Web. Ou seja, a integração é efetuada em tempo de codificação do serviço. Disponibilizar experimentos remotos na forma de Serviços Web Semânticos possibilita a integração com outros experimentos remotos, assim como com ambientes Web, por meio da descoberta, invocação e composição automáticas dos serviços.

Experimentos remotos são uma categoria de aplicações que impõem diversas e rígidas restrições no que diz respeito à disponibilidade dos recursos. O autor propôs a plataforma RALOWS, do inglês *Remote Access Laboratory Ontology and Web Service* (Ontologia para laboratório de acesso remoto e Serviços Web), para investigação de como experimentos remotos podem ser disponibilizados como Serviços Web Semânticos.

A plataforma RALOWS [Prazeres et al., 2007] é uma abordagem proposta neste trabalho que reutiliza e estende ontologias conhecidas para prover descrição formal de experimentos remotos. Essa abordagem apresenta uma combinação de conceitos temporais e descrição de recursos e serviços que permitem a especificação de informações de agendamento, bem como prover restrições no uso dos recursos envolvidos em uma experimentação remota.

A plataforma RALOWS tira proveito das características da ontologia OWL-S para descrever experimentos remotos na forma de Serviços Web. A ontologia de tempo (*OWL-Time*) é utilizada para definir as restrições temporais associadas aos recursos e serviços providos. A ontologia de recursos (*Resource Ontology*) é utilizada para descrever os recursos que são disponibilizados por meio dos experimentos remotos.

A Figura 3.9 ilustra a ontologia RALOWS que reutiliza como ontologia de nível

superior a ontologia OWL-S (classes *Service*, *Process*, *Grounding*, *Profile*, *Output* e *Input*) com as extensões da ontologia de recursos (classe *Resource*) e da ontologia de tempo (classes *DateTimeInterval* e *ProperInterval*), assim como as classes próprias da RALOWS (*InputResource*, *OutputResource* e *Instrument*) e as propriedades também de RALOWS incluídas para prover as extensões (*availableTime*, *readyTime*, *hasInputResource* e *hasOutputResource*).

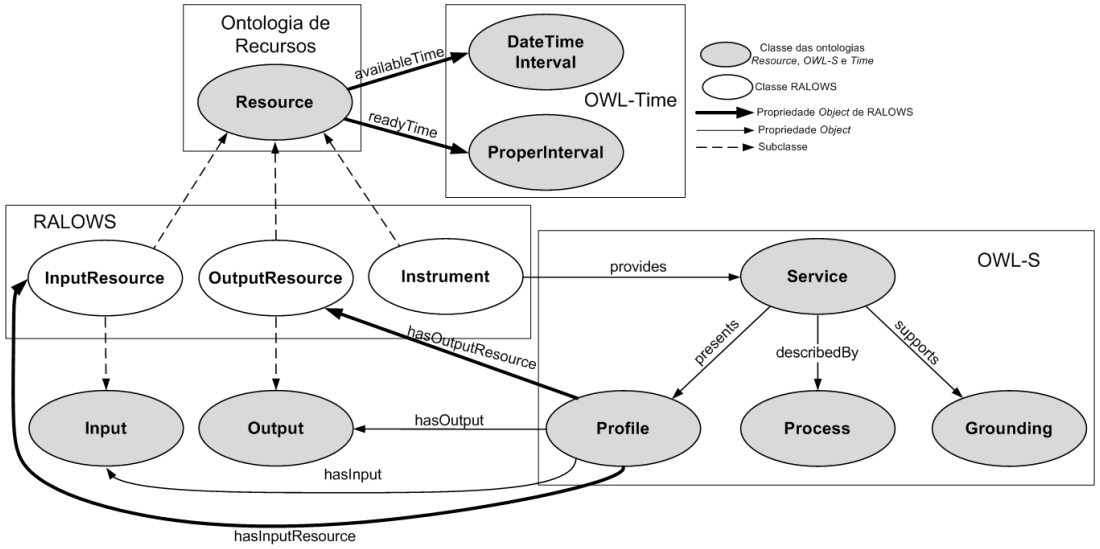

Figura 3.9: Ontologia RALOWS: Ontologia para laboratório de acesso remoto e Serviços Web.

As ontologias de recurso e de tempo utilizadas na RALOWS estão propostas em conjunto com a especificação de OWL-S para descrição de recursos e conceitos temporais, respectivamente, envolvidos no uso de Serviços Web. É relevante observar que na versão 1.2 [Martin et al., 2007a] do OWL-S, a ontologia de tempo é utilizada, de maneira limitada, na especificação de processos, e a ontologia de recursos ainda não é utilizada. Martin et al. [2007a] planejam que nas próximas versões as ontologias de recurso e de tempo sejam utilizadas plenamente na especificação do perfil e dos processos do serviço. Dessa forma, a proposta de RALOWS de integrar essas três ontologias na descrição de serviços que envolvem experimentação remota vai em direção aos objetivos dos proponentes do OWL-S.

3.3.1 Recursos

A propriedade *provides* (ver Figura 3.9) de RALOWS indica que um recurso pode prover um serviço. No âmbito de experimentos remotos, três tipos de recursos são considerados como detalhado na Figura 3.9: *InputResource* (recursos de entrada), *OutputResource* (recursos de saída) e *Instrument* (instrumento). Esses tipos de

recursos são extensões definidas na ontologia de recursos para prover descrição de experimentos remotos, conforme é apresentado no Documento 3, no qual as linhas 6 a 9 apresentam a extensão para *InputResource*, as linhas 10 a 13 apresentam a extensão para *OutputResource* e as linhas 14 a 17 para *Instrument*.

Documento 3 Tipos de recurso da ontologia RALOWS.

```
1  <!-- A resource defined in RALOWS -->
2
3  <owl:Class rdf:about="&resource;Resource">
4   <rdfs:label>Resource</rdfs:label>
5  </owl:Class>
6  <owl:Class rdf:about="&resource;InputResource">
7   <rdfs:label>Input Resource</rdfs:label>
8   <rdfs:subClassOf rdf:resource="&resource;Resource"/>
9  </owl:Class>
10 <owl:Class rdf:about="&resource;OutputResource">
11  <rdfs:label>Output Resource</rdfs:label>
12  <rdfs:subClassOf rdf:resource="&resource;Resource"/>
13 </owl:Class>
14 <owl:Class rdf:about="&resource;Instrument">
15  <rdfs:label>Instrument Resource</rdfs:label>
16  <rdfs:subClassOf rdf:resource="&resource;Resource"/>
17 </owl:Class>
```

Um instrumento é um tipo de recurso que pode ser acessado e controlado por meio da Web para prover algum serviço (e.g. um osciloscópio). Um recurso definido na ontologia de recursos pode ser um recurso atômico ou um recurso agregado, tal como apresentado na Figura 2.4 da Seção 2.1.3 (*AtomicResource* e *AggregateResource*). Com isto, um experimento remoto pode ser modelado como um único recurso que provê um serviço ou como um conjunto de recursos agregados.

Na plataforma RALOWS um instrumento ou agregações de instrumentos são empacotados como um Serviço Web Semântico em OWL-S. Assim, considerando que todo instrumento provê um serviço, cada instrumento: apresenta um perfil de serviço; é descrito por um processo; e precisa ser concretizado de alguma maneira, tal como é na ontologia OWL-S e conforme é ilustrado na Figura 3.9. O Documento 4 apresenta a propriedade *provides* da classe *Instrument* que indica mais uma extensão proposta em RALOWS para descrever que um instrumento provê um serviço.

Os outros tipos de recursos definidos em RALOWS são os recursos que atuam como entrada e saída de experimentos remotos. Alguns exemplos de recurso de entrada são: uma solução química demandada em um experimento de Química; um técnico necessário para supervisionar a execução do experimento; um rato utilizado em experimentos da psicologia, dentre outros. Exemplos de recursos de saída são recursos de entrada que sofreram alguma alteração após a execução do experimento, ou mesmo novos recursos originados da execução do experimento: uma solução

Documento 4 Um instrumento em RALOWS provê algum tipo de serviço.

```
1  <!-- An Instrument provides a service -->
2
3  <owl:ObjectProperty rdf:ID="&resource;provides">
4   <rdfs:domain rdf:resource="&resource;Instrument"/>
5   <rdfs:range rdf:resource="&service;Service"/>
6  </owl:ObjectProperty>
```

química que sofreu alguma reação química; um rato que teve seu comportamento alterado após um experimento de psicologia comportamental, dentre outros.

Os recursos de entrada (classe *InputResource*) e de saída (classe *OutputResource*) são ambos subclasses de recurso (classe *Resource*). O que difere um recurso de entrada de um recurso de saída em RALOWS é que o primeiro é subclasse de entrada (classe *Input* de OWL-S) e o segundo é subclasse de saída (classe *Output* de OWL-S), conforme pode ser observado na Figura 3.9.

Conforme apresentado na Figura 3.9 e detalhado no Documento 5, um recurso de entrada é uma subclasse de *Input* (linhas 3 a 6 do Documento 5) que é a classe de todas as entradas em OWL-S. Essa classe especial de entrada foi criada por motivos de descoberta de experimentos e pelo fato de que alguns desses recursos podem conter restrições temporais bem como demandar agendamento prévio para a sua utilização.

A Figura 3.9 também apresenta a propriedade *hasInputResource* do perfil (*Profile*) do serviço para a classe de recursos de entrada (*InputResource*). Essa propriedade, tal como definida no Documento 5 (linhas 7 a 11), é uma extensão, na forma de sub-propriedade, da propriedade *hasInput* do OWL-S.

Ainda na Figura 3.9 e no Documento 5, um recurso de saída é uma subclasse de *Output* (linhas 13 a 16 do Documento 5) que é a classe de todas as saídas em OWL-S. Essa classe especial de saída foi criada estritamente por motivos de descoberta de experimentos, visto que a descoberta em OWL-S é baseada na transformação das entradas em saídas.

A Figura 3.9 também apresenta a propriedade *hasOutputResource* do perfil (*Profile*) do serviço para a classe de recursos de saída (*OutputResource*). Essa propriedade, tal como definida no Documento 5 (linhas 17 a 21), é uma extensão, na forma de sub-propriedade, da propriedade *hasOutput* do OWL-S.

3.3.2 Entradas, saídas, pré-condições e resultados

As entradas, saídas, pré-condições e resultados (IOPR) de um instrumento são especificados no perfil do serviço em OWL-S. Cada instrumento tem seu próprio conjunto de IOPRs, que é intrínseco ao instrumento. Ainda, as IOPRs de um experimento contêm as IOPRs de todos os instrumentos envolvidos. Os IOPRs específicos

Documento 5 Definição de recursos de entrada e saída em RALOWS.

```
1  <!-- InputResource and OutputResource are extensions as a subclass
        of Input and Input, and hasInputResource and hasOutputResource
        are extensions as a sub-property of hasInput and hasOutput -->
2
3  <owl:Class rdf:about="&resource;InputResource">
4   <rdfs:label>Input Resource</rdfs:label>
5   <rdfs:subClassOf rdf:resource="&process;#Input"/>
6  </owl:Class>
7  <owl:ObjectProperty rdf:ID="&profile;hasInputResource">
8   <rdfs:subPropertyOf rdf:resource="&process;#hasInput"/>
9   <rdfs:domain rdf:resource="&profile;Profile"/>
10  <rdfs:range rdf:resource="&resource;InputResource"/>
11 </owl:ObjectProperty>
12
13 <owl:Class rdf:about="&resource;OutputResource">
14  <rdfs:label>Output Resource</rdfs:label>
15  <rdfs:subClassOf rdf:resource="&process;#Output"/>
16 </owl:Class>
17 <owl:ObjectProperty rdf:ID="&profile;hasOutputResource">
18  <rdfs:subPropertyOf rdf:resource="&process;#hasOutput"/>
19  <rdfs:domain rdf:resource="&profile;Profile"/>
20  <rdfs:range rdf:resource="&resource;OutputResource"/>
21 </owl:ObjectProperty>
```

de cada instrumento e experimento correspondem às configurações específicas de experimentos (esquema *TSCD* da Figura 3.3) apresentadas na Seção 3.2.2.

Porém, alguns IOPRs podem ser comuns à maioria dos experimentos e correspondem às configurações comuns a experimentos (esquema *TSCD* da Figura 3.3) apresentadas na Seção 3.2.2. São exemplos desses IOPRs:

- entradas: *UserName_In*, *Password_In* e *IntervalEvent_In*;
- saídas: *Results_Out* e *ExperimentLog_Out*;
- pré-condições: *UserExist*, *ScheduledIntervalEventExist* e *IsValidIntervalEvent*;
- resultados: *ExperimentExecuted*, *ResultsGenerated* e *ScheduledIntervalEventInvalided*.

Os experimentos podem apresentar ainda recursos que são consumidos pelo experimento e são descritos em RALOWS (na Seção 3.3.1) como recursos de entrada e saída (apresentados na Figura 3.9 e definidos nos Documentos 3 e 5 como *InputResource* e *OutputResource*).

No perfil do serviço também é especificada a categoria do experimento, por exemplo: experimentos de química, experimentos de física, experimentos biológicos, experimentos da psicologia. O perfil do serviço em OWL-S possui uma propriedade

opcional (*serviceCategory*) cujo valor pode ser definido por uma ontologia que descreve categoria de serviços.

Empacotar instrumentos como serviços em OWL-S permite automatizar a descoberta de experimentos que são compostos por esses instrumentos. O perfil é utilizado para anunciar os experimentos com o propósito de descoberta. O Capítulo 4 apresenta um algoritmo de descoberta baseado na categoria do serviço, entradas e saídas, além de levar em consideração as restrições temporais associadas ao serviço.

3.3.3 Intervalos de disponibilidade e agendamento

Na ontologia RALOWS são utilizadas, a princípio, duas classes da ontologia de tempo (*OWL-Time*) descrita na Seção 2.1.3: *DateTimeInterval* e *ProperInterval*. De fato, a própria classe *DateTimeInterval* é uma subclasse de *ProperInterval*. Esses conceitos temporais são associados a recursos através das propriedades *availableTime* e *readyTime* como ilustrado na Figura 3.9.

A propriedade *availableTime* descreve intervalos de tempo em que um recurso está disponível para execução ou para agendamento de execução. A propriedade *readyTime* descreve um intervalo de tempo, entre duas execuções seguidas do mesmo experimento, necessário para algum tipo de preparação prévia dos recursos envolvidos no experimento ou, ainda, para a substituição de recursos consumidos na experimentação anterior por novos recursos.

Os detalhes envolvendo as restrições de tempo na descrição e na disponibilização de experimentos como Serviços Web Semânticos, bem como um algoritmo para descoberta de serviços que leva em consideração as restrições temporais são descritos no Capítulo 4.

3.4 RALOWS: análise usando um estudo de caso

Esta seção apresenta uma instância de RALOWS no experimento denominado de "câmara de condicionamento operante" (também conhecida por Caixa de Skinner) [Skinner, 1959]. Esse experimento foi utilizado como base para análise e avaliação da plataforma RALOWS [Prazeres et al., 2007]. A Seção 3.4.1 apresenta o experimento, explicando sua forma de execução e que objetivos e resultados são esperados. As Seções 3.4.2 e 3.4.3 apresentam, respectivamente, o experimento de Skinner descrito apenas por OWL-S, e o experimento descrito segundo a abordagem RALOWS.

3.4.1 Experimento Caixa de Skinner com ratos

A variação do experimento Caixa de Skinner descrita[5] objetiva condicionar um rato dentro de uma Caixa de Skinner a pressionar uma barra e liberar água. O rato é previamente privado de água por um determinado período de tempo a ponto de deixá-lo sedento. A Caixa de Skinner, ilustrada na Figura 3.10, tem uma barra e um dispensador de água que libera gotículas de água quando acionado. O dispensador pode ser acionado externamente à caixa, por um executor de experimento, ou internamente à caixa, pelo rato pressionando a barra.

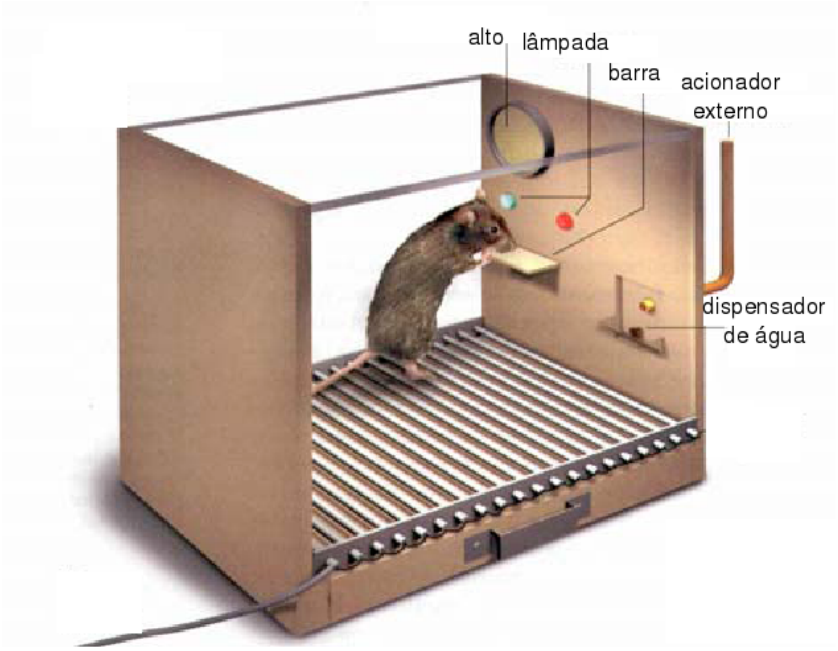

Figura 3.10: Uma Caixa de Skinner.

A execução do experimento ocorre com o executor do experimento empregando a técnica de modelagem de respostas por aproximações sucessivas para ensinar a resposta de pressão à barra ao rato. Essa técnica exige que o experimentador considere o conjunto das respostas emitidas pelo rato (os diversos movimentos que ele faz dentro da caixa) e escolha, em diferentes momentos, aquelas que devem ser seguidas pelo acionamento do dispensador de água. O acionamento do dispensador é controlado pelo experimentador na etapa inicial do experimento. Posteriormente, a critério do experimentador, pode ser acionado sempre que a barra de respostas fosse pressionada pelo rato [Teixeira et al., 2005].

Uma outra variação comum desse experimento é inserir outras variáveis ao ambiente (Caixa de Skinner), tais como uma lâmpada que pode ou não ser acesa pelo experimentador, uma argola que pode ser instalada no interior da caixa, de tal forma

[5]Para esse exemplo não foram modelados os reforços negativos, tal como choque elétrico.

que, por exemplo, o experimentador só libere a água quando o rato tiver determinado comportamento e a lâmpada estiver acesa.

Para executar o experimento remotamente, uma Caixa de Skinner foi automatizada e conectada a um computador e uma aplicação contendo uma janela de vídeo e três botões conforme ilustrado na Figura 3.11 [Capobianco et al., 2005]. A janela de vídeo mostra a imagem capturada de uma câmera colocada de frente para a Caixa de Skinner automatizada. Um dos botões aciona o dispensador de água, o outro acende ou apaga a lâmpada e o terceiro emite um som que pode ser usado, por exemplo, para acordar o rato caso ele durma durante o experimento.

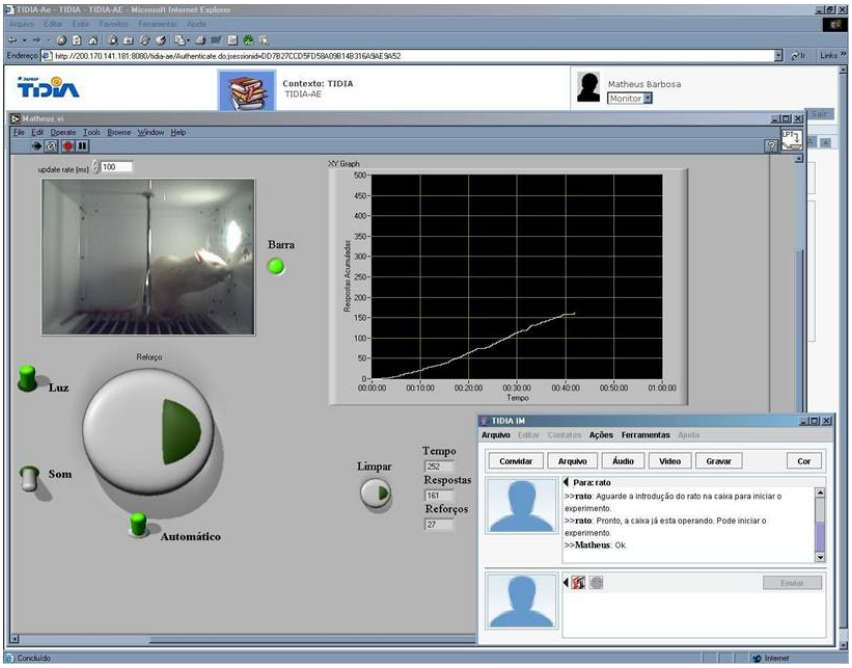

Figura 3.11: Caixa de Skinner automatizada.

3.4.2 Experimento Caixa de Skinner descrito com OWL-S

Esta seção apresenta, por meio dos Documentos 6, 7 e 8, a descrição do experimento Caixa de Skinner por OWL-S e sem as extensões propostas na plataforma RALOWS.

O Documento 6 descreve as entradas, saídas, pré-condições e resultados (ou efeitos) do experimento sob a forma de IOPR no perfil do serviço. No documento a única pré-condição é que o rato esteja privado do uso de água.

O Documento 7 contém o principal processo do experimento (*SkinnerBoxProcess*). Esse processo é composto por uma seqüência (linha 5) de dois outros processos. O primeiro é um processo atômico chamado de *SignIn* (linha 7), que define que o executor do experimento deve ser um usuário validado no sistema. O segundo

Documento 6 Descrição do perfil e dos IOPRs em OWL-S.

```
1  <!-- Descriptions of IOPRs in the Profile -->
2  <!-- Inputs -->
3  <profile:hasInput rdf:resource="&sb_process;#Light_In"/>
4  <profile:hasInput rdf:resource="&sb_process;#Sound_In"/>
5  <profile:hasInput rdf:resource="&sb_process;#Water_In"/>
6  <!-- Preconditions -->
7  <profile:hasPrecondition rdf:resource="&sb_process;#RatIsWaterPrivated"/>
8  <!-- Effects -->
9  <profile:hasEffect rdf:resource="&sb_process;#ExperimentExecuted"/>
10 <profile:hasEffect rdf:resource="&sb_process;#ResultsGenerated"/>
11 <!-- Outputs -->
12 <profile:hasOutput rdf:resource="&sb_process;#PressureBar_Out"/>
13 <profile:hasOutput rdf:resource="&ba_process;#Results_Out"/>
14 <profile:hasOutput rdf:resource="&ba_process;#ExperimentLog_Out"/>
```

é um processo composto chamado *ExecuteExperiment* (linha 8), que é descrito no Documento 8.

Documento 7 Descrição do processo composto *SkinnerBoxProcess* em OWL-S.

```
1  <!-- SkinnerBoxProcess is a composite process.
   It is composed of a sequence whose components are
   1 atomic processes, SignIn and a composite process,
   ExecuteExperiment. -->
2
3  <process:CompositeProcess rdf:ID="SkinnerBoxProcess">
4   <process:composedOf>
5    <process:Sequence>
6     <process:components rdf:parseType="Collection">
7      <process:AtomicProcess rdf:about="#SignIn"/>
8      <process:CompositeProcess rdf:about="#ExecuteExperiment"/>
9     </process:components>
10   </process:Sequence>
11  </process:composedOf>
12 </process:CompositeProcess>
```

O Documento 8 apresenta a descrição do processo composto *ExecuteExperiment*. Na linha 3 foi definida uma condição (*Condition* em OWL-S). Esse processo é composto de quatro processos atômicos (linhas 10 a 13). Para compor o processo foram utilizados dois construtores do OWL-S: *Repeat-While* e *Choice*. O construtor *Repeat-While* especifica uma interação com uma condição de parada (*UserNotExit* que está definida na linha 3). Por sua vez, o construtor *Choice* indica que apenas um dos processos entre os listados deve ser escolhido por vez. Em outras palavras, o experimento permite ao executor acender ou desligar a lâmpada, liberar água, sendo que apenas uma ação por vez é permitida.

Documento 8 Descrição do processo composto *ExecuteExperiment* em OWL-S.

```
1  <!-- ExecuteExperiment is a composite process.
   It is composed of a choice whose components are
   4 atomic processes, TurnOnLight, TurnOffLight,
   PlaySound and GiveWater. -->
2
3  <process:Condition rdf:ID="UserNotExit"/>
4
5  <process:CompositeProcess rdf:ID="ExecuteExperiment">
6   <process:composedOf>
7    <process:Repeat-While
      whileCondition="#UserNotExit">
8     <process:Choice>
9      <process:components rdf:parseType="Collection">
10      <process:AtomicProcess rdf:about="#TurnOnLight"/>
11      <process:AtomicProcess rdf:about="#TurnOffLight"/>
12      <process:AtomicProcess rdf:about="#PlaySound"/>
13      <process:AtomicProcess rdf:about="#GiveWater"/>
14     </process:components>
15    </process:Choice>
16   </process:Repeat-While>
17  </process:composedOf>
18 </process:CompositeProcess>
```

3.4.3 Experimento Caixa de Skinner descrito com RALOWS

Esta seção apresenta, por meio dos Documentos 9, 10, 11, 12 e 13, a descrição do experimento Caixa de Skinner por OWL-S e com as extensões propostas na plataforma RALOWS.

A primeira extensão é mostrada no Documento 9, em que uma instância de *Instrument* é criada para descrever a Caixa de Skinner como um instrumento na plataforma RALOWS (um *Instrument* está ilustrado na Figura 3.9 e definido no Documento 3). O Documento 9 define três propriedades do tipo *availableTime* (linhas 4 a 6) para o instrumento "SkinnerBox". Essas propriedades incluem períodos de tempo (descritos pela ontologia *OWL-Time*) em que um dado recurso está disponível para uma experimentação remota. A linha 7 especifica que o instrumento (Caixa de Skinner) provê um serviço.

O Documento 10 especifica o rato como sendo um recurso de entrada (*InputResource*). Esse recurso não provê um serviço, pois é utilizado como entrada para o experimento e deve ser agendado, o que demanda a associação da propriedade *availableTime* (linha 4).

As entradas, saídas, pré-condições e resultados (ou efeitos) são descritos no Documento 11 sob a forma de IOPRs no perfil do serviço (experimento). Esse documento difere do Documento 6 por incluir: (a) uma nova entrada que faz referência

Documento 9 Um instrumento provê um serviço.

```
1 <!-- An instrument provides a service  -->
2
3  <resource:Instrument rdf:ID="SkinnerBox">
4   <resource:availableTime rdf:resource="&time;#Interval_1"/>
5   <resource:availableTime rdf:resource="&time;#Interval_2"/>
6   <resource:availableTime rdf:resource="&time;#Interval_3"/>
7   <resource:provides rdf:resource="&service;#SkinnerBoxExperiment"/>
8 </resource:Instrument>
```

Documento 10 Um recurso é uma entrada do serviço (experimento).

```
1 <!-- An resource is an input  -->
2
3 <resource:InputResource rdf:ID="Rat_in">
4  <resource:isAvailableBy rdf:resource="&time;#Interval_1"/>
5 </resource:InputResource>
```

para o *IntervalEvent* (linha 3) que deve ser agendado para a execução do experimento; (b) duas novas pré-condições especificando que deve existir um intervalo de tempo agendado para a execução do experimento (linha 10) e que o intervalo de tempo deve ser válido (linha 11); (c) um efeito *IntervalEventInvalided* (linha 15) explicitando que o intervalo de tempo deve ser tornado inválido após a execução do experimento; (d) um recurso definido como entrada (*hasInputResource* na linha 7); e (e) um recurso definido como saída (*hasOutputResource* na linha 20). Essas diferenças foram todas modeladas segundo as extensões propostas na plataforma RALOWS, utilizando a ontologia de tempo e a ontologia de recursos combinadas ao OWL-S.

Especificamente para o experimento de Caixa de Skinner, houve uma transformação em um recurso de entrada (*Rat_In*) para um recurso de saída (*Rat_Out*), ou seja, o rato que serviu de entrada para o experimento sofreu uma transformação, por meio da execução do experimento, e gerou uma nova saída. Em outras palavras, foi realizado o condicionamento do rato, que é a funcionalidade principal do experimento de Skinner. Essa transformação poderia ocorrer em outros experimentos, tal como em um experimento químico que, após uma reação química, realiza uma transformação nos recursos de entrada. É nessa transformação que se baseiam os algoritmos para descoberta em OWL-S e, portanto, a descoberta pode ser realizada com base nas funcionalidades dos serviços (experimentos).

Os Documentos 12 e 13 correspondem, respectivamente, aos Documentos 7 e 8 adaptados com as extensões propostas na plataforma RALOWS. No Documento 12, um novo processo atômico foi incluído: *SelectIntervalEvent* (linha 8). Esse processo testa se o usuário tem um *IntervalEvent* válido para executar o experimento.

No Documento 13, mudanças foram realizadas na condição de parada. No

Documento 11 Descrição do perfil e dos IOPRs em RALOWS.

```
1  <!-- Descriptions of IOPEs in the ServiceProfile -->
2  <!-- Inputs -->
3  <profile:hasInput rdf:resource="&sb_process;#IntervalEvent_In"/>
4  <profile:hasInput rdf:resource="&sb_process;#Light_In"/>
5  <profile:hasInput rdf:resource="&sb_process;#Sound_In"/>
6  <profile:hasInput rdf:resource="&sb_process;#Water_In"/>
7  <profile:hasInputResource rdf:resource="&sb_process;#Rat_In"/>
8  <!-- Preconditions -->
9  <profile:hasPrecondition rdf:resource="&sb_process;#RatIsWaterPrivated"/>
10 <profile:hasPrecondition rdf:resource="&sb_process;#IntervalEventExist"/>
11 <profile:hasPrecondition rdf:resource="&sb_process;#IsValidIntervalEvent"/>
12 <!-- Effects -->
13 <profile:hasEffect rdf:resource="&sb_process;#ExperimentExecuted"/>
14 <profile:hasEffect rdf:resource="&sb_process;#ResultsGenerated"/>
15 <profile:hasEffect rdf:resource="&sb_process;#IntervalEventInvalided"/>
16 <!-- Outputs -->
17 <profile:hasOutput rdf:resource="&sb_process;#PressureBar_Out"/>
18 <profile:hasOutput rdf:resource="&ba_process;#Results_Out"/>
19 <profile:hasOutput rdf:resource="&ba_process;#ExperimentLog_Out"/>
20 <profile:hasOutputResource rdf:resource="&sb_process;#Rat_Out"/>
```

Documento 8, a condição de parada ocorre quando o usuário espontaneamente sai da experimentação remota. Já no Documento 13 a condição de parada também ocorre quando o *IntervalEvent* agendado para o usuário expirar.

Ao utilizar OWL-S para descrever um experimento como um Serviço Web Semântico pode-se importar todos os aspectos relacionados à ontologia OWL-S para o domínio de experimentos remotos: descrição explícita de funcionalidades, descoberta automática, descrição do fluxo do serviço através dos processos, dentre outros. O fato de que RALOWS combina outras ontologias e adiciona conceitos temporais ao serviço e à descrição dos recursos torna possível realizar agendamento e impor restrições no uso dos recursos envolvidos na experimentação.

3.5 Trabalhos relacionados

Este capítulo apresentou resultados referentes a três esforços: configuração e descrição de experimentos remotos em ambientes Web (Seção 3.2); integração de experimentos remotos em sistemas de gerenciamento de aprendizado (LMS) na Web (Seção 3.2); e disponibilização de experimentos remotos descritos como Serviços Web Semânticos (seções 3.3 e 3.4). Esta seção apresenta comparativos do trabalho realizado com trabalhos reportados na literatura.

