Universidade Federal de Pernambuco

Graduação em Engenharia da Computação

Centro de Informática

2 0 0 8 . 2

Web Semântica na Automação de Composição de Web Services

Filipe Luiz Mélo da Costa Monteiro

TRABALHO DE GRADUAÇÃO

Recife

2008

Universidade Federal de Pernambuco
Centro de Informática

Filipe Luiz Mélo da Costa Monteiro

Web Semântica na Automação de Composição de Web Services

Monografia apresentada ao Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do Grau de Engenheiro Computação.

Orientador: *Prof. Ph.D.* Frederico Luiz Gonçalves de Freitas

Recife
2008

A meu pai

AGRADECIMENTOS

Em primeiro lugar, agradeço a meu pai por tudo. Agradeço por todos os momentos vividos e conselhos valiosos que a mim foram dados.

A minha mãe e Afonso, os quais sempre me ajudaram de todas as maneiras possíveis, e sem os quais eu provavelmente não teria sequer realizado uma graduação.

Aos amigos de universidade, que estiveram comigo todos esses anos, inclusive nas intermináveis noites “viradas“ trabalhando.

Ao meu orientador Fred Freitas por me mostrar o caminho a seguir em mais esta fase da minha vida.

Aos companheiros de Chemtech, especialmente Gustavo Melim, que me possibilitaram realizar este trabalho com maior tranqüilidade.

RESUMO

Os *Web Services* Semânticos consistem na parte dinâmica de um novo conceito de *Web*: a *Web* Semântica (também chamada de *Web* 3.0). A *Web* Semântica ajudará os usuários a delegarem tarefas aos *softwares*. Graças à inserção de significado, o processamento de requisições de usuários passará por deduções lógicas, de forma a se alcançar resoluções automaticamente.

O objetivo deste trabalho de graduação é explorar a *Web* Semântica dentro do contexto das técnicas aplicadas na composição e interoperabilidade de *Web Services*. Busca-se, primeiramente, uma análise sobre os principais avanços no uso de tecnologias associadas a tais serviços (com destaque para OWL-S). Posteriormente, propõe-se uma ferramenta que viabilize a síntese de novos serviços de maneira semi-automática, a partir de outros pré-existentes - com composição de funcionalidades.

Palavras-chave: *Web* Semântica, *Web Services*, *Web Services* Semânticos, composição de *Web Services*.

ABSTRACT

Semantic Web Services represent the dynamic part of a new Web concept: the Semantic Web (also known as Web 3.0). The Semantic Web will help its users delegating software additional tasks. Due to the insertion of semantics, the processing of user requisitions will pass through logical deductions achieving resolutions automatically.

The main purpose of this work is to explore the Semantic Web in means of techniques applied on composition and interoperability of Web Services. Initially, it is presented an analysis of the main technologies associated with those services (featuring OWL-S). Finally, it is proposed a tool responsible for the synthesis of new services (from other existing ones) in a semi-automatic fashion - with composition of functionalities.

Keywords: Semantic Web, Web Services, Semantic Web Services, Web Services composition.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
COMPOWS	Compositor de *Web Services*
GUI	*Graphical User Interface*
HTML	*Hyper Text Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
MVC	*Model-View-Controller*
OWL	*Ontology Web Language*
OWL-S	*Ontology Web Language for Services*
RDF	*Resource Description Framework*
RDFS	*Resource Description Framework Schema*
RDQL	*RDF Data Query Language*
SOAP	*Simple Object Access Protocol*
SPARQL	*Simple Protocol and RDF Query Language*
UDDI	*Universal Description, Discovery and Integration*
URI	*Universal Resource Identifier*
URL	*Universal Resource Locator*
W3C	*World Wide Web Consortium*
WS-BPEL	*Businness Process Execution Language for Web Services*
WSDL	*Web Services Definition Language*
WWW	*World Wide Web*
XHTML	*eXtensible HyperText Markup Language*
XML	e*Xtensible Markup Language*

1. INTRODUÇÃO

1.1. Contexto

Partindo de tão somente um repositório de mídia para uma provedora global de serviços, a *Web* evoluiu rapidamente nos últimos anos [1]. Tais serviços - *Web Services* - são providos por aplicações, bancos de dados, sensores e uma vasta gama de aparelhos conectados à rede mundial de computadores.

Com esse processo evolutivo, encaixa-se a necessidade de recursos computacionais responsáveis por automatizar o uso dos *Web Services* [2]. Seria necessário que agentes de *software* fossem capazes de automatizar os processos de descobrimento (como exemplo, encontrar serviços *on-line* para aluguel de carros), execução (como comprar um ingresso para um espetáculo) e composição (por exemplo, tomar todas as ações necessárias para o agendamento de uma viagem, desde a compra de passagens ao aluguel do hotel) de tais serviços.

Dentro desse contexto, torna-se extremamente desejável que a informação esteja mais bem definida na rede. Através da inserção de significado (semântica), torna-se possível a obtenção de uma melhor forma para o tratamento das necessidades dos usuários - agora, humanos e máquinas. Os recursos dos provedores de serviços na *Web* deverão ser, portanto, compreensíveis por programas de computador - criando-se desta forma *Web Services* Semânticos [2].

Os *Web Services* Semânticos consistem, portanto, na parte dinâmica dessa nova Web: a *Web* Semântica (também chamada de *Web* 3.0). Ela ajudará os usuários a delegarem tarefas aos *softwares*. Graças à inserção de significado, o processamento de requisições passará por deduções lógicas, de forma a se alcançar resoluções automaticamente.

1.2. Objetivo Principal

O objetivo principal deste trabalho de graduação é explorar a concepção da *Web* Semântica no contexto dos *Web Services*. Busca-se, mais objetivamente, uma análise sobre os principais avanços no uso das tecnologias propostas atualmente (com destaque para OWL-S). Esta análise tem como foco principal a resolução dos

principais problemas de automação computacional associados ao uso de *Web Services*: descobrimento, composição e execução.

1.3. Objetivos Específicos

Neste contexto, propõe-se a especificação de uma ferramenta (e desenvolvimento de um protótipo) que viabilize a síntese de novos serviços a partir de outros pré-existentes – agregando funcionalidades. Tal ferramenta – denominada COMPOWS (acrônimo para Compositor de *Web Services*) - deve atacar principalmente o problema da composição automática de *Web Services,* no tocante ao uso de semântica. Desta forma, abstrai-se um eventual módulo planejador, utilizando-se em seu lugar um usuário que deverá ser capaz de sintetizar composições de maneira semi-automática (de acordo com as descrições de serviços disponíveis em uma base de dados).

1.4. Relevância

O projeto escolhido teve como principal motivação a importância conferida atualmente aos *Web Services.* Diversas empresas utilizam os mesmos como principal meio de integração de sistemas. Adicionalmente, grandes corporações disponibilizam na *Web* serviços próprios, de forma que sejam utilizáveis por aplicações de terceiros, estendendo o alcance de seus produtos e influência [3]. Desta forma, considera-se relevante a realização de um estudo no sentido de demonstrar os principais benefícios a tais aplicações com o advento da *Web* Semântica [2].

1.5. Estrutura do Trabalho

Este trabalho é composto por quatro capítulos.

O **capítulo dois** contém uma discussão sobre os fundamentos de *Web Services*, bem como os avanços providos pela introdução de semântica aos mesmos.

O **capítulo três** mostra, inicialmente, a especificação da ferramenta COMPOWS para composição semi-automática de *Web Services*, com suas funcionalidades e arquitetura. Em um segundo momento, são tratados detalhes de

desenvolvimento da solução, e o uso de tecnologias na sua realização. Por fim, é mostrado um estudo de caso, ou seja, a utilização do protótipo da ferramenta na geração de uma composição e os resultados obtidos.

Finalizando, no **capítulo quatro** serão feitas considerações finais concernentes ao trabalho, juntamente com sugestões e trabalhos futuros.

2. WEB SERVICES

Neste capitulo, será feita uma discussão a cerca de *Web Services*. Inicialmente, serão mostradas definições e tecnologias utilizadas atualmente pela indústria. Em seguida, o foco passará às principais tarefas de automação relacionadas a esses serviços. E, por último, será demonstrada a viabilidade de resolução de tais problemas por meio do uso de *Web* Semântica.

2.1. Definição

Com a evolução da *Web* com o passar dos anos, as páginas da rede passaram de entes antes puramente estáticos a mecanismos dinâmicos, reagindo às requisições do usuário. Esta evolução foi responsável, portanto, por transformar a *Web* em uma provedora de serviços [2]. Dentro desse contexto, encaixam-se os *Web Services,* que nada mais são do que serviços que se utilizam da *Web* como meio físico.

Neste trabalho, define-se *Web Service* [4] como um sistema de software disponibilizado como um serviço na rede, constituindo, desta forma, um recurso. Tal serviço tem como propósito realizar a interoperabilidade de máquinas para um fim específico. Assim, entende-se que tais recursos são APIs (*Application Programming Interfaces*) de acesso via rede, de forma que outras aplicações possam usufruir de funcionalidades presentes no provedor.

Outra definição aplicável a *Web Services* deste trabalho é a utilizada em [5]: aplicações empresariais baseadas em *Web* que usam padrões XML [6] e protocolos de transporte abertos para troca de dados entre seus clientes solicitantes. Assim, entende-se que *Web Services* devem ser idealmente baseados em padrões abertos, independentes de plataforma e aplicações, além de poderem realizar intercâmbio de dados e recursos livremente.

2.2. Utilização

Ilustrando a utilização de tais sistemas, consideremos o seguinte cenário: uma empresa de grande porte tem vários de seus sistemas internos feitos em tecnologia *Web* (pensando em portabilidade e praticidade, já que eles necessitam apenas de um *browser* para execução), e deseja que tais aplicações compartilhem um único mecanismo de verificação e autorização de usuários (com acesso a um banco de dados). Uma saída bastante viável é a criação de outro sistema: um *Web Service* para realizar tal operação. Como conseqüência, os outros programas passarão a tratar a autorização de usuários de maneira única (usando-o como uma biblioteca *Web*) e sem a necessidade de múltiplos pontos de acesso ao banco de dados. Como exemplo de uso desta abordagem, a empresa Chemtech [7] se utiliza de *Web Services* como integradores de sistemas em alguns de seus projetos.

Agora, analisemos a seguinte situação: uma grande livraria, que vende seus produtos através da rede, deseja disponibilizar a páginas de terceiros o serviço de procura e listagem de itens em seu catálogo, de acordo com a necessidade do visitante. Tal serviço seria viabilizado por esta empresa através da disponibilização de *Web Services* públicos, de forma que desenvolvedores pudessem utilizar livremente essas bibliotecas em suas aplicações. Como exemplo desta possibilidade, temos as empresas Amazon (tendo vários *Web Services* disponibilizados em [8]) e eBay, esta última disponibilizando alguns serviços públicos para desenvolvedores *Web* com sucesso: 47% de suas listagens de produtos foram efetuadas através dessa tecnologia durante o ano de 2006 [3].

Os dois cenários mostram as facetas mais importantes de utilização atualmente dos *Web Services*: como um meio integrador de sistemas e como uma API de utilização pública.

2.3. Tecnologias

Uma característica marcante para *Web Services* é o uso de tecnologias específicas. Por exemplo, em [9] eles são definidos da seguinte forma: serviços distribuídos que processam mensagens SOAP [10] (*Simple Object Access Protocol*) codificadas em um arquivo XML [6] (*eXtensible Markup Language*), mandadas através de um pacote HTTP [11] (*Hypertext Transfer Protocol*, protocolo utilizado

para comunicação com transferência de dados em intranets e *World Wide Web*), e descritas em WSDL [12] (*Web Services Description Language*). Tal definição também pode ser estendida aos *Web Services* tratados neste trabalho, ainda que não nos restrinjamos exclusivamente a essas tecnologias. Nas próximas subseções, discutiremos em mais detalhes estas tecnologias associadas.

2.3.1. XML

XML [6] (*eXtensible Markup Language*) é uma linguagem de marcação desenvolvida pela W3C (*World Wide Web Consortium*) em 1996.

Criada com o intuito de facilitar o compartilhamento de informações através da *Internet*, o seu *design* prezou por uma série de pontos, dentre eles: seus documentos são legíveis e claros, a criação de processadores para arquivos é razoavelmente fácil, documentos XML são facilmente criáveis e sua estrutura dá suporte a uma vasta gama de aplicações.

Outra característica de destaque é a separação clara entre sua formatação e conteúdo: a sintaxe é baseada em *tags* (marcações), sendo as mesmas responsáveis por encapsular toda a informação do documento.

Adicionalmente, XML dá liberdade de criação ao programador, resultando em infinitas possibilidades de modelagem para uma mesma informação. Isto ocorre devido ao fato de XML não dar significado ao modo de se aninhar as *tags* (na figura 2.1 pode ser visto um exemplo desta característica), deixando a semântica para as aplicações.

XML é de grande importância para este trabalho, tendo várias das tecnologias da *Web* Semântica se utilizado da mesma como forma de representação sintática mais eficiente para tratamento computacional.

```
<curso nome="Matematica Discreta">
<professor>Davi Vila</professor>
</curso>
```

```
<professor nome="Davi Vila">
<curso>Matematica Discreta</curso>
</professor>
```

Figura 2.1: Trechos de dois arquivos XML distintos que contém a mesma informação.

2.3.2. WSDL

WSDL (*Web Services Description Language*) [12] é uma linguagem de descrição de *Web Services*, tornando-se uma recomendação da W3C em sua versão 2.0. Tal linguagem se utiliza de um formato XML para a modelagem de serviços como uma série de *endpoints* (ou portas) operando mensagens (contendo procedimentos ou documentos) que trafegam na rede.

Em WSDL, serviços são descritos a partir da definição de seis pontos básicos:

- *types* (tipos): representam as definições de tipos de dados usados nas mensagens trocadas;
- *message* (mensagem): constitui uma definição abstrata da informação enviada pelo *Web Service*;
- *portType* (tipo da porta): são as operações que o serviço pode realizar. Cada operação referencia uma série de mensagens de entrada e saída;
- *binding* (ligação): aqui devem ser especificados os protocolos (por exemplo, SOAP [10]) e formatos de dados utilizados nas mensagens e operações definidas por um determinado *portType*;
- *port* (porta): especifica o endereço para um *binding*, definindo, portanto, um único *endpoint*;
- *service* (serviço): é utilizado para agregar uma série de portas.

Dessa forma, uma aplicação que requer um serviço pode ler a descrição em WSDL de um *Web Service* de forma a “entender” como realizar a conexão com o mesmo, já que todas as informações a cerca dos protocolos e mensagens utilizadas devem estar bem definidas. Na figura 2.2, temos uma ilustração de como este processo ocorreria, partindo do pressuposto que o agente requerente do serviço tenha acesso ao WSDL publicado pelo *Web Service.*

WSDL é de fundamental importância para esta monografia, sendo o processo de execução de serviços viabilizado através de uma ligação (denominada g*rounding*) entre a descrição semântica (em OWL-S) e a “física” (em WSDL). Mais detalhes a cerca deste procedimento serão tratados na seção referente à OWL-S.

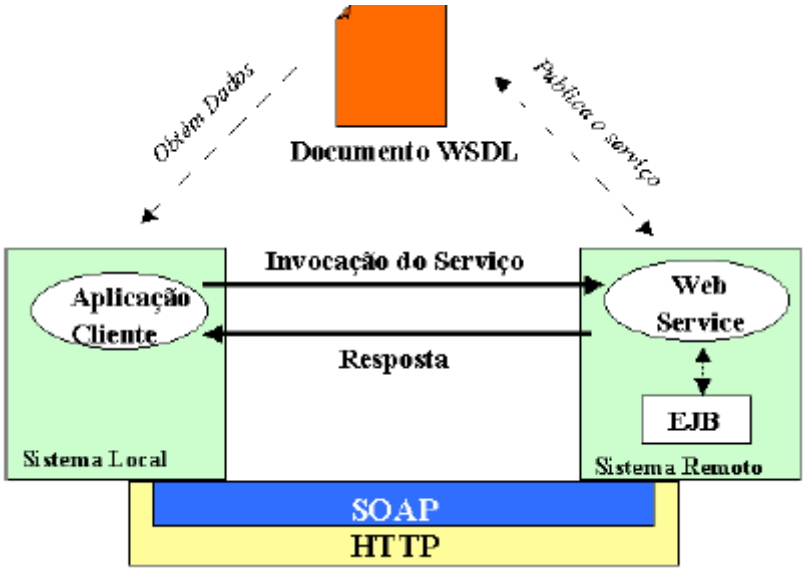

Figura 2.2: Cliente obtém dados a cerca de como usar um serviço publicado pelo *Web Service* através de sua descrição em WSDL. Tais dados são posteriormente usados para a conexão cliente-servidor (via SOAP e HTTP) [13].

2.3.3. SOAP

SOAP (*Service Oriented Architecture Protocol*) [10] é um protocolo para troca de informações estruturadas em um arquivo XML, para ambientes descentralizados e distribuídos. Trata-se de um paradigma *stateless* (sem-estado), de mensagens únicas normalmente encapsuladas em pacotes HTTP para tráfego de rede. Sistemas que utilizam SOAP podem criar padrões de interação mais complexos combinando o mesmo com outros protocolos a serem utilizados.

Atualmente, SOAP é o protocolo de escolha para a implementação de grande parte dos *Web Services* – sendo estes os chamados "*big*" *Web Services,* em oposição aos RESTful *Web Services* [14]. Nesta monografia, são tratados apenas os "*big*" *Web Services*, já que a arquitetura dos mesmos vem sendo utilizada há mais tempo dentro do contexto trabalhado.

2.4. Tarefas de Automação

Associado ao uso de *Web Services* temos algumas tarefas de automação importantes. O usuário (humano ou máquina) deveria poder executar determinadas atividades com dificuldade mínima, de forma a ter suas necessidades atendidas

automaticamente [15]. Nas próximas subseções serão detalhadas quais seriam essas tarefas de interesse.

2.4.1. Descobrimento

Entende-se por descobrimento automático [2] o processo de automação pelo qual se localizam serviços que fornecem um determinado conjunto de funcionalidades, e atendem às restrições especificadas pelo usuário.

Como exemplo, consideremos o caso em que um usuário deseja localizar um *Web Service* de venda de passagens aéreas entre duas determinadas cidades, e que aceite um cartão de crédito específico. A automação desta tarefa passa pela ausência de necessidade do usuário efetuar manualmente cada uma das seguintes atividades[15]:

- procura em um sítio de busca;
- análise das respostas encontradas;
- execução de um serviço selecionado a fim de determinar se o mesmo satisfaz às suas restrições.

2.4.2. Execução

A tarefa de execução automática [2] de *Web Services* compreende o conjunto de atividades que culminam na execução de um determinado serviço de interesse por um agente computacional, automaticamente. Este procedimento deve apenas considerar que o agente possui um meio de acessar a descrição do serviço (por exemplo, em WSDL), não sendo o mesmo pré-programado para comunicação com aquele sistema em particular.

Ilustrando o processo de execução, tem-se o caso em que o usuário deseja efetuar uma requisição de compra de passagem aérea para um determinado vôo em um serviço já conhecido. Assim, o sistema executor deve ser capaz de chamar uma série de procedimentos remotos para o usuário, de forma que o mesmo atinja o objetivo traçado com atuação direta mínima [15].

2.4.3. Composição

A atividade denominada Composição Automática de *Web Services* [15] envolve a seleção, composição e interoperabilidade automática dos serviços envolvidos para a execução de uma tarefa complexa, dada uma descrição em alto-nível do objetivo final.

Ilustrando o conceito definido, tem-se a situação em que o usuário deseja realizar todos os preparativos para uma viagem a uma conferência [2]. Desta forma, o sistema compositor deve ser capaz de planejar uma composição de *Web Services*, selecionando os serviços de interesse (no exemplo, serviços de compra de passagens áreas e reserva em hotéis) de forma que as entradas e saídas dos mesmos se complementem - ainda no caso de estudo, uma saída do serviço de passagens aéreas poderia ser a data de chegada ao destino final, fazendo com que o compositor utilizasse a mesma como entrada do *Web Service* de reserva em hotéis (para a data de início da reserva). Posteriormente, a aplicação compositora deveria executar automaticamente a composição formada a fim de atender à requisição do usuário. Por fim, a execução da composição pode ser dada com orquestração (onde o sistema compositor atua regendo a composição de *Web Services* como em uma orquestra, ativando a execução de cada serviço quando necessário) ou coreografia (onde cada *Web Service* sabe de sua função na composição, atuando da mesma forma que dançarinos em uma coreografia, executando-se automaticamente quando necessário).