Bagnasco et al. [2002] propõem o uso de XML para descrever dados que dizem respeito a instrumentos, experimentos e atividades dos usuários do ambiente de

Documento 12 Descrição do processo composto *SkinnerBoxProcess* em RALOWS.

```
1  <!-- SkinnerBoxProcess is a composite process.
   It is composed of a sequence whose components are
   2 atomic processes, SignIn and SelectIntervalEvent,
   and a composite process, ExecuteExperiment. -->
2
3  <process:CompositeProcess rdf:ID="SkinnerBoxProcess">
4   <process:composedOf>
5    <process:Sequence>
6     <process:components rdf:parseType="Collection">
7      <process:AtomicProcess rdf:about="#SignIn"/>
8      <process:AtomicProcess rdf:about="#SelectIntervalEvent"/>
9      <process:CompositeProcess rdf:about="#ExecuteExperiment"/>
10    </process:components>
11   </process:Sequence>
12  </process:composedOf>
13 </process:CompositeProcess>
```

experimentos remotos. Os autores argumentam que sua abordagem possibilita escalabilidade, interoperabilidade e reutilização de informações e configurações. Os autores ainda sugerem a utilização de uma ferramenta de edição de documentos XML comercial (e externa à implementação proposta por eles) para a criação dos documentos XML de configuração dos experimentos.

As pesquisas de Bagnasco et al. [2002] são semelhantes ao trabalho reportado nesta tese na Seção 3.2, porém limita-se a apenas descrever o ambiente (experimentos, instrumentos, etc.) de experimentação remota utilizando documentos XML. Já a abordagem apresentada neste na Seção 3.2 utiliza esquemas XML para descrição de todo o ambiente de experimentação remota (WebLabs, recursos, experimentos, etc.). A escolha por esquemas XML permite ainda a validação dos dados de entrada para as configurações, fato que não ocorre na proposta de Bagnasco et al. [2002]. Uma outra diferença importante é o fato de que o trabalho reportado na Seção 3.2 utiliza fluxos de transformações XSLT para a geração automática das interfaces gráficas utilizadas para a configuração, atividade essa para a qual Bagnasco et al. [2002] sugerem a utilização de ferramentas para edição de documentos XML.

Agrawal and Srivastava [2007] apresentam uma arquitetura em três camadas e modelam os experimentos em termos de entradas, saídas e um conjunto de restrições. Os autores descrevem três papéis referentes a atores no processo de experimentação remota: administrador de WebLab, administrador de experimentos e aluno. Os WebLabs são modelados como um conjunto de experimentos que compartilham políticas de administração e acesso comuns, fornecendo um conjunto de ferramentas para registro de WebLabs e de experimentos que capturam os metadados do WebLab e dos experimentos com o mínimo de programação envolvida. Os autores também

Documento 13 Descrição do processo composto *ExecuteExperiment* em RALOWS.

```
1  <!-- ExecuteExperiment is a composite process.
   It is composed of a choice whose components are
   4 atomic processes, TurnOnLight, TurnOffLight,
   PlaySound and GiveWater. -->
2
3  <process:Condition rdf:ID="IntervalEventIsValid"/>
4
5  <process:CompositeProcess rdf:ID="ExecuteExperiment">
6   <process:composedOf>
7    <process:Repeat-While whileCondition="#IntervalEventIsValid">
8     <process:Choice>
9      <process:components rdf:parseType="Collection">
10      <process:AtomicProcess rdf:about="#TurnOnLight"/>
11      <process:AtomicProcess rdf:about="#TurnOffLight"/>
12      <process:AtomicProcess rdf:about="#PlaySound"/>
13      <process:AtomicProcess rdf:about="#GiveWater"/>
14     </process:components>
15    </process:Choice>
16   </process:Repeat-While>
17  </process:composedOf>
18 </process:CompositeProcess>
```

apresentam uma proposta para agendamento da execução do experimento.

A principal diferença entre a abordagem de Agrawal and Srivastava [2007] e o trabalho reportado na Seção 3.2 é que o foco de Agrawal and Srivastava [2007] é na arquitetura e na infra-estrutura de comunicação envolvidas na execução de um experimento do tipo *batch*. Já o foco do trabalho descrito na Seção 3.2 é na descrição do experimento para possibilitar sua integração a ambientes Web de aprendizagem. Entretanto, três resultados se assemelham nos dois trabalhos: definição de papéis de usuários, ferramentas para administração e agendamento comuns a todos os experimentos, e a utilização de XML na configuração dos parâmetros iniciais do experimento.

A principal limitação da abordagem reportada por Agrawal and Srivastava [2007] é que os experimentos gerenciados pela arquitetura de três camadas proposta são todos experimentos do tipo *batch*. Ou seja, o agendamento da execução dos experimentos é realizado por ordem de chegada das entradas por meio de uma fila implementada na arquitetura em três camadas, diferente do trabalho apresentado na Seção 3.2 que apresenta possibilidade de agendar também experimentos do tipo interativo. A configuração por meio de documentos XML apresentada por Agrawal and Srivastava [2007], assim como em Bagnasco et al. [2002], também não utiliza esquemas XML para validação e não possui interface para a criação das configurações (geradas de forma automática como na Seção 3.2).

As pesquisas de Ozdogru and Cagiltay [2007] e Sancristobal et al. [2008],

apresentam propostas para integração de experimentos remotos em LMS. Essas pesquisas possuem objetivos similares aos trabalhos apresentados nas Seções 3.2 e 3.3: incluir, da forma mais automatizada possível, atividades de experimentação remota em sistemas de gerenciamento de aprendizado na Web.

Ozdogru and Cagiltay [2007] apresentam problemas e soluções da integração de experimentos remotos no LMS *Moodle*[6]. Os autores sugerem a utilização do módulo *Scorm* do *Moodle* para efetuar a integração dos experimentos. A proposta é descrever as atividades de experimentação remota na forma de um documento XML *Scorm*, formato que pode ser adicionado diretamente ao LMS Moodle. Entretanto, os autores não apresentam uma arquitetura da implementação e integração de fato da proposta. As propostas apresentadas nas seções 3.2 e 3.3 são de propósito genérico, independente das características particulares de cada ambiente no qual os experimentos devam ser integrados.

Sancristobal et al. [2008] apresentam as vantagens da utilização de LMS para gerenciamento de experimentos remotos. Os autores sugerem que todas as funcionalidades de cadastro de WebLabs e de experimentos, das configurações do experimentos, da atribuição das atividades de experimentação remota sejam feitos via o LMS, pois isso permite a reutilização de todas as funcionalidades educacionais existentes no LMS. Como solução ao problema da integração, os autores sugerem a utilização do *iLabs* [Harward et al., 2008] como plataforma de disponibilização dos experimentos e a criação de um *middleware* para efetuar a integração do LMS ao *iLabs*. Sancristobal et al. [2008] deixam claro que sua proposta é um trabalho em progresso e que esse *middleware* para prover a integração de fato ainda será implementado.

A integração de experimentos remotos em LMS é uma das idéias centrais da abordagem apresentada na Seção 3.2 e também no trabalho de Sancristobal et al. [2008]. A Seção 3.2 apresenta toda a proposta da integração, que é baseada no fluxo de transformação de documentos XML, e que não está atrelada a uma plataforma de disponibilização de experimentos como a *iLabs*. Os experimentos podem ser definidos *ad hoc* e depois incluídos no LMS por meio das descrições em esquema XML.

As Seções 3.3 e 3.4 apresentam a plataforma RALOWS para descrição semântica de experimentos remotos na forma de Serviços Web Semânticos. No momento da redação desta tese, uma revisão na literatura a respeito da descrição e publicação de experimentos remotos como Serviços Web Semânticos não encontrou trabalhos. Alguns trabalhos ([Yan et al., 2005; 2006a;b; Harward et al., 2008; Wei-Feng et al., 2009]) descrevem esforços para a disponibilização de experimentos remotos na forma de Serviços Web tradicionais. Essas propostas, com exceção de Yan et al. [2006a], têm foco na infra-estrutura de comunicação necessária para disponibilizar

[6]http://moodle.org/

os experimentos remotos como Serviços Web, enquanto que os trabalhos descritos nas Seções 3.3 e 3.4 desta tese têm foco na configuração, descrição e integração dos dados envolvidos na experimentação remota.

Yan et al. [2006a] apresentam uma arquitetura baseada em SOA para disponibilização de experimentos remotos na forma de Serviços Web. Os autores utilizam os padrões WSDL, UDDI e SOAP na implementação da arquitetura. A exceção citada no parágrafo anterior é devido ao fato de que os os autores também abordam a descrição e integração dos dados envolvidos na experimentação remota. Os experimentos são descritos por meio do WSDL, que provê três tipos de informação: parâmetros de entrada/saída necessários à operação dos instrumentos; informações sobre a interface gráfica de acesso aos instrumentos; e metadados a respeito do instrumento. A descrição de experimentos por meio do WSDL não provê semântica, enquanto que os trabalhos descritos nas Seções 3.3 e 3.4 deste trabalho apresentam uma plataforma para descrição semântica dos experimentos remotos por meio dos Serviços Web Semânticos.

3.6 Considerações finais

Este capítulo apresentou os primeiros resultados obtidos neste trabalho. Esses resultados abordaram o domínio específico de experimentos remotos dada a motivação inicial de integração de experimentos remotos de propósitos variados a ambientes de gerenciamento de aprendizado eletrônico (LMS).

Partindo da motivação inicial para o trabalho, foi especificada, implementada e analisada uma primeira proposta [Prazeres et al., 2005; 2006b; Prazeres and Teixeira, 2006] para integração de experimentos remotos em ambientes de aprendizagem na Web. Essa proposta, apresentada na Seção 3.2, é baseada no uso de esquemas XML para a descrição da sintaxe das configurações envolvidas na experimentação remota e do uso de transformações de documentos para prover a configuração de forma dinâmica e extensível.

Comparando com trabalhos relacionados, a proposta do uso de esquemas XML mostrou-se bastante adequada para a integração de experimentos remotos em um ambiente de aprendizado eletrônico. Entretanto, o autor verificou que o uso de esquemas XML para realizar configuração de parâmetros é uma abordagem que também pode ser efetuada com a utilização de Serviços Web.

Dado a característica de interoperabilidade dos Serviços Web e de interoperabilidade dinâmica dos Serviços Web Semânticos, o autor observou ser oportuno investigar a utilização dos padrões e tecnologias de Serviços Web e de Serviços Web Semânticos na descrição de experimentos remotos. Essa investigação resultou na definição da plataforma RALOWS apresentada nas Seções 3.3 e 3.4.

A plataforma RALOWS permite adicionar descrição semântica a experimentos remotos por meio das ontologias OWL-S, *OWL-Time* e ontologia de recursos. Dessa forma, os experimentos na plataforma são publicados sob a forma de Serviços Web Semânticos. A abordagem inicial, que utiliza esquemas XML, tem foco na sintaxe das configurações do experimento remoto, enquanto que a plataforma RALOWS é baseada em descrições semânticas.

A abordagem apresentada pela plataforma RALOWS também foi aplicada [Escobedo et al., 2007] como um estudo de caso em um outro domínio de aplicações que envolvem dispositivos físicos, tal como em experimentos remotos – ambientes inteligentes. Esse estudo de caso demonstra o uso das ontologias OWL-S, de recurso e OWL-Time para modelar serviços relacionados à automação de ambientes inteligentes (automação de descoberta e composição).

A característica de interoperabilidade automática dos Serviços Web Semânticos provê resultados interessantes caso aplicada à integração de experimentos remotos a ambientes Web, tal como um ambiente de aprendizado eletrônico. Com os experimentos descritos como Serviços Web Semânticos, a integração a ambientes Web pode ser realizada de forma automática por meio da descoberta e composição automáticas.

Embora a motivação inicial e os resultados apresentados neste capítulo tenham se referido a aplicações modeladas como Serviços Web Semânticos no domínio de experimentação remota, as propostas de descoberta automática (Capítulo 4), composição automática (Capítulo 5) e também a infra-estrutura para Serviços Web Semânticos apresentada no Capítulo 6 não levam em consideração o domínio da aplicação.

CAPÍTULO

4

Descoberta de Serviços Web Semânticos

A linguagem de ontologia Web para serviços (OWL-S) apresentada no Capítulo 2 permite atribuir semântica explícita quando da modelagem de Serviços Web, conforme descrito no Capítulo 3 no domínio de experimentação remota. Semântica explícita permite a flexibilização e a automação da utilização e da prestação de serviços bem como apóia a construção de ferramentas e metodologias [Martin et al., 2007a]. Tarefas como descoberta e composição de Serviços Web podem ser automatizadas por meio da utilização das descrições semânticas nos Serviços Web.

O processo de descoberta de Serviços Web com UDDI é limitado à busca sintática por palavras-chave e não permite a realização de inferências baseadas em taxonomias. Por sua vez, o processo de descoberta automática de Serviços Web Semânticos descrito por Martin et al. [2007a] é baseado no *matching* semântico entre funcionalidades expostas por meio da sub-ontologia do perfil do serviço provido em OWL-S e as funcionalidades requeridas pelo consumidor do serviço.

Os atributos funcionais e não funcionais de um serviço em OWL-S são descritos por meio de ontologias de domínio em OWL e expostos no perfil do serviço para prover a descoberta. Isto supera a limitação da busca sintática possibilitando o *matching* semântico. *Matching* semântico, ou simplesmente *matching* (forma que é utilizada neste trabalho), é o processo de comparação entre dois conceitos com o objetivo de encontrar similaridades. O processo de descoberta baseado em *matching* semântico é denominado *capability matching* [Gao et al., 2002], no qual o objetivo é descobrir

serviços com determinadas funcionalidades requeridas.

O algoritmo para descoberta automática de Serviços Web Semânticos apresentado neste capítulo é baseado no *matching* semântico das funcionalidades do serviço. O algoritmo apresenta graus de similaridade entre as funcionalidades requeridas e as funcionalidades dos serviços anunciados, seleção de serviço baseada nas categorias do serviço e uma fase final de seleção que classifica os serviços com base nos graus de similaridades.

Adicionalmente, o autor verificou que as estratégias de descoberta reportadas na literatura não levam em consideração que alguns serviços podem não estar disponíveis ou podem requerer acesso exclusivo. Com base nisso, o algoritmo proposto para descoberta [Prazeres et al., 2008] estende os trabalhos reportados na literatura por adicionar restrições temporais possivelmente envolvidas na utilização do serviço e por prover descoberta baseada também na disponibilidade do serviço.

Este capítulo apresenta na Seção 4.1 o algoritmo utilizado na tarefa de para descoberta automática de Serviços Web Semânticos. Esse algoritmo também é utilizado para permitir descoberta automática nas tarefas de publicação e de composição de serviços como está detalhado na Seção 4.2. A Seção 4.3 apresenta um estágio adicional do algoritmo de descoberta cujo objetivo é permitir a descoberta com base na disponibilidade do serviço. As Seções 4.4 e 4.5 apresentam, respectivamente, trabalhos relacionados à descoberta automática de Serviços Web Semânticos e as considerações finais referentes a este capítulo.

4.1 Algoritmo para descoberta de Serviços Web Semânticos

Os algoritmos para descoberta de Serviços Web Semânticos em OWL-S reportados na literatura (e.g. [Paolucci et al., 2002; Srinivasan et al., 2006; Klusch et al., 2006; Li and Horrocks, 2003; Yao et al., 2006; Jaeger et al., 2005]) e denominados de algoritmos de *matching* são baseados no *macthing* das funcionalidades (entradas e saídas) e da categoria dos serviços. Esses algoritmos têm algumas características em comum, sendo que as duas principais são o uso de *matching* com classificação (*ranking*) dos resultados (por ordem melhor *matching*) e o fato de que o *matching* é realizado utilizando inferência baseada em *subsumption*.

A inferência baseada em *subsumption* é executada sobre o esquema conceitual (ontologias). Partindo dessa premissa, o *matching* não é sintático e sim baseado nas relações e conceitos de uma ontologia OWL. Aspectos de uma ontologia como conceitos, propriedades, restrições, intersecções, uniões e propriedades simétricas, transitivas e inversas, são utilizados para efetuar a inferência baseada em *subsumption*.

Relações de *subsumption* são baseadas na hierarquia de classes de uma ontologia.

Alguns algoritmos de *matching* baseados em *subsumption* definem graus de exatidão do "matching", tal como, *exact*, *plugin*, *subsume* e *fail* [Paolucci et al., 2002]. A Figura 4.1 apresenta um fragmento de uma ontologia de domínio para veículos. Por relações de *subsumption* dessa figura pode-se inferir, por exemplo, que a classe "Carro" *subsumes* a classe "Sedan", ou seja, a classe "Carro" é superclasse da classe "Sedan".

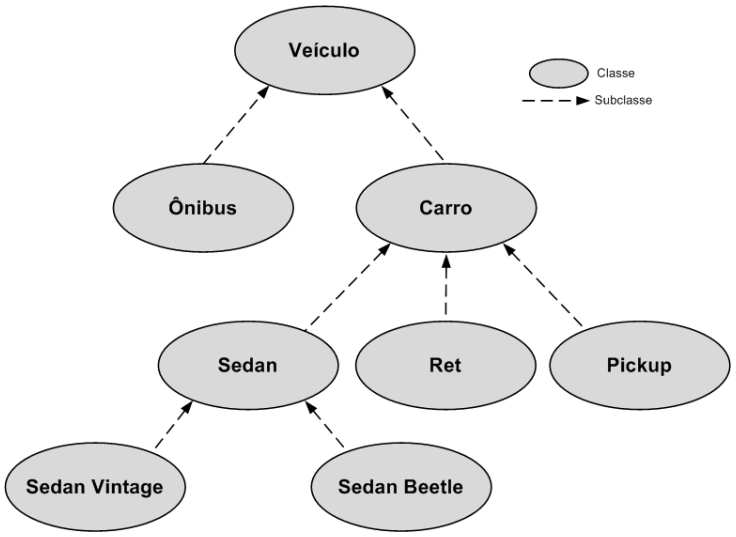

Figura 4.1: Fragmento de uma ontologia para veículo.

O perfil de um serviço descrito por OWL-S é a principal estrutura utilizada para descoberta de serviços. A descoberta acontece da seguinte forma: os provedores de serviço anunciam o perfil de seus serviços com os atributos funcionais (IOPRs) e os não funcionais, e o cliente do serviço também deve gerar um perfil do serviço (com os atributos funcionais e não funcionais desejados) que vai ser comparado (*matchmaking of service*) com os perfis de serviços anunciados por provedores de serviços. Então, o processo de descoberta retorna o(s) serviço(s) encontrado(s) estabelecendo o grau de exatidão da busca que é baseado em *subsumption*.

O algoritmo [Prazeres et al., 2008] apresentado nesta seção, conforme ilustrado na Figura 4.2, possui cinco estágios: *matching* nas entradas, *matching* nas saídas, seleção baseada nas pré-condições, seleção baseada na categoria do serviço e seleção final do serviço.

O primeiro e segundo estágios, ilustrados na Figura 4.2 e descritos na Seção 4.1.2 e na Seção 4.1.3 respectivamente, utilizam inferência baseada em *subsumption* para definir o grau de similaridade do *matching* na funcionalidade do serviço (entradas e saídas).

Os três últimos estágios (ilustrados na Figura 4.2 e descritos na Seção 4.1.4, na Seção 4.1.5 e na Seção 4.1.6) do algoritmo de descoberta realizam seleção de serviço.

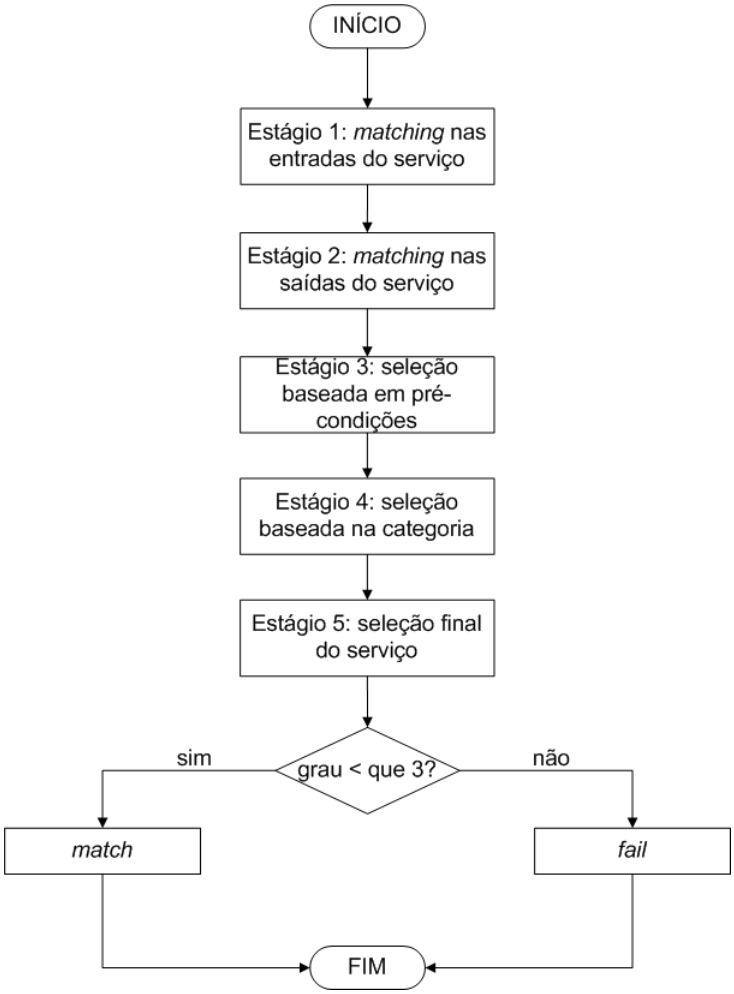

Figura 4.2: Algoritmo para descoberta de Serviços Web Semânticos.

A seleção tem por objetivo filtrar a quantidade de resultados obtidos com os dois estágios iniciais do algoritmo (*matching* das entradas e das saídas).

O terceiro estágio do algoritmo realiza seleção com base nas pré-condições do serviço e tem por objetivo analisar se as entradas informadas pelo usuário satisfazem as condições de execução do serviço.

O quarto estágio também utiliza inferência baseada em *subsumption* (conforme os dois primeiros estágios) com objetivo de realizar seleção de serviço com base na descrição das categorias.

O quinto e último estágio apresenta uma classificação dos serviços que mais se aproximam da requisição inicial do do usuário.

A Seção 4.3.3 ainda apresenta um estágio extra que é opcional para o algoritmo para descoberta de Serviços Web Semânticos. Esse estágio é executado apenas caso o serviço apresente restrições temporais relativas à disponibilidade do serviço.

4.1.1 Graus de similaridade

Os graus de similaridade utilizados no algoritmo para os três primeiros estágios são baseados na proposta de Paolucci et al. [2002] conforme apresentado no Documento 14. São quatro os graus de similaridade: *exact*, *plugIn*, *subsumes* e *fail*.

Documento 14 Os graus de similaridade definidos por Paolucci et al. [2002].

```
1  //graus de similaridade
2
3  degreeOfMatch(R, A):
4  if A=R then return exact
5  if R subclassOf A then return exact
6  if A subsumes R then return plugIn
7  if R subsumes A then return subsumes
8  otherwise fail
```

O grau é dito *exact* em duas situações distintas como pode ser visto nas linhas 4 e 5 do Documento 14. Na primeira situação (linha 4), os conceitos comparados (Requerido e Anunciado) são equivalentes, ou seja, são os mesmos conceitos de uma ontologia. A segunda situação (linha 5) ocorre caso o conceito requerido seja uma sub-classe direta do conceito anunciado. Segundo Paolucci et al. [2002] a utilização de sub-classes diretas garante a consistência do que um serviço espera como entrada ou do que o usuário do serviço espera como saída.

Nos casos em que o conceito requerido é sub-classe indireta do conceito anunciado, o grau é dito ser um *plugIn* (linha 6). O grau *plugIn* significa que o conceito anunciado pode substituir o requerido garantindo a consistência, porém não garante que vai satisfazer o conceito requerido.

O grau *subsumes* (linha 7) ocorre nos casos em que o conceito anunciado é sub-classe, direta ou indireta, do conceito requerido. Nesses casos, o conceito anunciado não satisfaz por completo o conceito requerido.

A classificação de Paolucci et al. [2002] diz respeito a cada conceito individualmente. Entretanto, para os dois primeiros estágios (Seções 4.1.2 e 4.1.3) do algoritmo foi necessário a criação de uma nova classificação para o grau de similaridade. A classificação deve abranger um conjunto de conceitos em vez de conceitos individualmente. Essa classificação diz respeito ao grau de similaridade do conjunto das entradas e saídas do serviço requerido em comparação com o conjunto das entradas e saídas do serviço anunciado.

A Figura 4.3 ilustra o momento de um *matching* nas funcionalidades de um serviço (*Service1*). Cada entrada requerida (*InR1* e *InR2* na Figura 4.3) e cada saída requerida (*OutR1* e *OutR2* na Figura 4.3) é comparada com as entradas e saídas (respectivamente) de um serviço anunciado e o grau de similaridade é atribuído a cada *matching* realizado como especificado no Documento 14. A classificação é

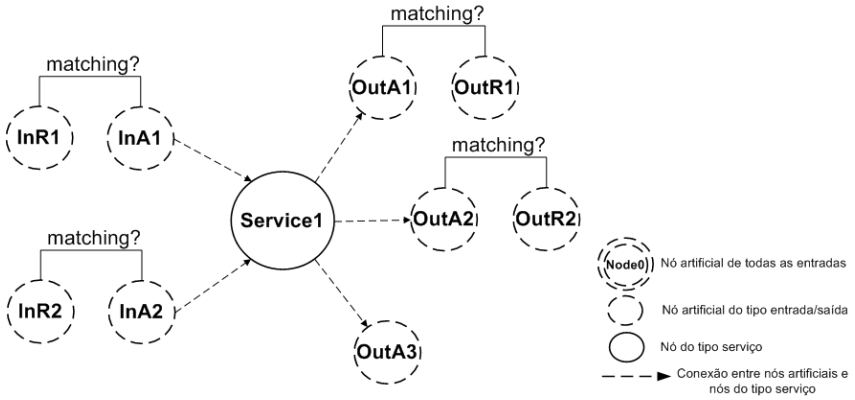

Figura 4.3: *Matching* nas entradas e saídas de um serviço anunciado.

atribuída para cada conjunto de conceitos (entradas e saídas) separadamente logo após a computação do *matching* em todas as entradas e saídas da seguinte forma:

- *Matching* **completo**: se o serviço anunciado possuir todas as entradas requeridas, o *matching* é dito completo para o conjunto das entradas requeridas. Da mesma forma, se o serviço anunciado possuir todas as saídas requeridas, o *matching* é dito completo para o conjunto das saídas requeridas.

- *Matching* **sub-conjunto**: se o serviço anunciado possuir um sub-conjunto das entradas requeridas, o *matching* é dito sub-conjunto para o conjunto das entradas requeridas. Da mesma forma, se o serviço anunciado possuir um sub-conjunto das saídas requeridas, o *matching* é dito sub-conjunto para o conjunto das saídas requeridas.

- *Matching* **super-conjunto**: se o serviço anunciado possuir um super-conjunto das entradas requeridas, o *matching* é dito super-conjunto para o conjunto das entradas requeridas. Da mesma forma, se o serviço anunciado possuir um super-conjunto das saídas requeridas, o *matching* é dito super-conjunto para o conjunto das saídas requeridas.

- *Matching* **intersecção**: se o serviço anunciado possuir uma intersecção com as entradas requeridas, o *matching* é dito intersecção para o conjunto das entradas requeridas. Da mesma forma, se o serviço anunciado possuir uma intersecção nas saídas requeridas, o *matching* é dito intersecção para o conjunto das saídas requeridas.

- *Matching* **disjunto**: se o serviço anunciado não possuir ao menos uma das entradas requeridas, o *matching* é dito disjunto para o conjunto das entradas requeridas. Da mesma forma, se o serviço anunciado não possuir ao menos uma

das saídas requeridas, o *matching* é dito disjunto para o conjunto das saídas requeridas.

Os graus de similaridade para conjunto de conceitos apresentados nesta seção e os graus de similaridade propostos por Paolucci et al. [2002] são utilizados no algoritmo da forma que está descrita nas seções 4.1.2, 4.1.3 e 4.1.5 do presente trabalho.

4.1.2 *Matching* nas entradas do serviço

A primeira etapa do algoritmo determina o quanto os parâmetros de entrada de um serviço anunciado satisfazem o serviço requerido pelo usuário. O conjunto de entradas do serviço requerido é chamado neste trabalho de entradas disponíveis, pois são as entradas que o usuário, requisitante do serviço, tem à sua disposição para a descoberta e posterior execução do serviço.

Para cada entrada disponível, o *matching* é realizado com cada entrada de todos os serviços anunciados. O grau do *matching*, apresentado na Tabela 4.1, é atribuído a cada serviço anunciado avaliando o conjunto por inteiro das entradas disponíveis (*ReqInput*) com o conjunto por inteiro das entradas do serviço anunciado (*AdvInput*).

A Tabela 4.1 apresenta a atribuição dos graus para os possíveis *matching*: 0 para *matching* completo, 1 nos casos em que o *matching* é um sub-conjunto, 2 para um super-conjunto, 2 para intersecção e 3 nos casos em que não existe *matching*.

O *matching* é sempre atribuído em relação ao serviço anunciado. Por exemplo, o *matching* é dito ser super-conjunto se o conjunto das entradas do serviço anunciado for um super-conjunto do conjunto das entradas disponíveis. Assim, o serviço anunciado é um super-conjunto do serviço requerido em relação às entradas.

Tabela 4.1: Graus de similaridade para *matching* nas entradas

Matching	**Explicação**	**Grau**
Completo	Se ReqInput $\equiv$ AdvInput	0
Sub-conjunto	Se ReqInput $\supsetneq$ AdvInput	1
Super-conjunto	Se ReqInput $\subsetneq$ AdvInput	2
Intersecção	Se ReqInput $\cap$ AdvInput	2
Disjuntos	Não existem entradas em comum: o *matching* falhou	3

O *matching* é dito completo apenas caso o serviço anunciado possua exatamente todas as entradas disponíveis. Nesse caso, é atribuído o menor grau de *matching* (grau 0 na Tabela 4.1), pois é o melhor caso para a descoberta das entradas devido ao fato de que o serviço pode usar todas as entradas que o usuário tem disponível e o serviço não exige outras entradas indisponíveis ao usuário.

Super-conjunto, sub-conjunto e intersecção são graus de *matching* intermediários e ainda aceitáveis no caso de não serem encontrados serviços com *matching* completo.

Pode-se notar na Tabela 4.1 que, para as entradas, o algoritmo atribui um grau de *matching* pior para o resultado de super-conjunto e de intersecção. Essa decisão é em função da disponibilidade das entradas, ou seja, no caso de super-conjunto e intersecção, o serviço anunciado pode requerer mais entradas do que o usuário tem disponível. O usuário pode nunca conseguir executar o serviço caso não disponha de uma ou mais entradas.

O último grau de *matching* da Tabela 4.1 é o pior caso e por isso recebe grau igual a 3. O *matching* das entradas é dito disjunto (ou falho) se o serviço anunciado não possuir ao menos uma das entradas disponíveis.

Os graus de *matching* para as entradas são importantes para a fase de seleção do serviço (Seção 4.1.6) e para a composição do serviço apresentada no Capítulo 5. A Seção 4.2 descreve como o grau de *matching* para as entradas é levado em consideração no momento da decisão de se compor ou não um serviço.

4.1.3 *Matching* nas saídas do serviço

A segunda etapa do algoritmo determina o quanto os parâmetros de saída de um serviço anunciado satisfazem o serviço requerido pelo usuário. O conjunto de saídas do serviço requerido é chamado no presente trabalho de saídas requeridas, pois são as saídas necessárias para o usuário executar a tarefa desejada.

Para cada saída requerida, o *matching* é realizado com cada saída de todos os serviços anunciados. O grau do *matching*, apresentado na Tabela 4.2, é atribuído a cada serviço anunciado avaliando o conjunto por inteiro das saídas requeridas (*ReqOutput*) com o conjunto por inteiro das saídas do serviço anunciado (*AdvOutput*).

A Tabela 4.2 apresenta a atribuição dos graus para os possíveis *matching*: 0 para *matching* completo, 1 nos casos em que o *matching* é um super-conjunto, 2 para um sub-conjunto, 2 para intersecção e 3 nos casos em que não existiu *matching*.

O *matching* é sempre atribuído em relação ao serviço anunciado. Por exemplo, o *matching* é dito ser sub-conjunto se o conjunto das saídas do serviço anunciado for um sub-conjunto do conjunto das saídas. Assim, o serviço anunciado é um sub-conjunto do serviço requerido em relação às saídas.

Tabela 4.2: Graus de similaridade para *matching* nas saídas

Matching	Explicação	Grau
Completo	Se ReqOutput $\equiv$ AdvOutput	0
Super-conjunto	Se ReqOutput $\subsetneq$ AdvOutput	1
Sub-conjunto	Se ReqOutput $\supsetneq$ AdvOutput	2
Intersecção	Se ReqOutput $\cap$ AdvOutput	2
Disjuntos	Não existe saídas em comum: *matching* falhou	3

O *matching* é dito completo apenas caso o serviço anunciado possua exatamente todas as saídas requeridas. Nesse caso, é atribuído o menor grau de *matching* (grau 0 na Tabela 4.2), pois é o melhor caso para a descoberta das saídas devido ao fato de que o serviço fornece todas as saídas requeridas pelo usuário.

Super-conjunto, sub-conjunto e intersecção são graus de *matching* intermediários e ainda aceitáveis no caso de não se encontrar serviços com *matching* completo. Pode-se notar na Tabela 4.2 que, para as saídas, o algoritmo atribui um grau de *matching* melhor para o resultado de super-conjunto. Essa decisão é em função de que o *matching* super-conjunto para as saídas significa que o serviço, assim como o *matching* completo, também fornece todas as saídas requeridas pelo usuário e algumas saídas não requeridas. Enquanto que o *matching* sub-conjunto e intersecção não apresentam algumas das saídas que o usuário solicitou.

O último grau de *matching* da Tabela 4.2 é o pior caso e por isso recebe grau igual a 3. O *matching* das saídas é dito disjunto (ou falho) se o serviço anunciado não possuir ao menos uma das saídas requeridas.

Os graus de *matching* para as saídas são importantes para a fase de seleção do serviço (Seção 4.1.6) e para a composição do serviço apresentada no Capítulo 5. A Seção 4.2 descreve como o grau de *matching* para as saídas é levado em consideração no momento da decisão de se compor ou não um serviço.

4.1.4 Seleção baseada em pré-condições

Conforme apresentado no Capítulo 2, além das entradas e saídas, o perfil do serviço descrito em OWL-S expõe ainda mais duas informações: as pré-condições e os resultados. Pré-condição é uma proposição que deve ser verdadeira para que o serviço funcione corretamente. Já os resultados são especificações de saídas e de efeitos, que são proposições que serão verdadeiras ao final da execução do serviço.

As pré-condições podem estar diretamente relacionadas com uma ou mais entradas de um serviço e, se isso ocorrer, significa que essas entradas são obrigatórias para a execução do serviço. Por exemplo, um serviço que requer, como requisito para a sua execução, que o usuário tenha uma conta (usuário e senha) válida no sistema, pode apresentar a seguinte pré-condição: *hasAcctID(SignInInfo, AcctID)*.

A pré-condição *hasAcctID* possui dois parâmetros. O primeiro (*SignInInfo*) corresponde à uma entrada do serviço que deve ser obrigatória para a execução do serviço. O segundo (*AcctID*) é uma variável local e, para o algoritmo aqui proposto, não tem influência na descoberta do serviço.

A seleção baseada nas pré-condições corresponde ao terceiro estágio do algoritmo e define dois níveis de classificação tal como apresentado na Tabela 4.3. O grau igual a 0 é o melhor caso devido ao fato de que o serviço não tem pré-condições diretamente relacionadas com as entradas as quais o usuário não dispõe. O grau 3 indica que

Tabela 4.3: Classificação para pré-condição

Matching	Explicação	Classificação
Exact	Não existem pré-condições ou não existem pré-condições relacionadas a entradas indisponíveis	0
Fail	Existem pré-condições relacionadas a entradas indisponíveis	3

ao menos uma pré-condição pode não ser satisfeita com o conjunto de entradas informadas pelo usuário. Dessa forma, o algoritmo aqui descrito também utiliza as descrições das pré-condições em OWL-S no processo de descoberta automática.

4.1.5 Seleção baseada na categoria do serviço

A seleção baseada na categoria do serviço, quarto estágio do algoritmo, determina, por meio de *matching*, a conectividade semântica entre a categoria de um serviço requerido (*ReqCategory*) e a categoria de um serviço anunciado (*AdvCategory*). Esse estágio é opcional dentro do algoritmo para descoberta de Serviços Web Semânticos aqui proposto.

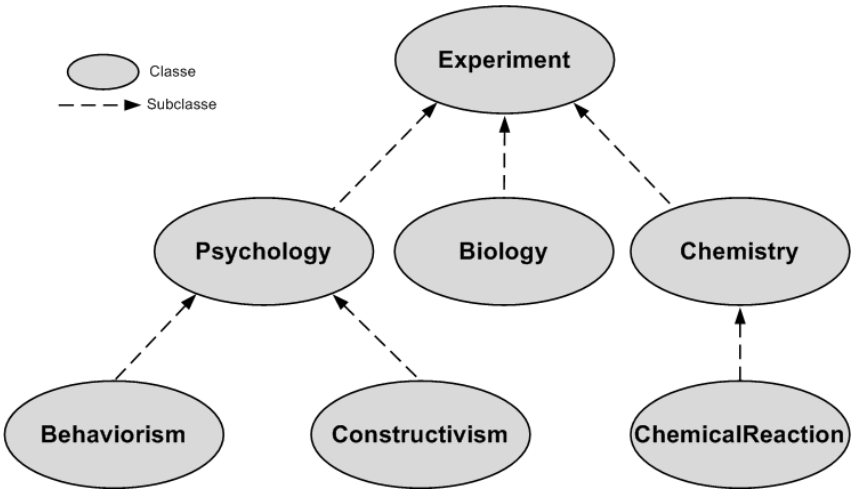

Figura 4.4: Hierarquia de categorias de experimentos.

A Figura 4.4 apresenta um fragmento de uma ontologia de domínio que descreve a hierarquia de classes para categorização de experimentos. O algoritmo de *matching* utiliza ontologias de domínio na classificação dos serviços para realizar inferência baseada em *subsumption* no *matching* (da mesma forma que ocorre no *matching* de cada entrada e saída do serviço nos dois primeiros estágios). Por exemplo, uma requisição por um serviço de experimento "Caixa de Skinner" define uma categoria de serviços chamada *Behaviorism* (*ReqCategory*). O algoritmo de matching encontra um serviço publicado com um experimento em que a categoria do serviço (*AdvCategory*)

é *Psychology*. Nesse caso, tal como ilustra a Figura 4.4, o grau de matching não é *exact*. O algoritmo reconhece, por meio da inferência baseada em *subsumption*, que *Behaviorism* é um *PlugIn* (ver Tabela 4.4) de *Psychology*.

Tabela 4.4: Classificação para *matching* na categoria

Matching	Explicação	Classificação
Exact	Se AdvCategory e ReqCategory são equivalentes	0
PlugIn	Se ReqCategory é sub-conceito de AdvCategory	1
SubSume	Se ReqCategory é super-conceito de AdvCategory	2
Fail	Em qualquer outro caso, o *match* é dito falho	3

O grau de *matching* para categoria está definido na Tabela 4.4. De acordo com a tabela, se o matching por uma dada categoria retorna um super-conceito, é atribuído a classificação 1 (*PlugIn*), e nos casos em que retorna um sub-conceito é atribuído a classificação 2 (*SubSume*).

4.1.6 Seleção final do serviço

O quinto e último estágio do algoritmo de *matching* (seleção do serviço) resulta em uma classificação geral do *matching*, apresentado na Tabela 4.5. Nesse estágio, o algoritmo de *matching* apenas tem *Fail* como resultado final, se ao menos um dos dois estágios iniciais (*matching* das entradas e das saídas) tiveram ao menos um grau 3 como resultado.

Tabela 4.5: Classificação final para seleção do serviço

Matching	Explicação	Classificação
Match	O resultado de cada estágio obteve grau 0	0
Match	Se ao menos um resultado em um estágio obteve grau 1 e os outros resultados não apresentaram grau maior que 1	1
Match	Se ao menos um resultado em um estágio obteve grau 2 e os outros resultados não apresentaram grau maior que 2	2
Fail	Em ao menos um estágio o resultado foi igual a 3	3

Os níveis de *matching* apresentados na forma classificada na Tabela 4.5 definem o grau de exatidão do processo de descoberta. Os serviços com classificações menores são os mais próximos da requisição do usuário, sendo que os serviços classificados com 0 atendem 100% à requisição. Os serviços classificados entre 1 e 2 atendem parcialmente à requisição do usuário e correspondem a soluções parciais para o problema inicial (tarefa que o usuário quer realizar).

A vantagem da utilização de graus para a classificação dos resultados é que no estágio de seleção final pode-se escolher até que nível de resultados serão

apresentados para o usuário final. Por exemplo, se existiu *matching* igual a 0 em todos os estágios, significa que um ou mais serviços "perfeitos" foram encontrados. Partindo dessa premissa, pode-se tomar a decisão de não exibir os resultados para os outros graus. Por outro lado, se não existiu *matching* igual a 0 em todos os estágios, pode-se tomar a decisão de, por exemplo, de exibir os resultados com graus iguais a 1 e 2.

4.2 Descoberta na publicação e composição de serviços

A descoberta automática de Serviços Web Semânticos é uma atividade essencial para o processo de composição automática. As abordagens para composição automática de Serviços Web existentes na literatura e apresentadas no Capítulo 5 (como trabalhos relacionados a este trabalho) levam em consideração o resultado da descoberta para realizar a composição automática. Caso não ocorram, entre os resultados do processo de descoberta, serviços que atendam completamente à requisição do usuário, surge a oportunidade e o desafio de se realizar composição automática de serviços, que é um dos problemas investigados neste trabalho.

Neste trabalho, não apenas o resultado do processo de descoberta é utilizado para composição automática, como também o próprio processo de descoberta é utilizado algumas vezes durante o processo de composição, conforme descrito na Seção 4.2.2 e detalhado no Capítulo 5.

A descoberta automática de Serviços Web Semânticos também é utilizada no presente trabalho durante a publicação do serviço. Porém, o objetivo é ainda com foco em composição, tal como descrito na Seção 4.2.1 e também detalhado no Capítulo 5.

4.2.1 Publicação

O modelo utilizado para realizar composição automática de serviços neste trabalho é baseado em grafos. Um grafo é criado no momento da publicação dos serviços por questões de otimização do tempo de composição. A Figura 4.5 apresenta um exemplo de publicação de serviço da forma como é proposta neste trabalho. O exemplo neste momento, não entra em todos os detalhes da publicação e da criação do grafo, apenas apresenta a forma que a descoberta é utilizada na criação de um grafo.

O serviço *Service3* na Figura 4.5 está sendo publicado e a figura representa o momento em que é calculado o *matching* entre dois pares de conceitos: *Out1* de *Service3* com *In1* de *Service1*; e *Out2* de *Service3* com *In1* de *Service2*.

O objetivo desses dois *matchings* é verificar a conectividade semântica entre dois serviços. Em outras palavras, verificar se a saída de um serviço tem a mesma semântica da entrada de um outro serviço. A conectividade semântica entre dois serviços, na prática, significa que a saída de um serviço (e.g. *Out1* de *Service3*), em

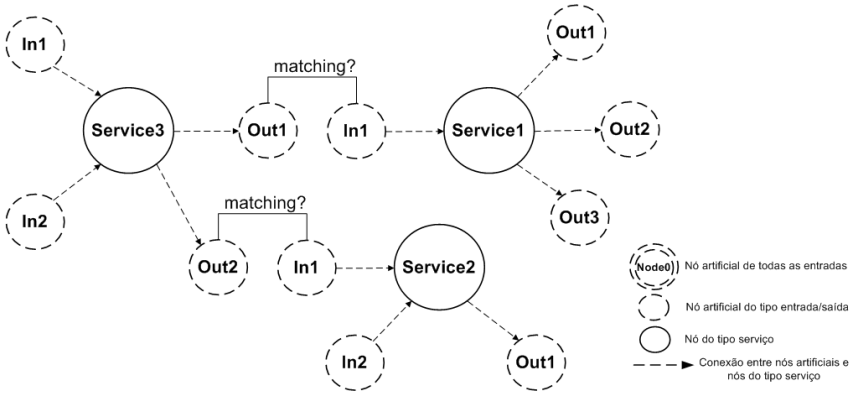

Figura 4.5: Uso da descoberta na publicação do serviço.

uma composição de serviços, pode funcionar como insumo (e.g. *In1* de *Service1*) para um outro serviço.

O processo de descoberta na publicação é utilizado para encontrar todos os serviços que possuem conectividade semântica com o serviço que está sendo publicado e essa informação é utilizada para a criação do grafo. Os detalhes da modelagem e da criação do grafo não serão tratados nesta seção, visto que são apresentados em detalhes no capítulo referente a composição de serviços (Capítulo 5). O objetivo de descrever o uso da descoberta no momento da publicação é ressaltar como o processo de descoberta descrito na Seção 4.1 influencia diretamente o processo de composição automática de serviços apresentado neste trabalho.

4.2.2 Composição

O processo de descoberta também é utilizado no presente trabalho no momento da composição automática de um serviço. Nos casos em que um usuário solicita um serviço e informa os requisitos de entrada e saída, a descoberta é efetuada normalmente utilizando o algoritmo conforme descrito na Seção 4.1.

Pode ocorrer do resultado da descoberta não ser satisfatório, tal como apresentado na fase de seleção do serviço descrita na Seção 4.1.6. Neste trabalho, ainda que os resultados não sejam satisfatórios eles são reutilizados no processo de composição automática de três maneiras:

1. auxiliar na tomada de decisão sobre efetuar ou não uma composição;

2. fornecer um conjunto de todos os serviços que possuem ao menos uma entrada disponível ou uma saída requerida;

3. verificar a composição final.

Os resultados da descoberta fornecem a informação de quão um serviço anunciado atende aos requisitos do usuário. Essa informação é crucial para se decidir sobre compor ou não um novo serviço. Uma decisão, nos casos em que se precisa que todo o processo seja automatizado, é sempre tentar compor caso não exista ao menos um serviço que atenda completamente os requisitos do usuário. Soluções mais flexíveis podem levar em consideração a opinião do usuário que pode decidir sobre compor um novo serviço ou utilizar um serviço que atende parcialmente aos seus requisitos.

Nos casos em que a opção é pela composição, os resultados da descoberta são utilizados para informar ao processo de composição quais serviços atendem parcialmente às solicitações do usuário. Esses serviços devem ser o ponto de partida para uma composição automática, assim como ocorre na abordagem para composição automática apresentada no Capítulo 5 deste trabalho.

Após o processo de composição ser realizado com sucesso, ou seja, uma composição que atende aos requisitos do usuário ter sido encontrada, a descoberta é utilizada para verificar o quanto a composição final atende aos requisitos iniciais do usuário. O processo de descoberta descrito na Seção 4.1 é repetido uma única vez, tendo como parâmetros os requisitos do usuário (conjuntos de entradas disponíveis e saídas requeridas) e as entradas e saídas da nova composição. Nesse ponto, o resultado da composição é comparado com os resultados da descoberta inicial e pode-se avaliar a eficiência da composição.

4.3 Descoberta de disponibilidade

O algoritmo para descoberta automática de Serviços Web Semânticos apresentado na Seção 4.1 é genérico e baseado no *matching* das funcionalidades do serviço. Como uma extensão a esse algoritmo o autor propôs a descoberta baseada na disponibilidade do serviço com a finalidade de também permitir a descoberta de serviços específicos que incluem restrições temporais de acesso como no caso dos experimentos remotos apresentados no Capítulo 3 [Prazeres et al., 2008].

A Seção 4.3.3 apresenta um estágio extra e opcional do algoritmo para prover a descoberta com base na disponibilidade do serviço. Esse estágio extra é executado apenas se o serviço estiver descrito com as extensões temporais específicas para experimentos remotos previstas na plataforma RALOWS e generalizadas tal como apresentado nas Seções 4.3.1 e 4.3.2.

4.3.1 RALOWS-Gen

A ontologia RALOWS, apresentada e discutida na Seção 3.3, tem como proposta modelar experimentos remotos na forma de Serviços Web Semânticos ao mesmo tempo em que descreve as restrições temporais e de acesso aos recursos envolvidos

na experimentação remota. Um experimento, em RALOWS, pode ser qualquer tipo de serviço em que os recursos tenham restrições temporais. Partindo desta premissa, é possível generalizar a ontologia RALOWS para permitir a descrição de quaisquer tipos de Serviços Web que possuam restrições temporais e de acesso aos recursos envolvidos.

A Figura 4.6 apresenta a ontologia RALOWS-Gen, na qual a classe *Instrument* da ontologia RALOWS foi omitida de forma que instrumentos (dispositivos que na RALOWS proviam serviços) serão modelados também como recursos de entrada do serviço (*InputResource*). Exemplos de serviços que possuem fortes restrições temporais tal como os experimentos remotos são, por exemplo, agendamento de reuniões remotas, reserva de quartos em hotel, reserva de assentos em vôos, dentre outros que possuem recursos que não podem ser compartilhados para utilização simultânea ou mesmo que precisam de um agendamento prévio para a sua utilização.

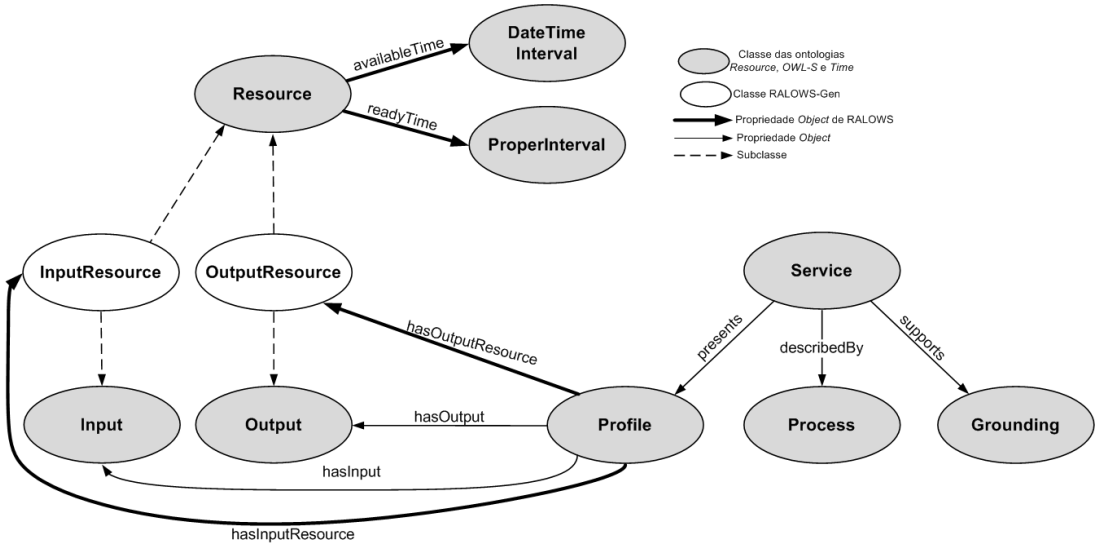

Figura 4.6: Ontologia RALOWS-Gen.

Restrições de tempo envolvem a associação de conceitos temporais aos recursos envolvidos em um Serviço Web. Conceitos como instantes e intervalos de tempo são associados aos recursos e serviços, tal como apresentado na Seção 4.3.2, definindo pré-condições para a utilização do serviço e efeitos pós-utilização dos serviços.

4.3.2 Conceitos temporais

A ontologia de tempo *OWL-Time* define alguns conceitos temporais básicos: *Instant*, *Interval*, *Instant Event*, *Interval Events* e algumas subclasses desses conceitos (ver Seção 2.1.3). Pan [2005] descreve o uso da ontologia de tempo para representar conceitos temporais agregados a Serviços Web Semânticos.

Baseando-se nas principais classes de OWL-Time e na proposta de Pan [2005], esta seção apresenta alguns conceitos temporais agregados a serviços e a seus recursos. Conceitos como disponibilidade, duração de sessão, tempo mínimo de sessão e tempo máximo de sessão são apresentados e discutidos nas seções seguintes.

Conceitos temporais para recursos

Com o objetivo de permitir o agendamento de serviços foram definidos dois intervalos de tempo que são associados aos recursos pertencentes ao serviço. Esses intervalos são definidos por duas propriedades conforme ilustrado na Figura 4.6: *availableTime* (definido no Documento 15) e *readyTime* (definido no Documento 16).

A propriedade *availableTime* (linhas 3 a 6 no Documento 15) da classe *Resource* define os intervalos de tempo em que o recurso está disponível para ser utilizado. Um recurso é declarado com a cardinalidade de zero ou mais com respeito à propriedade *availableTime* (linhas 8 a 11). Como resultado, um recurso pode ter zero ou mais intervalos de disponibilidade.

Documento 15 Propriedade *availableTime* de um recurso.

```
1  <!-- Defining the property availableTime -->
2
3  <owl:ObjectProperty rdf:ID="&resource;availableTime">
4   <rdfs:domain rdf:resource="&resource;Resource"/>
5   <rdfs:range rdf:resource="&time;DateTimeInterval"/>
6  </owl:ObjectProperty>
7
8  <owl:Restriction>
9   <owl:onProperty rdf:resource="&resource;availableTime"/>
10  <owl:minCardinality
        rdf:datatype="&xsd;#nonNegativeInteger">0</owl:minCardinality>
11 </owl:Restriction>
```

No caso do recurso ter zero intervalos de disponibilidade, o recurso não está disponível e não pode ser agendado; como resultado, todos os serviços que necessitam do recurso em questão também não poderão ser agendados. Caso o recurso possua um ou mais intervalos de disponibilidade, a disponibilidade do recurso é definida como a união dos intervalos de disponibilidade, ou seja:

$$\begin{aligned}&\text{ResourceAvailability} \equiv \exists \text{availableTime.Interval1} \cup \\ &\exists \text{availableTime.Interval\_2} \cup \\ &\exists \text{availableTime.Interval\_3} \cup \\ &\vdots \\ &\exists \text{availableTime.Interval\_n}\end{aligned}$$

Alguns recursos precisam de um certo tempo de preparação para tornarem-se prontos para uso. Por exemplo, o tempo associado com a preparação de uma solução química demandada por um experimento de Química, o tempo necessário para preparação de uma sala que será utilizada em uma reunião por meio de vídeo-conferência, dentre outros. A propriedade *readyTime* (linhas 3 a 6 do Document 16) da classe *Resource* define o tempo de preparação necessário a um recurso. Um recurso é declarado com a cardinalidade de zero (linhas 8 a 11) ou um (linhas 13 a 16) com respeito à propriedade *readyTime*.

Documento 16 Propriedade *readyTime* de um recurso.

```
1  <!-- Defining the property readyTime. -->
2
3  <owl:ObjectProperty rdf:ID="&resource;readyTime">
4   <rdfs:domain rdf:resource="&resource;Resource"/>
5   <rdfs:range rdf:resource="&time;ProperInterval"/>
6  </owl:ObjectProperty>
7
8  <owl:Restriction>
9   <owl:onProperty rdf:resource="&resource;readyTime"/>
10  <owl:minCardinality
         rdf:datatype="&xsd;#nonNegativeInteger">0</owl:minCardinality>
11 </owl:Restriction>
12
13 <owl:Restriction>
14  <owl:onProperty rdf:resource="&resource;readyTime"/>
15  <owl:maxCardinality
         rdf:datatype="&xsd;#nonNegativeInteger">1</owl:maxCardinality>
16 </owl:Restriction>
```

Conceitos temporais para serviços

Dado que um serviço pode ter como entradas um ou mais recursos que precisam de um agendamento prévio, a disponibilidade de um serviço como um todo pode ser computada pela intersecção da disponibilidade de todos os recursos envolvidos, ou seja:

$$\begin{aligned}&\text{ServiceAvailability} \equiv \exists \text{resource\_1.ResourceAvailability} \cap \\ &\exists \text{resource\_2.ResourceAvailability} \cap \\ &\exists \text{resource\_3.ResourceAvailability} \cap \\ &\vdots \\ &\exists \text{resource\_n.ResourceAvailability}\end{aligned}$$

O intervalo agendado deve ser um intervalo válido de acordo com os intervalos de

disponibilidade dos recursos e não pode sobrepor um outro intervalo agendado.

Outro importante conceito temporal de serviços é a duração de uma sessão, que é o intervalo de tempo que o usuário gasta executando o serviço. Alguns serviços podem ter limites mínimos e máximos para a duração de uma sessão de execução.

No Documento 17, ambos os limites de *minSessionDuration* (linhas 16 a 19) e de *maxSessionDuration* (linhas 21 a 24) são definidos como propriedades da classe *SessionDuration*. A cardinalidade 1 (linhas 5 e 10) para as duas propriedades na definição da classe *SessionDuration* indica que uma instância de *SessionDuration* deve ter um e apenas um valor de propriedade para *minSessionDuration* e *maxSessionDuration*.

Documento 17 Definição da classe *SessionDuration*.

```
1  <!-- Defining the class SessionDuration. -->
2
3  <owl:Class rdf:ID="SessionDuration">
4   <rdfs:subClassOf>
5    <owl:Restriction owl:cardinality="1">
6     <owl:onProperty rdf:resource="#minSessionDuration"/>
7    </owl:Restriction>
8   </rdfs:subClassOf>
9   <rdfs:subClassOf>
10   <owl:Restriction owl:cardinality="1">
11    <owl:onProperty rdf:resource="#maxSessionDuration"/>
12   </owl:Restriction>
13  </rdfs:subClassOf>
14 </owl:Class>
15
16 <rdf:Property rdf:ID="minSessionDuration">
17  <rdfs:domain rdf:resource="#SessionDuration"/>
18  <rdfs:range rdf:resource="&time;#Interval"/>
19 </rdf:Property>
20
21 <rdf:Property rdf:ID="maxSessionDuration">
22  <rdfs:domain rdf:resource="#SessionDuration"/>
23  <rdfs:range rdf:resource="&time;#Interval"/>
24 </rdf:Property>
```

Como exemplo, para definir uma sessão que deve durar entre 30 (no mínimo) e 120 (no máximo) minutos (*Session30-120minutesDuration* definida no Documento 19), são definidos a duração mínima da sessão (30 minutos) e a duração máxima da sessão (120 minutos).

Considerando que as faixas (*rdfs:range* nas linhas 18 e 23 do Documento 17) das propriedades *minSessionDuration* e *maxSessionDuration* são intervalos (*Interval* de *OWL-Time*), intervalos com durações específicas devem ser criados previamente. Por exemplo, para uma sessão de 30 a 120 minutos (*Session30-120minutesDuration* definida no Documento 19), os intervalos *Interval30Minutes* e *Interval120Minutes*

devem ser definidos previamente conforme ilustrado no Documento 18.

Documento 18 Definição das classes *Interval30Minutes* and *Interval120Minutes*.

```
1  <!-- Defining the classes
        Interval30Minutes and Interval120Minutes-->
2
3  <owl:Class rdf:ID="Interval30Minutes">
4   <!-- intervals with a duration of 30 minutes -->
5   <rdfs:subClassOf rdf:resource="&time;#Interval"/>
6   <owl:subClassOf>
7    <owl:Restriction>
8     <owl:onProperty rdf:resource="&time;#durationDescriptionDataType"/>
9     <owl:hasValue rdf:datatype="&xsd;duration">
10     PT30M
11    </owl:hasValue>
12   </owl:Restriction>
13  </owl:subClassOf>
14 </owl:Class>
15
16 <owl:Class rdf:ID="Interval120Minutes">
17  <!-- intervals with a duration of 120 minutes -->
18  <rdfs:subClassOf rdf:resource="&time;#Interval"/>
19  <owl:subClassOf>
20   <owl:Restriction>
21    <owl:onProperty rdf:resource="&time;#durationDescriptionDataType" />
22    <owl:hasValue rdf:datatype="&xsd;duration">
23     PT120M
24    </owl:hasValue>
25   </owl:Restriction>
26  </owl:subClassOf>
27 </owl:Class>
```

As duas definições (*Interval30Minutes* e *Interval120Minutes*) do Documento 18 utilizam a propriedade *durationDescriptionDataType*, uma propriedade simples de um intervalo (classe *Interval* de *OWL-Time*) que utiliza o tipo de dados *duration* do esquema XML na sua faixa de valores. *PT30M* (linha 10 do Documento 18) e *PT120M* (linha 23 do Documento 18) são valores descritos pelo tipo de dados *duration* do esquema XML e significam, respectivamente, 30 minutos e 120 minutos.

A definição da sessão *Session30-120minutesDuration* restringe os valores de *minDeliveryDuration* e *maxDeliveryDuration* para as classes *Interval30Minutes* e *Interval120Minutes*, respectivamente, tal como ilustrado no Documento 19 (linhas 8 e 15).

Por fim, um serviço modelado nas plataformas RALOWS ou RALOWS-Gen pode ter uma entrada (*Input*) chamada *DateTimeSession* (linhas 3 a 5 do Documento 20) do tipo *Session30-120minutesDuration*. Isto significa que uma sessão de execução desse serviço deve ter uma duração mínima de 30 minutos e máxima de 120 minutos.

Documento 19 Definição da classe *Session30-120minutesDuration*.

```
1  <!-- Defining the class
        Session30-120minutesDuration -->
2
3  <owl:Class rdf:ID="Session30-120minutesDuration">
4   <rdfs:subClassOf rdf:resource="#DeliveryDuration"/>
5   <rdfs:subClassOf>
6    <owl:Restriction>
7     <owl:onProperty rdf:resource="#minDeliveryDuration"/>
8     <owl:allValuesFrom rdf:resource="#Interval30Minutes"/>
9    </owl:Restriction>
10  </rdfs:subClassOf>
11
12  <rdfs:subClassOf>
13   <owl:Restriction>
14    <owl:onProperty rdf:resource="#maxDeliveryDuration"/>
15    <owl:allValuesFrom rdf:resource="#Interval120Minutes"/>
16   </owl:Restriction>
17  </rdfs:subClassOf>
18 </owl:Class>
```

Documento 20 Definição da entrada *DateTimeSession*.

```
1  <!--Defining the input DateTimeSession
               for the experiment-->
2
3  <process:Input rdf:ID="DateTimeSession">
4   <process:parameterType rdf:resource="#Session30-120minutesDuration"/>
5  </process:Input>
```

4.3.3 *Matching* na disponibilidade do serviço

Na Seção 4.3.2 foram definidos conceitos temporais para experimentos e recursos. No *matching* da disponibilidade do serviço, os conceitos temporais são utilizados para executar o *matching* baseado na disponibilidade do serviço e dos recursos envolvidos. Conceitos como *ResourceAvailability*, *ServiceAvailability* e *SessionDuration* (definidos na Seção 4.3.2), por exemplo, são comparados para realizar a descoberta do serviço requerido.