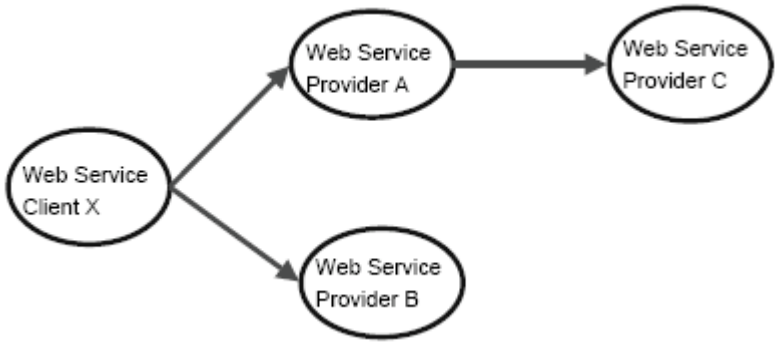

Figura 2.3: Exemplo de composição de *Web Services* [5].

2.5. Tecnologias Associadas às Tarefas de Automação

Nenhuma das três tarefas de automação associadas a *Web Services* é plenamente realizável com a *Web* atual [2]. A causa primordial para tal é a falta de uma linguagem capaz de caracterizar o conteúdo de tais serviços semanticamente. Apesar disto, diversas tecnologias foram criadas no sentido de se atingir, ao menos parcialmente, resoluções para estas atividades. Nas próximas subseções, discutiremos algumas destas tecnologias de interesse.

2.5.1. UDDI

Para a tarefa de descobrimento e execução automáticos, tem-se como destaque UDDI [16] (*Universal Description, Discovery and Integration*), definido como um repositório global de descrições de *Web Services*. Ao se utilizar de UDDI, a aplicação solicitante deve selecionar um serviço publicado (comunicando-se através de mensagens SOAP) de forma a receber um documento WSDL correspondente ao *Web Service* selecionado. A figura 2.4 (abaixo) ilustra como funciona tal procedimento.

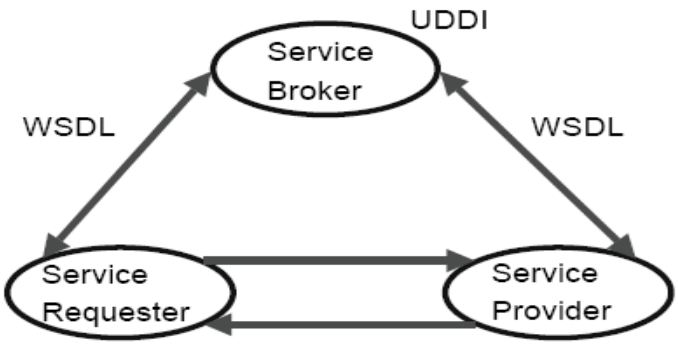

Figura 2.4: Cliente obtém dados a cerca de como usar um serviço publicado pelo servidor UDDI através de sua descrição em WSDL (obtida do provedor). [5].

Todavia, tal tecnologia apesar de ajudar na tarefa de descobrimento de maneira fundamental, não vingou como uma solução global: não existem muitos servidores UDDI na *Internet*, e poucas descrições de serviços publicados são encontradas dentro dos mesmos [17]. Dessa forma, UDDI passou a ser uma boa solução apenas para casos de utilização em *intranets*, e pouco eficiente fora delas [17].

2.5.2. WS-BPEL

Considerando-se a atividade de composição com interoperabilidade automática de serviços, destaca-se WS-BPEL [18] (*Businness Process Execution Language for Web Services*) como solução aplicada atualmente pela indústria. WS-BPEL é uma linguagem que se utiliza de uma gramática XML para a definição e padronização de estruturas utilizadas na orquestração de *Web Services* [5].

Dessa forma, a composição formada (denominada processo em sua terminologia) por WS-BPEL é baseada em um *workflow* (fluxo de trabalho) pré-modelado, composto por um conjunto de atividades (que correspondem à troca de mensagens, ou à transformação de resultado intermediária) [5]. Como exemplo destas atividades, temos:

- Atividades básicas: aquelas que efetivamente realizam algo palpável (por exemplo, *wait* que força uma espera);
- Atividades estruturadas: aquelas que organizam as atividades básicas, sem realizar nenhuma transação efetivamente (por exemplo, *sequence* que organiza uma execução seqüencial de atividades básicas). Sendo tais atividades as responsáveis por montar efetivamente a composição de serviços.

WS-BPEL dá ainda suporte a uma série de controles de fluxo, em especial:

- Variáveis: que armazenam mensagens ou dados;
- Manipulação de erros: permitindo a captura e tratamento de erros;
- Manipulação de compensação: permitindo desfazer passos da composição que já foram completados;
- Correlação de mensagens: permitindo que diferentes processos participem de conversações *stateful* (com controle de estado).

Entretanto, WS-BPEL foca a representação de composições nas quais o fluxo de processos e conexões entre os serviços são conhecidos a priori, ou seja, estaticamente [19]. Portanto, considera-se inadequado seu uso para o problema de composição dinâmica de *Web Services*, mais desafiador que o anterior, nos quais os serviços são requisitados sob demanda [19]. Tal tarefa de composição dinâmica [20] consiste na utilização de serviços existentes para a realização de uma

funcionalidade necessária, a qual não poderia ser diretamente realizada pelos *Web Services* presentes na composição até o momento [21].

A composição dinâmica de serviços requer a compreensão das capacidades de *Web Services* (em termos do que eles podem fazer) e a compatibilidade destes serviços [19]. WS-BPEL, então, não seria capaz de realizar tal operação com sucesso devido ao fato de não ter um bom controle de seu fluxo de trabalho em tempo de execução de composições [5].

Outra dificuldade mais forte a ser superada para a composição dinâmica seria a ponte conceitual entre as diferentes definições de serviços e o que elas realmente significam (ou seja, a extração de semântica), uma fraqueza de WS-BPEL [5]. Esta última dificuldade, portanto, deve ser superada com o uso de tecnologias que agreguem significado para as diferentes terminologias utilizadas pelos humanos e seus equivalentes formatos de dados em linguagem de máquina [19]. Aqui, encaixa-se a necessidade de tecnologias associadas a um novo conceito de *Web*, a *Web* Semântica [19].

Nas próximas seções, serão abordados temas referentes à *Web* Semântica, sua definição e tecnologias associadas. Em seguida, voltaremos a focar os serviços na WWW, discutindo a concepção de *Web Services* Semânticos.

2.6. Web Semântica

A *Web* Semântica [22], conceito propagado pela W3C, é uma extensão para a *Web* atual onde o conteúdo publicado será mais facilmente processável pelas máquinas (devido à inserção de significado, semântica), de forma a se automatizarem tarefas que antes dependiam exclusivamente (ou pesadamente) de um agente humano.

A concepção atual da *Web* Semântica se baseia na evidência de que a grande maioria do conteúdo da WWW destina-se ao consumo exclusivamente humano. Até o conteúdo que é gerado automaticamente a partir de bancos de dados é normalmente apresentado sem sua informação estrutural associada (encontrada originalmente nos bancos de dados). Assim, o uso típico da *Web* envolve pessoas pesquisando e processando informações, procurando e contatando outras pessoas, revisando catálogos de lojas *online* e comprando produtos através

do preenchimento de extensos formulários [23]. A visão inicial da *Web* Semântica prevê, portanto, um papel mais importante para o significado da informação do que o atual, fazendo com que boa parte destas atividades desnecessárias seja passível de automação por sistemas inteligentes.

É importante ressaltar que o desenvolvimento da *Web* Semântica tem o suporte da indústria, e governos de diversos países têm investido pesadamente. O governo dos Estados Unidos estabeleceu o *DARPA Agent Markup Language (DAML) Project*, e a *Web* Semântica também figura entre os principais pontos do *Sixth Framework Programme* da União Européia [23].

Na próxima seção, discutir-se-á a cerca das bases estruturais para que a visão da *Web* Semântica se torne realidade.

2.7. Bases Estruturais da *Web* Semântica

Para que se atinja a visão da *Web* Semântica (como mostrada na seção anterior), faz-se necessária a utilização de um conjunto de tecnologias, constituindo uma extensão para a WWW atual. Aqui, apresenta-se uma visão estruturada para esta nova versão da *Web*: fazendo com que cada camada se baseie em uma tecnologia totalmente compatível com as inferiores a ela mesma [23]. Nas figuras 2.5 e 2.6 (mais abaixo), temos formatos de pilhas tecnológicas de interesse para a *Web* Semântica.

Nas próximas subseções, trataremos de cada uma das mais importantes camadas separadamente, em uma abordagem seguindo o fluxo de dependências (ou seja, de baixo para cima), focando apenas as tecnologias de interesse para este trabalho. Excetuamos desta análise a camada XML, pois tal tecnologia já foi debatida na subseção 2.3.1. Porém, frisa-se aqui que XML representa o formato de serialização de escolha para os documentos da *Web* Semântica, fazendo com que muitas das tecnologias a serem discutidas se utilizem de sua notação específica.

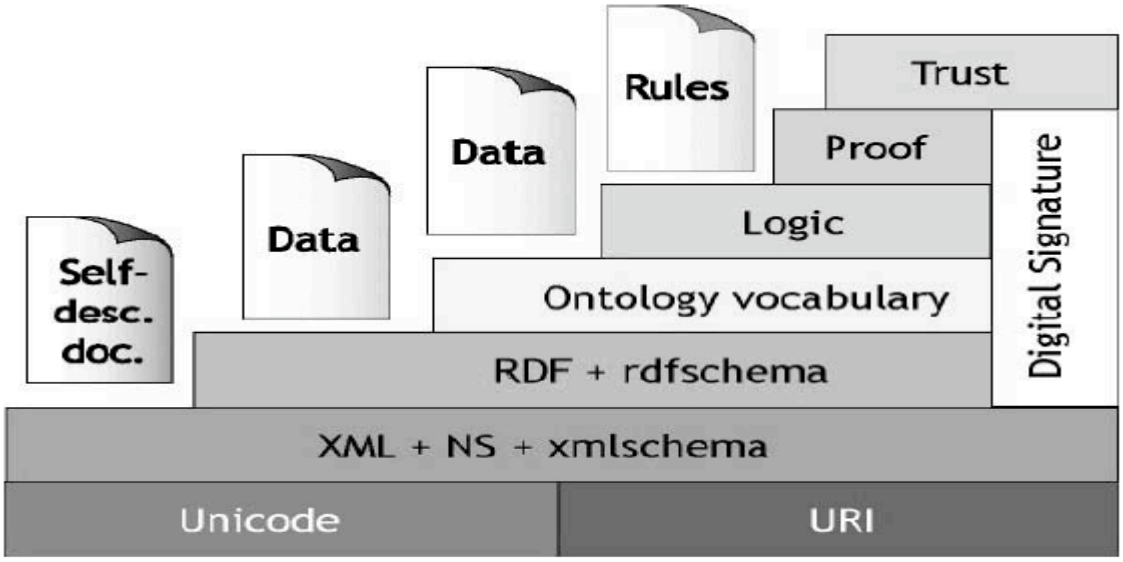

Figura 2.5: Estrutura de pilha em camadas para a *Web* Semântica [23].

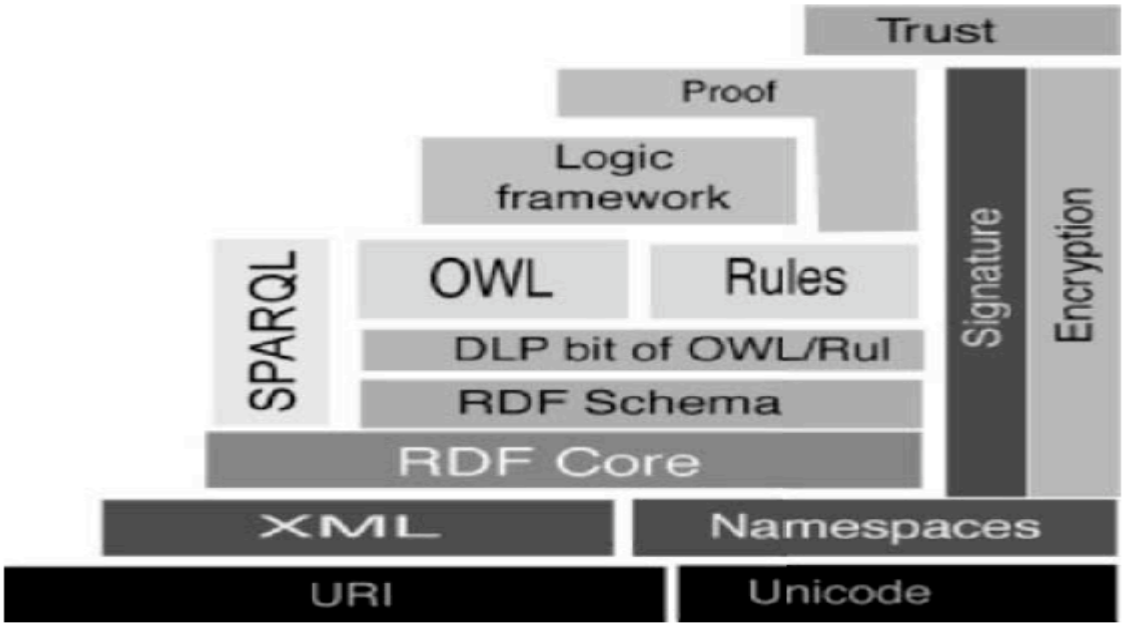

Figura 2.6: Estrutura alternativa de pilha em camadas para a *Web* Semântica [23].

2.7.1. RDF e RDF *Schema*

Imediatamente acima de XML (em ambas as pilhas mostradas) está RDF (*Resource Description Framework*) [24]. RDF é um tipo de modelagem de dados básica, como um modelo entidade-relacionamento, para se descrever afirmações (indicações, *statements*) sobre objetos *Web* (ou seja, recursos, *resources*) [23]. É

importante destacar que RDF não depende inteiramente de XML: porém, provê uma sintaxe baseada na mesma, desta forma se localizando acima na pilha da *Web* Semântica [23].

Para a concretização da visão da *Web* Semântica, a W3C considerou a necessidade de substituição de HTML (*Hyper Text Markup Language*) [25], esta sendo a linguagem predominante para a escrita de páginas *Web* atualmente. Tal mudança faz-se necessária devido a uma fraqueza de HTML: a formatação do conteúdo de suas páginas não tem tratamento semântico (em nível de marcação), pois seus documentos destinam-se ao consumo exclusivamente humano. A idéia é que uma mudança de tecnologia nesse nível propiciaria a ausência de necessidade de criação de sistemas inteligentes complexos específicos para extração de informações presentes na *Web*. A tecnologia substituta deveria explicitar os metadados (dados sobre dados) associados, provendo significado aos mesmos (ou seja, semântica) especificamente para a *Web* [23].

Dessa forma, surge RDF, aparecendo como uma alternativa à utilização de XML - ou até mesmo de XHTML (*eXtensible HyperText Markup Language*) [26] que une XML e HTML, já que XML também explicita dados sobre dados - devido às falhas desta última do ponto de vista da *Web* Semântica: a falta de amarras ao modo de se aninhar as marcações (*tags*) gera ambigüidade (como mostrado na subseção referente à XML), o que dificulta a extração de significado de suas estruturas [23]. Assim, RDF corrige as falhas de XML, oferecendo uma forma sintática nesta linguagem (mais adequada ao tratamento computacional).

RDF se baseia em três conceitos fundamentais [23], sendo estes:

- *Resources* (recursos): são as coisas, objetos, aos quais se referem as indicações em RDF. Os recursos são, portanto, qualquer coisa que se queira modelar: inclusive as próprias páginas *Web*, ou objetos do mundo real. Todo recurso dispõe de um URI (*Universal Resource Identifier*) para identificação única deste objeto na *Web* – sendo URL (*Universal Resource Locator*) uma especialização de URI. Porém, um URI não necessariamente provê acesso ao recurso, podendo consistir apenas em um identificador.

- *Properties* (propriedades): As propriedades são responsáveis por definir relacionamentos entre recursos. Por exemplo, a propriedade “escrito por” poderia ser aplicada a um recurso “livro”.
- *Statements* (indicações): Por fim, o terceiro elemento de RDF define uma tripla, agregando uma propriedade a um recurso e um valor. Desta forma, um valor poder ser outro recurso ou um literal qualquer (como um tipo primitivo, por exemplo, uma seqüência de caracteres ou um número inteiro).

Assim, uma modelagem em RDF pode ser vista como um conjunto de triplas, que corresponderiam a uma frase com sujeito (recurso), predicado (propriedade) e objeto (o valor). Nas figuras 2.7 e 2.8 (na página seguinte), temos um exemplo de modelagem RDF com diferentes representações (em formatos de XML e grafo, respectivamente).

2.7.1.1. RDFS

Perto de RDF na pilha da *Web* Semântica, temos RDF *Schema* (RDFS) [27]. Tal tecnologia corresponde a uma linguagem de descrição para estabelecer o vocabulário de propriedades e classes para recursos em RDF [23]. RDF *Schema* provê, portanto, primitivas de modelagem para a organização hierárquica de objetos *Web* - com conceitos como: classes, propriedades, subclasses, “subpropriedades”, relacionamentos, e restrições de domínio e alcance.

2.7.1.2. SPARQL

SPARQL (*Simple Protocol and RDF Query Language*) [28] é uma linguagem de consulta a modelagens RDF, tornando-se uma recomendação da W3C para a *Web* Semântica em 2008. SPARQL pode ser utilizada para expressar consultas a diversas fontes de dados, estando o dado originalmente em RDF ou apenas visualizado em RDF através de *middlewares*. SPARQL também oferece teste de valores e imposição de restrições às suas consultas no grafo fonte da modelagem RDF usada.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://www.cin.ufpe.br/">
    <dc:title>Centro de Informática</dc:title>
  </rdf:Description>
</rdf:RDF>
```

Figura 2.7: Modelagem RDF em XML de uma tripla que estabelece o título da página *Web* de URL http://www.cin.ufpe.br como "Centro de Informática".

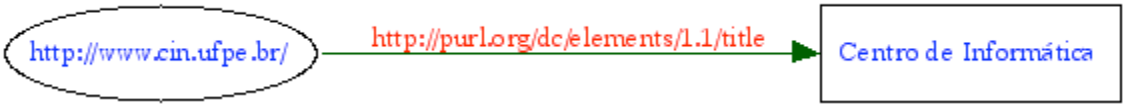

Figura 2.8: Modelagem RDF como um grafo para a mesma tripla da figura 2.7 (gerado pela ferramenta W3C RDF Validator [29]).

2.7.2. Ontologia

Acima de RDF na pilha da *Web* Semântica, temos a camada de ontologia, ou de vocabulário ontológico. O termo ontologia, em computação, refere-se a uma especificação formal explícita de um conceito formado, definindo-se um domínio de discurso - no caso da *Web,* provendo um conceito compartilhado de um referido domínio.

Uma ontologia tipicamente estabelece uma lista de termos (conceitos importantes, como classes) e seus relacionamentos (hierarquias) [23]. Como visto na seção anterior, RDF *Schema* estabelece conceitos equivalentes a esses, podendo ser vista como uma linguagem primitiva para a definição de ontologias. Porém, existe a necessidade de uma ferramenta ontológica mais expressiva que expanda RDF *Schema*, permitindo representações de relacionamentos mais complexos entre objetos *Web.*

Na próxima subseção, discutir-se-á a cerca de uma linguagem ontológica poderosa, construída a partir de RDF para a *Web*.

2.7.2.1. OWL

A necessidade de uma linguagem ontológica mais poderosa para a *Web* surgiu quando a W3C identificou a necessidade de mais expressividade para a

definição de alguns casos de uso característicos da *Web* Semântica. Tais casos de uso não poderiam ser modelados a partir da utilização simples de RDF e RDF *Schema* devido a uma série de limitações encontradas. Algumas destas limitações mais importantes são [23]:

- Escopo local de propriedades: *rdfs:range* define o alcance de uma propriedade globalmente, não permitindo restrições que só se apliquem localmente a uma propriedade (por exemplo, dentro de uma classe específica);
- Disjunção entre classes: RDFS não prevê que algumas classes possam ser disjuntas entre si. Por exemplo, as classes “Macho” e “Fêmea” deveriam ser disjuntas, porém não há como se modelar isto em RDFS;
- Combinações booleanas de classes: A construção de novas classes a partir de operadores de união, intersecção e complemento não é provida por RDFS. Por exemplo, a classe “Pessoa” poderia ser a união das classes disjuntas “Mulher” e “Homem”;
- Cardinalidade: A definição de restrições de cardinalidade, estabelecendo quantos valores distintos uma determinada propriedade pode assumir. Por exemplo, a classe “Pessoa” poderia ter uma restrição na propriedade “pais” que determinasse cardinalidade dois. Novamente, tais restrições são impossíveis em RDFS;
- Características especiais de propriedades: RDFS não prevê a utilização de transitividade (por exemplo, “maior que”), unicidade (“por exemplo, “é mãe de”) e inversão (por exemplo, “come” e “é comido por”) em propriedades, dentre outras possibilidades.

Dessa forma, OWL (*Web Ontology Language*) [30] é uma linguagem de descrição específica para a *Web*, que dispõe de mais expressividade do que RDF *Schema* para a definição de propriedades e classes. OWL foi construída a partir de RDF e RDFS (também mantendo uma forma de representação sintática em XML de uso primário), definindo meios de modelagem para as limitações discutidas. Na figura 2.9 (abaixo) temos um trecho de uma ontologia em OWL para a modelagem da natureza. Observa-se a utilização dos construtores *owl:disjointWith* para

disjunção entre classes, *owl:someValuesFrom* para restrição de valores em propriedades e *owl:intersectionOf* para intersecções entre classes.

```
<owl:Class rdf:ID="animal"> <rdfs:comment> Classe Animal </rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="planta">
<rdfs:comment> Classe Planta. Uma planta não pode ser um animal. </rdfs:comment>
<owl:disjointWith rdf:resource="#animal"/> </owl:Class>

<owl:Class rdf:ID="arvore"> <rdfs:comment>  Uma árvore é uma subclasse de planta.
<rdfs:subClassOf rdf:resource="#planta"/>
</owl:Class>

<owl:Class rdf:ID="carnivoro">
<owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#animal"/>
<owl:Restriction>
<owl:onProperty rdf:resource="#come"/>
<owl:someValuesFrom rdf:resource="#animal"/>
</owl:intersectionOf>
</owl:Class>
```

Figura 2.9: Trecho de uma modelagem em OWL para a natureza. Observa-se que um carnívoro é um animal que come outros animais.

Apesar da alta capacidade expressiva de OWL, é importante ressaltar que seu *design* prezou pela eficiência no suporte à construção de *softwares* lógicos para inferência em cima de suas modelagens. Porém, como geralmente existe um *trade-off* entre poder de expressão e de raciocínio em cima de linguagens ontológicas, a W3C definiu OWL em três versões - de acordo com a necessidade de sua utilização. São elas:

- OWL *Lite*: Trata-se de uma versão restrita de OWL em termos de expressividade, suportando principalmente classificação hierárquica e restrições simples. Assim, OWL *Lite* é de mais fácil compreensão do que OWL DL e OWL *Full*;
- OWL DL (*Description Logic*): É uma extensão de OWL *Lite* da linguagem orientada a manter expressividade máxima até o limite da eficiência plena de raciocínio lógico computacional. OWL DL faz isso apenas impondo restrições às construções de OWL, sem as retirar da linguagem. Uma desvantagem de OWL DL é a falta de compatibilidade total com RDF, que também é um dos benefícios de OWL *Lite*.

Ressalta-se que conclusões lógicas e ontologias válidas em OWL *Lite* também têm validade em OWL DL;

- OWL *Full*: Trata-se da versão completa de OWL, de máxima expressividade. Um de seus benefícios é a completa compatibilidade sintática e semântica com RDF. Porém, devido ao seu alto poder de expressão, perde-se capacidade de inferência lógica computacional completa sobre suas modelagens. Ressalta-se que conclusões lógicas e ontologias válidas em OWL DL também têm validade em OWL *Full*.

2.8. *Web Services* Semânticos

Graças à inserção de semântica à *Web*, espera-se que agentes de software tornem-se capazes de localizar, selecionar, utilizar, compor, e monitorar serviços *Web* automaticamente, de forma a atenderem as demandas de seus usuários [2].

Entretanto, para fazer uso de um *Web Service,* um agente de *software* precisa de uma descrição de serviço que seja interpretável computacionalmente, e que forneça os meios pelos quais o mesmo pode ser acessado. Desta forma, a utilização da *Web* Semântica tornará viável a existência de uma rede semântica de serviços, desde que suas propriedades, capacidades, interfaces e efeitos estejam codificados em um formato livre de ambigüidades [2].

Um ponto importante da Web Semântica é, portanto, o estabelecimento de um *framework* para viabilizar a criação e compartilhamento de descrições para serviços. Deve-se utilizar, nesta extensão da *Web*, uma ontologia definida como padrão, que consista em um conjunto básico de classes e propriedades para a declaração e descrição de *Web Services* [15]. Para isto, sabe-se que OWL tem uma representação apropriada, compatível com o novo modelo de *Web* proposto, onde uma nova ontologia pode estender suas capacidades especificamente para o tratamento de serviços [15].

Na próxima seção, trataremos da ontologia feita para a *Web* Semântica que foca justamente na utilização de serviços, construída a partir de OWL.

2.9. OWL-S

OWL-S (*Web Ontology Language for Services*) [15] é a ontologia *Web* voltada para serviços na rede, construída em cima de OWL a partir de um esforço colaborativo entre diversos membros da W3C para a viabilização de uma rede semântica de serviços.

Nesta seção, primeiramente discutiremos sobre a ontologia de serviços de OWL-S, e suas sub-ontologias (denominadas *profile*, *process* e *grounding*). Posteriormente, mostraremos como OWL-S torna possível a resolução das tarefas de automação ligadas a *Web Services* (como mostrado na seção 2.4), já que é esta a principal motivação para o desenvolvimento de tal tecnologia.

Ressalta-se que exemplos de descrições de *Web Services* em OWL-S podem ser encontradas no Apêndice II.

2.9.1. Ontologia de Serviços

Em OWL-S, *Web Services* são considerados simples (atômicos, *atomic*) ou complexos (compostos, *composite*) [15]. Os serviços atômicos são aqueles que apenas um agente acessível na *Web* está envolvido no processamento da requisição do usuário – recebendo a mensagem de requisição e atuando para produzir o efeito desejado. Em contraste, serviços complexos são compostos de múltiplos serviços primitivos, e podem requerer uma interação maior entre o autor da requisição e o conjunto de serviços utilizados. OWL-S foi feito para suportar ambas as categorias de serviços, tendo os serviços complexos motivado muito dos elementos de tal ontologia.

A seguir, discutiremos as sub-ontologias de serviços de OWL-S, que respondem às principais questões ligadas a descrição semântica de *Web Services.*

2.9.1.1. Classe *Profile*

Profile (ou *ServiceProfile*), em OWL-S, é a sub-ontologia que se encarrega de determinar o que um serviço provê a seus clientes. Desta forma, diz-se que em OWL-S um serviço apresenta as suas capacidades através de seu *Profile,* como pode ser visto na figura 2.10. A referência para uma instância da classe *Profile* em

uma dada instância *Service* é, por conseguinte, determinada pela propriedade *presents*.

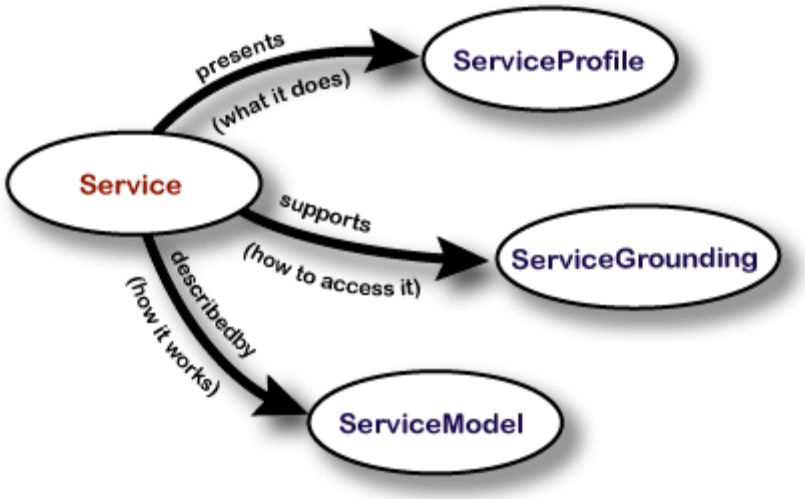

Figura 2.10: Estrutura da ontologia de serviços de OWL-S [15].

Assim, nesta sub-ontologia estão definidas todas as informações necessárias para que um agente computacional possa descobrir o serviço que apresente determinadas características necessárias. Similarmente, tal estrutura torna possível que um determinado algoritmo de casamento de composições (*matchmaker*) determine se dois dados serviços podem ser acoplados. Mais especificamente, em um *ServiceProfile* de um dado serviço podemos encontrar: descrição das atividades realizadas, limitações de aplicabilidade, questões de qualidade, requerimentos para que se torne possível sua invocação, e até mesmo o contato de alguém responsável pelo mesmo.

2.9.1.2. Classe *ProcessModel*

ProcessModel, ou *ServiceModel*, é a estrutura que se encarrega de determinar como um serviço pode ser utilizado por seus clientes. Assim, a propriedade *describedBy* (descrito por) referencia, em uma instância de *Service*, um determinado *ProcessModel* (como pode ser visto na figura 2.10).

Para determinar como um serviço funciona, o *ProcessModel* realiza: o detalhamento semântico do conteúdo das requisições esperadas pelo *Web Service*, as condições em que determinadas resoluções ocorrerão, e, onde necessário, a descrição do processo seqüencial que leva a tais resoluções. Ou seja, tal estrutura

define como requerer o serviço, e o que acontece quando o mesmo é executado. Adiante, tal descrição pode ser utilizada das seguintes maneiras por agentes que tratam *Web Services* complexos:

- Análise profunda para determinação se um determinado serviço não-trivial realmente atende às necessidades requeridas (complementando informações do *ServiceProfile*);
- Realização de composições de descrições de serviços, a partir de múltiplos *Web Services*, para se alcançar um objetivo específico. Isto é viabilizado a partir da utilização de estruturas de controle parecidas com as encontradas em WS-BPEL, tais como: *Sequence* (atividades a serem realizadas em ordem), *Split* (atividades que são executadas paralelamente), *Repeat-Until* (atividades que são realizadas até que uma determinada condição se torne verdade), dentre outras;
- Durante o curso da execução do serviço, coordenação de atividades dos diferentes participantes;
- Monitoramento de execução de um *Web Service*.

2.9.1.3. Classe *Grounding*

A questão de como um agente pode interagir com um *Web Service* é respondida pelo *grounding*, ou *ServiceGrounding*. O *grounding* provê os detalhes de conhecimento necessários a cerca de protocolos de transporte utilizados pelo serviço. Instâncias da classe *Service* devem ter uma propriedade *supports* (suporta) que referencie um *ServiceGrounding*.

Uma instância de *grounding* contém informações a cerca de meios de acesso ao serviço referenciado, especificando: protocolos de comunicação, formatos de mensagens e outros detalhes específicos (como número de portas usadas ao se contatar um determinado *Web Service*). Adicionalmente, o *grounding* deve determinar, para cada tipo semântico de entrada ou saída definido no *ServiceModel*, um meio desprovido de ambigüidades para o intercâmbio de elementos de dados daquele tipo com o *Web Service* a que se refere (ou seja, as técnicas de serialização utilizadas).

Assim, a função central do *grounding* em OWL-S é mostrar como entradas e saídas abstratas de processos atômicos podem ser realizadas concretamente como mensagens, que carreguem tais entradas e saídas em algum formato de transmissão específico. Aqui, encaixa-se WSDL (discutido na subseção 2.3.2), que é utilizado por OWL-S para a realização do *grounding* devido a sua larga utilização, com sucesso, pela indústria na especificação concreta de mensagens para *Web Services*.

2.9.2. Tarefas de Automação em OWL-S

A seguir, detalha-se como a visão de OWL-S viabiliza as três tarefas de automação ligadas a *Web Services*.

2.9.2.1. Descobrimento

A partir da utilização de OWL-S, a informação necessária a realização da atividade de descobrimento automático de serviços poderia ser especificada em um formato de marcação específico (semanticamente tratável computacionalmente) nos provedores de serviços, enquanto registros de serviços (como UDDI, porém melhorados a partir da utilização de semântica) ou um agente de procura de serviços na Internet (viabilizado pela ontologia de serviços de OWL-S) poderiam ser utilizados para efetuar a localização automática de *Web Services.* Unindo as idéias propostas, um servidor poderia publicar sua descrição OWL-S em um registro de serviços, de forma que agentes pudessem encontrá-lo quando procurassem por um serviço na rede que estivesse publicado em registros de serviços conhecidos.

As idéias propostas para a resolução desta atividade são viabilizadas graças à possibilidade de publicação declarativa de propriedades e capacidades de serviços em OWL-S (em um *Profile*). Exemplos relacionados a este tipo de declaração em OWL-S podem ser encontrados no Apêndice II.

2.9.2.2. Execução

A execução de um serviço pode ser entendida como um conjunto de chamadas a procedimentos remotos, já que um *Web Service* nada mais é que uma API de acesso via rede. Assim, OWL-S é uma linguagem declarativa,

computacionalmente interpretável, que inclui a semântica dos parâmetros a serem especificados quando no momento da execução destas chamadas, e a semântica daquilo que é retornado nas mensagens após a falha ou sucesso de execução de tais serviços. Assim, um agente de software poderia ser capaz de interpretar esta linguagem de marcação a fim de entender quais entradas são necessárias para invocar um serviço, bem como a informação a ser retornada. Desta forma, OWL-S, em conjunto com ontologias de domínio especificadas em OWL, provê meios definidos como padrão para a especificação de APIs para *Web Services* que torne esta atividade de execução automática possível.

2.9.2.3. Composição

A partir da utilização de OWL-S, a informação necessária para a seleção e composição de serviços na *Web* será codificada nos próprios serviços. Assim, torna-se possível que um *software* seja programado para a manipulação destas representações, para um fim específico, de forma a se alcançar automaticamente tal resolução. Para isso, OWL-S provê mecanismos de declaração para especificação de pré-requisitos e conseqüências associadas a utilização de *Web Services*, além de uma linguagem para a descrição de composições (definindo, desta forma, serviços compostos a partir de outros atômicos) e interação com fluxo de dados (similarmente às estruturas de controle de WS-BPEL).

3. FERRAMENTA PARA COMPOSIÇÃO DE *WEB SERVICES*

Neste capitulo, será detalhada a ferramenta COMPOWS (Compositor de *Web Services*). Nos tópicos iniciais, referentes à especificação da ferramenta, serão mostrados: a visão geral do projeto, sua arquitetura, e a solução aplicada para a tarefa de composição de serviços. Posteriormente, será apresentado o protótipo da ferramenta, juntamente com as tecnologias utilizadas em seu desenvolvimento. Por fim, será analisada a utilização do protótipo da ferramenta em um estudo de caso.

Ressalta-se, aqui, que as principais idéias expostas neste capítulo têm forte relação com os trabalhos [31] e [19], tendo este último inspirado a concepção da ferramenta COMPOWS.

3.1. Visão Geral

A possibilidade de aplicação de tecnologias associadas à *Web* Semântica na tarefa de automação de composição de *Web Services* torna viável a realização da mesma, como mostrado no capítulo anterior. Porém, para que uma ferramenta execute composições de maneira plenamente automatizada, a mesma deve dispor de um componente capaz de montar um planejamento de acordo com o objetivo estabelecido pelo usuário. Neste trabalho, analisamos a tarefa de composição de serviços apenas no que concerne à aplicação de *Web* Semântica, portanto, abstraindo a presença de um eventual módulo planejador. Em seu lugar, está localizada a interface com o usuário, o qual deverá executar composições de maneira semi-automatizada, de acordo com serviços cadastrados previamente na ferramenta.

Espera-se que o sistema COMPOWS guie o usuário na composição dinâmica de *Web Services*. O processo semi-automatizado constitui, portanto, na apresentação de serviços que gozem de possibilidade de composição com outros serviços previamente incluídos na composição, a partir da utilização de estruturas de controle do *ProcessModel* de OWL-S. Posteriormente, a composição gerada deve ser passível de execução através do *grounding* WSDL dos serviços atômicos, em um processo de orquestração onde a ferramenta atua como regente (como mostrado no capítulo anterior).

Dessa forma, analisemos como seria o fluxo de utilização do sistema COMPOWS, do ponto de vista do usuário:

1) **Abertura:** um usuário abre a ferramenta, a fim de executar uma composição de serviços. Ao entrar no sistema, ele deverá selecionar, a partir de uma lista de opções, um serviço pelo qual deseje iniciar a composição. Ao selecionar um serviço para composição, o mesmo deverá ter suas principais informações disponibilizadas na tela: entradas, saídas, pré-condições e efeitos;
2) **Acoplamento:** neste ponto, o usuário deverá se utilizar de uma estrutura de controle do *ProcessModel* de OWL-S (como *sequence*, *if-then-else* ou s*plit*) e acoplar a um serviço já presente na composição. Posteriormente, o sistema apresenta, para cada estrutura de controle selecionada, opções de serviços viáveis de inserção na composição;
3) **Seleção:** Aqui, o usuário deve selecionar um serviço, dentre os listados no passo anterior, para a realização da composição. Posteriormente, o usuário pode voltar ao passo 2 (para inserir uma nova estrutura de controle à composição) ou seguir para o próximo passo;
4) **Composição:** Finalizando a composição, o usuário deve selecionar quais saídas de serviços atômicos presentes na composição serão também saídas do *Web Service* composto. Ressalta-se que entradas de serviços atômicos que não estejam conectadas a uma estrutura de controle serão definidas automaticamente como entradas do *Web Service* composto.
5) **Execução:** Finalizando a utilização da ferramenta, esta deve dispor de um mecanismo de execução da composição efetuada (de acordo com parâmetros de entrada passados pelo usuário). Adicionalmente, também deve ser possível visualizar a descrição OWL-S do novo serviço composto.

3.2. Arquitetura

Para a concretização do sistema imaginado na seção anterior, a ferramenta COMPOWS está estruturada em uma série de módulos, representados na figura 3.1. Focando principalmente um sistema modulado, o modelo de arquitetura aqui apresentado foi baseado no padrão MVC (*Model-View-Controller*) [32], prezando pela separação clara entre as diferentes camadas de *software*. Portanto, o modelo utilizado visa garantir que mudanças em uma camada da aplicação não tenham reflexos nas demais.

O módulo identificado com o rótulo GUI (*Graphical User Interface*) é o responsável pela apresentação das informações ao usuário. Através dela o usuário deverá montar e visualizar suas composições. Tal módulo, então, deverá ter uma interface com o restante do sistema, repassando as ações do usuário para as camadas imediatamente inferiores.

Um dos módulos que se comunicam diretamente com o GUI é o Compositor. Tal componente deverá ser responsável por tratar as requisições de composição de serviços realizadas pelo usuário, e recebidas através do GUI. É neste componente que as composições ocorrem efetivamente.

Ao lado do módulo Compositor, encontra-se o Descobridor. Esta é a parte do sistema que se responsabiliza por tratar o descobrimento de novos serviços na *Web*. Desta forma, espera-se que o módulo GUI repasse ao mesmo uma determinada composição em OWL-S, sendo a mesma então validada, e posteriormente cadastrada no sistema.

Também se comunicando diretamente com o módulo GUI, está o módulo Executor. Este é o componente responsável por orquestrar a execução de uma composição efetuada pelo usuário (como visto no capítulo anterior), a partir da descrição em OWL-S do novo serviço composto.

Na camada inferior da arquitetura do sistema, temos o módulo KB (*knowledge base*, base de conhecimento). Esta é a parte do sistema responsável por guardar, com persistência, as descrições de serviço em OWL-S cadastrados no sistema através do Descobridor. Desta forma, informações escritas em OWL são passíveis de conversão em triplas RDF, e estocagem em sua base de dados. O KB também possui um motor de inferência, o qual possui axiomas para regras de inferência

OWL. Tais axiomas podem ser aplicados aos fatos presentes no KB para o descobrimento de relações entre classes, tais como herança, de acordo com a necessidade do módulo Compositor.

Na próximas seção, será detalhado como os módulos mais importantes do sistema atuam para a realização das tarefas ligadas à composição de serviços.

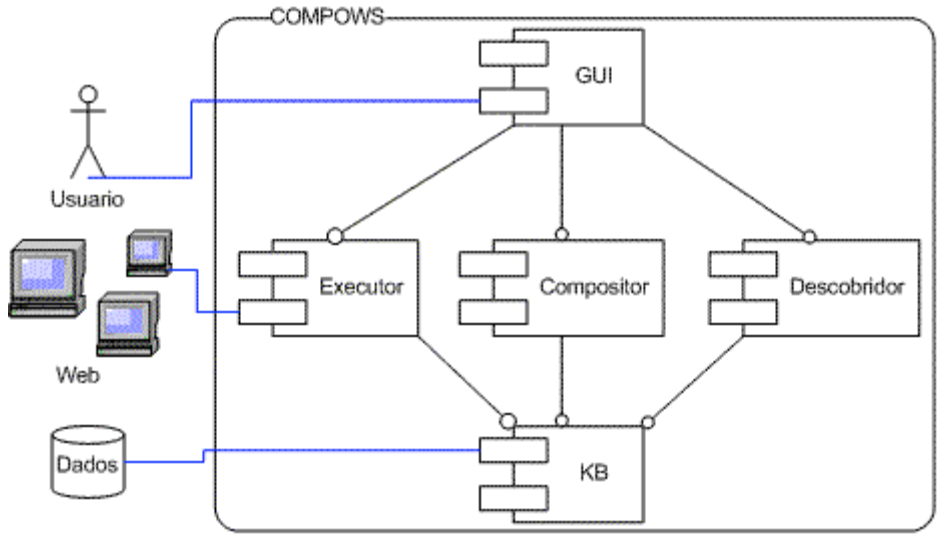

Figura 3.1: Arquitetura modular da ferramenta compositora de serviços COMPOWS.

3.3. Composição de Serviços

Como visto na seção referente à visão geral do sistema, a ferramenta COMPOWS deve guiar o usuário na criação de um fluxo de interação entre *Web Services*, apresentando possibilidades de escolha entre os serviços cadastrados a cada momento. Desta forma, analisemos os dois pontos de consideração mais importantes concernentes ao fluxo de utilização da ferramenta que atingem o módulo Compositor diretamente (como visto na primeira seção do capítulo):

- Sempre que o usuário efetuar uma escolha (isto é, selecionando um serviço para introdução na composição), uma consulta deve ser enviada pelo Compositor ao KB. Esta consulta consiste na recuperação de informações referentes ao serviço selecionado (entradas, saídas, pré-condições e efeitos) presentes na descirção OWL-S do mesmo, mais especificamente em seu *ServiceProfile*;

- O usuário define uma composição de serviços ao aplicar uma estrutura de controle em alguma(s) entrada(s) de um *Web Service* previamente selecionado. Assim, uma consulta deve ser realizada ao KB, para cada uma das entradas de cada serviço de interesse, para a formação de uma lista de serviços que possam suprir o tipo de dado esperado - de acordo com a estrutura de controle utilizada na acoplagem dos serviços.

Para o segundo ponto, um pouco mais complexo que o primeiro, tem-se um algoritmo especial utilizado para a definição de acoplagem entre serviços, adaptado a partir da função *outputMatch* proposta em [31] (como pode ser visto na figura 3.2). Neste procedimento, determina-se a relação entre a entrada de interesse de um determinado serviço presente na composição e cada um dos elementos da lista de saídas de um determinado serviço a ser analisado. Tal relação pode ser definida, então, em quatro níveis de acoplagem:

- *Exact*: trata-se de uma acoplagem completa, onde os dois tipos de dados envolvidos são considerados plenamente equivalentes. Isto acontece quando os dois parâmetros envolvidos são da mesma classe, ou quando a saída considerada é subclasse da entrada de interesse. Por exemplo, se a saída analisada pertencer à classe Ônibus (segundo a ontologia de veículos e seus derivados que consta na figura 3.2), e a entrada esperada à classe Veículo, têm-se uma acoplagem exata - tendo-se em vista que um ônibus é tão somente uma especialização de veículo.
- *Plugin*: aqui, tem-se um exemplo de uma acoplagem quase completa, onde o tipo de dado da saída do serviço a ser analisado pertence a um subconjunto do tipo de dado da entrada do serviço presente na composição. Por exemplo, se a saída analisada for da classe Micro-Ônibus (subclasse de Ônibus, que por sua vez é subclasse de Veículo), e a entrada esperada da classe Veículo.
- *Subsume*: aqui, tem-se um exemplo de uma acoplagem incompleta, onde o tipo de dado da entrada do serviço presente na composição pertence a um subconjunto do tipo de dado da saída do serviço a ser analisado. Por exemplo, se a saída analisada for da classe Veículo, e

a entrada esperada da classe Micro-Ônibus (novamente, onde esta última é subclasse de Ônibus, que por sua vez é de Veículo). Esta já é uma relação fraca, sendo a composição formada incompleta, possivelmente não atingindo os objetivos do usuário plenamente.

- *Fail:* Ocorre uma falha completa quando nenhuma das outras três relações pode ser identificada, ou seja, quando não existem relações pertinentes entre os tipos de dados envolvidos.

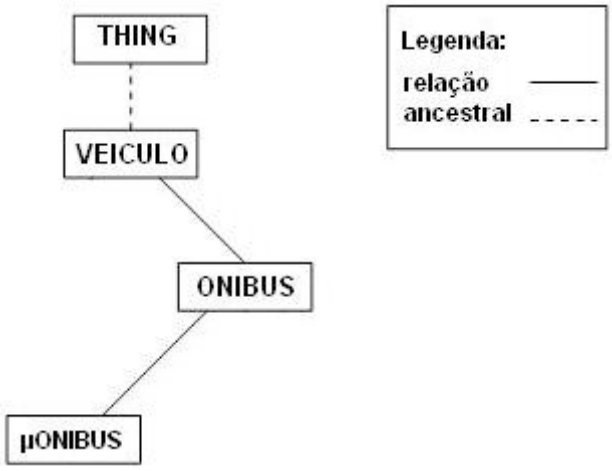

Figura 3.2: Ontologia para veículos e seus derivados.

Relações de qualquer tipo diferente de *exact* e *plugin* não são consideradas válidas para a ferramenta COMPOWS, por não produzirem composições plenamente confiáveis. Por conseguinte, ao efetuar uma consulta ao KB, o módulo compositor recebe uma lista de serviços que tenham saídas com acoplamento possível com uma entrada de interesse. Tais consultas são efetuadas em uma linguagem específica para as triplas RDF cadastradas no sistema (que se referem ao serviços OWL-S), no caso, SPARQL (como mostrado no capítulo anterior), tratadas pelo *reasoner* presente no KB.

Assim, baseando-se na escolha do usuário sobre quais serviços devem participar da composição (e os pontos onde estas ocorrem, isto é, as entradas e saídas dos serviços), o sistema gera um novo *CompositeService* passível de publicação, descobrimento e participação em outras composições. Para isto, é produzido um *ServiceProfile* (a partir das entradas e saídas definidas pelo usuário) e um *CompositeProcess* (correspondente ao *ProcessModel*) próprios para cada novo

serviço composto. Posteriormente, a composição pode ser executada a partir da simples utilização de *groundings* dos serviços atômicos participantes, em um processo de orquestração guiado pelo módulo Executor.