O estágio de *matching* na disponibilidade considera a disponibilidade do serviço e seus recursos e utiliza a classificação de Paolucci et al. [2002]. O objetivo desse estágio é descobrir se os serviços encontrados nos estágios anteriores têm ou não tempo disponível (*ServiceAvailability*) para execução e se a sua execução pode ser agendada para a data e horário requeridos (*ReqAvailability*).

A Tabela 4.6 mostra que, se o tempo requerido (*ReqAvailability*) for igual ao tempo disponível (*ServiceAvailability*), ou se o tempo requerido for um sub-conceito do tempo disponível, o *matching* tem *ranking* igual a 0. Esse valor reflete o fato de que se o

Tabela 4.6: Classificação para *matching* na disponibilidade do serviço

Match	Explicação	Classificação
Exact	Se ReqAvailability e ServiceAvailability são equivalentes	0
PlugIn	Se ReqAvailability é sub-conceito de ServiceAvailability	0
SubSume	Se ReqAvailability é super-conceito de ServiceAvailability	3
Fail	Em qualquer outro caso, o *match* é dito falho	3

tempo requerido for igual (*Exact*), ou estiver contido (*PlugIn*), no tempo disponível, o serviço está disponível para execução ou para agendamento prévio. Se mais tempo (*Subsume*) do que o disponível for requerido, o serviço não possui disponibilidade suficiente para sua execução (na data e horário requeridos).

4.4 Trabalhos relacionados

O processo de descoberta descrito neste capítulo é similar a trabalhos que envolvem a utilização de *matching* semântico por meio de inferência baseada em *subsumption* e a trabalhos que também utilizam graus de similaridade para prover classificação dos serviços descobertos.

Alguns algoritmos para *matching* em Serviços Web Semânticos com OWL-S são reportados na literatura. Esses algoritmos são baseados principalmente na capacidade (funcionalidade) dos serviços. Esses algoritmos verificam o *matching* nas entradas e saídas do serviço, e alguns deles levam em consideração também a categoria dos serviços.

Paolucci et al. [2002] apresentam um algoritmo de *matching* semântico baseado nas funcionalidades do serviço. O algoritmo considera as entradas e saídas e define um grau de *matching* calculado com a similaridade entre as entradas e saídas dos serviços requeridos e anunciados. São quatro os graus de *matching* definidos por Paolucci et al. [2002]: *exact*, *plugin*, *subsume* e *fail*. Esses graus de *matching* são utilizados diretamente neste trabalho nas fases de seleção do serviço, enquanto que para as fases de matching entre as funcionalidades do serviço requerido e do serviço, anunciado novos graus de similaridade são propostos no presente trabalho.

Sycara et al. [2003] propõem um mecanismo para *matching* que implementa o algoritmo de Paolucci et al. [2002] e adiciona *matching* semântico ao UDDI. Srinivasan et al. [2004; 2006] apresentam um mapeamento entre o perfil do serviço em OWL-S para o UDDI e estende as propostas de Sycara et al. [2003] e Paolucci et al. [2002] implementando *matching* também nas propriedades de categorização do OWL-S. O algoritmo descrito neste trabalho na Seção 4.1 também utiliza as propriedades de categorização do OWL-S como proposta para classificação dos serviços descobertos nas

fases de *matching* da funcionalidade. A implementação da descoberta apresentada no Capítulo 6 deste trabalho utiliza o mapeamento do OWL-S para o UDDI proposto por Sycara et al. [2003] e estendido mais tarde por Srinivasan et al. [2004; 2006].

Outros trabalhos geralmente estendem o *matching* baseado em *subsumption* com outras técnicas, tais como técnicas de recuperação da informação e lógica difusa.

Klusch et al. [2006] usam métricas de similaridade baseadas em técnicas de recuperação da informação para aperfeiçoar o processo de *matching* no perfil do serviço em OWL-S. Os autores estendem os trabalhos de Paolucci et al. [2002] e Sycara et al. [2003] criando novos graus de similaridade baseados agora nas métricas de recuperação da informação que eles utilizam. A proposta dos autores é a de efetuar um *matching* híbrido que utiliza inferência baseada em *subsumption* e em recuperação da informação.

Fenza et al. [2008] também apresentam *matching* híbrido, porém os autores utilizam lógica difusa em um segundo momento da descoberta: nos casos em que não ocorreu o *matching* exato entre os serviços disponíveis. Os autores estabelecem graus de relevância para cada tipo de parâmetro utilizado para a descoberta. Os parâmetros, na ordem de relevância, são as entradas, saídas, pré-condições, resultados e descrições textuais. Dessa forma, assim como no presente trabalho, eles também levam em consideração as pré-condições necessárias à execução do serviço, além de outros parâmetros.

Em Lomuscio et al. [2007], os autores propõem inferência baseada em restrições temporais para Serviços Web. Os autores têm foco na utilização das descrições temporais para sincronização entre serviços no momento da composição. Já neste trabalho o foco é na disponibilidade do serviço e no momento da sua descoberta.

O algoritmo discutido neste trabalho leva em consideração a categoria do serviço, suas entradas e saídas e, ainda, diferentemente dos algoritmos apresentados nesta seção, considera as pré-condições envolvidas na execução do serviço e a disponibilidade para execução ou agendamento do serviço. Este trabalho, conforme apresentado na Seção 4.2, também propõe a utilização de *matching* semântico em outras tarefas relacionadas aos Serviços Web Semânticos, além de pura e simplesmente na descoberta: publicação e composição.

4.5 Considerações finais

Este capítulo apresentou um processo de descoberta automática de Serviços Web Semânticos baseado em um algoritmo que executa *matching* semântico entre a funcionalidade requerida pelo usuário e os serviços anunciados. As principais características do algoritmo descrito na Seção 4.1 são: *matching* baseado em *subsumption*; utilização de graus de similaridade; classificação por categoria e pré-condições;

seleção final do serviço baseada no grau de similaridade das funcionalidades e das fases de classificação; e a possibilidade de um estágio extra para realizar seleção do serviço com base na sua disponibilidade.

O processo de descoberta baseado no *matching* semântico em OWL-S apresentado neste capítulo pode: (i) descobrir um ou mais serviços que atendem 100% ao que foi solicitado; (ii) não descobrir serviço algum; (iii) ou descobrir vários serviços (sub-serviços) que atendem parcialmente à solicitação do usuário. O primeiro é o caso ótimo: a solicitação do usuário pode ser resolvida com a execução de um único serviço. O segundo é o pior caso: a solução pode ser a busca por serviços similares com base nas categorias do serviço. O terceiro caso pode ser o mais comum: a maioria dos serviços da Web são composições fixas de mais de um serviço; por exemplo, lojas *on-line* são composições de serviços de busca de produtos, carrinho de compras, métodos de pagamento, serviços de boleto, serviços de cartão de crédito, etc.

As composições fixas de serviços são realizadas em tempo de criação e de codificação dos serviços. Alguns padrões da indústria são utilizados para descrever a composição de serviços de forma manual e estática e garantir que a composição seja executada. O WSBPEL [Andrews et al., 2003; Arkin et al., 2005] é um exemplo desse tipo de padrão, em que o desenvolvedor do serviço planeja e codifica a composição manual explorando seu conhecimento sobre as funcionalidades que o Serviço Web deverá prover.

Para automatizar a composição do serviço, a expertise do programador deve ser codificada em um padrão que seja passível de ser processado e interpretado por agentes de software. Segundo Martin et al. [2007a], nenhum dos padrões da indústria é capaz de codificar esse tipo de conhecimento. A linguagem OWL-S tem potencial para promover a realização de composição automática de Serviços Web porque possibilita a descrição semântica das funcionalidades, das condições necessárias à execução e do controle do fluxo e do fluxo de dados dos Serviços Web.

A descoberta automática de serviços é um passo essencial visando a composição automática. Dessa forma, este capítulo também apresentou na Seção 4.2 a utilização da descoberta nos processos de publicação e composição de serviços. Essa forma de utilização da descoberta está detalhada no Capítulo 5 que trata da composição automática de serviços.

CAPÍTULO
5

Composição de Serviços Web Semânticos

O processo de descoberta automática de Serviços Web Semânticos apresentado no Capítulo 4 pode não ser concluído com sucesso, ou seja, não encontrar serviços que atendam os requisitos de funcionalidade solicitados pelo usuário. Esse fato sugere a oportunidade e o desafio de se utilizar serviços que atendem parcialmente a solicitação do usuário para compor serviços de forma automática que, em conjunto, atendam a funcionalidade esperada pelo usuário.

Composição automática de Serviços Web Semânticos, que é uma das contribuições deste trabalho, é o processo de automaticamente *planejar*, *selecionar* e *executar* serviços Web para alcançar o objetivo do usuário [Ankolekar, 2003].

O *planejamento* tem o foco de, a partir de uma tarefa a ser executada, encontrar (descobrir) serviços que, em conjunto, realizem a tarefa esperada, e além disso montar um plano (*workflow*) de execução para esses serviços. O planejamento pode ser efetuado com base nos atributos funcionais do serviço (funcionalidade ou tarefa que o serviço executa) e é dessa forma que é realizado neste trabalho.

O objetivo da *seleção* é sempre o de otimizar a composição. A seleção é uma atividade pós-planejamento e visa selecionar os melhores serviços utilizando os atributos funcionais e também pode se basear em atributos não funcionais (*QoS*, custo do serviço, disponibilidade, localização geográfica, dentre outros).

Por fim, a fase de *invocação* (ou execução) deve garantir a execução da composição efetuada nas fases de descoberta, planejamento e seleção de serviços. Em outras palavras, a invocação deve garantir, dentre outras coisas, a tolerância a falhas

(disponibilidade dos serviços envolvidos na composição) e o *QoS* requerido na fase de seleção.

Este capítulo apresenta os resultados da modelagem de composição automática de Serviços Web sob a forma de grafos direcionados e com custos estabelecidos. O trabalho desenvolvido nesta tese apresenta resultados em planejamento e seleção (também envolvendo descoberta) de Serviços Web para realizar a composição com base nos atributos funcionais e não-funcionais do serviço. A fase de invocação automática dos serviços envolvidos na composição não é o objetivo deste trabalho e por esse motivo a invocação automática não é abordada neste capítulo.

Nos resultados apresentados neste capítulo, as fases de planejamento e seleção estão fundidas em uma única etapa, pois o planejamento é realizado à medida que os serviços são selecionados para a composição. Inicialmente o problema da composição foi modelado como um grafo direcionado e sem diferenciação de custos, no qual foi aplicado o algoritmo de Dijkstra para cálculo do caminho de custo mínimo. Os resultados foram analisados e verificou-se que, apesar de realizar uma composição com as funcionalidades requeridas pelo usuário, os serviços selecionados para a composição, em relação à funcionalidade do serviço requerido (atributos funcionais) não foram os esperados.

Em um segundo momento, foram aplicados custos relativos à funcionalidade (entradas e saídas) dos serviços com o objetivo de tornar as composições obtidas pelo algoritmo de Dijkstra próximas às composições esperadas como ideais em relação à requisição do usuário. Ou seja, o objetivo da aplicação dos custos era o de otimizar a seleção dos serviços para a composição final.

As composições obtidas sem a aplicação de custos diferenciados foram então comparadas com as composições obtidas levando-se em consideração os custos relativos aos atributos funcionais. Os resultados foram confrontados e verificou-se que a consideração dos custos contribuíram para uma melhor seleção dos serviços.

Este capítulo apresenta na Seção 5.1 o modelo em grafo utilizado para a execução do cálculo de caminhos de custo mínimo na composição de serviços. A Seção 5.2 apresenta as políticas utilizadas para a atribuição dos custos funcionais e não-funcionais no modelo em grafo. Os resultados referentes à aplicação do algoritmo de cálculo do caminho de custo mínimo para a composição automática de serviços estão apresentados na Seção 5.3. A Seção 5.4 apresenta os resultados referentes à geração do grafo que representa a composição final. Uma avaliação da composição em relação à requisição inicial do usuário é apresentada na Seção 5.5. A Seção 5.6 apresenta a estratégia utilizada para gerar a descrição da composição, utilizando os construtores de fluxo da linguagem OWL-S, a partir do grafo da composição final. As Seções 5.7 e 5.8 apresentam, respectivamente, os trabalhos relacionados à composição automática de Serviços Web Semânticos e as considerações finais

referentes a este capítulo.

5.1 Modelo em grafo para Serviços Web Semânticos

Para realizar a composição automática utilizando algoritmos de cálculo do caminho de custo mínimo, o repositório de Serviços Web Semânticos foi modelado como um grafo direcionado e com custos que foram determinados e inseridos no grafo conforme detalhado na Seção 5.2.

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*) a uma entrada de um outro serviço **B**, então é criada uma aresta de **A** para **B**.

O grafo é criado ou modificado em dois momentos: durante a publicação do serviço e no momento da composição. Cada serviço é inserido no grafo no momento em que o serviço é criado e adicionado ao repositório UDDI (publicação do serviço), tal como descrito na Seção 5.1.1. No momento da composição, o grafo e alguns dos seus custos são modificados para atender aos requisitos do usuário (entradas disponíveis e saídas requeridas são adicionadas ao grafo), conforme descrito na Seção 5.1.2.

5.1.1 Publicação do serviço

O grafo é criado apenas no momento da publicação do serviço. Dessa forma, o grafo não precisa ser montado toda vez que uma composição é requerida. Todos os serviços do repositório UDDI, que pode estar distribuído entre diversos UDDIs na Internet, fazem parte do grafo, e por isso a abordagem de se criar o grafo no momento da publicação do serviço e não no ato da composição. Essa abordagem foi adotada pois, a partir de uma quantidade de serviços, o tempo para a criação do grafo inteiro no momento da composição seria um limitador da abordagem (avaliações estão apresentadas na Seção 6.3).

Uma decisão tomada no momento da modelagem do grafo foi que um par de serviços são conectados, com aresta de saída, apenas uma vez (conforme ilustrado na Figura 5.1), mesmo que tenham, na prática, mais de um par entrada/saída "compatíveis". Isto ocorre com cada par para o qual pelo menos uma entrada de um dos pares é do mesmo tipo da saída do outro par, baseado na ontologia de domínio que descreve as entradas e saídas. Essa decisão foi tomada pois não fazia sentido existirem duas arestas conectando os mesmos nós de um grafo e na mesma direção. Pode-se verificar que essa decisão tem a limitação de não corresponder à realidade pois, entre um par de serviços, poderia haver mais de um par entrada/saída correspondentes entre si. Porém, esta limitação é atenuada pelas políticas de custo estabelecidas neste trabalho na Seção 5.2.

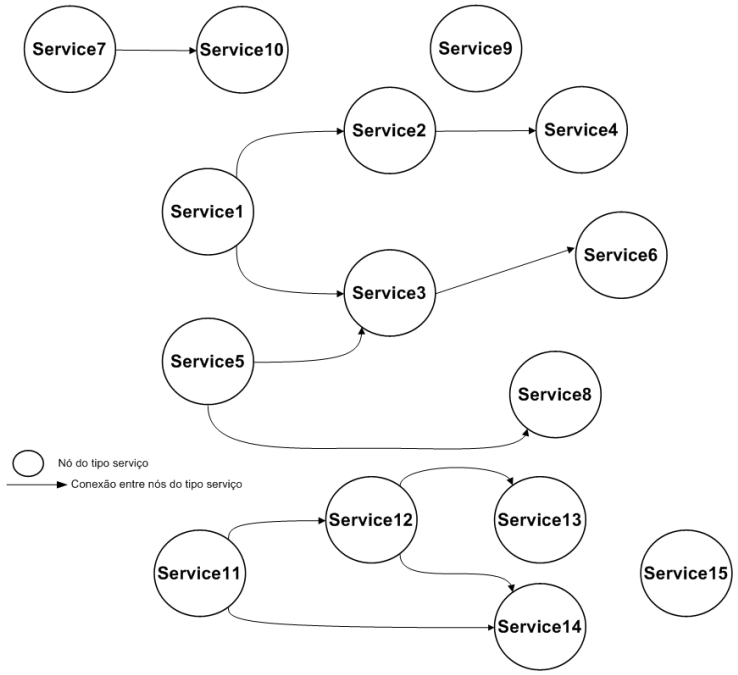

Figura 5.1: Repositório de serviços como grafo.

Uma vez criado o grafo, ele é armazenado e pode ser modificado se ocorrer alguma alteração no UDDI (inclusão ou exclusão de serviços). Além disso, o grafo também é modificado no momento da composição (Seção 5.1.2), no momento em que o usuário informa as funcionalidades (entradas e saídas) do serviço requerido. A Figura 5.1 ilustra parte do grafo montado a partir de um repositório UDDI. Os exemplos e os resultados apresentados neste capítulo são baseados nesse sub-conjunto de quinze serviços. A quantidade de quinze serviços foi escolhida apenas como forma de ilustrar em uma figura legível como os serviços estão conectados entre si no grafo.

Uma característica importante observada na Figura 5.1 é que trata-se de um grafo desconexo. Isto ocorre devido ao fato de que serviços existentes em repositórios UDDI têm, muitas vezes, funcionalidades distintas e dessa forma, é natural que não existam conexões (entradas/saídas) semânticas entre muitos dos serviços.

O Documento 21 apresenta o algoritmo para publicação de um novo serviço no repositório. Esse algoritmo também é responsável por atualizar o grafo cada vez que um novo serviço é registrado 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. A Seção 4.2 apresentou como o *matching* semântico é utilizado no presente trabalho para realizar a conexão semântica entre dois serviços no momento da publicação.

O procedimento *insereServico* (na linha 1 do Documento 21) recebe como parâmetro a descrição em OWL-S do serviço a ser publicado. Inicialmente um novo nó é

Documento 21 Algoritmo para inserir um novo serviço.

```
1  procedimento insereServico (owlsServico) retorna booleano
2  inicio
3    grafo.adicionaVertice();
4    para i=1 até Nservico faça //para cada serviço no repositório UDDI
5    inicio
6      se existe(saida em servico[i] similar a entrada em owlsServico) entao
7        //existe ao menos uma saida em servico[i]
8        //que conecta com entrada em owlsServico
9        grafo.insereAresta(servico[i], owlsServico, 0);
10
11     se existe(saida em owlsServico similar a entrada em servico[i]) entao
12       //existe ao menos uma saida em owlsServico
13       //que conecta com entrada em servico[i]
14       grafo.insereAresta(owlsServico, servico[i], 0);
15   fim para
16   //registra o serviço no UDDI
17   retorna registraUDDI(owlsServico);
18 fim insereServico
```

adicionado ao grafo na linha 3 do Documento 21. Da linha 4 até a linha 15 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 6 a 14 verificam se existe conectividade e, caso exista, novas arestas são inseridas no grafo.

As novas arestas criadas podem ser tanto conectando um serviço já existente no grafo ao novo serviço (novo nó no grafo) como o contrário. O primeiro caso pode ser visto na linha 9 do Documento 21: ao *servico[i]* (do repositório) é adicionada uma aresta direcionada ao novo serviço (*owlsServico*) com o custo igual a 0. O segundo caso é mostrado na linha 14 do Documento 21: ao *owlsServico* (novo serviço) é adicionada uma aresta direcionada a um serviço do repositório (*servico[i]*) utilizando o custo também igual a 0.

Inicialmente o grafo foi criado com custos nulos para avaliar o comportamento do algoritmo de cálculo do caminho de custo mínimo tal como apresentado na Seção 5.3.1 e depois traçar um comparativo com o comportamento do algoritmo após estabelecidas as políticas de custos (Seção 5.3.2). Os detalhes a respeito dos custos, tais como seus valores, seus objetivos, a forma como os custos são interpretados no grafo, dentre outros, foram propositalmente omitidos nesta seção. Esses detalhes estão descritos na Seção 5.2 em que o algoritmo apresentado no Documento 21 é modificado para refletir a política de custo estabelecida.

5.1.2 Entradas e saídas requeridas

Uma composição satisfatória do ponto de vista das funcionalidades (entradas e saídas) requeridas pelo usuário deve: priorizar a utilização das entradas disponíveis,

evitar a utilização de entradas não informadas pelo usuário (possivelmente indisponíveis), e garantir as saídas requeridas. Tomando como exemplo um serviço de vendas de livros e um usuário que possui apenas as informações de "nome do livro" e "nome do autor" (entradas disponíveis), e requer as saídas "preço do livro" e "pagamento por boleto", o processo de descoberta e composição deve:

- **evitar** serviços que exijam, adicionalmente às informações disponíveis ao usuário, o "ISBN" referente ao livro procurado;
- **priorizar** a utilização das entradas "nome do livro" e "nome do autor", embora seja aceitável que se alcance as saídas requeridas apenas com um sub-conjunto dessas entradas;
- e **garantir** as saídas "preço do livro" e "pagamento por boleto", embora seja aceitável que o serviço descoberto, ou composto, ofereça saídas adicionais.

Em resumo, uma composição ideal deve garantir que o usuário alcance o máximo dos resultados esperados com um mínimo de esforço. Isto implica garantir todas as saídas requeridas e não utilizar entradas não informadas pelo usuário. Entretanto, o processo de composição deve ser flexível o suficiente para oferecer composições alternativas caso não seja possível compor serviços com as informações disponíveis e requeridas pelo usuário.

Para aplicar o cálculo da composição de serviços usando algoritmos de cálculo do caminho de custo mínimo foram utilizados dois artifícios no grafo no momento da composição. Esses artifícios correspondem à adição de alguns nós "artificiais" que não correspondem a serviços do repositório UDDI. Utilizando o grafo da Figura 5.1 e adicionando os nós artificiais tal como ilustrado na Figura 5.2 têm-se que:

- foi criado um nó raiz único (*Node0* na Figura 5.2) que possui arestas apontando para todos os serviços que possuem ao menos uma das entradas disponíveis (nós *InR1* a *InR4* na Figura 5.2);
- foi criado um nó para cada saída requerida (nós *OutR1* a *OutR5* na Figura 5.2) para as quais cada serviço que possui a saída aponta;

Esses dois artifícios garantem que todas as saídas requeridas, se existirem no grafo, estarão na composição final e também possibilitam diminuir o espaço de busca do algoritmo de cálculo do caminho de custo mínimo, pois esse pode partir de um ponto único (o nó das entradas disponíveis - *Node0* na Figura 5.2). As outras entradas e saídas de cada serviço foram omitidas do grafo da Figura 5.2 apenas para permitir uma melhor visualização.

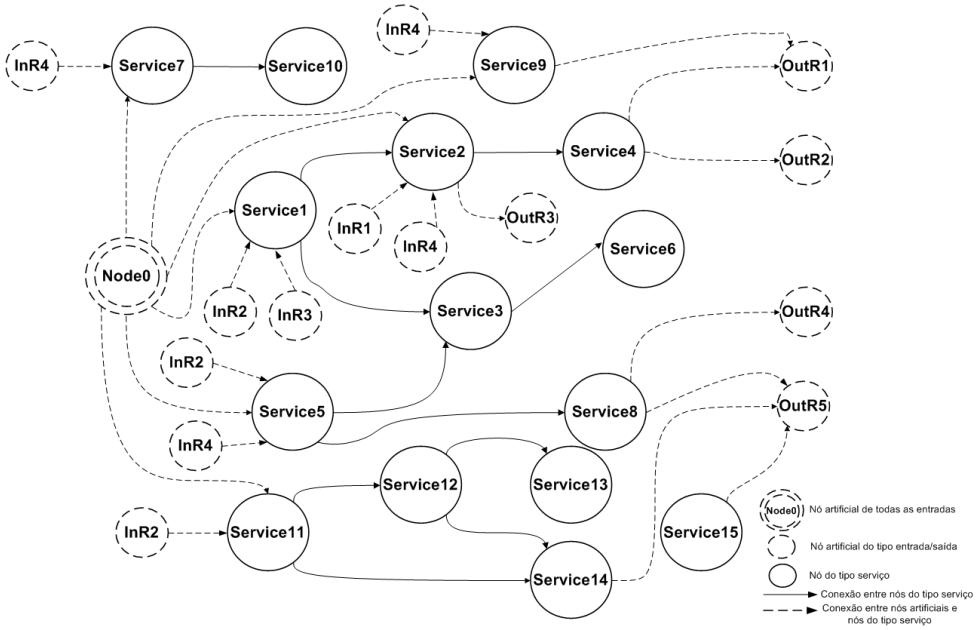

Figura 5.2: Grafo com nó de entrada e nós de saídas "artificiais".

A utilização desses artifícios apresenta a limitação de que não garante o uso de todas as entradas disponíveis e pode até forçar o uso de entradas não disponíveis. Porém, essa limitação é intencional e ao mesmo tempo estratégica, dado que permite sugerir ao usuário composições com outras entradas, caso seja impossível com as entradas disponíveis. Essa limitação é eliminada, ou atenuada, com a adoção de políticas de custos como apresentado na Seção 5.2.

Ao se comparar a Figura 5.1 com a Figura 5.2 deve-se perceber que alguns nós foram adicionados: eles correspondem às entradas e saídas requeridas pelo usuário no momento da busca pelo serviço. Neste trabalho, no processo de descoberta e composição de serviços, apenas as saídas são chamadas de requeridas, pois as entradas correspondem à informação que o usuário tem disponível para alcançar as saídas (resultados do serviço) requeridas.

O Documento 22 apresenta o algoritmo utilizado para adicionar os nós artificiais referentes às entradas disponíveis e saídas requeridas no grafo. O procedimento *conectaDispReqGrafo*, na linha 1 do Documento 22, recebe dois parâmetros: a lista das entradas disponíveis (*entradasDisp*) e a lista das saídas requeridas (*saidasReq*). Esse algoritmo é executado cada vez que um usuário solicita uma composição e as mudanças ocasionadas no grafo não devem interferir no grafo original. Dessa forma, a linha 3 mostra a criação de uma cópia do grafo original.

O objetivo do algoritmo do Documento 22 é inserir no grafo os nós artificiais tal como ilustrado na Figura 5.2. As linhas 6 a 10 do Documento 22 criam os nós *Node0* e todos os nós de saídas requeridas, inicialmente sem nenhuma aresta. A seguir, as

Documento 22 Algoritmo para inserir os nós artificiais no grafo.

```
1  procedimento conectaDispReqGrafo (entradasDisp, saidasReq)
2  inicio
3    grafoAux = grafo;
4    //o nó zero é guardado sempre para criar
5    //o nó artificial das entradas disponíveis
6    grafoAux.adicionaVertice(0);
7
8    //para cada saída requerida cria um nó artificial
9    para i=1 até saidasReq.tamanho faça
10     grafoAux.adicionaVertice(saidasReq[i]);
11
12   para j=1 até Nservico faça //para cada serviço no repositório UDDI
13   inicio
14     se existe(entrada em servico[j] similar alguma entradasDisp) entao
15       //existe ao menos uma entrada disponível em servico[j]
16       grafoAux.insereAresta(Node0, servico[j], 0);
17
18     para i=1 até saidasReq.tamanho faça //para cada saída requerida
19       se existe(saida em servico[j] similar saidasReq[i]) entao
20         //existe ao menos uma saída requerida em servico[j]
21         grafoAux.insereAresta(servico[j], saidasReq[i], 0);
22     fim para
23   fim para
24 fim insereServico
```

arestas começam a ser adicionadas, nas linhas 12 a 23, conectando os novos nós ao grafo original. A linha 12 faz percorrer todos os serviços contidos no repositório, que são os mesmos serviços contidos no grafo original.

Para cada serviço no repositório que possua ao menos uma entrada disponível cria-se uma aresta partindo do *Node0* para o serviço em questão com peso igual a 0 (linha 13 a 16 do Documento 22). A Figura 5.2 ilustra exemplos dessas arestas que saem do *Node0* para serviços.

No caso das saídas requeridas, para cada serviço no repositório que possua ao menos uma saída requerida cria-se uma aresta partindo desse serviço para a saída requerida em questão com peso igual a 0 (linha 21 do Documento 22). É importante notar, tal como ilustra a Figura 5.2, que cada saída é um nó no grafo, diferentemente das entradas. Assim, é necessário verificar a conectividade de cada serviço para cada saída requerida.

Da mesma forma que na Seção 5.1.1, os custos ainda continuam nulos e por esse motivo o algoritmo do Documento 22 também será modificado na Seção 5.2 para refletir a política de custo estabelecida.

5.2 Políticas de custo propostas

Conforme descrito na Seção 5.1, inicialmente o grafo foi criado com pesos nulos nas arestas, o que é equivalente a dizer que todas as arestas têm o mesmo custo. Essa abordagem inicial, simples, visava analisar o comportamento do algoritmo de cálculo do caminho de custo mínimo para a composição de serviços.

O principal problema encontrado foi que as composições não se aproximavam das ideais: ao se comparar as composições geradas com a composição tida como ideal por um conhecedor do serviço, os resultados não eram os esperados. Naturalmente, o uso de algoritmos de cálculo do caminho de custo mínimo em grafo faz sentido nos casos em que existem custos associados às arestas: ficou claro que era necessário estabelecer uma política de custos para eliminar os serviços "não desejados" da composição. A Seção 5.3 apresenta a comparação da execução do algoritmo de cálculo do caminho de custo mínimo com custos nulos e com os custos tal como estabelecidos nesta seção.

Custos em grafos são estabelecidos nas arestas que conectam um nó a outro, isto denota que existe um custo associado a uma mudança de um dado nó para o seu posterior. Porém, no presente trabalho assume-se que o custo está associado a cada serviço (a cada nó no grafo de serviços da Figura 5.2). O objetivo é que o custo esteja associado à utilização do serviço. Para modelar essa característica toda aresta de entrada de um nó do tipo serviço tem o valor do custo total do serviço que recebe a aresta. Essa visão não chega a ser uma mudança na estrutura do grafo, é apenas uma forma de interpretação que serve para representar o custo de utilização de cada serviço. Na prática, quer dizer que o custo da aresta corresponde ao custo de utilização do próximo serviço e faz o algoritmo de cálculo do caminho de custo mínimo decidir qual o melhor caminho a ser escolhido no grafo. Assim, custo neste trabalho sempre será chamado de peso ou custo do serviço.

Devido ao fato de que a descoberta e a composição dos serviços têm como objetivo inicial encontrar um serviço ou uma composição que atenda aos requisitos de entrada e saída (funcionalidade) do usuário, a proposta principal para política de custo foi a de estabelecer custos mais altos para os serviços que possuem muitas entradas não disponíveis e saídas não requeridas.

O objetivo principal da composição nesse trabalho é o de encontrar composições que atendam a uma requisição de entradas e saídas do usuário. Nesse sentido, os custos estabelecidos visaram eliminar da composição serviços com entradas não disponíveis e saídas não requeridas, tal como descrito na Seção 5.2.1: os chamados custos funcionais. Todavia, outros custos, aqui chamados de custos não-funcionais e apresentados na Seção 5.2.2, podem ser estabelecidos e no presente trabalho estão divididos em dois grupos: custos não-funcionais de qualidade de serviço e custos

não-funcionais escolhidos pelo usuário.

5.2.1 Atributos funcionais

Como o objetivo principal da composição neste trabalho é a de encontrar uma composição que atenda às funcionalidades requeridas pelo usuário, o primeiro custo considerado foi o oferecimento de funcionalidades por parte dos serviços, as quais são dadas pelas entradas e saídas dos serviços. Ou seja, o primeiro objetivo da composição é encontrar serviços que combinados possam prover a funcionalidade que não foi encontrada em um único serviço no processo de descoberta apresentado no Capítulo 4.

Entretanto, como o grafo é criado no momento da publicação do serviço, quando ainda não é possível estabelecer quais são as entradas disponíveis ao usuário e quais são as saídas requeridas pelo usuário para realizar a composição, optou-se por estabelecer custos funcionais uniformes para todas as entradas e saídas dos serviços e, depois, oferecer descontos funcionais para cada entrada disponível e saída requerida em cada serviço.