```
outputMatch(outputsRequest, outputsAdvertisement) {
   globalDegreeMatch= Exact
   forall outR in outputsRequest do {
      find outA in outputsAdvertisement such that
         degreeMatch= maxDegreeMatch(outR,outA)
         if (degreeMatch=fail) return fail
         if (degreeMatch<globalDegreeMatch)
            globalDegreeMatch= degreeMatch
   return sort(recordMatch);}
```

```
degreeOfMatch(outR,outA):
   if outA=outR then return exact
   if outR subclassOf outA then return exact
   if outA subsumes outR then return plugIn
   if outR subsumes outA then return subsumes
   otherwise fail
```

Figura 3.3: Algoritmo de determinação de nível de acoplamento entre *Web Services* [31].

3.4. Protótipo COMPOWS

Nas próximas subseções serão tratados tópicos referentes ao desenvolvimento de um protótipo para a ferramenta COMPOWS, especificada nas seções anteriores. A codificação desta aplicação prezou pela disponibilização das funcionalidades consideradas mais importantes, de forma a viabilizar uma análise de estudo de caso, a qual está presente na próxima seção deste trabalho. No Apêndice I encontra-se o código de tal protótipo desenvolvido.

3.4.1. Tecnologias Utilizadas

O protótipo da ferramenta COMPOWS, aqui apresentado, foi codificado na linguagem de programação Java [33], em sua versão 1.6. A sua escolha foi motivada por sua utilização em outros trabalhos relacionados a este (a saber, [19]). O ambiente de codificação utilizado foi o Eclipse 3.2 [34]. Apesar da IDE (*Integrated Development Environment*) em questão estar em uma versão mais recente, a anterior foi escolhida devido à possibilidade de integração com o *plugin* Visual Editor [35], usado para construção de toda a interface gráfica do sistema.

Toda a base de dados foi simulada em arquivos de texto simples, tornando mais genérico o acesso aos dados necessários para a consulta de modelagens OWL-S eventualmente cadastradas no sistema.

Visando a máxima produtividade de desenvolvimento, algumas APIs, detalhadas nas subseções seguintes, foram utilizados para poupar trabalho e garantir robustez ao resultado final obtido.

3.4.1.1. Jena

Jena [36] é um *framework* Java para a construção de aplicações destinadas à *Web* Semântica. A mesma foi desenvolvida com o intuito de prover um ambiente de codificação para RDF, RDFS, OWL e SPARQL, incluindo um mecanismo de inferência axiomático. Jena é *open source*, e fruto do trabalho do HP *Labs Semantic Web Programme*. O protótipo aqui apresentado utilizou-se das seguintes funcionalidades presentes em Jena: API RDF, leitura e escrita de RDF em RDF/XML e triplas, API OWL, estocagem em memória e motor de consulta SPARQL.

3.4.1.2. Apache Axis

A API Axis [37] é uma implementação do protocolo SOAP (debatido no capítulo anteiror). Esta biblioteca tem como objetivo oferecer ao desenvolvedor uma abstração dos detalhes necessários ao se lidar com SOAP e WSDL ao se planejar interações com *Web Servies*. Neste protótipo, portanto, Axis é utilizado para a codificação de requisições enviadas à *Web Services*, e posterior decodificação das respostas obtidas, ao se orquestrar a execução de composições de serviços.

3.4.1.3. OWL-S API

A OWL-S API [38] provê ao usuário uma biblioteca Java para acesso programático de leitura, execução e escrita de modelagens OWL-S para serviços. Este framework constitui, desta forma, uma importante parte do protótipo desenvolvido, inclusive dispondo de interfaces com as outras duas APIs discutidas nas subseções anteriores. A OWL-S API foi desenvolvida pela equipe *Mindswap* conjuntamente com o *Fujitsu Labs of America* e *College Park*.

3.4.1.4. Pellet

Pellet [39] é um *reasoner* OWL DL *open-source* baseado em Java que pode ser utilizado juntamente à API Jena (como feito neste protótipo). Entre suas

funcionalidades, as mais importantes utilizadas neste trabalho foram: checagem de validação de ontologias e execução de consultas RDQL (linguagem de consultas sobre RDF a partir da qual surgiu SPARQL).

3.4.2. Interface Gráfica

A interface gráfica do protótipo da aplicação COMPOWS está estruturada da seguinte forma (de acordo com a numeração estabelecida na figura 3.3, abaixo):

1) **Caixa de Ferramentas:** aqui estão localizados os controles OWL-S utilizados para acoplar os serviços e efetuar as composições, tais como: *sequence*, *split-join* e *repeat-until*;
2) **Console:** meio de comunicação responsável por levar mensagens internas do sistema ao conhecimento do usuário;
3) **Corpo da Aplicação:** nesta parte do sistema, caso a aba "Modelo" esteja selecionada, são mostradas composições ao usuário, de acordo com as escolhas efetuadas pelo mesmo durante a utilização do programa. No caso em que a aba "Texto" estiver selecionada, trata-se da faixa da GUI onde descrições OWL-S referentes a serviços, a serem cadastrados no sistema ou compostos pelo usuário, são inseridas;
4) **Abas:** Refere-se às abas "Texto" e "Modelo" que quando selecionadas alternam a visão do Corpo do sistema;
5) **Menu:** Este ponto se refere ao menu da ferramenta, com acesso aos seguintes sub-menus (ilustrados na figura 3.4, abaixo):
    - Compositor: Aqui, tem-se acesso às funcionalidades de composição de serviços do sistema. Ao clicar no sub-menu "Novo" , o usuário tema possibilidade de iniciar uma nova composição. Caso opte por selecionar "Abrir", tem-se a possibilidade de se escolher composições já realizadas anteriormente para visualização (e realização de alterações). O botão "Salvar" simplesmente guarda em arquivo o estado atual da composição utilizada, enquanto que o botão "Gerar Saída para Composição" torna todas as saídas de serviços

atômicos presentes na composição (e não atrelados a quaisquer controles) também saídas do serviço composto. Por último, ao se clicar em “Sair“ a aplicação é finalizada;

- Executor: Trata-se da parte do Menu responsável por realizar as execuções das composições efetuadas (ao se clicar em seu único sub-menu, denominado “Executar“). Ressalta-se que, ao se clicar nessa opção, abre-se uma nova janela (ilustrada na figura 3.5, abaixo) para a definição dos valores para as entradas da composição a ser executada. Essa janela dispõe de uma lista das entradas (1, para a seleção da entrada a ser valorada), uma caixa de texto (3, para a escrita dos valores) e um botão “OK” (2) para determinar o início da execução;
- Descobridor: Aqui, tem-se acesso às funcionalidades ligadas ao módulo Descobridor da ferramenta COMPOWS. Tem-se duas opções: “Validar Modelo” (valida uma descrição OWL-S escrita pelo usuário no Corpo do sistema, com a aba “Texto“ selecionada) e “Inserir Serviço“ (adiciona um serviço validado à base de conhecimento do sistema);
- Ajuda: Menu que trata de informações do sistema, ao se clicar na opção “Sobre COMPOWS“.

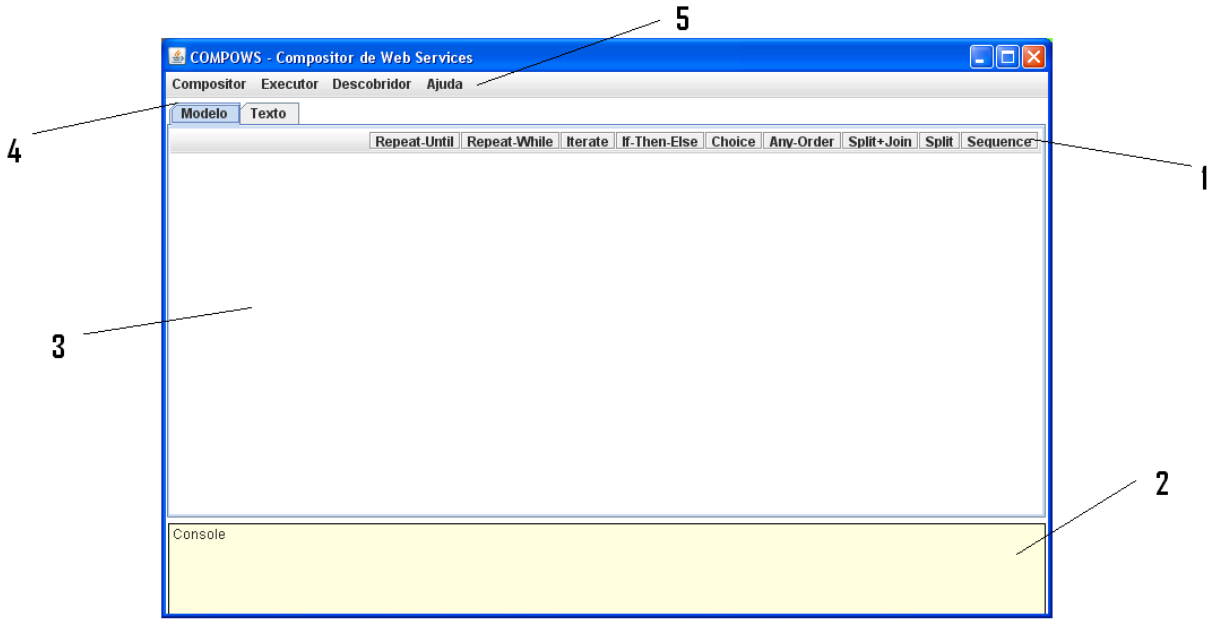

Figura 3.4: Visão geral da aplicação COMPOWS.

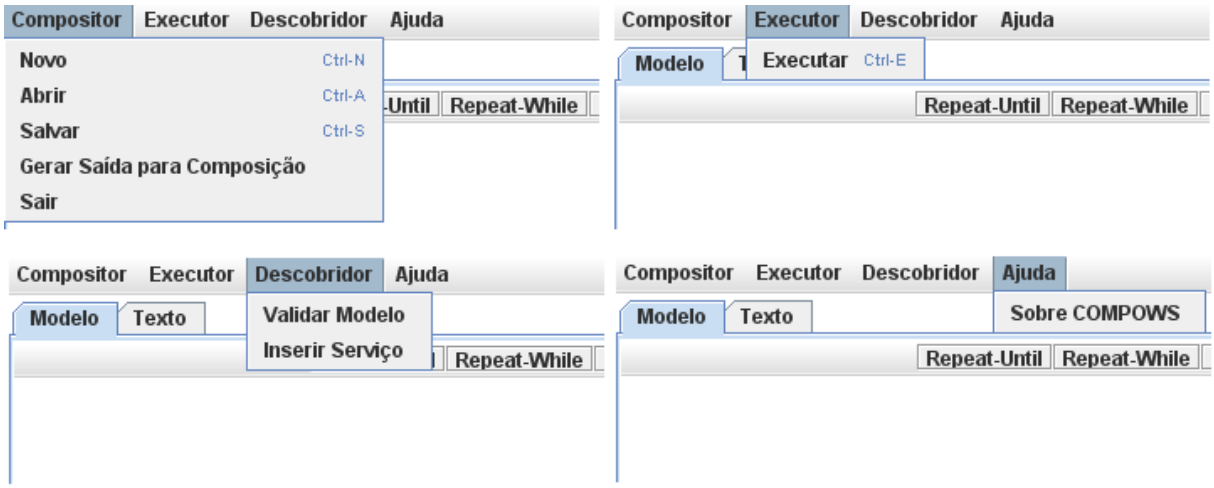

Figura 3.5: Menu da aplicação COMPOWS.

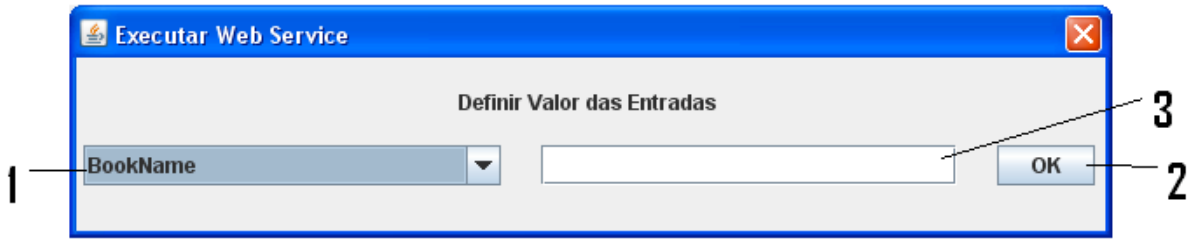

Figura 3.6: Janela de definição de valores para as entradas da composição a ser executada.

3.5. Estudo de Caso

Nesta seção será realizado um estudo de caso, efetuando-se uma composição simples de *Web Services* com o protótipo apresentado na seção anterior.

3.5.1. Utilizando o COMPOWS

Ao se executar o protótipo COMPOWS, tem-se uma visão da aplicação exatamente como mostrado na figura 3.3. Para começar uma nova composição, deve-se clicar no Menu Compositor, e posteriormente no sub-menu “Novo“ (figura 3.6, abaixo).

Neste momento, deve-se escolher (em uma *combobox* disponibilizada pelo sistema, como na figura 3.7, abaixo) um dos serviços cadastrados no sistema. Deseja-se, portanto, realizar um composição de serviços a partir do *Web Service BookFinderService* (cuja descrição OWL-S completa se encontra no Apêndice II), que retorna uma instância da classe livro de acordo com o nome de uma obra que se deseje procurar. A partir da escolha inicial deste serviço atômico, espera-se construir um novo serviço complexo que determine o menor preço para um livro (passando como parâmetro o nome do mesmo) entre duas lojas específicas que disponibilizem seus serviços na *Web*, e que se utilizem da mesma ontologia de domínio de *BookFinderService*.

Posteriormente, as principais informações do serviço selecionado são disponibilizadas no Console (entradas, saídas, pré-condições e efeitos), como na figura 3.8 abaixo.

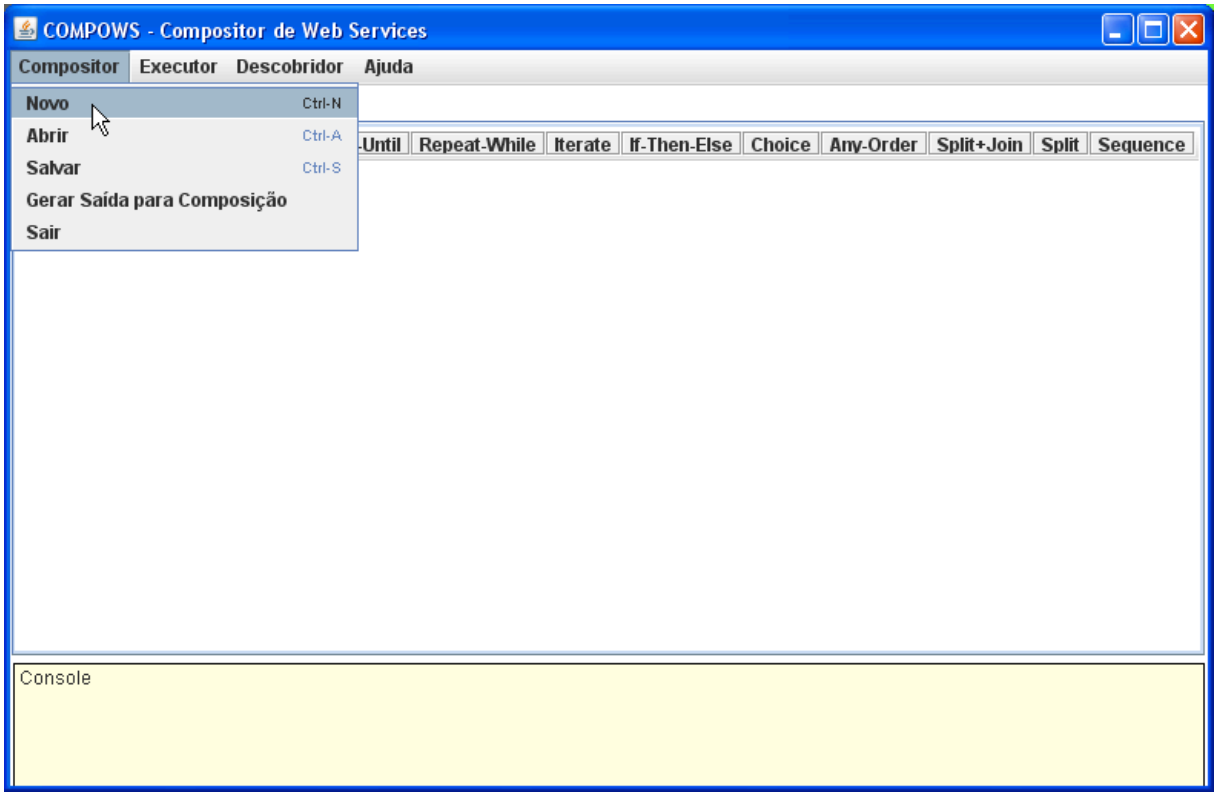

Figura 3.7: Início de uma nova composição.

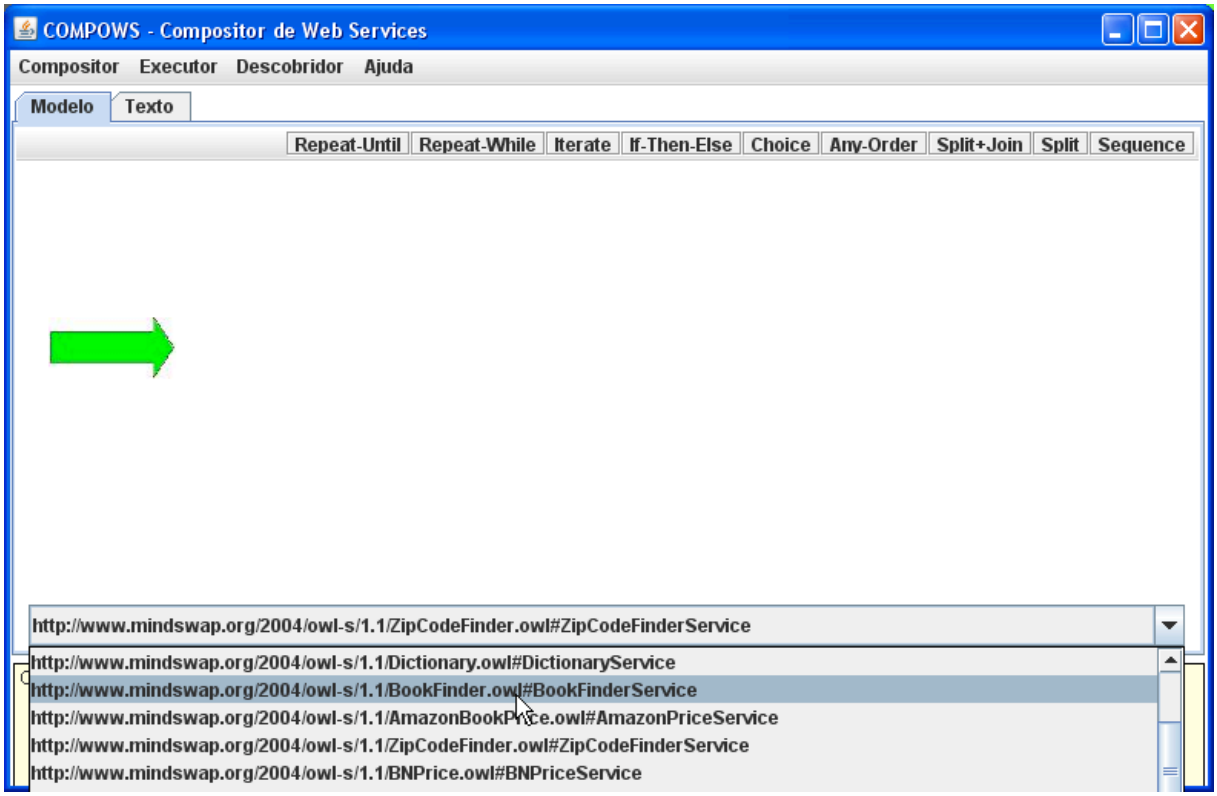

Figura 3.8: Seleção do primeiro serviço da nova composição.

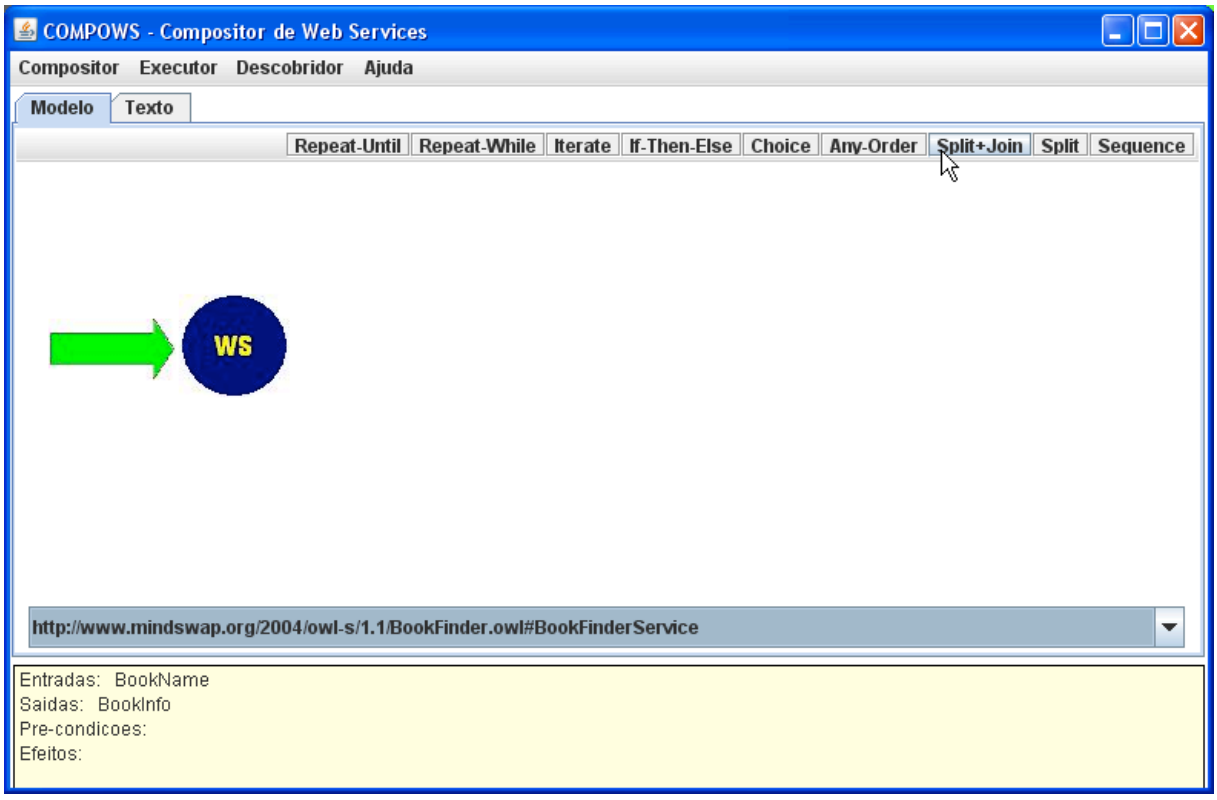

Figura 3.9: Serviço *BookFinderService* é selecionado e tem suas informações disponibilizadas no Console. Posteriormente, o usuário seleciona o controle *SpIt-Join*.

Seguindo com a composição, deve-se escolher portanto um controle para aplicar à saída *BookInfo* do *BookFinderService*. Para realizar o serviço imaginado, necessita-se que dois serviços de procura de preços de livros sejam avaliados paralelamente a partir de um livro como entrada (*BookInfo*). Desta forma, encaixa-se um controle *split-join* (que realiza a execução de serviços em paralelo simultaneamente), como mostrado na figura 3.9 abaixo. Ao se mostrar o controle na tela, a *combobox* do Corpo do sistema exibe possibilidades de acoplagem através deste controle com o primeiro serviço selecionado (definidos através do algoritmo de acoplagem discutido na seção 3.3). No caso de estudo, o sistema exibiu duas possibilidades de serviços a serem executados paralelamente de acoplagem *Exact* com a saída *BookInfo*: *BNPriceService* e *AmazonPriceService* (cujas descrições OWL-S encontram-se no Apêndice II). Estes serviços são, então, selecionados e devem executar uma procura de preço para o livro retornado por *BookFinderService*.

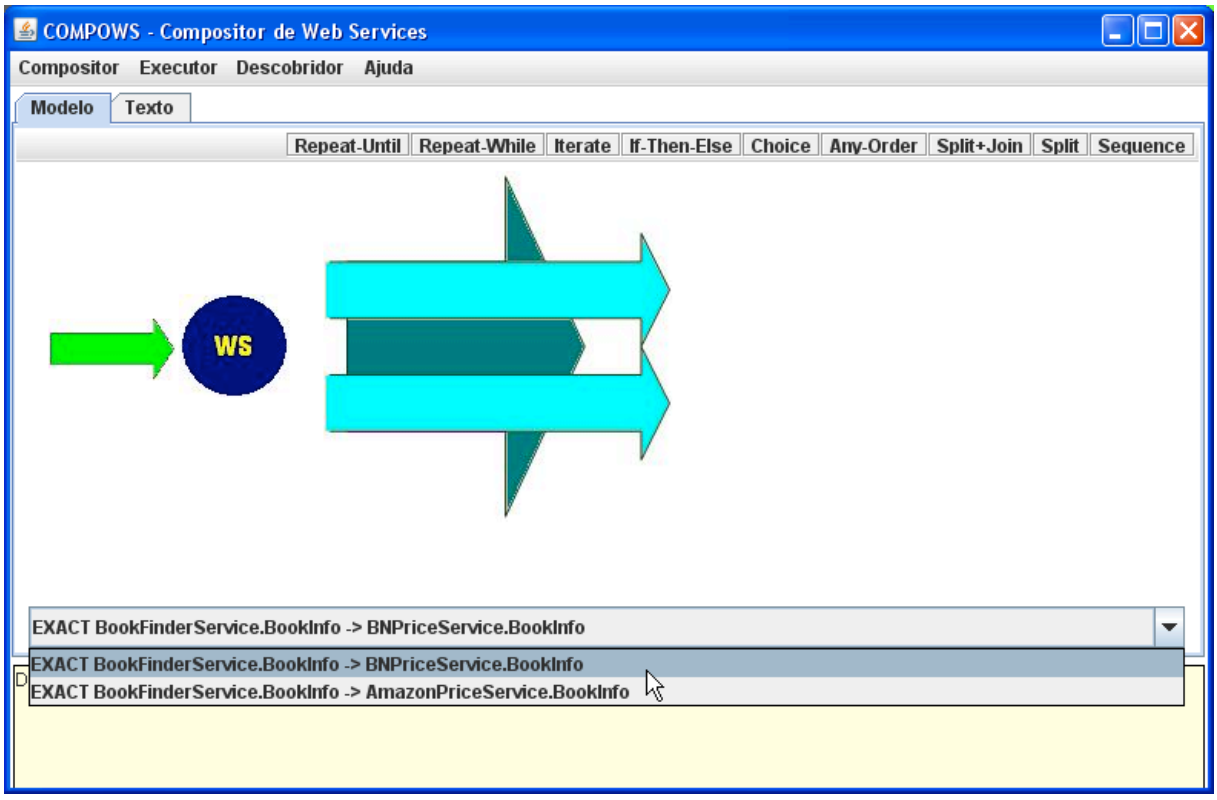

Figura 3.10: Controle *Split-Join* é mostrado na tela, e as possibilidades de acoplagem entre serviços a partir deste controle são disponibilizadas na *combobox*.

Todavia, para se determinar o menor preço entre os dois serviços avaliados, deve-se utilizar de um outro controle de OWL-S. Aqui, encaixa-se a necessidade de um controle *if-then-else* a fim de avaliar qual das duas saídas retornadas pelo *split-join* deve ser considerada (como mostrado na figura 3.10, abaixo). Desta forma, após a seleção deste controle, fica a cargo do usuário determinar qual deve ser a expressão verificadora aplicada. Para este caso de análise, deseja-se utilizar o operador “Menor Que“, de forma a retornar a saída de *BNPriceService* apenas se esta for de valor inferior a de *AmazonPriceService*, retornando justamente a saída desta última em caso contrário (como realizado na figura 3.11, abaixo).

Posteriormente, deve-se gerar a saída para a composição, como mostrado na figura 3.12 abaixo. Esta funcionalidade é responsável por marcar as saídas do controle *if-then-else* como saídas do *Web Service* composto, neste protótipo.

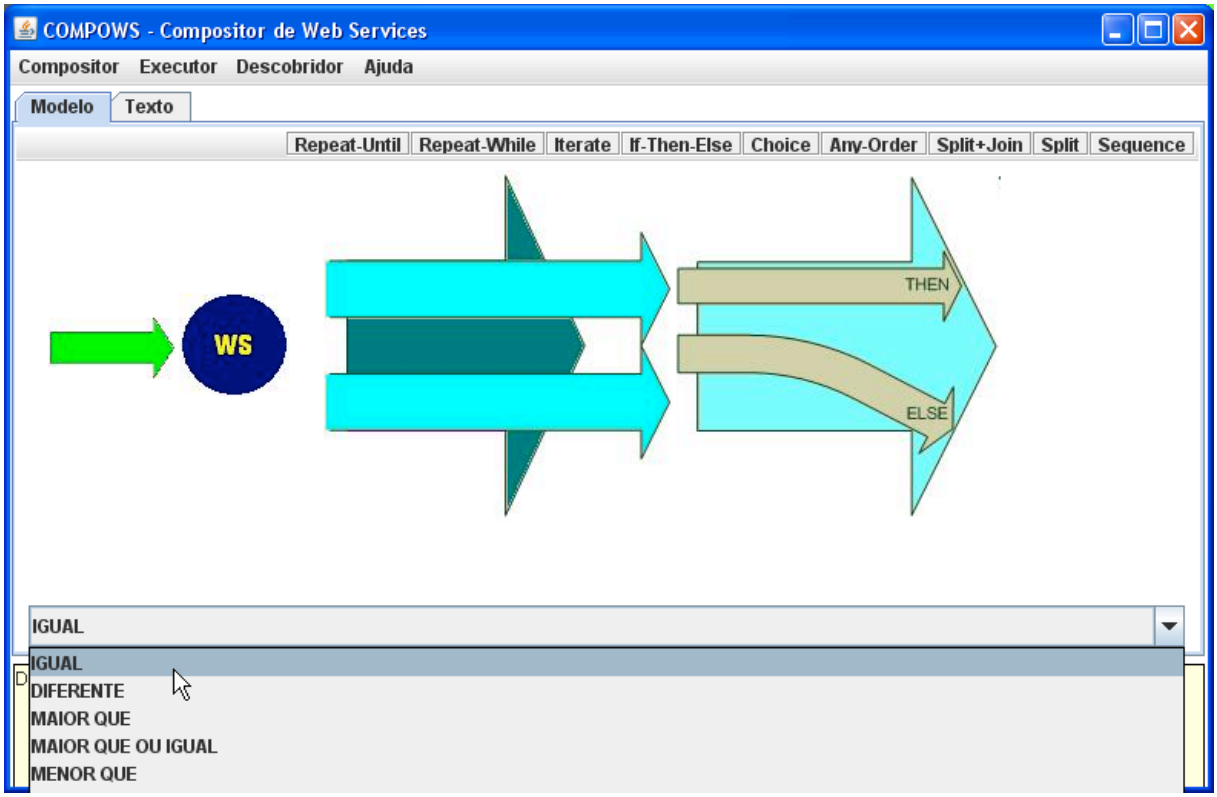

Figura 3.11: Controle *If-Then-Else* é mostrado na tela, e as possibilidades de montagem de expressão verificadora são disponibilizadas na *combobox*.

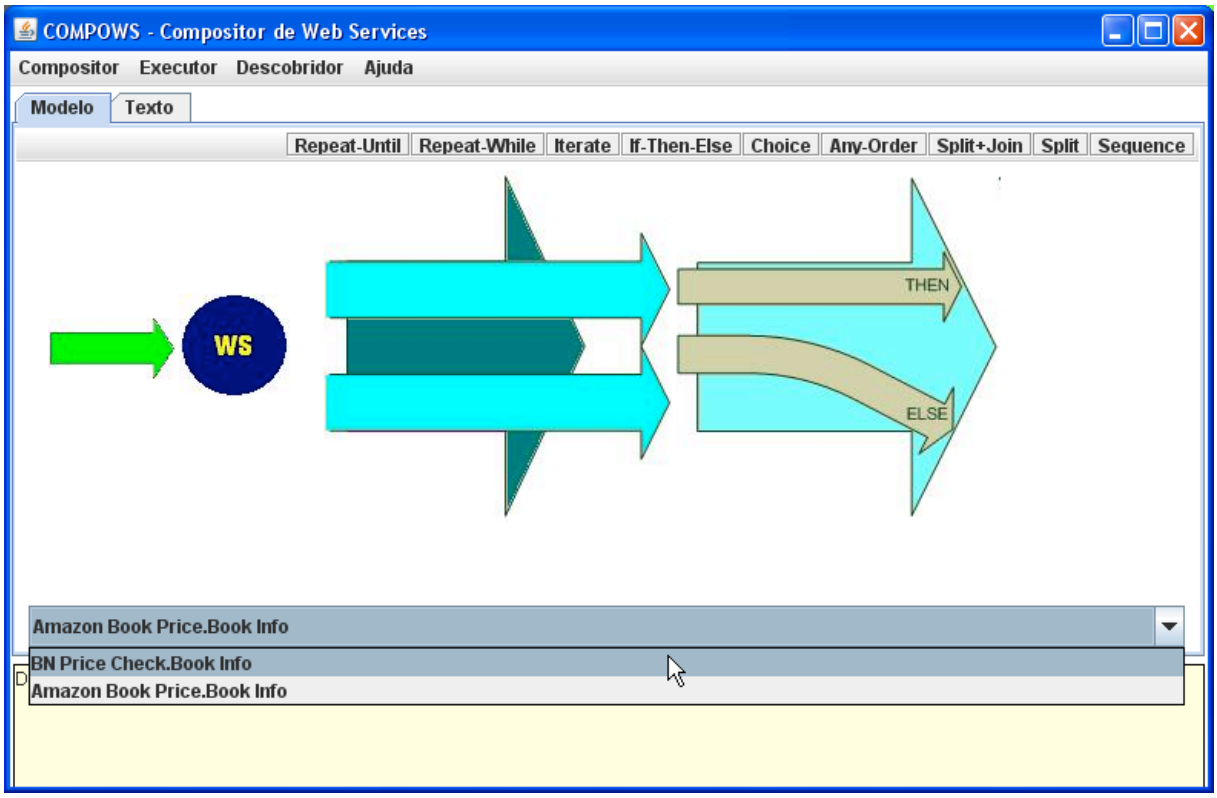

Figura 3.12: Definição das saídas de serviços participantes no controle *If-Then-Else*.

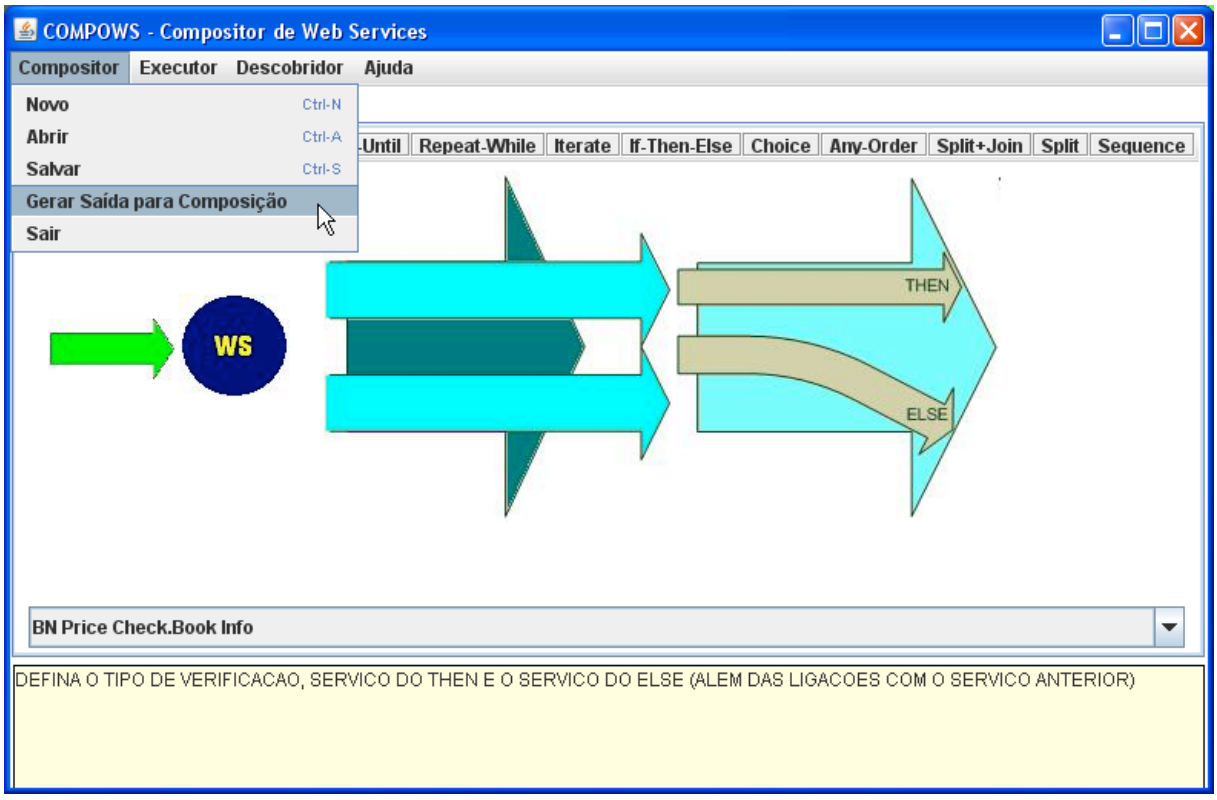

Figura 3.13: Gerar saída do serviço composto.

Após a composição ser gerada, a ferramenta disponibiliza, ao ser selecionada a aba “Texto“, a visualização concreta da descrição OWL-S do novo serviço composto anteriormente (como mostrado na figura 3.13, abaixo).

A partir da finalização da fase de composição de um novo serviço, o sistema disponibiliza ao usuário a possibilidade de execução do mesmo. Como mostrado na figura 3.14 abaixo, o usuário deve clicar no botão “Executar“ e posteriormente preencher os valores para cada entrada do serviço composto. Este último procedimento foi realizado na figura 3.15 abaixo, onde o usuário escreve o nome do livro que se deseja saber o menor preço dentre as lojas pesquisadas. Por fim, o sistema executa o serviço e retorna as respostas ao usuário através do Console (no caso, as saídas seriam o preço do livro e a loja na qual este preço foi encontrado), como mostrado na figura 3.16 abaixo.

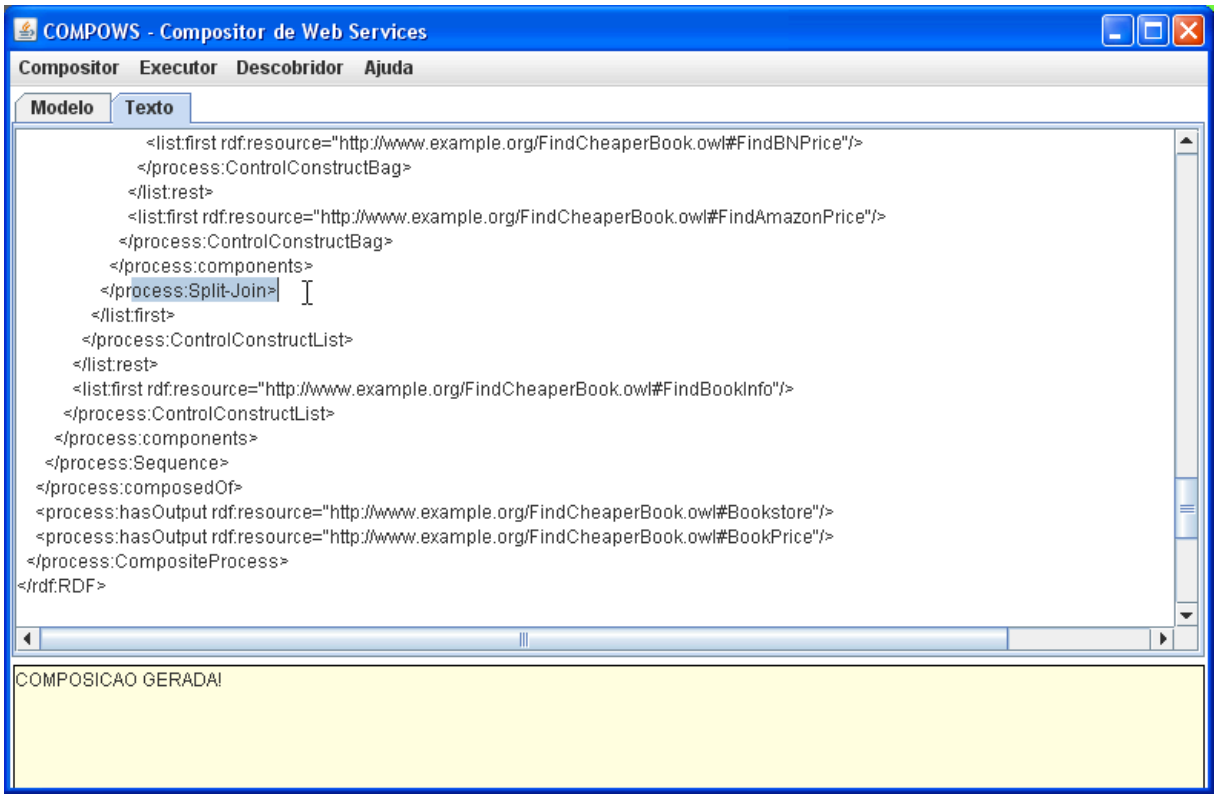

Figura 3.14: A descrição da composição é mostrada ao ser selecionada a aba "Texto".

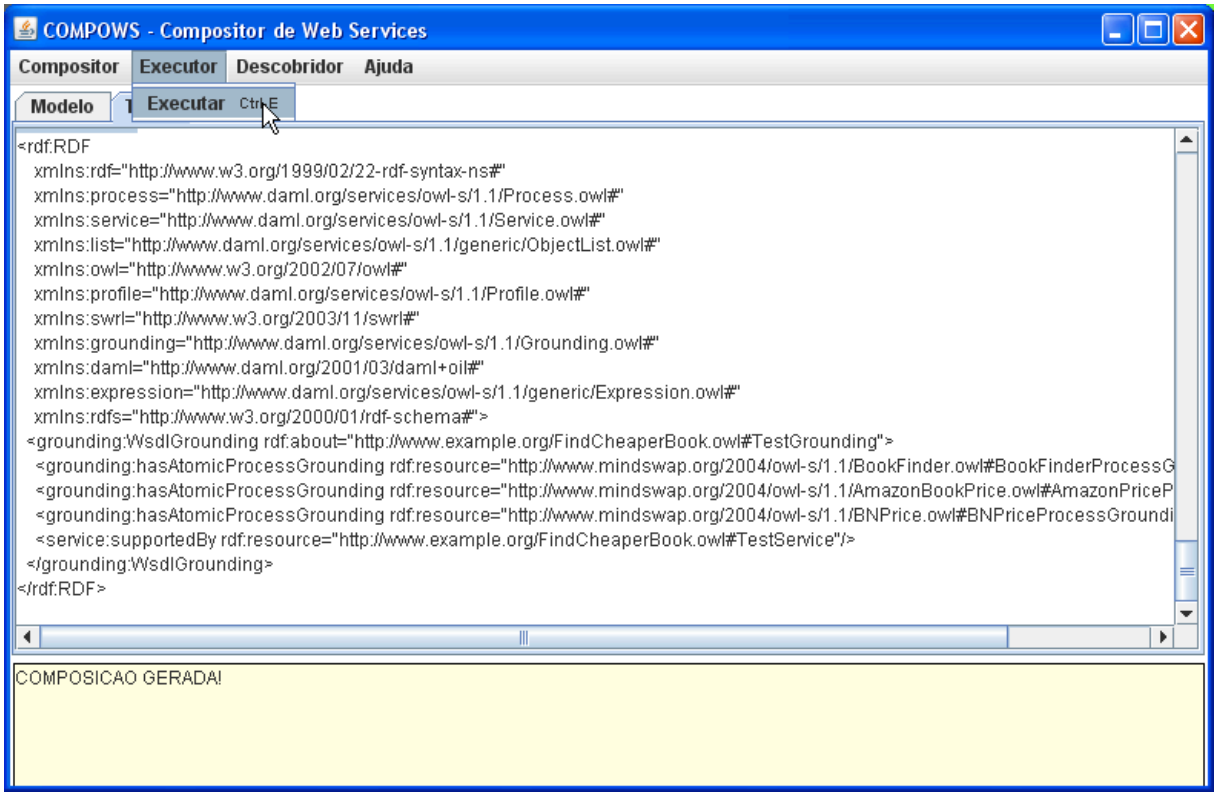

Figura 3.15: Usuário pressiona o botão "Executar" a fim de que a composição gerada seja executada.

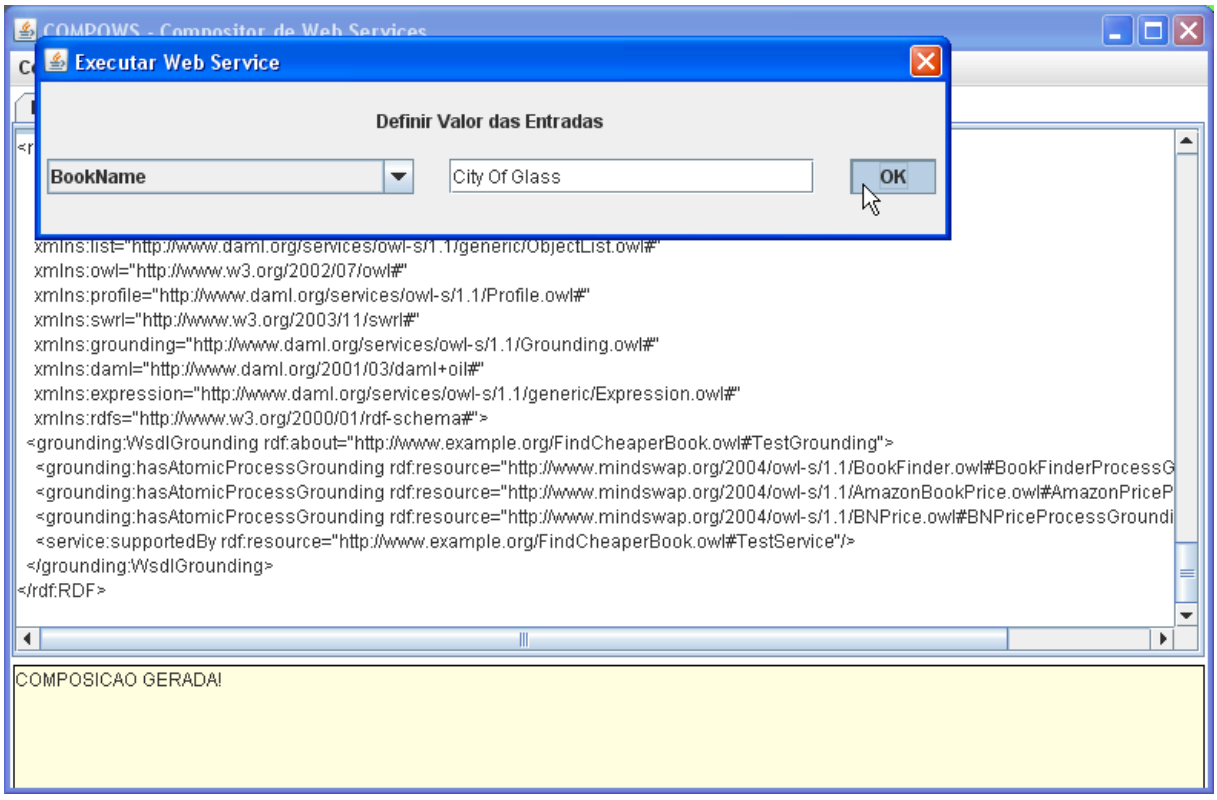

Figura 3.16: Usuário preenche o valor de entrada da composição a ser executada.

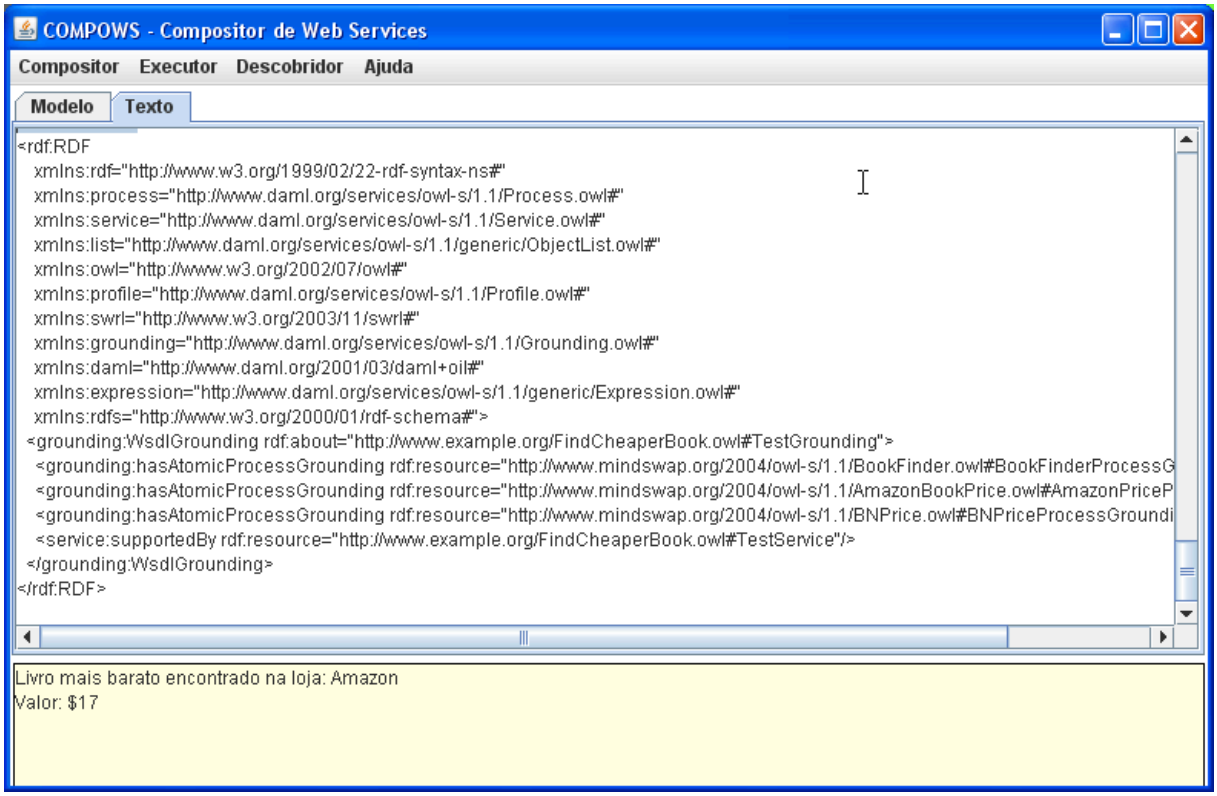

Figura 3.17: O sistema exibe o retorno da execução da composição gerada.

3.5.2. Considerações

O protótipo COMPOWS possibilita uma maior facilidade na criação e manipulação de composição de serviços *Web*. No estudo de caso apresentado na subseção anterior, uma composição razoavelmente complexa foi construída rapidamente. Este mesmo desenvolvimento feito de forma clássica (através da manipulação de *tags* OWL-S) levaria um tempo consideravelmente superior para sua concretização. A partir da utilização da ferramenta, um usuário desprovido de conhecimentos técnicos específicos pode gerar novos serviços a partir de outros pré-existentes, o que comprova que a aplicação auxilia de fato em um desenvolvimento semi-automático de *Web Services* complexos.

Todavia, a ferramenta ainda se encontra em fase inicial e impede que regras de negócio baseadas em pré-condições possam ser aplicadas às composições. Apesar de nem todos os controle OWL-S terem sido de fato implementados, as principais funcionalidades ligadas ao descobrimento, criação e execução de *Web Services* compostos foram completadas. Desta forma, a criação deste protótipo funcional veio a confirmar as teorias expostas no capítulo anterior sobre a viabilidade de uma ferramenta compositora de serviços semânticos – ainda que não plenamente automatizada.

4. CONCLUSÕES

Ao longo deste trabalho foram apresentados conceitos sobre *Web Services* e os avanços causados pela introdução de semântica aos mesmos. Inicialmente, foram abordados temas e tecnologias relacionados à resolução dos principais problemas de automação associados ao uso destes serviços: descobrimento, execução e composição. Dentre as tecnologia citadas, deu-se destaque à OWL-S, através da apresentação dos principais conceitos concernentes à concepção da tecnologia.

Posteriormente, foi apresentada a ferramenta COMPOWS para composição semi-automática de *Web Services*. Foram tratadas as principais características, arquitetura e funcionalidades durante a sua especificação, bem como algoritmos aplicados. Posteriormente, utilizou-se um protótipo da ferramenta (cuja estrutura fora tratada anteriormente) para a realização de um estudo de caso, com o intuito de comprovar a eficácia da solução proposta.

4.1. Principais Contribuições

Podem ser consideradas como as principais contribuições deste trabalho apresentado, a estruturação dos conhecimentos adjacentes à criação de uma ferramenta compositora de *Web Services* e a especificação de sua estrutura arquitetural.

4.2. Trabalhos Futuros