Custos funcionais

A definição do valor de custos para as entradas e saídas é realizada utilizando como base os graus de similaridade definidos por Paolucci et al. [2002]. A idéia foi a de atribuir grau máximo para cada entrada e saída do serviço no momento da sua publicação. O objetivo era assumir *a priori* o pior caso: nenhuma entrada e nenhuma saída do serviço iria atender a requisição do usuário, ou seja, o *matching* seria igual a *Fail* e receberia o grau 3. A partir do grau 3 para cada entrada e saída do serviço o custo inicial para todo e qualquer serviço é calculado no momento da publicação da seguinte maneira:

$C_{is} = (N_{in} * 3) + (N_{out} * 3)$, em que:

- C_{is}: custo inicial do serviço;
- N_{in}: número de entradas do serviço que é multiplicado pelo grau 3;
- N_{out}: número de saídas do serviço que é multiplicado pelo grau 3.

Para inserir os custos no momento da publicação do serviço o algoritmo apresentado no Documento 21 na Seção 5.1.1 foi alterado tal como apresentado no Documento 23. Foram basicamente três alterações: na linha 4 foi inserido o cálculo do custo para o novo serviço; na linha 10, o custo inicial para o novo serviço é utilizado em todas as novas arestas que apontam para o novo serviço; e na linha 15 os custos

dos serviços que apontam para o novo serviço são utilizados nas novas arestas que saem de algum serviço para o novo serviço.

O custo inicial para cada serviço do repositório (*servico[i].ci* na linha 15 do Documneto 23) foi calculado, no momento da sua publicação, exatamente da mesma forma que o custo para o novo serviço é calculado na linha 4 do Documento 23.

Documento 23 Algoritmo para inserir um novo serviço com o custo inicial.

```
1  procedimento insereServico (owlsServico) retorna booleano
2  inicio
3    grafo.adicionaVertice();
4    ci = (owlsServico.Nin*3) + (owlsServico.Nout*3);
5    para i=1 até Nservico faça //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
```

Uma vez que o grafo tem os custos iniciais calculados e associados aos serviços no momento da sua publicação, isto pode ser utilizado pelo algoritmo de cálculo do caminho de custo mínimo no momento da composição do serviço. Porém, alguns serviços devem ter custos diferenciados e isto é esse fato que determina o comportamento do algoritmo.

Para determinar o comportamento do algoritmo de cálculo do caminho de custo mínimo em relação aos custos funcionais é proposto neste trabalho o conceito de descontos. Os descontos, como o próprio nome sugere, visam estabelecer descontos nos custos dos serviços que mais se aproximem da requisição inicial do usuário.

Descontos funcionais

Na Seção 5.2.1 foram definidos custos para todas as entradas e saídas de um serviço (nó do grafo). Porém, entradas disponíveis e saídas requeridas não devem possuir custos, pois fazem parte da requisição do usuário. Para cada entrada disponível e saída requerida o serviço recebe um "desconto" no seu custo inicial.

O objetivo dos descontos é forçar que o algoritmo de custos mínimos se comporte de forma a priorizar a escolha sempre por serviços que possuam entradas disponíveis

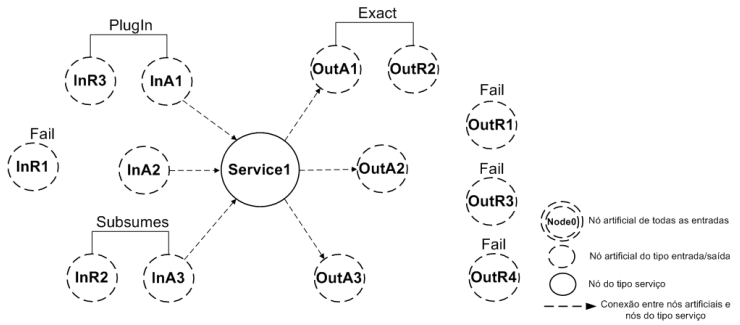

Figura 5.3: *Matching* pata atribuição dos descontos aos serviços.

e/ou saídas requeridas. A Figura 5.3 apresenta o processo de *matching* e atribuição dos graus de similaridades entre as entradas disponíveis e saídas requeridas. Na figura, cada entrada e saída fornecida pelo usuário no momento da requisição do serviço é comparada com as entradas e as saídas dos serviços no repositório, da mesma forma que é realizado no processo de descoberta, e os graus de *matching* são estabelecidos.

Todas as entradas e saídas do serviço recebem o custo inicial igual a 3 conforme apresentado na Seção 5.2.1, que corresponde ao grau de *matching* de distância maior de similaridade entre dois conceitos. Dessa forma, se o serviço possuir alguma entrada ou saída semanticamente similar a, respectivamente, alguma das entradas disponíveis ou saídas requeridas, o serviço deve obter um desconto.

A Tabela 5.1 apresenta os possíveis descontos para os serviços que possuem entradas similares às entradas disponíveis informadas pelo usuário na requisição pelo serviço. Essa tabela indica que o desconto 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. O desconto total em relação às entradas de um serviço é dado pela fórmula:

$DT_{in} = (\sum_{i=1}^{N_{in}} D_{in}) * \frac{1}{\lambda}$, em que:

- DT_{in}: desconto total em relação às entradas do serviço;
- N_{in}: número de entradas do serviço;
- D_{in}: desconto de cada entrada do serviço que é dado pela Tabela 5.1;
- λ: é um parâmetro ajustável pelo usuário. O valor padrão de λ é 1. Esse parâmetro pode ser utilizado para aumentar ou diminuir o desconto do serviço

em relação às entradas.

O λ é um parâmetro importante no desconto relativo às entradas. A composição final pode requerer entradas que o usuário não tenha disponível. Ao aumentar o valor de λ o desconto relativo às entradas diminui e, como resultado, os serviços que têm entradas não disponíveis terão custos ainda maiores, e o algoritmo de cálculo do caminho de custo mínimo irá buscar composições que, caso existam, requeiram um número menor de entradas de que o usuário não dispõe.

Tabela 5.1: Descontos funcionais para as entradas com base nos graus de *matching*.

Matching	**Ocorrência**	**Desconto**
Exact	Se InServico e InR são equivalentes	3
PlugIn	Se InServico é sub-conceito de InR	2
SubSume	Se InServico é super-conceito de InR	1
Fail	InServico e InR não possuem similaridade semântica	0

A Tabela 5.2 apresenta os possíveis descontos para os serviços que possuem saídas similares às saídas requeridas informadas pelo usuário na requisição pelo serviço. Essa tabela indica que o desconto 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. O desconto total em relação às saídas de um serviço é dado pela fórmula:

$DT_{out} = (\sum_{i=1}^{N_{out}} D_{out}) * \frac{1}{\lambda}$, em que:

- DT_{out}: desconto total em relação às saídas do serviço;
- N_{out}: número de saídas do serviço;
- D_{out}: desconto de cada saída do serviço que é dado pela Tabela 5.2;
- λ: é um parâmetro ajustável pelo usuário. O valor padrão de λ é 1. Esse parâmetro pode ser utilizado para aumentar ou diminuir o desconto do serviço em relação às saídas.

O parâmetro λ no desconto relativo às saídas tem a mesma função que a apresentada para o desconto relativo às entradas. Porém, a ocorrência de saídas não requeridas na composição final não causam tanto impacto em relação à utilização do serviço. O usuário pode apenas descartar as saídas que não são do seu interesse.

Tabela 5.2: Descontos funcionais para as saídas com base nos graus de *matching*.

Matching	Ocorrência	Desconto
Exact	Se OutServico e OutR são equivalentes	3
SubSume	Se OutServico é super-conceito de OutR	2
PlugIn	Se OutServico é sub-conceito de OutR	1
Fail	OutR e OutServico não possuem similaridade semântica	0

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.

De posse do desconto total do serviço, os custos dos serviços que possuem entradas disponíveis e saídas requeridas são alterados para os novos custos da seguinte forma:
$C_{es} = C_{is} - DT_s$, em que:

- C_{es}: custo efetivo do serviço;
- C_{is}: custo inicial do serviço;
- DT_s: desconto total do serviço.

Os custos e descontos funcionais "forçam" o algoritmo de cálculo do caminho de custo mínimo a encontrar caminhos partindo das entradas disponíveis e chegando nas saídas requeridas, e evitando serviços que possuem muitas entradas não disponíveis e saídas não requeridas. Os resultados da aplicação dos custos e descontos funcionais estão apresentados na Seção 5.3, que apresenta os resultados referentes à execução do algoritmo para cálculo do caminho de custo mínimo.

5.2.2 Atributos não-funcionais

Os atributos não-funcionais são todos os atributos de um serviço que não têm relação direta com a funcionalidade oferecida pelo serviço. Geralmente os atributos não-funcionais são parâmetros de Qualidade de Serviço (*QoS*) ou parâmetros definidos pelo usuário.

Kalepu et al. [2003] elencam um conjunto de atributos de *QoS* para Serviços Web, dentre eles estão disponibilidade, confiabilidade, latência, tempo de resposta, segurança e confiabilidade. Existem alguns trabalhos (e.g. [Zhou et al., 2004], [Kritikos and Plexousakis, 2007] e [Zhang et al., 2007]) que têm por objetivo estender a ontologia OWL-S com a adição de informações de *QoS*. Apesar de que *QoS* não é o foco deste trabalho, é possível utilizar os atributos de *QoS* para definir os custos dos serviços e, com isso, realizar composição automática com base em atributos de *QoS*.

Os tempos de execução dos serviços não são conhecidos *a priori*. Entretanto, com as descrições de OWL-S é possível definir o controle de fluxo e fluxo de dados internos a cada serviço por meio dos construtores da sub-ontologia de processos apresentada no Capítulo 2. Uma análise desses construtores (apresentados na Tabela 5.3) pode ser utilizada para definição do desempenho do serviço com base em análise de complexidade de algoritmos.

Além dos custos não-funcionais relativos aos atributos de *QoS*, os custos dos serviços podem ser customizados de acordo com as necessidades do usuário, levando em consideração o objetivo final da composição. O usuário pode escolher um dos parâmetros (entradas e saídas) que ele utilizou para a requisição do serviço como base para a definição dos custos dos serviços. Um exemplo disso é, por exemplo, compor serviço com preço final menor.

Custos relativos à complexidade do serviço

A linguagem OWL-S descreve o fluxo interno a cada serviço por meio de construtores para controle de fluxo e para fluxo de dados que fazem parte da sub-ontologia de processos do OWL-S apresentada no Capítulo 2. Alguns desses construtores representam construtores de linguagens de programação, são eles:

- *Sequence*: uma lista de componentes a ser executada em ordem;
- *Split*: um conjunto de componentes que devem ser executados de forma concorrente;
- *Split+Join*: um conjunto de componentes que devem ser executados de forma concorrente com sincronização. A execução completa no momento em que todos os componentes tiverem sido executados;
- *Any-Order*: um conjunto de componentes que devem ser executados sem ordem pré-estabelecida e sem concorrência. A execução completa no momento em que todos os componentes tiverem sido executados;
- *Choice*: escolha para execução de um único componente dado um conjunto de componentes;

- *If-Then-Else*: testa condição, se verdadeira executa um conjunto de ações, se não, executa outro conjunto de ações;
- *Repeat-While*: realiza iterações enquanto uma condição é verdadeira ou falsa;
- *Repeat-Until*: realiza iterações até uma condição se tornar verdadeira ou falsa.

A Tabela 5.3 apresenta a complexidade de cada construtor OWL-S tratados de forma isolada. Assim como em algoritmos, a complexidade final de cada serviço vai depender da sua lógica interna de execução que é dada pela combinação dos construtores apresentados na Tabela 5.3 e das suas complexidades.

Tabela 5.3: Complexidade dos construtores de fluxo da ontologia OWL-S.

Construtor OWL-S	Complexidade
Split	O(log n)
Split+Join	O(n log n)
Any-Order	O(n log n)
Choice	O(1)
If-Then-Else	O(1)
s*Repeat-While*	O(n)
Repeat-Until	O(n)

Cada processo composto, em um serviço descrito por OWL-S, utiliza os construtores de fluxo para descrever seu fluxo. Dessa forma, é possível analisar a complexidade de cada construtor de fluxo do OWL-S, analisar a complexidade de cada processo de um serviço, tal como é realizado em análises de complexidade de algoritmos, e atribuir um custo relativo para cada serviço com base nessa análise de complexidade. Serviços com processos que possuem ordem de complexidade maior deverão ter um custo de utilização maior. Os custos baseados na complexidade dos construtores de fluxo da ontologia OWL-S foram definidos tal como indicado na Tabela 5.3. Entretanto, esses custos não foram implementados para avaliação na infra-estrutura descrita no Capítulo 6.

Custos definidos pelo usuário

Os custos discutidos até esse ponto são custos "transparentes" ao usuário, ou seja, definidos no presente trabalho, e que visam garantir a funcionalidade requerida para a composição do serviço ou garantir composições baseadas na complexidade do serviço. Além desses custos, a abordagem proposta e implementada neste trabalho possibilita que o próprio usuário escolha o que é custo para ele. Assim, qualquer parâmetro do serviço (entrada, saída ou variáveis internas) pode se tornar o custo do serviço.

O principal exemplo do que seria o custo para um usuário é o preço de um produto ou serviço oferecidos por meio de um Serviço Web. O usuário pode escolher que quer serviços com menor preço, ou com menor distância geográfica da sua casa, ou ainda inverter o conceito de custo mínimo e escolher serviços com preços maiores, como por exemplo, sempre se hospedar em hotéis cinco estrelas.

Independentemente do custo escolhido, o algoritmo de cálculo do caminho de custo mínimo vai apresentar o mesmo comportamento: buscar caminhos que apresentem o custo mínimo para uma composição de serviços modelada como um grafo direcionado.

5.3 Algoritmo de cálculo do caminho de custo mínimo para a composição automática de Serviços Web

O algoritmo de Dijkstra foi utilizado neste trabalho para o cálculo dos caminhos de custo mínimo. Esse algoritmo possui complexidade da ordem de $O(N^2)$ para o pior caso e o tempo de execução para a composição de serviços utilizada no presente trabalho é apresentado na Seção 6.3. Algoritmos para cálculo do caminho de custo mínimo encontram um caminho de custo mínimo (se existente) entre todos os nós de um grafo conectado. Pode-se também utilizar o algoritmo para encontrar o caminho entre dois nós escolhidos e é dessa forma que o algoritmo de Dijkstra é empregado neste trabalho.

O objetivo da composição de serviços nesta tese é o de encontrar serviços que combinados produzam as saídas requeridas pelo usuário que requisitou o serviço. A Seção 5.1 descreve o nó *Node0*, que corresponde ao nó de todas as entradas disponíveis ao usuário, e os nós correspondentes às saídas requeridas.

A Figura 5.2 apresenta um grafo em que é possível visualizar os nós *OutR1* a *OutR5* que são os nós das saídas requeridas, bem como o nó *Node0*. Os nós das saídas requeridas são os nós "alvos" para o algoritmo de cálculo do caminho de custo mínimo. O algoritmo foi aplicado para calcular os caminhos de custo mínimo partindo do nó *Node0* para cada nó de saída requerida. O algoritmo, da forma como é aplicado neste trabalho, para realizar a composição de serviços, gera um caminho de custo mínimo para cada saída requerida. A Seção 5.4 apresenta a abordagem utilizada para gerar as composições de serviços a partir dos caminhos de custo mínimo.

Como forma de avaliar os resultados apresentados pelo algoritmo de Dijkstra na composição de serviços, o algoritmo foi aplicado inicialmente sem diferenciação de custos (todos os serviços com custo igual a "1") conforme descrito na Seção 5.3.1 e depois foram aplicados os custos e descontos funcionais (apresentados na Seção 5.2.1) e os resultados foram avaliados tal como descrito na Seção 5.3.2.

5.3.1 Cálculo dos caminhos sem diferenciação de custos

A implementação inicial do algoritmo de cálculo do caminho de custo mínimo foi sem diferenciação de custos nas arestas, ou seja, todas as arestas com os mesmos custos. Os resultados foram satisfatórios, do ponto de vista de que chegavam a uma composição que alcançava as saídas requeridas. Embora, muitas vezes, não fosse a composição ideal levando em consideração, principalmente, as entradas disponíveis as saídas requeridas.

A execução do algoritmo de cálculo do caminho de custo mínimo aplicado ao grafo da Figura 5.2 e sem diferenciação de custos gerou os caminhos (Figura 5.4-A, Figura 5.4-B, Figura 5.4-C, Figura 5.4-D e Figura 5.4-E) para cada saída requerida, conforme apresentado na Figura 5.4.

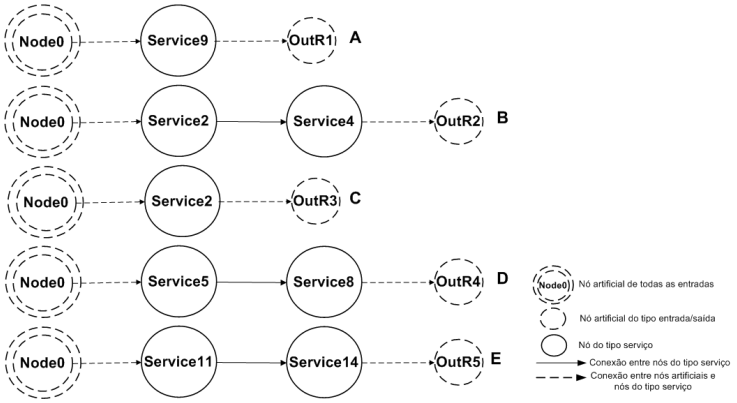

Figura 5.4: Caminhos parciais, sem diferenciação de custos, do nó 0 para todas as saídas requeridas.

Pode-se notar no grafo da Figura 5.2 que a saída requerida *OutR1* possui dois caminhos possíveis. O caminho da Figura 5.4-A foi escolhido pelo algoritmo e o caminho da Figura 5.5 não foi computado pelo algoritmo. Isto ocorreu devido ao fato de que, na implementação sem diferenciação de custos, todas as arestas possuem custos iguais e, por esse motivo, os caminhos que possuem menos arestas possuem, como conseqüência, custo menor.

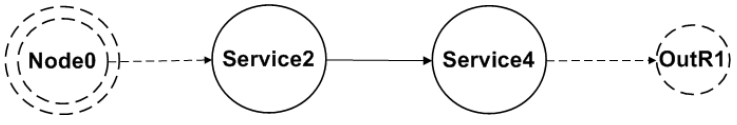

Figura 5.5: Caminho de *Node0* a *OutR1* não computado pelo algoritmo de cálculo do caminho de custo mínimo.

É importante notar que o caminho da Figura 5.5 seria mais interessante para a composição, pois o serviço *Service4* possui duas saídas requeridas enquanto que o

caminho da Figura 5.4-A possui apenas uma saída requerida.

Uma análise da saída requerida *OutR5*, que no grafo da Figura 5.2 possui dois caminhos possíveis e de igual tamanho (mesma quantidade de nós e arestas), identifica-se que o caminho da Figura 5.4-E foi escolhido pelo algoritmo e o caminho da Figura 5.6 não foi computado pelo algoritmo. Isso ocorreu porque todos os caminhos de igual tamanho, na implementação sem diferenciação de custos, possuem o mesmo custo e o algoritmo apenas apresenta um dos possíveis caminhos de custo mínimo (caminhos de custo mínimo podem não ser únicos).

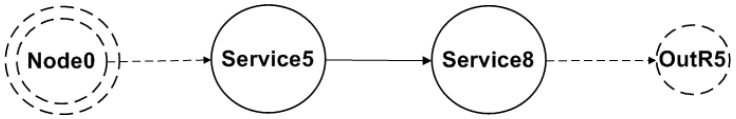

Figura 5.6: Caminho de *Node0* a *OutR5* não computado pelo algoritmo de cálculo do caminho de custo mínimo.

No caso da saída requerida *OutR5*, o caminho da Figura 5.6 seria mais "interessante" para a composição, pois o serviço *Service8* possui duas saídas requeridas enquanto que o caminho Figura 5.4-E possui apenas uma saída requerida.

Uma análise das entradas de cada serviço na Figura 5.2, e confrontado com os resultados obtidos pelo algoritmo, identifica-se que a abordagem sem diferenciação de custos não prioriza a escolha de serviços que possuem mais entradas disponíveis em comparação com outros. A análise dos resultados da execução do algoritmo com a aplicação dos custos e dos descontos funcionais é discutida na Seção 5.3.2.

5.3.2 Cálculo dos caminhos com diferenciação de custos nos atributos funcionais

Os custos foram estabelecidos conforme apresentado na Seção 5.2 com o objetivo de fazer com que o algoritmo selecionasse o melhor caminho de acordo com a funcionalidade requerida pelo usuário. A partir dai, o algoritmo começou a se comportar como esperado (composições ideais ou próximas das ideais). A aplicação de custos elevados associados às entradas indisponíveis e saídas não requeridas "forçam" o algoritmo a escolher caminhos contendo serviços com as funcionalidades mais próximas das requeridas.

Atribuição dos custos e descontos funcionais ao grafo

A Tabela 5.4 apresenta o custo das entradas, saídas e custo inicial (dado por C_{is}) para cada serviço do grafo apresentado na Figura 5.1 em que foi estabelecido que cada entrada e saída dos serviços do repositório receberiam inicialmente o custo máximo

igual a 3. Na Tabela 5.4 deve-se ler que, por exemplo, o serviço *Service1* tem custo igual a *15* pois possui duas entradas (custo *6*) e três saídas (custo *9*).

Tabela 5.4: Custo total dos serviços da Figura 5.1

Serviço	**Entradas**	**Saídas**	C_{is}
Service1	2	3	15
Service2	2	2	12
Service3	4	3	21
Service4	2	3	15
Service5	2	2	12
Service6	5	3	24
Service7	2	1	9
Service8	s	2	12
Service9	6	4	30
Service10	2	2	12
Service11	4	2	18
Service12	3	2	15
Service13	4	2	18
Service14	6	3	27
Service15	2	2	12

É importante observar que, se por algum motivo for necessário priorizar as entradas ou saídas, o atributo λ (apresentado na Seção 5.2.1) pode ser utilizado para ajustar os custos conforme desejado. Com o atributo λ é possível atribuir um peso maior ou menor aos descontos das entradas ou saídas. Para os resultados apresentados nas Tabelas 5.4 e Tabela 5.5, e na Figura 5.7, o valor padrão de $\lambda = 1$ foi mantido.

A Tabela 5.5 apresenta o custo inicial do serviço (dado por C_{is}), os descontos para as entradas disponíveis (dado por DT_{in}) e saídas requeridas (dado por DT_{out}), o desconto total atribuído ao serviço (dado por DT_s) e, por fim, o custo efetivo do serviço (dado por C_{es}).

O custo efetivo apresentado na Tabela 5.5 corresponde ao custo total do serviço menos o desconto provido para as entradas disponíveis e para as saídas requeridas. Ao se colocar custos nas entradas e saídas pode-se ter uma medida de quais serviços possuem mais entradas e saídas e quais possuem menos. Isto é importante para a avaliação da composição, pois serviços que possuem muitas entradas, por exemplo, serão custosos apenas nos casos em que as entradas não são disponíveis. Entretanto, não impossibilita do algoritmo escolher serviços com entradas não disponíveis caso o contrário não seja possível.

A Figura 5.7 ilustra o mesmo grafo apresentado na Figura 5.2 após a inclusão dos

Tabela 5.5: Custo efetivo dos serviços da Figura 5.2

Serviço	C_{is}	DT_{in}	DT_{out}	DT_s	C_{es}
Service1	15	6	0	6	9
Service2	12	6	3	9	3
Service3	21	0	0	0	21
Service4	15	0	6	6	9
Service5	12	6	0	6	6
Service6	24	0	0	0	24
Service7	9	3	0	3	6
Service8	12	0	6	6	6
Service9	30	3	3	6	24
Service10	12	0	0	0	12
Service11	18	3	0	3	15
Service12	15	0	0	0	15
Service13	18	0	3	3	15
Service14	27	0	3	3	24
Service15	12	0	3	3	9

custos com descontos da Tabela 5.5 nas arestas do grafo. A utilização dos descontos tem como objetivo priorizar, na composição, os serviços que possuem entradas disponíveis e saídas requeridas tal como discutido a seguir com a apresentação dos caminhos parciais.

Caminhos parciais

A execução do algoritmo de cálculo do caminho de custo mínimo aplicado ao grafo da Figura 5.7 com diferenciação de custos gerou os caminhos (Figura 5.8-A, Figura 5.8-B, Figura 5.8-C, Figura 5.8-D e Figura 5.8-E) para cada saída requerida, conforme apresentado na Figura 5.8.

As duas diferenças principais entre os caminhos gerados sem diferenciação de custos (Figura 5.4) e os caminhos gerados com diferenciação de custos (Figura 5.8) podem ser notadas nos caminhos escolhidos pelo algoritmo para as saídas *OutR1* e *OutR5*.

Os custos mais altos atribuídos aos serviços *Service9*, *Service11* e *Service14*, devido às suas entradas não disponíveis e os descontos concedidos aos serviços *Service4* e *Service8*, devido às suas outras saídas requeridas, adicionais em relação aos serviços *Service9* e *Service14* respectivamente, fizeram com que o algoritmo tivesse um comportamento melhor em relação ao apresentado na Seção 5.3.1.

Com a diferenciação dos custos, as saídas *OutR1* e *OutR5* tiveram seus caminhos selecionados passando por serviços que já possuíam outras saídas, otimizando, com

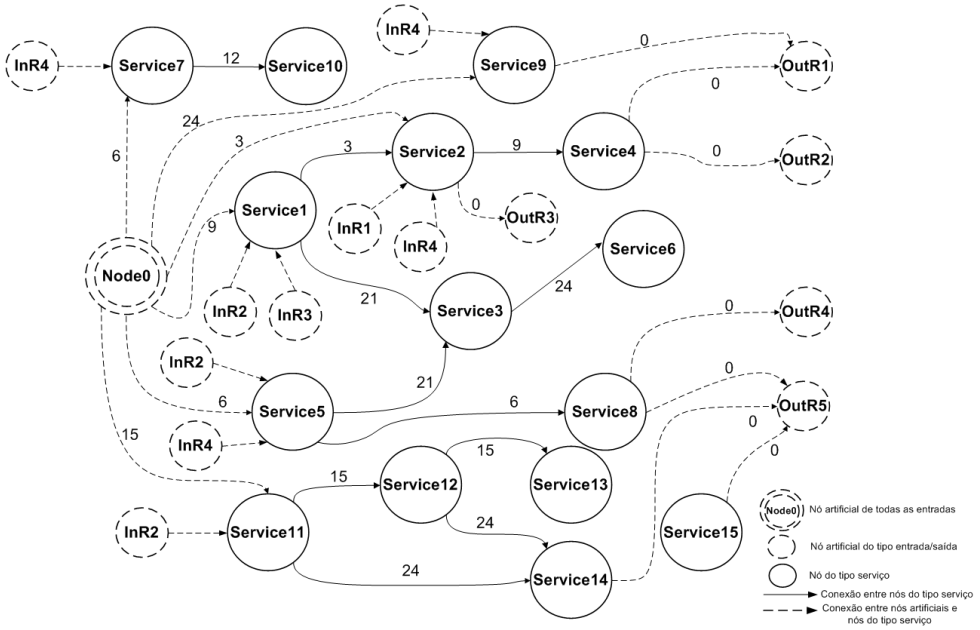

Figura 5.7: Grafo com custos e descontos funcionais.

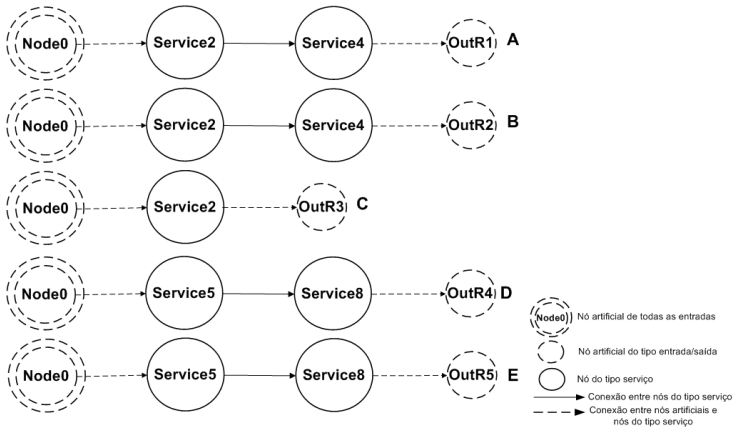

Figura 5.8: Caminhos parciais, com diferenciação de custos, do nó 0 para todas as saídas requeridas.

isso, a composição final apresentada na Seção 5.4.

5.4 Composição de serviços

O algoritmo de cálculo do caminho de custo mínimo não gera a composição final do serviço requerido, ele gera um caminho de custo mínimo para cada saída requerida. Os caminhos são analisados de forma automática, com o objetivo de encontrar intersecções entre caminhos para saídas distintas e, a partir desse ponto as composições são geradas.

A tendência observada é que os caminhos podem possuir intersecções entre eles, desde que existam saídas requeridas em um mesmo fluxo do grafo. Porém, também pode ocorrer caminhos que não se cruzam no grafo. Os caminhos conexos (com intersecções) devem ser combinados em um único caminho com os construtores seqüenciais de OWL-S e os caminhos desconexos devem ser mapeados para construtores paralelos de OWL-S.

Ainda utilizando o exemplo do grafo da Figura 5.2, esta seção apresenta e traça um comparativo das composições finais para os caminhos encontrados sem a diferenciação de custos (Seção 5.4.1) e a composição baseada nos custos e descontos funcionais (Seção 5.4.2).

5.4.1 Serviço composto sem diferenciação de custos

As Figuras 5.9 e 5.10 apresentam duas possíveis composições, tendo como base os possíveis caminhos de custo mínimo no grafo da Figura 5.2 para as saídas requeridas. Essas composições foram construídas com os caminhos parciais da Seção 5.3.1 que não levou em consideração os custos das arestas (custo uniforme).

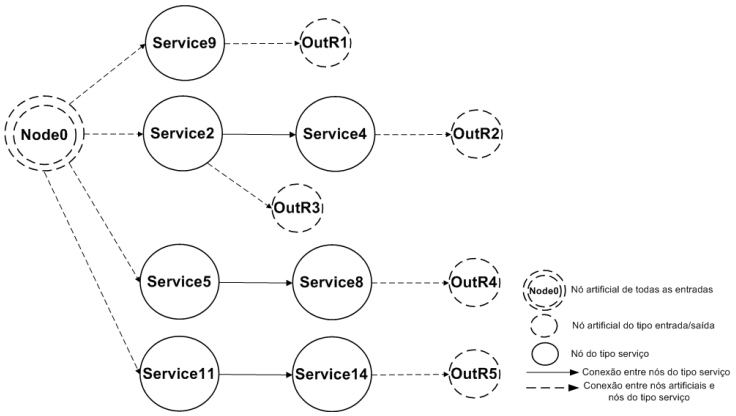

Figura 5.9: Composição baseada nos caminhos encontrados pelo algoritmo de cálculo do caminho de custo mínimo sem a diferenciação de custos.

A Figura 5.9 apresenta a composição baseada nos caminhos mínimos da Figura 5.4 encontrados pelo algoritmo de cálculo do caminho de custo mínimo. Essa composição apresenta as saídas *OutR4* e *OutR5* sendo geradas por serviços diferentes (*Service8* e *Service14*), pois foram os caminhos de custo mínimo encontrados na execução do algoritmo.

Na Figura 5.10 as saídas *OutR4* e *OutR5* são geradas pelo mesmo serviço (*Service8*): essa opção possibilitaria a eliminação de dois serviços (*Service11* e *Service14*) na composição e ainda assim alcançando todas as saídas requeridas. Porém, sem a diferenciação de custos o caminho para *OutR5* por meio de *Service8* não foi computado. Outro caminho, não apresentado nas Figuras 5.9 e 5.10, é o caminho

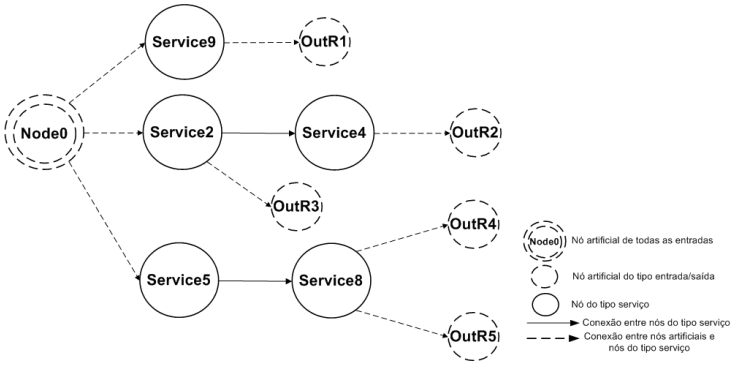

Figura 5.10: Uma outra possível composição ainda sem a diferenciação de custos.

para a saída *OutR1* apresentado na Figura 5.5. Esse caminho também eliminaria um serviço (*Service9*) da composição final, porém também não foi computado pelo algoritmo.

Para eliminar caminhos tais como os das Figuras 5.4-A e 5.4-E da composição, substituindo-os pelos caminhos das Figuras 5.5 e 5.6 respectivamente, e otimizar as composições em relação à funcionalidade do serviço requerido, os custos para as entradas não disponíveis e saídas não requeridas foram atribuídos na Seção 5.3.2 e os resultados da composição final estão apresentados na Seção 5.4.2.

5.4.2 Serviço composto com custos funcionais

A composição dos serviços com base nos caminhos com diferenciação de custos (Figura 5.8) ocorre da mesma forma que foi apresentado para a composição sem custo diferenciado (Seção 5.4.1): é necessário compor um grafo único e conectado, eliminando os caminhos duplicados por meio da análise das intersecções entre os caminhos encontrados.

A Figura 5.11 apresenta a composição baseada nos caminhos mínimos da Figura 5.8 encontrados pelo algoritmo de cálculo do caminho de custo mínimo. Como diferença à composição apresentada na Figura 5.9, a composição da Figura 5.11 apresenta as saídas *OutR1* e *OutR2* sendo geradas pelo mesmo serviço (*Service4*) e também as saídas *OutR4* e *OutR5* sendo geradas pelo mesmo serviço (*Service8*), pois foram os caminhos de custo mínimo encontrados na execução do algoritmo.

Comparando-se a composição sem custo diferenciado (Figura 5.9) com a composição com custo diferenciado (Figura 5.11), pode-se verificar que nas duas composições as saídas requeridas foram alcançadas. Entretanto, com a utilização dos custos e descontos funcionais, a composição final possui três serviços a menos e ainda eliminou algumas duplicações de resultados (as saídas *OutR1* e *OutR5*, na composição sem custos diferenciados, seriam geradas duas vezes por dois serviços distintos cada

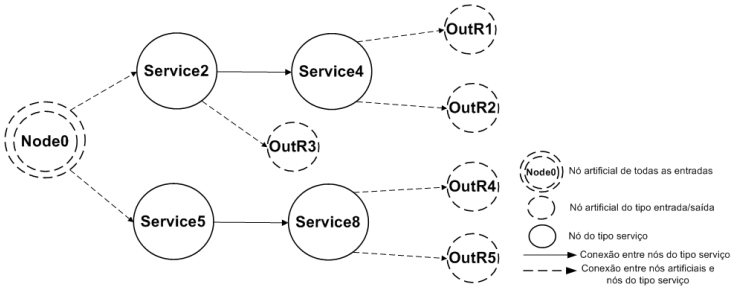

Figura 5.11: Serviço composto levando em consideração os custos.

uma).

A diferenciação de custos com base na funcionalidade do serviço requerido faz com que o algoritmo escolha o melhor caminho (baseado nos custos estabelecidos) dentre as possibilidades de caminhos para uma mesma saída. Entretanto, mesmo com os custos, não se pode garantir que um caminho para uma saída requerida será único. O que se pode garantir é que o caminho escolhido atenderá aos requisitos de custo estabelecidos *a priori*.

5.5 Avaliação da composição final

Após o processo de composição ser realizado com sucesso, ou seja, uma composição que atende aos requisitos do usuário ter sido encontrada, a descoberta é utilizada para verificar o quanto a composição final atende aos requisitos iniciais do usuário. O processo de descoberta apresentado no Capítulo 4 é repetido, uma única vez, tendo como parâmetros os requisitos do usuário (conjuntos de entradas disponíveis e saídas requeridas) e o serviço composto. Dessa forma, o resultado da composição é comparado com os resultados da descoberta inicial e pode-se avaliar a eficiência da composição.

Analisando a Figura 5.12, que apresenta uma composição final e todas as suas entradas e saídas referentes a todos os serviços pertencentes à composição, pode-se observar que: uma das entradas disponíveis não é utilizada (entrada *InR3* na Figura 5.12); duas entradas não disponíveis são necessárias à execução da composição (entradas *In5* e *In6* na Figura 5.12); e duas saídas não requeridas são produzidas (*Out6* e *Out7* na Figura 5.12).

Pode-se verificar que, apesar dos custos eliminarem serviços com muitas entradas não disponíveis e saídas não requeridas, essas ocorrências podem ainda aparecer na composição final. Isto ocorre devido ao fato de que o processo de composição é flexível e permite que composições desse tipo ocorram caso não seja possível uma composição que atenda 100% aos requisitos de entrada e saída. É aumentar o atributo λ referente aos descontos das entradas e saídas e o algoritmo de custos mínimos vai tentar

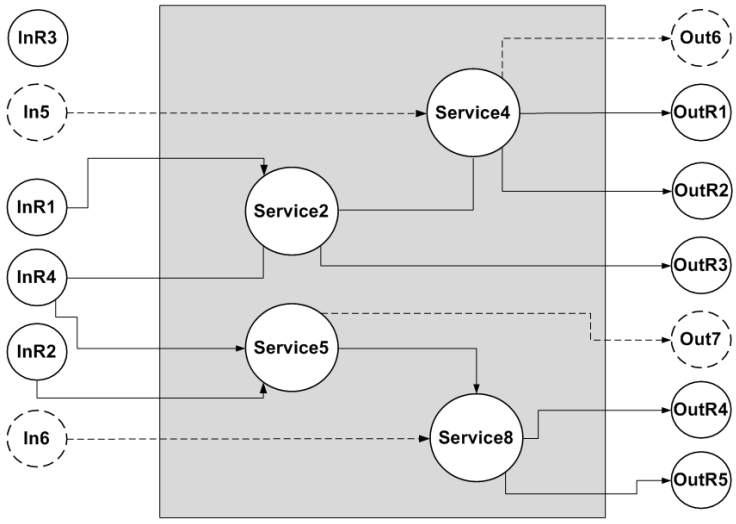

Figura 5.12: Entradas e saídas para a composição final.

encontrar outra composição com menos entradas ou saídas indesejadas. Se ocorrer de, mesmo com a variação de λ, o número das entradas ou saídas não se alterar é porque não existe uma composição que atenda completamente os requisitos de entrada e saída.

Entretanto, um outro fator pode causar a presença das entradas não disponíveis e das saídas não requeridas na composição final: a decisão inicial de modelar o grafo com uma única aresta ligando cada serviço mesmo nos casos em que existiam mais de uma conexão semântica entre eles. Essa decisão foi importante para se conseguir modelar o problema de composição através de grafos e para a aplicação do algoritmo de cálculo do caminho de custo mínimo.

Nada garante que, no exemplo da Figura 5.12, as entradas *In5* do serviço *Service4* e *In6* do serviço *Service8* não sejam, por exemplo saídas dos serviços *Service2* e *Service5* respectivamente. Isto ocorreria se os pares de serviços (*Service2*;*Service4*) e (*Service5*;*Service8*) tivessem duas conexões semânticas entre eles.

Essa é uma limitação de modelagem, pois o grafo não possui informações de quantas e quais conexões semânticas existem entre cada par de serviços contidos no repositório. Porém, a limitação é necessária ao modelo e pode ser tratada sem oferecer prejuízos à abordagem proposta. A solução é guardar as informações de conexão em uma outra estrutura de dados mais completa no sentido de refletir todas as conexões semânticas entre os serviços, utilizar o modelo de grafo com única aresta no processo de composição automática e, ao final do processo, mapear a composição final para a estrutura de dados que contém as informações mais completas.

A solução adotada para este trabalho é simples e funcional. No momento da publicação dos serviços são criados dois grafos em vez de um. Um dos grafos é

modelado da forma que foi apresentado na Seção 5.1, ou seja, apenas com nós serviços e suas conexões (arestas) únicas. O outro grafo, como apresentado na Figura 5.13, contém ainda todas as entradas e saídas modeladas como nós do grafo, ou seja, além dos serviços serem nós do grafo, as entradas e saídas também são modeladas como nós. Note que, com o objetivo de facilitar a visualização devido a grande quantidade de informação, o grafo da Figura 5.13 não é o mesmo grafo (Figura 5.1) que modela quinze serviços.

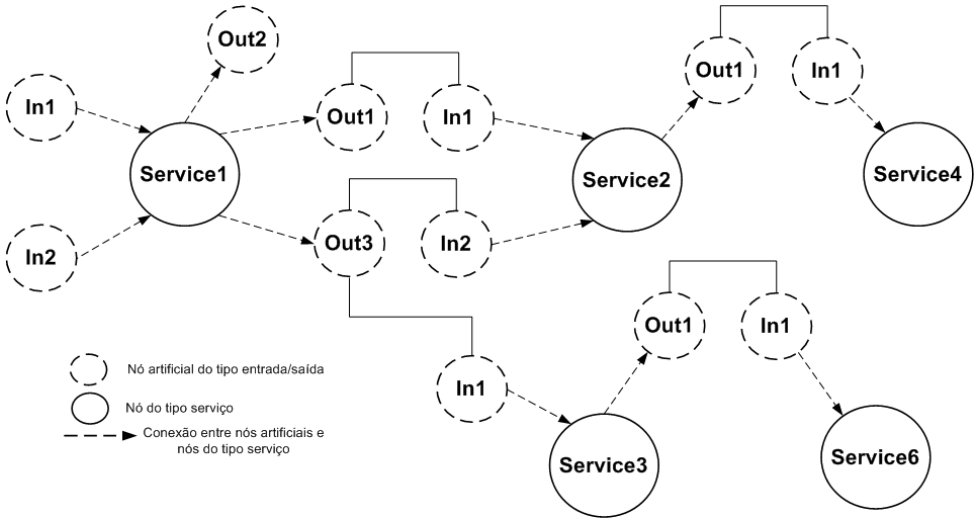

Figura 5.13: Grafo auxiliar utilizado para a composição final.

Após se obter os caminhos mínimos para todas as entradas requeridas, utiliza-se o grafo da Figura 5.13 para montar o grafo da composição final. Dessa forma, é possível diferenciar com certeza quais entradas realmente não são disponíveis das entradas que podem ser obtidas no fluxo interno da composição por meio de conexões semânticas.

5.6 Mapeamento da composição final para o OWL-S

A sub-ontologia de processos do OWL-S, apresentada no Capítulo 2, define o fluxo interno do serviço por meio de construtores que descrevem o *workflow* do serviço. Em um processo dois tipos de caminhos podem ocorrer [WfMC, 1996]:

- caminho paralelo: segmento de um processo em que duas ou mais atividades são executadas em paralelo dentro do *workflow*, podendo originar múltiplas *threads* de controle; Caminhos paralelos normalmente iniciam-se com um construtor *AND-Split* (Figura 5.14-A) e finalizam-se com um construtor *AND-Join* (Figura 5.14-B).

- caminho seqüencial: segmento de um processo em que as atividades são executadas em seqüência dentro do *workflow*, dando origem à uma única *thread* de controle. Caminhos seqüênciais não devem possuir construtores *AND-Split* e *AND-Join*.

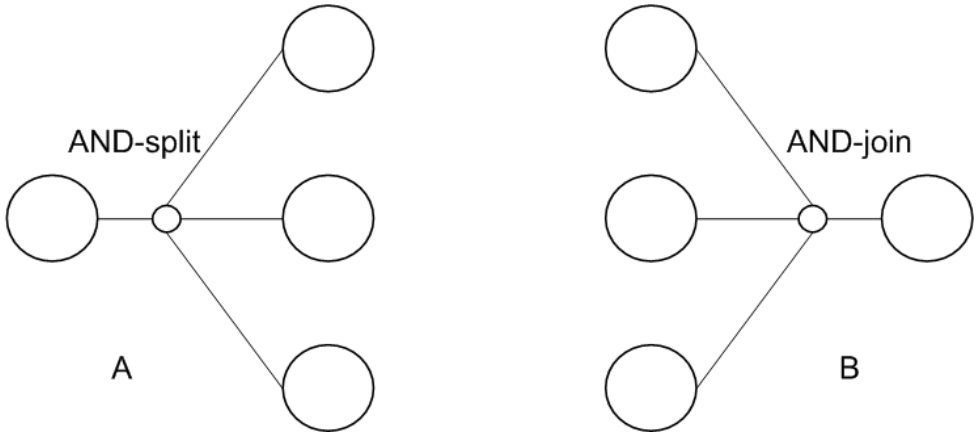

Figura 5.14: Construtores *AND-Split* e *AND-Join* para *workflow*.

Além dos construtores *AND-Split* e *AND-Join*, mais três construtores podem ocorrer em um processo [WfMC, 1996]:

- *OR-Split* (Figura 5.15-A): um ponto no *workflow* em que a *thread* de controle deve tomar a decisão sobre qual ramificação escolher dado várias alternativas de ramificações;
- *OR-Join* (Figura 5.15-B): um ponto no *workflow* em que duas ou mais alternativas de ramificações convergem para um único ponto em comum;
- *Iteration* (Figura 5.15-C): um ciclo no *workflow* que envolve a repetição de uma ou mais atividades até que uma condição de parada seja satisfeita.

Não ocorre execução paralela com os construtores *OR-Split* e *OR-Join* assim como em *AND-Split* e *AND-Join*. O fluxo em *OR-Split* é selecionado para apenas uma ramificação do *workflow* e em *OR-Join* é direcionado a convergir de volta para um único ponto.

Cada um dos construtores de fluxo das Figuras 5.14 e 5.15 podem ocorrer no grafo da composição final gerado para este trabalho. Esses construtores são mapeados direta ou indiretamente para um construtor de fluxo da sub-ontologia de processos da ontologia OWL-S como descrito na Tabela 5.6.

Um grafo que representa uma composição final, tal como os grafos apresentados nas Seções 5.4.1 e 5.4.2, é mapeado de volta para uma representação utilizando

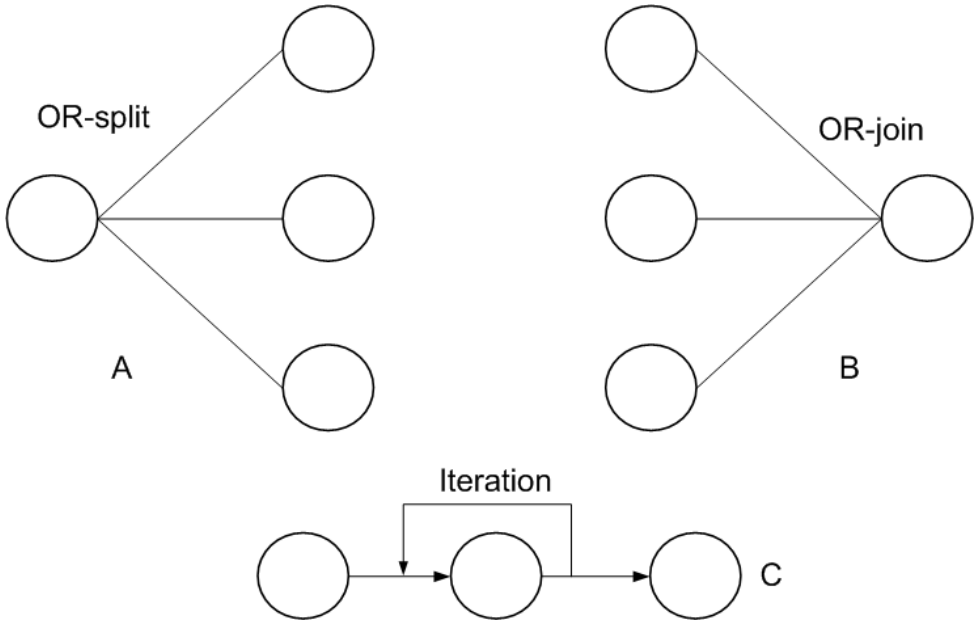

Figura 5.15: Construtores *OR-Split*, *OR-Join* e *Iteration* para *workflow*.

Tabela 5.6: Mapeamento do fluxo do grafo para os construtores de fluxo do OWL-S.

Grafo	**Construtor OWL-S**
Caminho seqüencial	*Sequence*
Caminho paralelo com barreira de sincronização apenas no início (*AND-Split*)	*Split*
Caminho paralelo 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*

a sub-ontologia de processos em OWL-S. Com isto o novo serviço composto pode ser tratado como um serviço único com seu fluxo interno baseados na lógica de processos do OWL-S. Além disso, a composição em OWL-S pode ser registrada no repositório de serviços e ser utilizada da próxima vez que um usuário buscar por serviços com as mesmas funcionalidades.

5.7 Trabalhos relacionados

Diversos esforços, tanto na academia como na indústria, têm sido empregados na direção da realização de composição de Serviços Web. Uma busca pelo assunto "*Web Services Composition*" na literatura apresenta uma vasta gama de resultados com focos muitas vezes distintos. Faz-se necessário situar o trabalho apresentado neste capítulo, cujo objetivo é composição automática de Serviços Web Semânticos, em relação aos encontrados na literatura.

As classificações apresentadas nesta seção não têm objetivo de completude e sim de situar o trabalho em composição de Serviços Web descrito neste capítulo. Esta seção apresenta as principais classificações encontradas na literatura, situa a composição de Serviços Web proposta no presente trabalho de doutorado dentre as classificações existentes e faz um comparativo com as abordagens diretamente relacionadas a este capítulo.

5.7.1 Composição estática *vs.* dinâmica e manual *vs.* automática

Dustdar and Schreiner [2005] apresentam uma pesquisa sobre composição de Serviços Web que, dentre outras coisas, discorre sobre os conceitos e diferenças entre composição estática *vs.* dinâmica e composição manual *vs.* automática. Segundo os autores, as classificações estática e dinâmica dizem respeito ao momento de realização da composição: estática é para a composição realizada em tempo de codificação do serviço; e dinâmica é para a composição realizada em tempo de execução do serviço. As classificações para a composição manual e automática dizem respeito à forma como a composição é realizada: manual é efetuada pelo desenvolvedor do serviço; e automática por ferramentas de software.

Os conceitos de composição "manual e estática" e de "automática e dinâmica" muitas vezes são fundidos na literatura. O autor desta tese entende que os conceitos de composição manual e automática são mais abrangentes, sendo que a composição manual é geralmente estática e a composição automática pode ser tanto estática como dinâmica e isto vai depender basicamente de quem e em que momento a composição está sendo realizada: um desenvolvedor de serviços no momento da criação de um novo serviço ou um usuário final no momento da utilização do serviço.

O desenvolvedor de serviços pode, por exemplo, utilizar ferramentas para buscar por serviços complementares ao serviço que ele está desenvolvendo e automaticamente realizar uma composição estática que nunca vai mudar, ou seja, todo usuário que for utilizar seu serviço tem que utilizá-lo com aquela mesma composição. Assim, o desenvolvedor realizou composição automática e estática.

O usuário final do serviço sempre realiza composição automática e dinâmica no momento da requisição por um serviço, ou seja, pode ocorrer que toda vez que um

mesmo usuário fizer uma mesma requisição por um serviço as composições geradas sejam diferentes.

A composição de Serviços Web apresentada neste trabalho é sempre automática e o fato de ser realizada de forma estática ou dinâmica não influencia a proposta aqui apresentada.

5.7.2 Composição automática de Serviços Web

Rao and Su [2004] e Martin et al. [2007a] elencam e descrevem os tipos de abordagens para a composição automática de Serviços Web. Milanovic and Malek [2004] e Alamri et al. [2006] também elencam e discutem as abordagens para realização de composição de Serviços Web. Entretanto, Milanovic and Malek [2004] não apresentam uma separação explícita entre as abordagens manuais e automáticas e Alamri et al. [2006] discutem apenas abordagens que são dinâmicas.

A partir das pesquisas apresentadas por Rao and Su [2004], Martin et al. [2007a], Milanovic and Malek [2004] e Alamri et al. [2006] pode-se dividir as abordagens para a composição automática de Serviços Web em: técnicas de planejamento em Inteligência Artificial; técnicas para síntese de programa; ou técnicas baseadas em *workflow*.

Exemplos de abordagens que utilizam técnicas de planejamento em Inteligência Artificial para a composição automática de serviços, dentre outros, são: a aplicação da linguagem Golog para gerar uma seqüência de Serviços Web baseada nas preferências e restrições do usuário [McIlraith and Son, 2002]; e o uso da técnica HTN (*Hierarchical Task Network*) para produzir uma seqüência de ações que executam uma atividade ou tarefa sempre dividindo as tarefas em sub-tarefas [Sirin et al., 2004b; Kuter et al., 2004; Paolucci et al., 2003a];

Abordagens que utilizam técnicas de síntese de programa são utilizadas em alguns trabalhos para a composição automática de Serviços Web: Lammermann [2002] utiliza a técnica de Síntese Estrutural de Programa (SSP – do inglês *Structural Synthesis of Program*); e Rao [2004] introduz um método que utiliza Lógica Linear.

A Seção 5.7.3 discorre sobre os trabalhos baseados em *workflow* e faz um comparativo à proposta apresentada neste capítulo.

5.7.3 Composição automática de Serviços Web baseada em *workflows*

As abordagens que utilizam técnicas baseadas em *workflow* para a composição automática de Serviços Web são diretamente relacionadas a este trabalho. A seguir são apresentadas essas abordagens comparando-as com o trabalho descrito neste capítulo.

Hashemian and Mavaddat [2005] e Mitra et al. [2007] apresentam abordagens similares que modelam o problema de composição utilizando estrutura de *Interface*

Automata [Alfaro and Henzinger, 2001] e *I/O Automata* [Lynch and Tuttle, 1987], respectivamente. Os autores modelam as conexões entre os serviços com base na relação de dependência entre entradas e saídas. A partir do modelo, aplica-se algoritmos para descobrir se uma saída é alcançada a partir de uma dada entrada. Essas abordagens não levam em conta os custos nas arestas e nem mesmo se a composição final pode conter muitas entradas não disponíveis ao usuário.

Gekas and Faslil [2007] apresentam algumas heurísticas com o objetivo de reduzir a quantidade de serviços presentes no grafo em que o algoritmo de composição vai ser executado. Os autores afirmam que o processo de criar o grafo em tempo de composição limita as abordagens baseadas em grafo. O trabalho reportado nesta tese apresenta uma abordagem para a criação do grafo no momento da publicação. Dessa forma, a criação do grafo não é um limitador da abordagem.

Silva et al. [2007] e Arpinar et al. [2005] descrevem abordagens em que utilizam grafos para realizar a composição automática de Serviços Web. As duas abordagens utilizam a similaridade semântica para efetuar a conexão do grafo de serviços.

Silva et al. [2007] não aplicam custos ao grafo e dessa forma não utilizam algoritmo para cálculo do caminho de custo mínimo tal como neste trabalho. O algoritmo proposto pelos autores implementa caminho inverso no grafo, partindo das saídas em direção às entradas. Não existe seleção do melhor serviço para a composição e o algoritmo pode gerar diversas composições ao final da sua execução. O algoritmo testa todas as possibilidades e se não houver caminho para chegar nas entradas nenhuma composição é gerada.

A proposta de composição desta tese tenta sempre garantir todas as saídas na composição final (caso seja possível), por isso a opção por realizar um caminho para cada saída, partido sempre do nó referente às entradas. Nos casos em que não existe uma composição com todas as entradas e saídas requeridas o algoritmo ainda assim encontra uma solução que pode agregar o máximo dos parâmetros requeridos pelo usuário.

Arpinar et al. [2005] apresentam uma abordagem em que a definição de custos nas arestas do grafo e um algoritmo para cálculo do caminho de custo mínimo são utilizados conforme neste trabalho. O objetivo dos autores é o de encontrar uma composição que produza as saídas requeridas levando em consideração o tempo de execução de cada serviço e a similaridade das conexões do grafo. O custo das arestas são definidos em função do tempo e da similaridade. Os caminhos são calculados de cada entrada para cada saída sem levar em consideração que um mesmo serviço pode conter mais de uma entrada ou saída requerida e nem mesmo que podem existir outras entradas e saídas adicionais.

Os custos e descontos funcionais descritos na Seção 5.2 garantem que os serviços que possuam mais entradas e saídas requeridas sejam escolhidos, ao mesmo tempo

que tenta evitar serviços com entradas e saídas não requeridas. Na abordagem proposta por Arpinar et al. [2005] pode ocorrer de muitos dos serviços intermediários na composição exigir entradas não disponíveis ao usuário, pois o custo baseado apenas na similaridade não executa uma boa seleção. A composição final baseada na abordagem de Arpinar et al. [2005] pode ser uma quantidade muito grande de caminhos em paralelo, partindo de uma entrada para uma saída e sem nenhum tipo de avaliação do fluxo entre os caminhos.zzz

5.8 Considerações Finais

Segundo Milanovic and Malek [2004], os principais problemas envolvidos na realização de composição automática estão em como identificar os serviços candidatos à composição, criar a composição de fato e então verificar o quanto que a composição atende ao requisito do usuário. A composição automática apresentada neste capítulo trata desses três problemas. Os custos e descontos funcionais (Seção 5.2), juntamente com a execução do algoritmo de cálculo do caminho de custo mínimo (Seção 5.3), identificam os serviços candidatos à composição. Os caminhos de custo mínimo e a análise de suas intersecções possibilitam criar a composição final (seções 5.3 e 5.4). A aplicação do *matching* da composição final em relação aos requisitos iniciais do usuário verifica o quanto que a composição atende ao requisito do usuário (Seção 5.5).

Este capítulo apresentou resultados da aplicação de algoritmo para cálculo de caminhos de custo mínimo na composição automática de Serviços Web Semânticos. Os resultados apresentados foram: a modelagem do problema da composição automática sob a forma de um grafo direcionado; a política de custos aplicada ao grafo para prover composições com base em atributos funcionais e não-funcionais; a aplicação do algoritmo de Dijkstra para cálculo dos caminhos de custo mínimo; e a composição final.

O modelo baseado em grafo possibilitou a utilização de um algoritmo de cálculo do caminho de custo mínimo para descobrir os caminhos de custo mínimo para cada saída requerida pelo usuário.

Os resultados das composições apresentaram-se satisfatórios à partir da implementação dos custos e descontos. Os custos e descontos referentes aos atributos funcionais fizeram o algoritmo se comportar de forma a selecionar os melhores serviços para a composição com base nas funcionalidades (entradas e saídas) requeridas pelo usuário.

O algoritmo de Dijkstra aplicado ao grafo dos Serviços Web Semânticos gera um caminho único para cada saída requerida com base nos custos estabelecidos. A execução do algoritmo parte de um nó artificial que simboliza as entradas disponíveis

para nós artificiais referentes a cada saída requerida.

Após a execução do algoritmo, uma composição final é gerada e pode ser confrontada com a requisição inicial do usuário para avaliar o quão que a composição final atende aos requisitos iniciais. Por fim, a composição gerada de forma automática é modelada e descrita utilizando os construtores de fluxo do OWL-S e armazenada no repositório UDDI para que possa ser reutilizada posteriormente em uma nova busca pelos mesmos requisitos.

O Capítulo 6 apresenta uma arquitetura de software e uma infra-estrutura para implantação de Serviços Web Semânticos. Essa infra-estrutura engloba, dentre outras coisas, a implementação de ferramentas e módulos com a finalidade de realizar a composição automática tal como descrita neste capítulo. Ainda no Capítulo 6 é realizada uma avaliação dessa referida implementação com o objetivo de demonstrar a viabilidade da composição automática.

CAPÍTULO 6

Infra-estrutura para desenvolvimento de Serviços Web Semânticos

Este capítulo descreve a infra-estrutura implementada para permitir o desenvolvimento de Serviços Web Semânticos conforme proposta desta tese. Essa infra-estrutura possui as funcionalidades para publicação, descoberta e composição de Serviços Web Semânticos.

A Seção 6.1 descreve as tecnologias e plataformas utilizados para desenvolvimento de Serviços Web Semânticos e das funcionalidades para publicação, descoberta e composição implementadas neste trabalho. Essas funcionalidades são descritas na Seção 6.2 juntamente com a arquitetura da implementação. A Seção 6.3 apresenta a avaliação com base na complexidade e nos tempos de resposta das funcionalidades desenvolvidas para publicação, descoberta e composição de Serviços Web Semânticos. Para finalizar, a Seção 6.4 traça algumas considerações finais referentes a este capítulo.

6.1 Infra-estrutura para Serviços Web Semânticos

Berners-Lee et al. [2001] afirmam que a Web Semântica deve ser uma extensão da Web atual com estruturação e adição de semântica ao conteúdo da Web, em vez da criação de uma nova Web. Dessa forma, a infra-estrutura 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 6.1 apresenta a infra-estrutura utilizada para desenvolvimento de

Serviços Web Semânticos bem como para as funcionalidades de publicação, descoberta e composição de serviços. A infra-estrutura 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*.

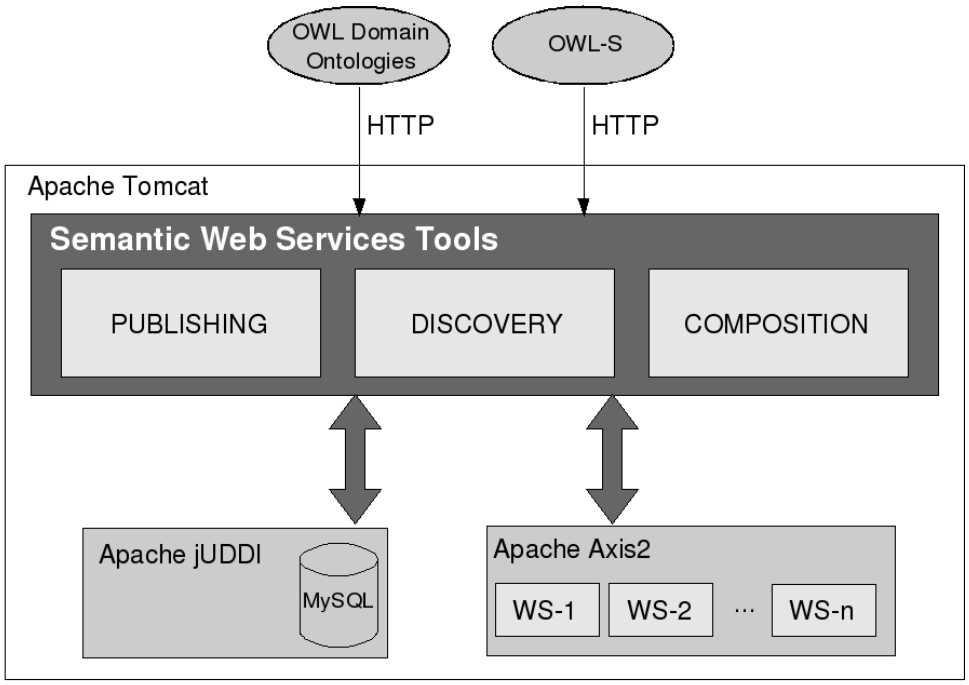

Figura 6.1: Infra-estrutura para Serviços Web Semânticos [Prazeres et al., 2009c].