A partir da análise desta monografia, espera-se que em trabalhos futuros a ferramenta COMPOWS:

- Possibilite manipulação de *tags* OWL-S diretamente no *ProcessModel* que se reflitam na interface gráfica, possibilitando a existência de mais liberdade ao usuário;
- Suporte a utilização de todos controles OWL-S existentes;
- Possibilite a escolha de quais saídas o usuário deseja que a composição efetivamente apresente, descartando as demais;

- Possibilite a criação de um *Profile* próprio para o novo serviço composto (com introdução de nome de entradas, saídas e descrição);
- Suporte a utilização de pré-condições e efeitos próprio da composição gerada;
- Disponha de um módulo Descobridor que possa efetivamente descobrir novos serviços na *Web* e introduzi-los na base de dados;
- Exiba de maneira mais clara para o usuário o estado atual da composição a ser gerada (graficamente);
- Disponha de um módulo Planejador que execute uma composição de serviços através da simples definição dos objetivos do usuário, estabelecendo um fluxo de trabalho que se utilize apenas dos serviços cadastrados no sistema. Desta forma, pode-se dizer que a ferramenta alcançaria o status de plenamente automatizada.

REFERÊNCIAS

[1]. MCILRAITH, S.; SON, T. C. Adapting Golog for Programming the Semantic Web. Working Notes of the Fifth International Symposium on Logical Formalizations of Commonsense Reasoning. Nova Iorque, p. 195-202, mai. 2001.

[2]. MCILRAITH, S. et al. Semantic Web Services. IEEE Intelligent Systems (Special Issue on the Semantic Web). Nova Iorque v. 16, n.2, p. 46-53, mar. 2001.

[3]. MOSKALYUK, A. Statistics on eBay Web Services Usage. Disponível em: < http://blogs.zdnet.com/ITFacts/?p=10326>. Acesso em: 8 out. 2008.

[4]. BOOTH, D. et al. Web Services Architecture. W3C Working Group Note, fev. 2004.

[5]. MAIGRE, R. Web services composition with WS-BPEL and OWL-S. Arvutiteaduse teooriaseminar (sügis 2005-kevad 2006). fev. 2006.

[6]. BRAY, T. et al. Extensible Markup Language (XML) 1.0 Third Edition. W3C Recommendation, fev. 2004. Disponível em: <http://web5.w3.org/TR/2004/RE C-xml-20040204/>. Acesso em: 14 out. 2008.

[7]. Chemtech - A Siemens Company. Disponível em: <http://www.chemtech.com.br/lportal/web/guest/home>. Acesso em: 20 out. 2008.

[8]. Amazon Web Services. 2006. Disponível em: <http://aws.amazon.com/>. Acesso em: 20 out. 2008.

[9]. FERGUSON, D. et al. Secure, Reliable, Transacted Web Services. 2003. Disponível em: <http://www.ibm.com/developerworks/webservices/library/ws-ecurtrans/>. Acesso em: 10 out. 2008.

[10]. MITRA, N.; LAFON, Y. SOAP Version 1.2. W3C Recommendation, jun. 2003. Disponível em: <http://www.w3.org/TR/soap12-part0/>. Acesso em: 10 out. 2008.

[11]. BERNERS-LEE, T. et al. Hypertext Transfer Protocol – HTTP/1.1. Request For Comments 2616, jun. 1999. Disponível em: <http://www.ietf.org/rfc/rfc2616.txt>. Acesso em: 14 out. 2008.

[12]. CHRISTENSEN, E. et al. Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001. Disponível em: <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>. Acesso em: 10 out. 2008.

[13]. CUNHA, D. Web Services, SOAP e Aplicações Web. 2002. Disponível em: <http://www.oficinadanet.com.br/index.php?acao=colunas_show&id=448>. Acesso em: 10 out. 2008.

[14]. PAUTASSO, C.; ZIMMERMAN, O.; LEYMANN, F. Restful web services vs. "big"' web services: making the right architectural decision. 17th International World Wide Web Conference Archieves. Pequim, p. 805-814, 2008.

[15]. MARTIN, D. et al. OWL-S: Semantic Markup for Web Services. W3C Member Submission, nov. 2004. Disponível em: <http://www.w3.org/Submission/2004/ SUBM-OWL-S-20041122/ >. Acesso em: 14 set. 2008.

[16]. The UDDI technical white paper. Disponível em: <http://uddi.xml.org/>. Acesso em: 16 out. 2008.

[17]. CHAPPELL, D. Who Cares About UDDI? Disponível em: <http://www.chappellassoc.com/articles/article_who_cares_UDDI.html>. Acesso em: 16 out. 2008.

[18]. CURBERA, F. et al. WS-BPEL - Business Process Execution Language for Web Services, Version 2.0. 2007. Disponível em: <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>. Acesso em: 16 out. 2008.

[19]. SIRIN, E.; PARSIA, B; HENDLER, J. Composition-driven Filtering and Selection of Semantic Web Services. AAAI Spring Symposium on Semantic Web Services. 2004.

[20]. BENNATALLAH, B. et al. Towards Patterns of Web Services Composition. Em GORLATCH, S. et al. Patterns and Skeletons for Parallel and Distributed Computing. Springer Verlag, Reuno Unido, nov. 2002.

[21]. MASUOKA, R. et al. Task computing - the semantic web meets pervasive computing. Proceedings of 2nd International Semantic Web Conference (ISWC2003). 2003.

[22]. BERNERS-LEE, T. et al. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Disponível em: <http://www.sciam.com/article.cfm?id=the-semantic-web>. Acesso em: 15 ago. 2008.

[23]. ANTONIOU, G.; HARMELEN, F. A Semantic Web Primer. 2.ed. Cambridge: MIT Press, 2008.

[24]. MANOLA, F.; MILLER, E. RDF Primer W3C Recommendation 10 February 2004. Disponível em: <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>. Acesso em: 27 out. 2008.

[25]. LE HORS, A.; JACOBS, I. HTML 4.01 Specification. W3C Recommendation 24 December 1999. <http://www.w3.org/TR/html401/>. Acesso em: 27 out. 2008.

[26]. PEMBERTON, S. et al. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). A Reformulation of HTML 4 in XML 1.0. W3C Recommendation 26 January 2000, revised 1 August 2002. Disponível em: <http://www.w3.org/TR/xhtml1/>. Acesso em: 28 out. 2008.

[27]. MCBRIDE, B. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10 February 2004. Disponível em: <http://www.w3.org/TR/rdf-schema/>. Acesso em: 03 nov. 2008.

[28]. PRUD'HOMMEAUX, E.; SEABORNE, A. SPARQL Query Language for RDF. W3C Recommendation 15 January 2008. Disponível em: <http://www.w3.org/TR/rdf-sparql-query/>. Acesso em: 03 nov. 2008.

[29]. CARROLL, J. W3C RDF Validator. 2007. Disponível em: <http://www.w3.org/RDF/Validator/ARPServlet>. Acesso em: 28 out. 2008.

[30]. DEAN, M. et al. Web Ontology Language (OWL) Reference Version 1.0. W3C Working Draft 12 November 2002. Disponível em: <http://www.w3.org/TR/2002/WD-owl-ref-20021112/>. Acesso em: 29 out. 2008.

[31]. PAOLUCCI, M. et al. Semantic Matching of Web Services Capabilities. First International Semantic Web Conference. Sardenha, jun. 2002.

[32]. REENSKAUG, T. MODELS - VIEWS - CONTROLLERS. Technical note, Xerox PARC. 1979.

[33]. Sítio Oficial da Tecnologia Java: The Source for Java Developers. Disponível em: <http://java.sun.com>. Acesso em: 03 nov. 2008.

[34]. Sítio Oficial da ferramenta Eclipse. Eclipse - plataforma aberta. Disponível em: <http://www.eclipse.org>. Acesso em: 03 nov. 2008.

[35]. Sítio Oficial do Visual Editor Project. Disponível em: <http://www.eclipse.org/vep/WebContent/main.php>. Acesso em: 03 nov. 2008.

[36]. Sítio Oficial do framework Jena - A Semantic Web Framework For Java. 2000. Disponível em: <http://jena.sourceforge.net/>. Acesso em: 03 nov. 2008.

[37]. Sítio Oficial do Projeto Apache Axis. 2006. Disponível em: <http://ws.apache.org/axis/>. Acesso em: 03 nov. 2008.

[38]. Sítio Oficial do Projeto Mindswap OWL-S API. Disponível em: <http://www.mindswap.org/2004/owl-s/api/>. Acesso em: 03 nov. 2008.

[39]. Sítio Oficial do Projeto Mindswap Pellet. Disponível em: <http://clarkparsia.com/pellet/>. Acesso em: 03 nov. 2008.

APÊNDICE I – CÓDIGO

Neste apêndice, será mostrado o código do protótipo da ferramenta COMPOWS referente apenas as classes tidas como importantes, cujo conhecimento da implementação se julgue importantes. Os próximos itens representam as classes de interesse.

1. Compositor