A plataforma *jUDDI* (Figura 6.1) 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 6.2.

A plataforma *Axis2* (Figura 6.1) é utilizada neste trabalho para desenvolver alguns

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

serviços que serviram como base de testes para algumas das implementações deste trabalho. Descrições em OWL-S para serviços simulados foram gerados 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 6.2.

Por fim, a infra-estrutura da Figura 6.1 está implantada no servidor Web *Apache Tomcat*. Tanto as plataformas *jUDDI* e *Axis2*, que são aplicações Web, como as funcionalidades para publicação, descoberta e composição estão funcionando nesse servidor Web.

6.2 Funcionalidades para Serviços Web Semânticos

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 6.2 apresenta a arquitetura de software utilizada nas implementações das funcionalidades de publicação, descoberta e composição que estão dispostas como *Semantic Web Services Tools* na infra-estrutura da Figura 6.1. Todos os módulos presentes na Figura 6.2 estão implementados na linguagem *Java* (*J2SE* 1.5 e *J2EE* 5.0) porque muitas das soluções implementadas para a Web Semântica utilizam essa linguagem.

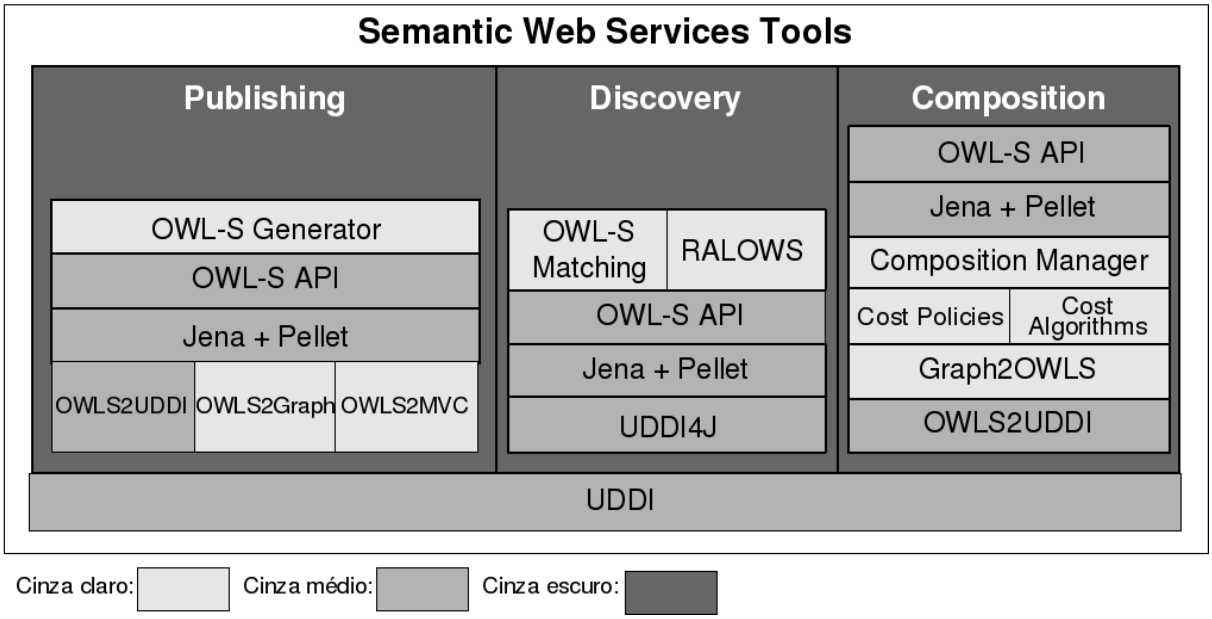

Figura 6.2: Arquitetura das funcionalidades para publicação, descoberta e composição de Serviços Web Semânticos [Prazeres et al., 2009c].

Os módulos apresentados em tons de cinza médio na Figura 6.2 são ferramentas ou *APIs*[4] (Interface de Programação de Aplicativos) 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 6.2.1 por serem necessários ao entendimento dos módulos que foram implementados para a validação deste trabalho (apresentados em tom de cinza claro na Figura 6.2) e estão descritos nas seções 6.2.2, 6.2.3 e 6.2.4.

Todos os módulos apesentados nesta seção estão implementados e parte deles está integrada da forma ilustrada na Figura 6.2. Entretanto, alguns desses módulos foram implementados separadamente e estão em processo de integração conforme a arquitetura proposta.

6.2.1 Ferramentas incorporadas de outros projetos

Todos os módulos apresentados em cinza médio na Figura 6.2 são componentes ou ferramentas de código fonte aberto, importados de outros projetos e utilizados com finalidades específicas neste trabalho: *Jena*[5], *OWL-S API*[6], Pellet[7], *OWLS2UDDI*[8] e *UDDI4J*[9].

O componente *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, que provê todos os métodos e construtores necessários para manipulação de documentos OWL-S, é utilizado na maioria dos módulos implementados neste trabalho e representado na arquitetura ilustrada na Figura 6.2. 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

[4] *Application Programming Interface* é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por programas aplicativos.
[5] http://jena.sourceforge.net/
[6] http://www.daml.ri.cmu.edu/owlsapi/
[7] http://clarkparsia.com/pellet/
[8] http://owl-s2uddi.projects.semwebcentral.org/
[9] http://uddi4j.sourceforge.net/

que realizam *matching* baseado em *subsumption*.

O componente *OWLS2UDDI* é uma ferramenta desenvolvida por Srinivasan et al. [2004] 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 infra-estrutura *jUDDI*, tal como neste trabalho, provê uma infra-estrutura 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 infra-estrutura do *jUDDI*.

6.2.2 Publicação de Serviços Web Semânticos

Como pode ser observado na Figura 6.2 são três os módulos implementados com o propósito de prover e validar a publicação de serviços neste trabalho. Esses módulos são o *OWL-S Generator*, o *OWLS2Graph* e o *OWLS2MVC*, que estão descritos a seguir.

Módulo *OWL-S Generator*

O módulo *OWL-S Generator* tem a função de gerar, de forma automática, descrições de serviços em OWL-S para servir de base de testes na descoberta e na composição de serviços cuja avaliação é apresentada na Seção 6.3. Esse módulo cria serviços artificias e os publica no repositório UDDI.

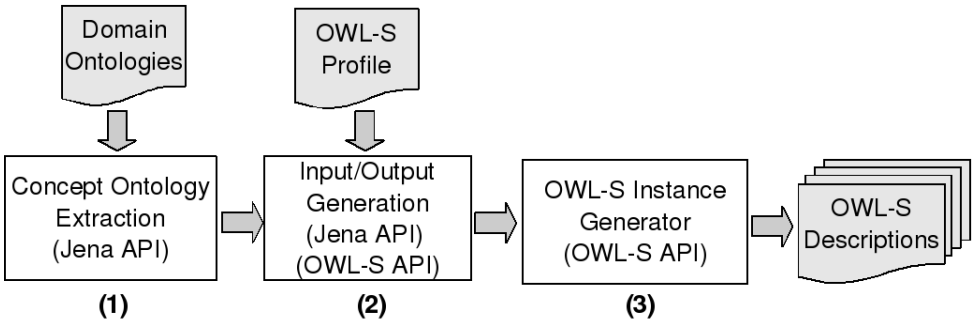

Figura 6.3: Visão geral do módulo *OWL-S Generator*.

A Figura 6.3 apresenta uma visão geral do módulo *OWL-S Generator* que tem como entrada ontologias de domínio e a sub-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 do perfil de um serviço é descrita com uma ontologia de domínio referente ao domínio do serviço em questão. Como pode ser visto na Figura 6.3 o módulo *OWL-S Generator*: (1) extrai conceitos de

ontologias de domínio; (2) cria entradas e saídas descritas com esses conceitos; (3) e gera instâncias de serviços descritos em OWL-S.

Módulo *OWLS2Graph*

O módulo *OWLS2Graph* é responsável por adicionar serviços descritos em OWL-S ao grafo de todos os serviços contidos no UDDI. Esse módulo possibilita tanto a adição de um único serviço e seus custos associados (no momento da publicação do serviço) ao grafo, como a transformação de todo o repositório UDDI em um grafo direcionado, com custos e com os serviços conectados entre si da forma descrita na Seção 5.1.

Toda vez que um serviço é publicado ele é adicionado ao grafo utilizado para realizar a composição automática de serviços. A Seção 5.1.1 descreve como as conexões são descobertas e estabelecidas. 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.

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, Prazeres et al. [2009a;b] observaram a oportunidade de propor o mapeamento entre os descritores e construtores do OWL-S e linguagens de programação. A Tabela 6.1 apresenta o mapeamento proposto para a linguagem Java utilizando uma implementação da arquitetura MVC (do inglês *Model-View-Controller*) [Krasner and Pope, 1988] para a Web.

Como pode ser visto na Tabela 6.1 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, enquanto que 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 6.1 tal como ilustrado na Figura 6.4. 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 infra-estrutura da Figura 6.1 e o OWL-S correspondente é inserido no repositório UDDI por meio do módulo *OWLS2UDDI*.

A Figura 6.4 apresenta uma visão geral do módulo *OWLS2MVC* que tem como entrada as instâncias OWL-S de um Serviço Web e folhas de estilo XSLT. Como pode ser visto na Figura 6.4, o módulo *OWLS2MVC* executa: (1) extração dos dados apresentados na Tabela 6.1 a partir dos documentos OWL-S de entrada; (2) extração dos esquemas referentes aos parâmetros de entrada e saída do OWL-S; (3)

Tabela 6.1: Mapeamento do OWL-S para a Java (Prazeres et al., 2009a;b).

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

transformação XSLT que gera interfaces gráficas para o Serviço Web.

A extração dos dados provenientes do documento OWL-S de entrada (Figura 6.4-(1)) gera os *Servlets* (Figura 6.4-(a)) que são os controladores da aplicação Web em MVC e também gera um sub-produto que são os esquemas referentes às entradas e saídas do serviço. Os esquemas são processados na Figura 6.4-(1) e então são gerados os *JavaBeans* (Figura 6.4-(b)) que são os modelos da aplicação Web em MVC. Finalmente, uma transformação XSLT, baseada na proposta de Prazeres and Teixeira [2006], é aplicada (Figura 6.4-(3)) aos esquemas para gerar algumas interfaces gráficas em *HTML* e *JSP* (Figura 6.4-(c)) que são as visões da aplicação Web em MVC.

O módulo *OWLS2MVC* é utilizado para, a partir de um documento OWL-S, gerar o esqueleto de uma aplicação Web descrita como um Serviço Web Semântico e 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 infra-estrutura *Axis2* apresentados na Seção 6.1.

6.2.3 Descoberta de Serviços Web Semânticos

São dois os módulos implementados, conforme pode ser visto na Figura 6.2, com o propósito de validar a descoberta de Serviços Web Semânticos apresentada no Capítulo 4 deste trabalho: os módulos *RALOWS* e o *OWL-S Matching* que estão descritos a seguir.

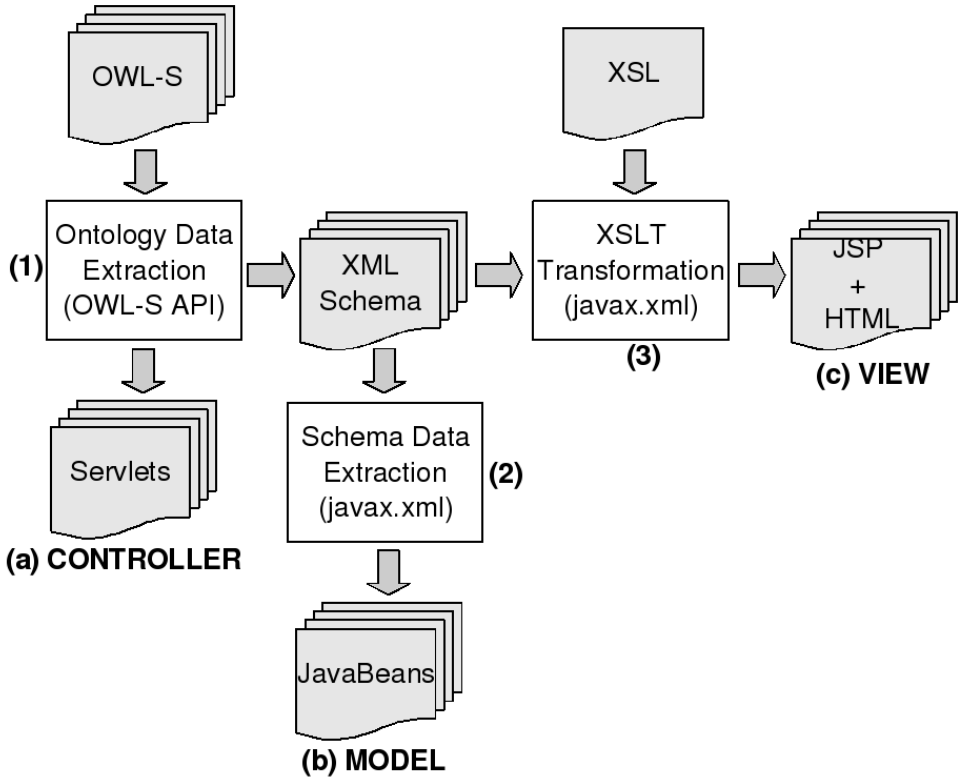

Figura 6.4: Visão geral do módulo *OWLS2MVC* [Prazeres et al., 2009a].

Módulo *RALOWS*

O módulo *RALOWS* [Prazeres et al., 2007] refere-se à extensão da *OWL-S API* com o intuito de prover as funcionalidades requeridas para a utilização da plataforma RALOWS apresentada na Seção 3.3. Para esse módulo foram estendidas as classes Java *Profile*, *Input* e *Output*, e todas as interfaces (*Interface Java*) e métodos utilizados para manipular essas classes na *OWL-S API*.

Módulo *OWL-S Matching*

O *matching*, que é realizado pelo algoritmo de descoberta nos seus estágios iniciais, utiliza o *framework* Jena em conjunto com a *OWL-S API* e a máquina de inferência *Pellet*. Esses três componentes são utilizados no módulo *OWL-S Matching* para prover a descoberta de Serviços Web Semânticos conforme proposto neste trabalho.

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 repositório *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.

6.2.4 Composição de Serviços Web Semânticos

Para validar e prover a composição automática de Serviços Web Semânticos apresentada no Capítulo 5 quatro módulos foram implementados, conforme pode ser visto na Figura 6.2: os módulos *Composition Manager*, *Cost Policies*, *Cost Algorithms* e *Graph2OWLS* que são descritos a seguir.

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.

Módulo *Cost Policies*

Os custos e os descontos apresentados na Seção 5.2 são definidos por meio da utilização do módulo *Cost Policies*. Esse módulo está implementado para receber custos referentes aos atributos funcionais ou não-funcionais. Todo o cálculo dos custos e da atribuição dos descontos ao grafo são efetuados no módulo *Cost Policies*.

O módulo *Cost Policies* recebe como parâmetro o grafo inicialmente com os custos uniformes 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. Inicialmente o desconto para cada serviço é calculado da forma descrita na Seção 5.2, e depois o grafo é percorrido alterando-se o custo para todos os serviços que obtiveram desconto. Por fim, o

módulo *Cost Policies* realiza as inserções dos nós virtuais (*Node0* e saídas requeridas), essenciais para a aplicação do algoritmo de cálculo do caminho de custo mínimo, e realiza todas as conexões semânticas (caso existam) entre os nós virtuais e os serviços do grafo.

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, como por exemplo, algoritmos que aceitam custos negativos[10] (que podem representar, por exemplo, promoções em preços de serviços pagos).

Para cada nó virtual referente a uma saída requerida, o módulo *Cost Algorithms* calcula um caminho de custo mínimo que parte do nó virtual *Node0*. Os detalhes e também uma avaliação da aplicação do algoritmo de Dijkstra para cálculo dos caminhos de custo mínimo para as saídas requeridas da requisição de um usuário são apresentados na Seção 5.3. Esses caminhos são utilizados pelo módulo *Graph2OWLS* para a geração da composição final.

Módulo *Graph2OWLS*

O módulo *Graph2OWLS* implementa o mapeamento da composição final para o OWL-S tal como descrito na Seção 5.6. 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.

Após gerar o grafo único que representa a composição final, o módulo *Graph2OWLS* realiza o mapeamento (apresentado na Tabela 5.6) dos construtores de *workflow* presentes no grafo para os construtores de fluxo do OWL-S. Toda composição já descrita em OWL-S é publicada no *jUDDI* para poder ser reutilizada posteriormente.

6.3 Avaliação da infra-estrutura implementada

Conforme descrito na Seção 6.2, 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

[10] O Algoritmo de Bellman-Ford é um algoritmo que resolve o mesmo problema do Algoritmo de Dijkstra e é normalmente usado apenas nos casos em que existem arestas de custo negativo.

ferramentas e APIs. Na prática, a ferramenta externa que realmente tem influência no desempenho da infra-estrutura apresentada na Seção 6.1 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, tal como descrito na Seção 4.2, 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.

Tabela 6.2: Configuração dos experimentos executados.

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

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 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 6.2: 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.

6.3.1 Descoberta

O algoritmo para descoberta apresentado no Capítulo 4 foi implementado no módulo *OWL-S Matching* da Figura 6.2 e o tempo de resposta para a busca por um serviço que possua as entradas e saídas requeridas pelo usuário é apresentado na Tabela 6.3. 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 6.3 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 6.2) foi utilizado na descoberta. O tempo da descoberta é influenciado pela inferência realizada pela máquina *Pellet* nos estágios iniciais do algoritmo de *matching*. 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 tal como pode ser observado na Tabela 6.3.

Tabela 6.3: 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

O tempo maior de resposta obtido nos testes foi de *1126ms*. Esse tempo máximo é utilizado nas seções 6.3.2 e 6.3.3 para avaliar também as funcionalidades de publicação e composição.

O resultado da descoberta por um serviço é uma lista de serviços que possuem similaridades com o serviço requerido. Essa lista é organizada na ordem de classificação definida pelos graus de similaridade apresentados na Seção 4.1.1.

6.3.2 Publicação

A publicação de um serviço como proposto neste trabalho executa duas funções distintas que são o registro do serviço no repositório UDDI e a inclusão do serviço no grafo utilizado para a composição. Essas funções são executadas em cinco procedimentos listados na Tabela 6.4.

Dessa forma, o desempenho da publicação do serviço depende dos cinco procedimentos (Tabela 6.4): 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 $In \rightarrow Out$).

Tabela 6.4: Tempo de resposta para a publicação de 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**

O registro no repositório UDDI é efetuado utilizando a ferramenta *OLWS2UDDI* descrita na Seção 6.2.1. A transformação do documento OWL-S para descritores UDDI demanda o maior tempo da publicação, como pode ser observado na Tabela 6.4.

A Tabela 6.4 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 6.3.

A Tabela 6.4 ainda 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.

Por fim, a Tabela 6.4 apresenta o tempo total de publicação de um serviço que é menor que *7752ms*. Esse tempo é aceitável uma vez que a publicação deve ser realizada pelo desenvolvedor do serviço e não influencia no tempo de descoberta e de composição do serviço. Entretanto, a decisão de montar o grafo no momento da publicação, inserindo um serviço por vez ao grafo, reduz consideravelmente o tempo da composição. Caso 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 ($|S| * 2 * 1126$), no qual $|S|$ é a quantidade de serviços. Considerando a amostra de mil serviços utilizada nesta seção, daria um tempo em torno de *38 minutos*, o que inviabilizaria a composição automática.

6.3.3 Composição

A realização da composição automática de serviços apresentada no Capítulo 5 executa cinco funções distintas: inserção do nó *Node0* 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 6.5[11].

Dessa forma, o desempenho da composição automática de serviços depende desses dez procedimentos (Tabela 6.5): 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

[11] 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.

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.

Tabela 6.5: Tempo de resposta para a composição de serviço.

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
Calculo dos descontos para entradas	$O(N_s * N_{InR})$	< 10
Calculo dos descontos para 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**

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 6.5.

A Tabela 6.5 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 do serviço 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 6.5 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 descrita na Seção 5.2, 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 6.5 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 6.5 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.

Por fim, a Tabela 6.5 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.

6.4 Considerações Finais

Este capítulo apresentou na Seção 6.1 uma infra-estrutura de software utilizada neste trabalho para a implantação de Serviços Web Semânticos. A infra-estrutura inclui funcionalidades para publicação, descoberta e composição de Serviços Web Semânticos que são descritas e avaliadas, respectivamente, nas seções 6.2 e 6.3.

A implementação e avaliação das funcionalidades para publicação, descoberta e composição de Serviços Web Semânticos tem como objetivo validar as propostas deste trabalho apresentadas nos capítulos 3, 4 e 5.

A arquitetura e os módulos da implementação realizada neste trabalho estão descritos na Seção 6.2. Cada módulo implementa uma função específica de acordo com os objetivos deste trabalho.

O desempenho, baseado na complexidade de algoritmos das referidas implementações e nos testes experimentais baseados nos tempos de respostas, apresentados na Seção 6.3, foram bastante satisfatórios e demonstram a viabilidade das propostas.

O Capítulo 7 discorre sobre as considerações finais desta tese, apresentando um resumo do trabalho realizado, as principais contribuições resultantes desta tese, as limitações das abordagens propostas bem como das referidas implementações e as

linhas de investigação e de trabalhos futuros visando a continuidade do referido trabalho.

CAPÍTULO 7

Conclusão

Nesta tese foram apresentados trabalhos que exploram a semântica explícita aplicada a Serviços Web para a automatização de tarefas que envolvem descoberta e composição de Serviços Web Semânticos. Este capítulo apresenta um resumo do trabalho realizado, as principais contribuições relativas a esta tese e as limitações das abordagens propostas no presente trabalho. Para finalizar, são apresentadas sugestões para continuidade deste trabalho.

7.1 Resumo do trabalho realizado

Motivado pelo problema inicial da integração de experimentos remotos em ambientes de aprendizado eletrônico na Web, o autor apresentou uma abordagem baseada em descrições sintáticas que possibilitava a integração dos experimentos remotos a um ambiente de aprendizado eletrônico [Prazeres et al., 2005; 2006b; Prazeres and Teixeira, 2006].

Entretanto, a integração baseada na sintaxe dos experimentos, que era manual e estática, envolve a participação de indivíduos na totalidade do processo. Nesse ponto, o autor passou a investigar os Serviços Web como meio de integração das aplicações e verificou que também demandam de integração manual em tempo de criação dos serviços. Considerando que a integração automática demandaria semântica explícita nas descrições das aplicações Web, o autor propôs uma plataforma para modelagem e descrição de aplicações Web na forma de Serviços Web Semânticos [Prazeres et al., 2007].

Estando os serviços descritos semanticamente, foi possível propor estratégias

para realizar a descoberta e a composição dos Serviços Web Semânticos de forma automática. Em relação à descoberta, foi proposto um algoritmo para descoberta automática de Serviços Web Semânticos [Prazeres et al., 2008]. Com relação à composição, foi proposto uma abordagem baseada em grafos e caminhos de custo mínimo para realização de composição automática de Serviços Web Semânticos [Prazeres et al., 2009c;d].

Para validar e avaliar os resultados, foi implementada uma infra-estrutura que provê funcionalidades para publicação, descoberta e composição de Serviços Web Semânticos [Prazeres et al., 2009b;a;c;d].

7.2 Contribuições

Esta tese apresenta contribuições envolvendo pesquisas relacionadas à Web de modo geral e a Serviços Web Semânticos de modo específico, áreas em que as tecnologias e padrões da Web Semântica são utilizados para prover automação de tarefas como descoberta, composição e invocação de serviços. As contribuições abrangem aspectos de modelagem e de descrição de serviços e incluem propostas para descoberta e composição automáticas. As principais contribuições referentes ao trabalho reportado nesta tese estão resumidas a seguir:

- Inicialmente foi proposta e implementada a modelagem e a descrição sintática de experimentos remotos com o objetivo de permitir sua integração a ambientes Web de aprendizado eletrônico [Prazeres et al., 2005; Prazeres and Teixeira, 2006]. Essa proposta foi validada por meio da integração do WebLab LaDABio[1] em um ambiente de aprendizado eletrônico, ambos no contexto do projeto TIDIA-Ae [Prazeres et al., 2006b].

- Com o objetivo de possibilitar descoberta e composição automáticas, tendo como motivação inicial os experimentos remotos, foi proposta a plataforma RALOWS para modelagem e descrição de aplicações Web na forma de Serviços Web Semânticos [Prazeres et al., 2007]. Posteriormente, a mesma abordagem foi aplicada em ambientes inteligentes para prover descoberta automática dos serviços disponíveis [Escobedo et al., 2007].

- Com o objetivo de promover descoberta automática de Serviços Web Semânticos foi proposto um algoritmo para descoberta de serviços com base nas funcionalidades providas pelo serviço, na categoria do serviço, nas pré-condições necessárias à execução do serviço e na disponibilidade do serviço ou necessidade de agendamento da sua execução [Prazeres et al., 2008].

[1] http://ladabio.deq.ufscar.br/kyatera.html

- Foi definido um modelo em grafo para composição automática de Serviços Web Semânticos. Além disso, políticas para atribuição de custos ao grafo dos serviços candidatos a participarem de uma composição foram estabelecidas. Definido o grafo, foi proposto um algoritmo para cálculo do caminho de custo mínimo. Os resultados foram utilizados com o objetivo de definir caminhos para realização da composição automática de serviços. O *workflow* da composição final foi gerado de forma automática a partir dos caminhos de custo mínimo. Por fim, foi realizado o mapeamento do *workflow* da composição final para construtores definidos em uma linguagem de ontologia para modelar *workflow* de processos [Prazeres et al., 2009c;d].

- Foi desenvolvida uma infra-estrutura para oferecer funcionalidades de descrição, publicação, descoberta e composição de Serviços Web Semânticos. Essa infra-estrutura possibilitou a validação e a avaliação das propostas de descoberta e composição automáticas e, como uma contribuição adicional necessária para as validações, incluiu funcionalidades para publicação de Serviços Web Semânticos [Prazeres et al., 2009a;c;d;b].

7.3 Limitações

Grande parte das limitações do trabalho reportado nesta tese são referentes a um ponto não coberto pela proposta: a invocação automática dos Serviços Web Semânticos. A descrição semântica para os Serviços Web, tal como já mencionado nesta tese, tem como objetivo principal a automação da descoberta, composição e invocação dos Serviços Web Semânticos. Entretanto, este trabalho não tratou das questões referentes à invocação automática e esta limitação poderá ser tratada em trabalhos futuros.

A invocação automática de Serviços Web é a limitação mais geral em termos dos objetivos gerais da proposta desta tese. Entretanto, também existem algumas limitações mais específicas deste trabalho que também podem ser tratadas em trabalhos futuros:

- algoritmos para cálculo de caminho de custo mínimo encontram apenas um único caminho mínimo dados um nó origem e um nó alvo, mesmo que existam mais de um caminho com os mesmos custos. Em relação aos custos e descontos funcionais propostos neste trabalho, caso existam mais de um caminho de custo mínimo é possível que se trate de caminhos que possuem serviços com as mesmas funcionalidades. Neste trabalho os outros caminhos de custo mínimo não são considerados;

- apesar de o algoritmo de cálculo de caminho de custo mínimo utilizado neste trabalho funcionar mesmo com a existência de ciclos no grafo (que são identificados e evitados), a modelagem de ciclos, se necessário, não foi tratada. Normalmente, serviços conectam-se entre si sem a existência de ciclos: cada serviço executa sua funcionalidade e passa os resultados para o próximo serviço que faz parte da composição. Entretanto, é possível que seja desejável a repetição no fluxo da composição e essa condição deve ser tratada;

- devido às limitações a respeito da invocação automática não ser tratada neste trabalho não é possível que o usuário estabeleça custos de forma dinâmica para a seleção *dinâmica* dos serviços. Isto ocorre porque caminhos únicos são criados a *priori* e as informações dinâmicas devem ser obtidas do serviço no momento da sua execução (invocação).

7.4 Trabalhos futuros

O trabalho apresentado nesta tese permite identificar alguns trabalhos futuros tanto para dar continuidade à pesquisa aqui reportada como para avaliar alternativas às limitações citadas na Seção 7.3. Uma lista não restritiva de sugestões para trabalhos futuros é a seguinte:

- investigação e avaliação das alternativas existentes na literatura para realizar invocação automática de Serviços Web Semânticos levando em consideração as propostas de descoberta e composição apresentados nesta tese;

- propor uma solução para modelar ciclos, caso necessário, na composição automática de serviços;

- investigação de outras abordagens para a modelagem do grafo para a composição, tal como a utilização de autômatos ou redes de fluxo;

- investigação do mapeamento da composição final gerada neste trabalho para padrões que descrevem a invocação de Serviços Web compostos (e.g. WSBPEL[2]) e das infra-estruturas que realizam a invocação com base nesses padrões (e.g. *Apache ODE*[3]);

- investigação de infra-estruturas baseadas em sistemas inteligentes multi-agentes e sua aplicação para a composição automática de serviços, de forma a possibilitar a seleção dinâmica dos serviços candidatos à composição;

[2]www.oasis-open.org/committees/wsbpel/
[3]http://ode.apache.org/

- aplicação dos resultados obtidos neste trabalho para composição automática para a integração de experimentos remotos a ambientes Web, motivação inicial deste trabalho, e também a integração entre experimentos remotos.

Referências Bibliográficas

Agrawal, A. and Srivastava, S. (2007). Weblab: A generic architecture for remote laboratories. In *ADCOM '07: Proceedings of the International Conference on Advanced Computing and Communications*, pages 301–306, Los Alamitos, CA, USA. IEEE Computer Society.

Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T., Sheth, A., and Verma, K. (2005). Web service semantics - WSDL-S. Disponível na Internet em http://www.w3.org/Submission/WSDL-S/. Visitado em 15/06/2009.

Akkiraju, R., Goodwin, R., Doshi, P., and Roeder, S. (2003). A method for semantically enhancing the service discovery capabilities of uddi. In *Web '03: Proceedings of the Workshop on Information Integration on the Web at the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 87–92.

Alamri, A., Eid, M., and Saddik, A. E. (2006). Classification of the state-of-the-art dynamic web services composition techniques. *Int. J. Web Grid Serv.*, 2(2):148–166.

Alfaro, L. and Henzinger, T. A. (2001). Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S. (2003). Business process execution language for web services, version 1.1. Disponível na Internet em http://www.ibm.com/developerworks/library/specification/ws-bpel/. Visitado em 15/06/2009.

Ankolekar, A. (2003). OWL-S: Semantic markup for web services. Disponível na Internet em http://www.daml.org/services/owl-s/1.0/owl-s.pdf. Visitado em 16/06/2009.

Ankolekar, A., Burstein, M., Hobbs, J. R., Lassila, O., Martin, D. L., McIlraith, S. A., Narayanan, S., Paolucci, M., Payne, T. R., and Sycara, K. (2001). DAML-S: A semantic markup language for web services. In *SWWS '01: Proceedings of the First Semantic Web Working Symposium*, Stanford University.

Antoniou, G. and van Harmelen, F. (2003). Web ontology language: OWL. In Staab, S. and Studer, R., editors, *Handbook on Ontologies in Information Systems*. Springer-Verlag.

Arkin, A., Askary, S., Bloch, B., Curbera, F., Goland, Y., Kartha, N., Liu, C. K., Thatte, S., Yendluri, P., and Yiu, A. (2005). Web Services Business Process Execution Language Version 2.0. Disponível na Internet em http://www.oasis-open.org/apps/org/workgroup/wsbpel/. Visitado em 15/06/2009.