```
     package compositor;

import java.io.FileNotFoundException;
import java.net.URISyntaxException;
import java.util.List;

import org.mindswap.owls.service.Service;

public class Compositor implements CompositorFachada{

    static Compositor comp;


    public static Compositor getInstance() throws FileNotFoundException,
URISyntaxException{
      if(comp == null){
            comp = new Compositor();
            Matchmaker.getInstance().getServices(); //load nos servicos.
      }
            return comp;
    }

    public Service createSequence(List listaServs) throws
FileNotFoundException, URISyntaxException{
      return Sequenciador.getInstance().createSequenceService(listaServs);
    }

    public Service getService(String uriServ) throws URISyntaxException,
FileNotFoundException{
      return Matchmaker.getInstance().getService(uriServ);
    }

   public List getServices() throws FileNotFoundException,
URISyntaxException{
       return Matchmaker.getInstance().getServices();
```

```
   }

   public List listarServicosSequence(Service serv) throws
FileNotFoundException, URISyntaxException{
        return Matchmaker.getInstance().listarServicosSequence(serv);
   }

    public String rodarExemploComplexo() throws Exception {
        return CreateComplexProcess.rodarExemplo();
    }

}
```

2. CreateComplexProcess

```
package compositor;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.net.URI;

import org.mindswap.owl.EntityFactory;
import org.mindswap.owl.OWLClass;
import org.mindswap.owl.OWLDataProperty;
import org.mindswap.owl.OWLDataType;
import org.mindswap.owl.OWLDataValue;
import org.mindswap.owl.OWLFactory;
import org.mindswap.owl.OWLIndividual;
import org.mindswap.owl.OWLKnowledgeBase;
import org.mindswap.owl.OWLOntology;
import org.mindswap.owl.vocabulary.XSD;
import org.mindswap.owls.OWLSFactory;
import org.mindswap.owls.grounding.Grounding;
import org.mindswap.owls.process.AtomicProcess;
import org.mindswap.owls.process.CompositeProcess;
import org.mindswap.owls.process.Condition;
import org.mindswap.owls.process.IfThenElse;
import org.mindswap.owls.process.Input;
import org.mindswap.owls.process.Local;
import org.mindswap.owls.process.Output;
import org.mindswap.owls.process.Perform;
import org.mindswap.owls.process.Process;
import org.mindswap.owls.process.ProcessList;
import org.mindswap.owls.process.Produce;
import org.mindswap.owls.process.Result;
import org.mindswap.owls.process.Sequence;
```

```
import org.mindswap.owls.process.SplitJoin;
import org.mindswap.owls.process.ValueData;
import org.mindswap.owls.process.execution.ProcessExecutionEngine;
import org.mindswap.owls.process.execution.SimpleProcessMonitor;
import org.mindswap.owls.profile.Profile;
import org.mindswap.owls.service.Service;
import org.mindswap.query.ValueMap;
import org.mindswap.swrl.Atom;
import org.mindswap.swrl.AtomList;
import org.mindswap.swrl.SWRLDataObject;
import org.mindswap.swrl.SWRLFactory;
import org.mindswap.swrl.SWRLFactoryCreator;
import org.mindswap.utils.URIUtils;


public class CreateComplexProcess {
    public static final URI baseURI = URI.create(
"http://www.example.org/FindCheaperBook.owl#" );
    public static final URI concepts = URI.create(
"http://www.mindswap.org/2004/owl-s/concepts.owl#" );

    OWLKnowledgeBase kb;

    OWLOntology ont;

    SWRLFactory swrl;

    Process BookFinder;
    Process BNPrice;
    Process AmazonPrice;

    OWLDataType xsdString;
    OWLDataType xsdFloat;
    OWLClass Price;
    OWLDataProperty amount;

    CompositeProcess ComparePricesProcess;

    Input CP_AmazonPrice;
    Input CP_BNPrice;

    Output CP_Bookstore;
    Output CP_OutputPrice;

    public CreateComplexProcess() {
    }

    private URI uri( String localName ) {
        return URIUtils.createURI( baseURI, localName );
    }
```

```
    private Service createService() throws Exception {
        Service service = ont.createService( uri( "TestService" ) );

        CompositeProcess process = createProcess();
        Profile profile = createProfile( process );
        Grounding grounding = createGrounding( process );

        service.setProfile( profile );
        service.setProcess( process );
        service.setGrounding( grounding );

        return service;
    }

    private Profile createProfile( Process process ) {
        Profile profile = ont.createProfile( uri( "TestProfile" ) );

        profile.setLabel( "Cheaper Book Finder" );
        profile.setTextDescription(
            "Find the price of book in Amazon and Barnes & Nobles " +
            "and return the cheaper price" );

        for(int i = 0; i < process.getInputs().size(); i++) {
            Input input = process.getInputs().inputAt( i );

            profile.addInput( input );
        }

        for(int i = 0; i < process.getOutputs().size(); i++) {
            Output output = process.getOutputs().outputAt( i );

            profile.addOutput( output );
        }

        return profile;
    }

    private Grounding createGrounding( CompositeProcess process ) {
        Grounding grounding = ont.createGrounding( uri( "TestGrounding" )
);

        ProcessList list = process.getComposedOf().getAllProcesses();
        for(int i = 0; i < list.size(); i++) {
            Process pc = list.processAt( i );
            if( pc instanceof AtomicProcess ) {
                grounding.addGrounding( ((AtomicProcess)
pc).getGrounding() );
            }
        }
```

```
        return grounding;
    }

    private CompositeProcess createProcess() {
        CompositeProcess process = ont.createCompositeProcess( uri(
"TestProcess") );

        Input inputBookName = process.createInput( uri( "BookName" ),
xsdString );

        Output outputBookstore = process.createOutput( uri( "Bookstore" ),
xsdString );

        Output outputBookPrice = process.createOutput( uri( "BookPrice" ),
Price );

        Sequence sequence = ont.createSequence();
        process.setComposedOf(sequence);

        Perform FindBookInfo = ont.createPerform( uri( "FindBookInfo" ) );
        FindBookInfo.setProcess( BookFinder );
        FindBookInfo.addBinding( BookFinder.getInput(),
Perform.TheParentPerform, inputBookName );

        sequence.addComponent( FindBookInfo );

        SplitJoin split = ont.createSplitJoin();

        Perform FindAmazonPrice = ont.createPerform( uri(
"FindAmazonPrice" ) );
        FindAmazonPrice.setProcess( AmazonPrice );
        FindAmazonPrice.addBinding( AmazonPrice.getInput(), FindBookInfo,
BookFinder.getOutput() );
        split.addComponent( FindAmazonPrice );

        Perform FindBNPrice = ont.createPerform( uri( "FindBNPrice" ) );
        FindBNPrice.setProcess( BNPrice );
        FindBNPrice.addBinding( BNPrice.getInput(), FindBookInfo,
BookFinder.getOutput() );
        split.addComponent( FindBNPrice );

        sequence.addComponent( split );

        CompositeProcess ComparePricesProcess =
createComparePricesProcess();
        Perform ComparePrices = ont.createPerform( uri( "ComparePrices" )
);
        ComparePrices.setProcess( ComparePricesProcess );
        ComparePrices.addBinding( CP_AmazonPrice, FindAmazonPrice,
```

```
AmazonPrice.getOutput() );
        ComparePrices.addBinding( CP_BNPrice, FindBNPrice,
BNPrice.getOutput() );

        sequence.addComponent( ComparePrices );

        Result result = process.createResult();
        result.addBinding( outputBookstore, ComparePrices, CP_Bookstore );
        result.addBinding( outputBookPrice, ComparePrices, CP_OutputPrice
);

        return process;
    }

    private CompositeProcess createComparePricesProcess() {
        CompositeProcess ComparePricesProcess =
            ont.createCompositeProcess( uri( "ComparePricesProcess" ) );

        CP_AmazonPrice = ComparePricesProcess.createInput( uri(
"CP_AmazonPrice"), Price );
        CP_BNPrice = ComparePricesProcess.createInput( uri( "CP_BNPrice"),
Price );

        Local CP_AmazonPriceAmount = ComparePricesProcess.createLocal(
uri( "CP_AmazonPriceAmount"), xsdFloat );
        Local CP_BNPriceAmount = ComparePricesProcess.createLocal( uri(
"CP_BNPriceAmount"), xsdFloat );

        CP_OutputPrice = ComparePricesProcess.createOutput( uri(
"CP_OutputPrice"), Price );
        CP_Bookstore = ComparePricesProcess.createOutput( uri(
"CP_Bookstore"), xsdString );

        Condition precondition1 = ont.createSWRLCondition();
        Atom atom1 = swrl.createDataPropertyAtom( amount, CP_AmazonPrice,
CP_AmazonPriceAmount );
        AtomList list1 = swrl.createList( atom1 );
        precondition1.setBody( list1 );
        ComparePricesProcess.addCondition( precondition1 );

        Condition precondition2 = ont.createSWRLCondition();
        Atom atom2 = swrl.createDataPropertyAtom( amount, CP_BNPrice,
CP_BNPriceAmount );
        AtomList list2 = swrl.createList( atom2 );
        precondition2.setBody( list2 );
        ComparePricesProcess.addCondition( precondition2 );