Arpinar, B., Zhang, R., Aleman-Meza, B., and Maduko, A. (2005). Ontology-driven web services composition platform. *Information Systems and E-Business Management*, 3(2):175–199.

Arroyo, S., Lara, R., Gomez, J. M., Berka, D., Ding, Y., and Fensel, D. (2005). Semantic aspects of web services. *The Practical Handbook of Internet Computing*, pages 17–31.

Bagnasco, A., Chirico, M., and Scapolla, A. M. (2002). XML technologies to design didactical distributed measurement laboratories. In *IMTC '02: Proceedings of the 19th IEEE Instrumentation and Measurement Technology Conference*, pages 651–655, Washington, DC, USA. IEEE Computer Society.

Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., and Tabet, S. (2005). Semantic web services framework (SWSF) overview – W3C member submission. Disponível na Internet em http://www.w3.org/Submission/SWSF/. Visitado em 15/06/2009.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). Web Ontology Language (OWL), W3C Recommendation. Disponível na Internet em http://www.w3.org/TR/owl-ref/. Visitado em 15/06/2009.

Beckett, D. (2004). RDF/XML syntax specification (revised), W3C Recommendation. Disponível na Internet em http://www.w3.org/TR/rdf-syntax-grammar/. Visitado em 15/06/2009.

Berners-Lee, T. (2000). Semantic Web on XML. Disponível na Internet em http://www.w3.org/2000/Talks/1206-xml2k-tbl/Overview.html. Visitado em 15/06/2009.

Berners-Lee, T., Cailliau, R., Groff, J.-F., and Pollermann, B. (1992a). World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, 1(2):74–82.

Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., and Secret, A. (1994). The World-Wide Web. *Commun. ACM*, 37(8):76–82.

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

Berners-Lee, T. J., Cailliau, R., and Groff, J.-F. (1992b). The World Wide Web. In *JENC: Conference Procedings on 3rd Joint European Networking Conference*, pages 454–459, New York, NY, USA. Elsevier North-Holland, Inc.

Bond, J., Law, D., D., A. L., and Roxburgh, H. P. (2004). *Teach Yourself J2EE in 21 Days*. SAMS. 1002 pages. Second Edition.

Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Keller, U., Kifer, M., König-Ries, B., Kopecky, J., Lara, R., Lausen, H., Oren, E., Polleres, A., Roman, D., Scicluna, J., and Stollberg, M. (2005). Web service modeling ontology (WSMO) – W3C member submission. Disponível na Internet em http://www.w3.org/Submission/WSMO/. Visitado em 15/06/2009.

Bruijn, J., Lausen, H., Polleres, A., and Fensel, D. (2006). The web service modeling language wsml: An overview. *The Semantic Web: Research and Applications*, 4011/2006:590–604.

Bulcão Neto, R. F., Prazeres, C. V. S., and Pimentel, M. G. C. (2006). *Web Semântica: Teoria e Prática*, chapter 2, pages 47–86. Tópicos em Sistemas Interativos e Colaborativos. SBC Press. Texto referente a mini-curso proferido no Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia'06), Natal-RN, Brasil.

Burstein, M. H., Hobbs, J. R., Lassila, O., Martin, D., McDermott, D. V., McIlraith, S. A., Narayanan, S., Paolucci, M., Payne, T. R., and Sycara, K. P. (2002). DAML-S: Web Service Description for the Semantic Web. In *ISWC '02: Proceedings of the First International Semantic Web Conference on the Semantic Web*, pages 348–363, London, UK. Springer-Verlag.

Capobianco, D., Barbosa, M. Q., Santos, F. S., Prazeres, C. V. S., and Teixeira, C. A. C. (2005). The learning action modeling of responses in rats executed in the

TIDIA-Ae Environment. In *Workshop TIDIA '05: Anais do II Workshop do projeto TIDIA (Tecnologia da Informacao para o Desenvolvimento da Internet Avançada)*, page 2, Sao Paulo, SP, Brazil.

DAML (2000). DAML-ONT. Disponível na Internet em http://www.daml.org/2000/10/daml-ont.html. Visitado em 15/06/2009.

Dustdar, S. and Schreiner, W. (2005). A survey on web services composition. *Int. J. Web Grid Serv.*, 1(1):1–30.

Dürst, M. and Freytag, A. (2007). Unicode in XML and other Markup Languages. Disponível na Internet em http://www.unicode.org/unicode/reports/tr20/. Visitado em 15/06/2009.

Escobedo, E. P., Prazeres, C. V. S., Teixeira, C. A. C., Pimentel, M. G. C., and Kofuji, S. (2007). SeCoAS: An Approach to Develop Semantic and Context-Aware Available Services. In *WebMedia '07: Proceedings of the 13th Brazilian Symposium on Multimedia and the Web*, pages 1–8, Gramado, Rio Grande do Sul, Brazil. ACM Press.

FAPESP (2005). Tecnologia da Informação no Desenvolvimento da Internet Avançada – Aprendizado Eletrônico-Ae (E-learning). Disponível na Internet em http://tidia-ae.usp.br/. Visitado em 28/02/2009.

Farrell, J. and Lausen, H. (2007). Semantic Annotations for WSDL and XML Schema. Disponível na Internet em http://www.w3.org/TR/sawsdl/. Visitado em 15/06/2009.

Fensel, D. (2001). *Ontologies: a silver bullet for knowledge management and electronic commerce*. Springer-Verlag New York, Inc., New York, NY, USA.

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

Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., and Domingue, J. (2006). *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Fenza, G., Loia, V., and Senatore, S. (2008). A hybrid approach to semantic web services matchmaking. *Int. J. Approx. Reasoning*, 48(3):808–828.

Ferreira, J. M. and Muller, D. (2004). The MARVEL EU project: A social constructivist approach to remote experimentation. In *REV '04: Proceedings of the 11st Remote Engineering and Virtual Instrumentation International Symposium*, page 11p.

Fjeldly, T. A. (2003). *Lab on the Web: Running Real Electronics Experiments via the Internet*, volume 1. John Wiley & Sons, Inc., New York, NY, USA.

Gao, X., Yang, J., and Papazoglou, M. P. (2002). The capability matching of web services. In *MSE '02: Proceedings of the Fourth IEEE International Symposium on Multimedia Software Engineering*, pages 56–63, Washington, DC, USA. IEEE Computer Society.

Gekas, J. and Faslil, M. (2007). Employing graph network analysis for web service composition. *International Journal of Information Technology and Web Engineering*, 2(4):21–40.

Gómez-Pérez, A., Fernández-López, M., and Corcho, O. (2004). *Ontological Engineering with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Advanced Information and Knowledge Processing. Springer, 1st edition.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220.

Grüninger, M. and Menzel, C. (2003). The process specification language (PSL) theory and applications. *AI Mag.*, 24(3):63–74.

Harward, V. J., del Alamo, J. A., Lerman, S. R., Bailey, P. H., Carpenter, J., DeLong, K., Felknor, C., Hardison, J., Harrison, B., Jabbour, I., Long, P. D., Mao, T., Naamani, L., Northridge, J., Schulz, M., Talavera, D., Varadharajan, C., Wang, S., Yehia, K., Zbib, R., and Zych, D. (2008). The iLab shared architecture: A Web Services Infrastructure to Build Communities of Internet Accessible Laboratories. *Proceedings of the IEEE*, 96:931–950.

Hashemian, S. and Mavaddat, F. (2005). A graph-based approach to web services composition. In *Applications and the Internet, 2005. Proceedings. The 2005 Symposium on*, pages 183–189.

Hirtle, D., Boley, H., Grosof, B., Kifer, M., Sintek, M., Tabet, S., and Wagner, G. (2008). Schema specification of RuleML 0.91. Disponível na Internet em http://www.ruleml.org/0.91/. Visitado em 15/06/2009.

Hobbs, J. R., Ferguson, G., Allen, J., Fikes, R., Hayes, P., McDermott, D., Niles, I., Pease, A., Tate, A., Tyson, M., and Waldinger, R. (2004). An OWL ontology of time. Disponível na Internet em http://www.isi.edu/ pan/time/owl-time-july04.txt. Visitado em 15/06/2009.

Hobbs, J. R. and Pan, F. (2004). An ontology of time for the semantic web. *ACM Transactions on Asian Language Information Processing (TALIP)*, 3(1):66–85.

Hobbs, J. R. and Pan, F. (2006). Time Ontology in OWL. Disponível na Internet em http://www.w3.org/2001/sw/BestPractices/OEP/Time-Ontology. Visitado em 15/06/2009.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Disponível na Internet em http://www.w3.org/Submission/SWRL/. Visitado em 15/06/2009.

Jaeger, M. C., Rojec-Goldmann, G., Mühl, G., Liebetruth, C., and Geihs, K. (2005). Ranked matching for service descriptions using OWL-S. In Müller, P., Gotzhein, R., and Schmitt, J. B., editors, *Kommunikation in verteilten Systemen (KiVS 2005)*, Informatik Aktuell, pages 91–102, Kaiserslautern, Germany. Springer.

Kalepu, S., Krishnaswamy, S., and Loke, S. W. (2003). Verity: A QoS metric for selecting web services and providers. In *Web Information Systems Engineering Workshops, International Conference on*, pages 131–139, Los Alamitos, CA, USA. IEEE Computer Society.

Karadimas, D. and Efstathiou, K. (2007). An integrated platform, implementing real, remote lab-experiments for electrical engineering courses. In *WBED'07: Proceedings of the sixth conference on IASTED International Conference Web-Based Education*, pages 631–636, Anaheim, CA, USA. ACTA Press.

Klusch, M., Fries, B., and Sycara, K. (2006). Automated semantic web service discovery with OWLS-MX. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922. ACM Press.

Krasner, G. E. and Pope, S. T. (1988). A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49.

Kritikos, K. and Plexousakis, D. (2007). Requirements for QoS-based Web Service Description and Discovery. In *COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference*, pages 467–472, Washington, DC, USA. IEEE Computer Society.

Kuter, U., Sirin, E., Nau, D., Parsia, B., and Hendler, J. (2004). Information gathering during planning for web service composition. In *ISWC '04: Proceedings of the International Semantic Web Conference*, pages 335–349. Springer Berlin / Heidelberg.

Lacy, L. W. (2005). *OWL: Representing Information Using the Web Ontology Language*. Trafford, Victoria, BC, Canada.

Lammermann, S. (2002). *Runtime Service Composition via Logic-Based Program Synthesis*. PhD thesis, Stockholm, Sweden. Supervisor - Enn Tyugu.

Lassila, O. (2001). Resource ontology. Disponível na Internet em: http://www.daml.org/services/owl-s/1.0/Resource.owl. Visitado em 15/06/2009.

Li, L. and Horrocks, I. (2003). A software framework for matchmaking based on semantic web technology. In *WWW '03: Proceedings of the 12th International Conference on World Wide Web*, pages 331–339, New York, NY, USA. ACM Press.

Lomuscio, A., Qu, H., Sergot, M. J., and Solanki, M. (2007). Verifying temporal and epistemic properties of web service compositions. In *ICSOC'07: International Conference on Service-Oriented Computing*, volume 4749 of *LNCS*, pages 456–461, Vienna, Austria. Springer Berlin / Heidelberg.

Lynch, N. A. and Tuttle, M. R. (1987). Hierarchical correctness proofs for distributed algorithms. In *PODC '87: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, New York, NY, USA. ACM.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2004a). OWL-S: Semantic markup for web services – W3C member submission. Disponível na Internet em http://www.w3.org/Submission/OWL-S/. Visitado em 15/06/2009.

Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D. L., Sirin, E., and Srinivasan, N. (2007a). Bringing semantics to web services with OWL-S. *World Wide Web*, 10(3):243–277.

Martin, D., Cheyer, A., and Moran, D. B. (1999). The open agent architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1-2):91–128.

Martin, D., Denker, M. B. G., Elenius, D., Giampapa, J., McDermott, D., McGuinness, D., McIlraith, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2006). OWL-S 1.2 release. Disponível na Internet em http://www.daml.org/services/owl-s/1.2/. Visitado em 15/06/2009.

Martin, D., Domingue, J., Brodie, M. L., and Leymann, F. (2007b). Semantic web services, part 1. *IEEE Intelligent Systems*, 22(5):12–17.

Martin, D., Domingue, J., Sheth, A., Battle, S., Sycara, K., and Fensel, D. (2007c). Semantic web services, part 2. *IEEE Intelligent Systems*, 22(6):8–15.

Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., and Sycara, K. (2004b). Bringing semantics to web services: The OWL-S approach. In *SWSWPC '04: Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*, pages 26–42, San Diego, California, USA.

Masuoka, R., Parsia, B., and Labrou, Y. (2003). Task computing—the semantic web meets pervasive computing. In *ISWC '03: Proceedings of the Second International Semantic Web Conference*, pages 866–881, Sanibel Island, Florida, USA.

McGuinness, D., Fikes, R., Hendler, J., and Stein, L. A. (2002). DAML+OIL: an ontology language for the semantic web. *IEEE Intelligent Systems*, 17(5):72–80.

McGuinness, D. L. and van Harmelen, F. (2004). OWL Web Ontology Language Overview, W3C Recommendation. Disponível na Internet em `http://www.w3.org/TR/owl-features/`. Visitado em 15/06/2009.

McIlraith, S. and Son, T. C. (2002). Adapting Golog for composition of semantic web services. In *KR '02: Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning*, pages 482–493.

McIlraith, S. A., Son, T. C., and Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53.

Milanovic, N. and Malek, M. (2004). Current solutions for web service composition. *Internet Computing, IEEE*, 8(6):51–59.

Mitra, S., Kumar, R., and Basu, S. (2007). Automated choreographer synthesis for web services composition using i/o automata. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 364–371.

Nandigam, J., Gudivada, V. N., and Kalavala, M. (2005). Semantic web services. *J. Comput. Small Coll.*, 21(1):50–63.

Noia, T. D., Sciacio, E. D., Donini, F. M., and Mongiello, M. (2003). Semantic matchmaking in a P2P electronic marketplace. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 582–586, New York, NY, USA. ACM.

Noy, N. F. and McGuinness, D. L. (2001). Ontology development 101: A guide to creating your first ontology. TR KSL-01-05, Stanford Knowledge Systems Laboratory.

OASIS (2004). Introduction to UDDI: Important Features and Functional Concepts. Disponível na Internet em http://uddi.org/pubs/uddi-tech-wp.pdf. Visitado em 15/06/2009.

OntoKnowledge (2000). OIL. Disponível na Internet em http://www.ontoknowledge.org/oil/. Visitado em 15/06/2009.

Ozdogru, B. and Cagiltay, N. (2007). How content management problem of a remote laboratory system can be handled by integrating an open source learning management system? problems and solutions. In *Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007. IEEE 18th International Symposium on*, pages 1–5.

Paladini, S., Silva, J. B. d., Alves, G. R., Fischer, B. R., and Alves, J. B. d. M. (2008). Using remote lab networks to provide support to public secondary school education level. In *CSEWORKSHOPS '08: Proceedings of the 2008 11th IEEE International Conference on Computational Science and Engineering - Workshops*, pages 275–280, Washington, DC, USA. IEEE Computer Society.

Pan, F. (2005). Temporal aggregates for web services on the semantic web. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, pages 831–832, Washington, DC, USA. IEEE Computer Society.

Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K. P. (2002). Semantic matching of web services capabilities. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 333–347, London, UK. Springer-Verlag.

Paolucci, M., Soudry, J., Srinivasan, N., and Sycara, K. (2004a). *A Broker for OWL-S Web Services*, volume 13 of *Multiagent Systems, Artificial Societies, and Simulated Organizations: Extending Web Services Technologies*, chapter 4, pages 79–98. Springer US.

Paolucci, M., Srinivasan, N., and Sycara, K. (2004b). Expressing WSMO mediators in OWL-S. In *SWS '04: Proceedings of the Semantic Web Services Workshop at the Third International Semantic Web Conference*, page 15p.

Paolucci, M., Sycara, K., and Kawamura, T. (2003a). Delivering semantic web services. In *WWW '03: Proceedings of the Twelfth International World Wide Web Conference*, pages 111–118.

Paolucci, M., Sycara, K., Nishimura, T., and Srinivasan, N. (2003b). Using DAML-S for P2P discovery. In *ICWS '03: Proceedings of the International Conference on Web Services*.

Payne, T. and Lassila, O. (2004). Guest editors' introduction: Semantic web services. *IEEE Intelligent Systems*, 19(4):14–15.

Pimentel, M. G., Prazeres, C. V. S., Ribas, H., Lobato, D., and Teixeira, C. (2005). Documenting the pen-based interaction. In *WebMedia '05: Proceedings of the 11th Brazilian Symposium on Multimedia and the web*, pages 1–8, New York, NY, USA. ACM.

Prazeres, C. V. S., Lucredio, D., Fortes, R., and Teixeira, C. A. C. (2006a). Uma proposta de navegação na web utilizando facetas. In *ERBD '06: Anais da Escola Regional de Banco de Dados*, pages 1–6.

Prazeres, C. V. S., Pimentel, M. G. C., and Teixeira, C. A. C. (2006b). A document-based approach for weblab configuration. In *Workshop TIDIA '06: Anais do III Workshop do projeto TIDIA (Tecnologia da Informacao para o Desenvolvimento da Internet Avançada)*, pages 112–114, Sao Paulo, SP, Brazil.

Prazeres, C. V. S., Pimentel, M. G. C., and Teixeira, C. A. C. (2007). Remote experiments as semantic web services. In Yu, H., Naphade, M., and Koiti, H., editors, *ICSC '07: Proceedings of the 1st IEEE International Conference on Semantic Computing*, pages 791–798. IEEE CS Press.

Prazeres, C. V. S., Santos, F. S., Barbosa, M., Capobianco, D., and Teixeira, C. A. C. (2005). Integrating tools for accessing resources remotely into an e-learning environment. In *Workshop TIDIA '05: Anais do II Workshop do projeto TIDIA (Tecnologia da Informacao para o Desenvolvimento da Internet Avançada)*, pages 1–8, Sao Paulo, SP, Brazil.

Prazeres, C. V. S. and Teixeira, C. A. C. (2006). A structured document-based approach for WebLab configuration. In *WebMedia '06: Proceedings of the 12th Brazilian symposium on Multimedia and the Web*, pages 1–10, Natal, Rio Grande do Norte, Brazil. ACM Press.

Prazeres, C. V. S., Teixeira, C. A. C., Munson, E. V., and Pimentel, M. G. C. (2009a). Semantic web services: from OWL-S via UML to MVC applications. In *SAC '09: Proceedings of the 2009 ACM Symposium on Applied Computing*, pages 675–680, New York, NY, USA. ACM.

Prazeres, C. V. S., Teixeira, C. A. C., Munson, E. V., and Pimentel, M. G. C. (2009b). Toward semantic web services as MVC Applications: from OWL-S via UML. *Journal of Web Engineering*, 8(3):1–22.

Prazeres, C. V. S., Teixeira, C. A. C., and Pimentel, M. G. C. (2008). Semantic web services discovery by matching temporal restrictions. In *SAINT '08: Proceedings of*

the 2008 International Symposium on Applications and the Internet, pages 26–32, Washington, DC, USA. IEEE Computer Society.

Prazeres, C. V. S., Teixeira, C. A. C., and Pimentel, M. G. C. (2009c). Descoberta e composição de serviços web semânticos: de condições a alternativas. In *Submetido ao WebMedia '09: 15th Brazilian Symposium on Multimedia and the Web*, pages 1–8.

Prazeres, C. V. S., Teixeira, C. A. C., and Pimentel, M. G. C. (2009d). Semantic web services discovery and composition: paths along workflows. In *Proceedings of the ECOWS '09: 7th IEEE European Conference on Web Services*, pages 1–9.

Rao, J. (2004). *Semantic Web Service Composition via Logic-based Program Synthesis*. PhD thesis, Trondheim, Norway. Supervisor - Mihhail Matskin.

Rao, J. and Su, X. (2004). A survey of automated web service composition methods. In *SWSWPC '04: Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*, volume 3387/2005 of *Lecture Notes in Computer Science*, pages 43–54. Springer Berlin, Heidelberg.

Ross, R. J., Boroni, C. M., Goosey, F. W., Grinder, M., and Wissenbach, P. (1997). Weblab! a universal and interactive teaching, learning, and laboratory environment for the World Wide Web. *SIGCSE Bull.*, 29(1):199–203.

Sancristobal, E., Martin, S., Díaz, R. G. G., Colmenar, A., Castro, M., Peire, J., Gomez, J., Lopez, E., and Lopez, P. (2008). Integration of Internet Based Labs and Open Source LMS. *Internet and Web Applications and Services, International Conference on*, 0:217–222.

Silva, E. M. G., Pires, L. F., and Sinderen, M. J. (2007). An algorithm for automatic service composition. In *ICSOFT '07: Proceedings of the 1st International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing*, pages 65–74. INSTICC Press.

Sirin, E., Parsia, B., and Hendler, J. (2004a). Filtering and selecting semantic web services with interactive composition techniques. *IEEE Intelligent Systems*, 19(4):42–49.

Sirin, E., Parsia, B., Wu, D., Hendler, J. A., and Nau, D. S. (2004b). HTN planning for Web Service composition using SHOP2. *Journal of Web Semantics*, 1(4):377–396.

Skinner, B. F. (1959). The concept of the reflex in the description of behavior. *Cumulative record*, 3(6):319–346.

Srinivasan, N., Paolucci, M., and Sycara, K. (2004). An efficient algorithm for OWL-S based semantic search in UDDI. In *SWSWPC '04: Proceedings of the International Workshop Semantic Web Services and Web Process Composition*.

Srinivasan, N., Paolucci, M., and Sycara, K. (2006). Semantic Web Service Discovery in the OWL-S IDE. In *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 109.2, Washington, DC, USA. IEEE Computer Society.

Stal, M. (2002). Web Services: beyond component-based computing. *Communications of the ACM*, 45(10):71–76.

Swartz, A. (2006). The Semantic Web In Breadth. Disponível na Internet em http://logicerror.com/semanticWeb-long. Visitado em 15/06/2009.

Sycara, K. P., Paolucci, M., Ankolekar, A., and Srinivasan, N. (2003). Automated discovery, interaction and composition of semantic web services. *J. Web Sem.*, 1(1):27–46.

Teixeira, C. A. C., Capobianco, D., Prazeres, C. V. S., Santos, F. S., and Barbosa, M. (2005). Processo de modelagem de resposta: Refinando requisitos de software de apoio a laboratórios de acesso remoto. In *SBIE '05: Anais do XVI Simposio Brasileiro de Informática na Educação*, pages 518–528, Juiz de Fora, MG, Brazil.

Toderick, L., Mohammed, T., and Tabrizi, M. H. N. (2005). A consortium of secure remote access labs for information technology education. In *SIGITE '05: Proceedings of the 6th conference on Information technology education*, pages 295–299, New York, NY, USA. ACM.

Uszok, A., Bradshaw, J. M., Johnson, M., Jeffers, R., Tate, A., Dalton, J., and Aitken, S. (2004). Kaos policy management for semantic web services. *IEEE Intelligent Systems*, 19(4):32–41.

W3C (2001a). Namespaces in xml 1.1. Disponível na Internet em http://www.w3.org/TR/xml-names11/. Visitado em 15/06/2009.

W3C (2001b). Web ontology working group. Disponível na Internet em http://www.w3.org/2001/sw/WebOnt/. Visitado em 15/06/09.

W3C (2002). Web Services Activity. Disponível na Internet em http://www.w3.org/2002/ws/. Visitado em 15/06/2009.

W3C (2003). Simple Object Access Protocol (SOAP) 1.2, W3C Recommendation. Disponível na Internet em http://www.w3.org/TR/soap12-part0/. Visitado em 15/06/2009.

W3C (2004a). Extensible markup language (xml) 1.1. Disponível na Internet em http://www.w3.org/TR/2004/REC-xml11-20040204/. Visitado em 15/06/2009.

W3C (2004b). Resource description framework (rdf). Disponível na Internet em http://www.w3.org/RDF/. Visitado em 15/06/2009.

W3C (2004c). Resource description framework (rdf) schema specification. Disponível na Internet em http://www.w3.org/TR/rdf-schema/. Visitado em 15/06/2009.

W3C (2004d). Xml schema. Disponível na Internet em http://www.w3.org/XML/Schema. Visitado em 15/06/2009.

W3C (2006). Web Services Description Language (WSDL) 2.0, W3C Candidate Recommendation. Disponível na Internet em http://www.w3.org/TR/wsdl20/. Visitado em 15/06/2009.

Wei-Feng, Y., Rong-Gao, S., and Zhong, W. (2009). Distributed remote laboratory using web services for embedded system. In *CISST'09: Proceedings of the 3rd WSEAS international conference on Circuits, systems, signal and telecommunications*, pages 56–59, Stevens Point, Wisconsin, USA. World Scientific and Engineering Academy and Society (WSEAS).

WfMC (1996). Workflow management coalition: Terminology & glossary. Technical Report (WFMC-TC-1011), The Workflow Management Coalition.

Yan, Y., Liang, Y., and Du, X. (2005). Controlling remote instruments using web services for online experiment systems. In *IEEE International Conference on Web Services*, pages 725–732, Los Alamitos, CA, USA. IEEE Computer Society.

Yan, Y., Liang, Y., Du, X., Saliah-Hassane, H., and Ghorbani, A. (2006a). Putting labs online with web services. *IT Professional*, 8(2):27–34.

Yan, Y., Liang, Y., Du, X., Saliah-Hassane, H., and Ghorbani, A. (2006b). Using web services to control remote instruments for online experiment systems. In *EKAW '06: Proceedings of the 14th International Conference on Knowledge Engineering and Knowledge Management*, volume 3865/2006 of *Lecture Notes in Computer Science*, pages 205–214. Springer Berlin, Heidelberg.

Yao, Y., Su, S., and Yang, F. (2006). Service matching based on semantic descriptions. In *AICT-ICIW '06: Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, page 126. IEEE Computer Society.

Zhang, Y., Huang, H., Qu, Y., and Zhao, X. (2007). *Semantic Service Discovery with QoS Measurement in Universal Network*, volume 4585/2007 of *Rough Sets and Intelligent Systems Paradigms*, chapter 4, pages 707–715. Springer US.

Zhou, C., Chia, L.-T., and Lee, B.-S. (2004). DAML-QoS ontology for web services. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, page 472, Washington, DC, USA. IEEE Computer Society.

Zin, A. M. and Harun, H. (2007). A study on the potential of using remote labs for e-learning engineering courses in malaysian universities. In *ACS'07: Proceedings of the 7th Conference on 7th WSEAS International Conference on Applied Computer Science*, pages 140–144, Stevens Point, Wisconsin, USA. World Scientific and Engineering Academy and Society (WSEAS).

Glossário

DAML-S: Acrônimo para *DAML for Services*. Uma das linguagens propostas para ontologia de Serviços Web que serviu de base para a criação da linguagem OWL-S.

DAML: Acrônimo para *DARPA Agent Markup Language*. Uma das linguagens propostas para ontologia na Web que serviu de base para a criação da linguagem OWL.

DARPA: Acrônimo para *Defense Advanced Research Projects Agency*. A agência de pesquisas que, dentre outros projetos, foi a responsável pela criação das linguagens DAML e DAML-S.

DTD: Acrônimo para *Data Type Definition*. Uma linguagem para especificação da sintaxe de documentos XML.

Esquema XML: Uma linguagem para especificação da sintaxe e de tipos de dados de documentos XML.

HTML: Acrônimo para *HyperText Markup Language*. Linguagem de marcação para hiper-documentos da *World Wide Web*. Sua principal função é na apresentação de documentos legíveis apenas para humanos.

HTTP: Acrônimo para *HyperText Transfer Protocol*. Um protocolo para transferência de arquivos (texto, gráfico, som, vídeo e outros arquivos multimídia) na *World Wide Web*.

Interoperabilidade: A habilidade do *software* ou do *hardware* de fabricantes diferentes interagirem.

IOPR: Acrônimo para *Input* (entrada), *Output* (saída), *Preconditions* (pré-condições) e *Results* (resultados). Os IOPRs fazem parte da descrição do perfil de um serviço em OWL-S e são utilizados para descoberta e composição de serviços.

Matching: Processo de comparação entre dois conceitos com o objetivo de encontrar similaridades.

Namespace: Um documento unicamente identificado na Web, através de uma URI, que é utilizado para estabelecer a estrutura lógica de outro documento.

OASIS: Acrônimo para *Organization for the Advancement of Structured Information Standards*. Um consórcio internacional que guia o desenvolvimento, a convergência e a adoção de padrões de negócios eletrônico.

OIL: Acrônimo para *Ontology Inference Layer*. Uma proposta baseada na Web para a camada de inferência em ontologias

Ontologia: Uma especificação formal de como representar objetos, conceitos e entidades em alguma área de interesse e os relacionamentos entre eles.

OWL-S: É uma ontologia para descrever Serviços Web dentro de um arcabouço baseado em OWL.

OWL: Acrônimo para *Web Ontology Language*. Uma linguagem de ontologia para descrever classes e relacionamentos entre informações e aplicações na Web.

Proxy: Um servidor que atua como intermediário entre uma aplicação cliente e a Internet normalmente para controle administrativo, segurança, *cache* e registro de informações.

RDFS: RDF *Schema* é uma linguagem para descrição de vocabulários em RDF (*RDF Vocabulary Description Language*). Consiste de algumas estruturas básicas para ontologia, tal como o conceito de classes.

RDF: Acrônimo para *Resource Descriptor Framework*. Um conjunto de especificações para um modelo de metadados em grafos com o objetivo de fazer sentenças sobre um recurso.

RFC: Acrônimo para *Request for Comments*. Um conjunto de anotações técnicas e organizacionais a respeito da Internet.

RPC: Acrônimo para *Remote Procedure Call*. Método para comunicação entre processos no qual o cliente envia uma mensagem de requisição para um sistema remoto como se estivesse chamando um procedimento local.

SOAP: Acrônimo para *Simple Object Access Protocol*. É um protocolo que especifica exatamente como codificar uma mensagem HTTP em um documento XML de forma que um programa em um computador pode chamar um programa em outro computador e passar informações a este último.

SOA: Acrônimo para *Service Oriented Architecture*. Uma arquitetura de software que é baseada no conceito de serviços. É a arquitetura utilizada nos Serviços Web.

UDDI: Acrônimo para *Universal Description, Discovery, and Integration*. Um registro baseado em XML para serviços na Internet.

URI: Acrônimo para *Universal Resource Identifier*. O conjunto genérico de todos os nomes e endereços que referem-se a objetos (tipicamente na Internet).

W3C: Acrônimo para *World Wide Web Consortium*. Um consórcio internacional que objetiva garantir a padronização e interoperabilidade para a evolução da Web.

Web Semântica: É uma extensão da Web por meio do uso de padrões, linguagens de marcação e ferramentas de processamento, com o objetivo de dar significado ao conteúdo dos documentos para Web para que possam ser entendidos por máquinas.

WSDL: Acrônimo para *Web Services Description Language*. Uma linguagem para descrever Serviços Web que é baseada na linguagem XML. Essa linguagem é utilizada para descrever a interface de serviços de um Serviço Web, ou seja, o formato das suas mensagens de requisição e resposta.

WWW: Acrônimo para *World Wide Web*. Uma rede de computadores na Internet que fornece informação em forma de hipertexto. Para visualizar a informação, pode-se usar um software chamado navegador (*browser*) para obter informações (chamadas documentos ou páginas) de servidores de Internet (ou sites) e mostrá-los na tela do usuário.

XML: Acrônimo para *Extensible Markup Language*. É um formato de texto simples e muito flexível derivado do SGML. Originalmente projetado para o desafio de publicação eletrônica de grande escala, o XML é um importante padrão para intercâmbio de dados na Web e em qualquer outra aplicação.