        IfThenElse ifThenElse = ont.createIfThenElse();
        ComparePricesProcess.setComposedOf( ifThenElse );
```

```
        Condition condition = ont.createSWRLCondition();
        Atom atom = swrl.createLessThan(
            (SWRLDataObject)
CP_BNPriceAmount.castTo(SWRLDataObject.class),
            (SWRLDataObject)
CP_AmazonPriceAmount.castTo(SWRLDataObject.class));
        AtomList list = swrl.createList( atom );
        condition.setBody( list );

        ifThenElse.setCondition( condition );

        ifThenElse.setThen( createProduceSequence( "BN", CP_BNPrice ) );
        ifThenElse.setElse( createProduceSequence( "Amazon",
CP_AmazonPrice ) );

        return ComparePricesProcess;
    }

    private Sequence createProduceSequence( String bookstore, Input price
) {
        Produce producePrice = ont.createProduce( uri( "Produce" +
bookstore + "Price" ) );
        producePrice.addBinding( CP_OutputPrice, Perform.TheParentPerform,
price );

        Produce produceName = ont.createProduce( uri( "Produce" +
bookstore + "Name" ) );
        ValueData valueData = ont.createValueData( ont.createDataValue(
bookstore ) );
        produceName.addBinding( CP_Bookstore, valueData);

        Sequence sequence = ont.createSequence();
        sequence.addComponent( producePrice );
        sequence.addComponent( produceName );

        return sequence;
    }

    public String run() throws Exception {
        kb = OWLFactory.createKB();

        ont = kb.createOntology();

      swrl = SWRLFactoryCreator.createFactory();

        BookFinder = kb.readService("http://www.mindswap.org/2004/owl-
s/1.1/BookFinder.owl#").getProcess();
        BNPrice = kb.readService("http://www.mindswap.org/2004/owl-
s/1.1/BNPrice.owl#").getProcess();
        AmazonPrice = kb.readService("http://www.mindswap.org/2004/owl-
```

```
s/1.1/AmazonBookPrice.owl#").getProcess();

        ont.addImport( BookFinder.getOntology() );
        ont.addImport( BNPrice.getOntology() );
        ont.addImport( AmazonPrice.getOntology() );

        xsdString = kb.getDataType( XSD.string );
        xsdFloat = kb.getDataType( XSD.xsdFloat );
        Price = kb.getClass( URIUtils.createURI( concepts, "Price" ) );
        amount = kb.getDataProperty( URIUtils.createURI( concepts,
"amount" ) );

        Service s = createService();

        File f = new File("temp.owl");

      try {
            f.createNewFile();
            BufferedWriter out = new BufferedWriter(new
FileWriter("temp.owl"));
            out.write("SERVICE: \n"+s.toRDF()+"\nPROFILE:
\n"+s.getProfile().toRDF()+"\nPROCESS:
\n"+s.getProcess().toRDF()+"\nGROUNDING: \n"+s.getGrounding().toRDF());
            out.close();
        } catch (IOException e1) {
            e1.printStackTrace();
      }catch (Exception e1) {
      }


        ProcessExecutionEngine exec = OWLSFactory.createExecutionEngine();

        exec.addMonitor( new SimpleProcessMonitor() );

         Process process = s.getProcess();

      ValueMap values = new ValueMap();
        values.setValue( process.getInput(),
            EntityFactory.createDataValue( "City of Glass" ) );

         values = exec.execute( process, values );

        String bookstore = values.getStringValue( process.getOutput(
"Bookstore" ) );
        OWLIndividual price = values.getIndividualValue(
process.getOutput( "BookPrice" ) );

        String retorno = "";
        retorno +=  "Livro mais barato encontrado na loja: " + bookstore ;
```

```
            price.setProperty( amount, "17" );

        retorno += "\nValor: $" + price.getProperty( amount );
        return retorno;
    }

    public static void main( String[] args ) throws Exception {
        CreateComplexProcess test = new CreateComplexProcess();
        String retorno = test.run();
        System.out.println(retorno);
    }

    public static String rodarExemplo() throws Exception{
       CreateComplexProcess test = new CreateComplexProcess();
         String retorno = test.run();
         return retorno;
    }
}
```

3. Match

```
package compositor;

import org.mindswap.owls.process.Input;
import org.mindswap.owls.process.Output;
import org.mindswap.owls.service.Service;

public class Match {
    public static String[] MATCHES = {"EXACT", "PLUGIN", "SUBSUME" ,
"FAIL"};
    public static int EXACT   = 0;
    public static int PLUGIN = 1;
    public static int SUBSUME = 2;
    public static int FAIL    = 3;

    public int matchType;
    public boolean listMatch;
    public Service outputService;
    public Output output;
    public Service inputService;
    public Input input;

    public Match(int matchType, Output output, Input input) {
        this.matchType = matchType;
        this.outputService = output.getService();
        this.output = output;
        this.inputService = input.getService();
        this.input = input;
```

```
    }

    public String toString() {
        String str = "";

        str += MATCHES[matchType] + " ";
        if(listMatch)
            str += ".LIST";
        str += outputService.getLocalName() + "." + output.getLocalName();
        str += " -> ";
        str += inputService.getLocalName() + "." + input.getLocalName();

        return str;
    }

}
```

4. Matchmaker

```
/*
 * Created on Oct 13, 2004
 */
package compositor;

import java.io.FileNotFoundException;
import java.net.URI;
import java.net.URISyntaxException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.mindswap.owl.OWLFactory;
import org.mindswap.owl.OWLKnowledgeBase;
import org.mindswap.owl.OWLModel;
import org.mindswap.owl.OWLType;
import org.mindswap.owls.process.Input;
import org.mindswap.owls.process.Output;
import org.mindswap.owls.service.Service;
import org.mindswap.query.ValueMap;

public class Matchmaker {

    OWLKnowledgeBase kb;
    static Matchmaker match;


    public Matchmaker() throws FileNotFoundException, URISyntaxException {
        kb = OWLFactory.createKB();
```

```
        kb.setReasoner("Pellet");
    }

    public static Matchmaker getInstance() throws FileNotFoundException,
URISyntaxException{
      if( match == null ){
            match = new Matchmaker();

                match.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/BNPrice.owl");
                match.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/BookFinder.owl");
                match.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/AmazonBookPrice.owl");
                match.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/Dictionary.owl");
                match.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/ZipCodeFinder.owl");
                match.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/FindLatLong.owl");
                match.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/BabelFishTranslator.owl#");
      }
      return match;
    }


    public List getServices(){
      return kb.getServices();
    }

    public void addOntology( String ont )  throws FileNotFoundException,
URISyntaxException {
        System.out.println( "Validando " + ont );
        kb.read( new URI( ont ) );
    }

    public void addOntology( URI ont )  throws FileNotFoundException {
        System.out.println( "Validando " + ont );
        kb.read( ont );
    }

    public List findServices(boolean getProducers) {
        String hasParameter = getProducers ? "process:hasOutput" :
"process:hasInput";

        String queryString =
            "SELECT * " +
            "WHERE " +
            "        (?process rdf:type process:Process)" +
```

```
            "       (?process " + hasParameter + " ?param)" +
            "USING " +
            "      process FOR <http://www.daml.org/services/owl-
s/1.1/Process.owl#>";

        return kb.query( queryString );
    }

    public List findOutputs() {
        return findServices(true);
    }

    public List findInputs() {
        return findServices(false);
    }

    public int getMatchType(OWLType outputType, OWLType inputType) {
        if(outputType.isEquivalent(inputType))
           return Match.EXACT;
        else if(outputType.isSubTypeOf(inputType))
           return Match.PLUGIN;
        else if(inputType.isSubTypeOf(outputType))
            return Match.SUBSUME;
        else
            return Match.FAIL;
    }

    public Service getService(String uriServ) throws URISyntaxException{
      Service modelo = kb.getService(new URI(uriServ));
      return modelo;
    }

    public List displayAllMatches() {
      List matches = new ArrayList();

      System.out.println( "Computing matches..." );

      List producers = findOutputs();
      List consumers = findInputs();

      Iterator i = producers.iterator();
      while( i.hasNext() ) {
          ValueMap binding = (ValueMap) i.next();
          Output output = (Output)
binding.getValue("param").castTo(Output.class);
          OWLType outputType = output.getParamType();

          Iterator j = consumers.iterator();
          while( j.hasNext() ) {
              binding = (ValueMap) j.next() ;
```

```
                Input input = (Input)
binding.getValue("param").castTo(Input.class);
                OWLType inputType = input.getParamType();

//              System.out.println("Trying " +
//                  URIUtils.getLocalName(outputType.getURI()) + " " +
//                  URIUtils.getLocalName(inputType.getURI()) + " " +
//                  producer.getLocalName() + " " +
//                  output.getLocalName() + " " +
//                  consumer.getLocalName() + " " +
//                  input.getLocalName()
//              );

              int matchType = getMatchType(outputType, inputType);
              if(matchType != Match.FAIL)
                  matches.add(new Match(matchType, output, input));

          }
      }

      return matches;
    }

    public List listarServicosSequence(Service serv){
      List matches = new ArrayList();

      List consumers = findInputs();
      List outputs = serv.getProfile().getOutputs();


      Iterator i = consumers.iterator();
      while( i.hasNext() ) {
          ValueMap binding = (ValueMap) i.next();
          Input inp = (Input)
binding.getValue("param").castTo(Input.class);
          OWLType inputType = inp.getParamType();

          Iterator j = outputs.iterator();
            while( j.hasNext() ){
                Output saida = (Output) j.next();
                //System.out.println(saida.getLocalName());
                OWLType outputType = saida.getParamType();

              int matchType = getMatchType(outputType, inputType);
              if(matchType != Match.FAIL && matchType != Match.SUBSUME)
                  matches.add(new Match(matchType, saida, inp));

          }
      }
      return matches;
```

```
    }

    public static void printIterator(Iterator i) {
        if(i.hasNext()) {
            while (i.hasNext())
                System.out.println( i.next() );
        }
        else
            System.out.println("<EMPTY>");

        System.out.println();
    }

//    public void run() throws FileNotFoundException, URISyntaxException {
//        Matchmaker matchmaker = new Matchmaker();
//
//        matchmaker.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/BNPrice.owl");
//        matchmaker.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/BookFinder.owl");
//        //matchmaker.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/CurrencyConverter.owl");
//        matchmaker.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/Dictionary.owl");
//        matchmaker.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/ZipCodeFinder.owl");
//        matchmaker.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/FindLatLong.owl");
//        matchmaker.addOntology("http://www.mindswap.org/2004/owl-
s/1.1/BabelFishTranslator.owl#");
//
//        List matches = matchmaker.displayAllMatches();
//        System.out.println();
//        System.out.println("Matches:");
//        printIterator(matches.iterator());
//    }

//    public static void main(String[] args) throws FileNotFoundException,
URISyntaxException {
//          Matchmaker test = new Matchmaker();
//          test.run();
//    }
}
```

5. Descobridor

```
package descobridor;
```

```
import java.io.File;
import java.io.PrintWriter;
import java.net.URI;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import org.mindswap.owl.OWLFactory;
import org.mindswap.owl.OWLKnowledgeBase;
import org.mindswap.owl.OWLOntology;
import org.mindswap.owls.service.Service;
import org.mindswap.utils.OutputFormatter;

public class Descobridor implements DescobridorFachada {
    boolean parseRDFFinished = false;

    List serviceURIs = new ArrayList();
    List errors = new ArrayList();
    List warnings = new ArrayList();
    List rdfProblems = new ArrayList();

    public Descobridor() {

    }

    public boolean validar(String fileURI) throws Exception {
      return validate(fileURI, new PrintWriter(System.out));
    }

    public boolean validate(String fileURI, PrintWriter output) throws
Exception {
      return validate(fileURI, output, false);
    }

    public boolean validate(String fileURI, PrintWriter output, boolean
isHTML) throws Exception {
      return validate(fileURI, new OutputFormatter(output, isHTML));
    }

    public boolean validate(String fileURI, OutputFormatter out) throws
Exception {
      URI uri = (new File("temp.owl")).toURI();

      OWLKnowledgeBase kb = OWLFactory.createKB();

      OWLOntology ont = kb.read(uri);

      List services = new ArrayList();
```

```
      //out.printBold("N˙mero de serviÁos descobertos:
").println(ont.getServices().size() +"");
      //out.printBold("N˙mero de serviÁos v·lidos:
").println(services.size()+"");
      out.println();

      printMesssages(out, "Problemas RDF : ", rdfProblems.iterator());
      printMesssages(out, "Erros: ", errors.iterator());
      printMesssages(out, "Warnings: ", warnings.iterator());

      if(services != null) {
            for(int i = 0; i < services.size(); i++) {
                  Service service = (Service) services.get(i);
                  out.printBold("ServiÁo:
").printLink(service.getURI().toString());
//                out.print(" (Version:
").print(service.getOWLSVersion()).print(")").println();
                  out.printBold("Nome: ").println(service.getLabel());
                  out.printBold("DescriÁ„o:
").println(service.getProfile().getTextDescription());
            }
      }

      out.flush();

      // FIXME return value for validation
      // return services.size() > 0 && errors.isEmpty();
      return true;
    }

    public void printMesssages(OutputFormatter out, String header,
Iterator i) {
      if(i.hasNext()) {
            out.printBold(header).println();

            while(i.hasNext()) {
                  String error = (String) i.next();

                  if(out.isFormatHTML()) out.print("<ul>");
                  if(out.isFormatHTML()) {
                        out.print("<li>");
                        out.print(format(error));
                        out.print("</li>");
                  }
                  else
                        out.println(error);
                  if(out.isFormatHTML()) out.print("</ul>");
            }

            out.println();
```

```
      }
    }


    public String format(String str) {
      return str;
//    return str.replaceAll("\n", "<br>").replaceAll(" ", " ");
//    StringBuffer buffer = new StringBuffer();
//    String link;
//    int start = 0, finish;
//    int index = str.indexOf("http://");
//
//    while(index != -1) {
//          buffer.append(str.substring(start, index));
//
//          finish = Math.min(str.indexOf(' ', index), str.indexOf('\n',
index));
//          if(finish > str.indexOf('<', index)) finish = str.indexOf('\n',
index);
//          if(finish == -1) finish = str.length();
//          link = str.substring(index, finish);
//          buffer.append("<a href=\"").append(link).append("\">");
//          buffer.append(link).append("</a>");
//          start = index + link.length();
//          index = str.indexOf("http://", start);
//    }
//    buffer.append(str.substring(start));
//
//    return buffer.toString().replaceAll("\n", "<br>");
    }

    public String getServiceURI() {
      return (String) serviceURIs.get(serviceURIs.size() - 1);
    }

    public void put(Map table, String key, String value) {
      List list = (List) table.get(key);

      if(list == null) {
            list = new ArrayList();
            table.put(key, list);
      }

      list.add(value);
    }

    /* (non-Javadoc)
     * @see
org.mindswap.owls.io.OWLSReaderListener#startService(java.lang.String)
```

```
     */
    public void startService(String serviceURI) {
      serviceURIs.add(serviceURI);
    }

    /* (non-Javadoc)
     * @see
org.mindswap.owls.io.OWLSReaderListener#finishService(java.lang.String)
     */
    public void finishService(String serviceURI) {
    }

    /* (non-Javadoc)
     * @see
org.mindswap.owls.io.OWLSReaderListener#warning(java.lang.String)
     */
    public void warning(String msg) {
        if(parseRDFFinished)
            warnings.add(msg);
        else
            rdfProblems.add(msg);
    }

    /* (non-Javadoc)
     * @see
org.mindswap.owls.io.OWLSReaderListener#error(java.lang.String)
     */
    public void error(String msg) {
        if(parseRDFFinished)
            errors.add(msg);
        else
            rdfProblems.add(msg);
    }


    /* (non-Javadoc)
     * @see org.mindswap.owls.io.OWLSReaderListener#startParseRDF()
     */
    public void startParseRDF() {
        parseRDFFinished = false;
    }

    /* (non-Javadoc)
     * @see org.mindswap.owls.io.OWLSReaderListener#finishParseRDF()
     */
    public void finishParseRDF() {
        parseRDFFinished = true;
    }

//  public static void main(String[] args) throws Exception {
```

```
//
//    Descobridor validator = new Descobridor();
//    System.out.println("Valid: " + validator.validate(""));
//  }

}
```

6. ProgramaSequence

```
package gui;

import java.io.FileNotFoundException;
import java.net.URISyntaxException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;

import org.mindswap.owl.EntityFactory;
import org.mindswap.owls.process.Condition;
import org.mindswap.owls.process.Input;
import org.mindswap.owls.process.Output;
import org.mindswap.owls.service.Service;
import org.mindswap.query.ValueMap;

import compositor.Compositor;
import compositor.Match;
import executor.Executor;

public class ProgramaSequence {


      public static void printIterator(Iterator i) {
            if(i.hasNext()) {
              while (i.hasNext())
                  System.out.println( i.next() );
            }
            else
                System.out.println("<EMPTY>");

            System.out.println();
        }

    public static void main(String[] args){

      try {
```

```
            System.out.println("listar servicos.");
      //listar servicos.
            List listaServicos;
            listaServicos = Compositor.getInstance().getServices();


            System.out.println("selecao do servico.");
      //selecao do servico.
            int selecionado = 4;
            Service servSelec = (Service) listaServicos.get(selecionado);
            System.out.println(servSelec.getName());

            System.out.println("mostrar informacoes do servico
selecionado.");
      //mostrar informacoes do servico selecionado.

    System.out.println(servSelec.getProfile().getTextDescription());

            System.out.println("Entradas: ");
            List inputs = servSelec.getProfile().getInputs();
            Iterator i = inputs.iterator();
            while( i.hasNext() ){
                  Input entrada = (Input) i.next();
                  System.out.println(entrada.getLocalName());
            }

            System.out.println("Saidas: ");
            List outputs = servSelec.getProfile().getOutputs();
            Iterator j = outputs.iterator();
            while( j.hasNext() ){
                  Output saida = (Output) j.next();
                  System.out.println(saida.getLocalName());
            }

            System.out.println("Pre-condicoes: ");
            List cond = servSelec.getProfile().getConditions();
            Iterator k = cond.iterator();
            while( k.hasNext() ){
                  Condition con = (Condition) k.next();
                  System.out.println(con.getLocalName());
            }

            System.out.println("Efeitos: ");
            List co = servSelec.getProfile().getResults();
            Iterator y = co.iterator();
            while( y.hasNext() ){
                  Condition con = (Condition) y.next();
                  System.out.println(con.getLocalName());
            }
```

```
            System.out.println("colocar operador sequence.");
      //colocar operador sequence.
            //meramente visual, tratado na gui.

            System.out.println("mostrar opcoes de composicao com o servico
(matchmaker).");
      //mostrar opcoes de composicao com o servico (matchmaker).
            List matches =
Compositor.getInstance().listarServicosSequence(servSelec);
            printIterator(matches.listIterator());

            System.out.println("escolha do outro servico (na verdade,
escolha de uma opcao da lista passada).");
      //escolha do outro servico (na verdade, escolha de uma opcao da lista
passada).
            selecionado = 1;
            Service servSelec2 = (Service) ((Match)
matches.get(selecionado)).inputService;
            System.out.println(servSelec2.getName());

            System.out.println("mostrar informacoes do servico
selecionado.");
      //mostrar informacoes do servico selecionado.

    System.out.println(servSelec2.getProfile().getTextDescription());

            System.out.println("Entradas: ");
            List inputs2 = servSelec2.getProfile().getInputs();
            Iterator i2 = inputs2.iterator();
            while( i2.hasNext() ){
                  Input entrada = (Input) i2.next();
                  System.out.println(entrada.getLocalName());
            }

            System.out.println("Saidas: ");
            List outputs2 = servSelec2.getProfile().getOutputs();
            Iterator j2 = outputs2.iterator();
            while( j2.hasNext() ){
                  Output saida = (Output) j2.next();
                  System.out.println(saida.getLocalName());
            }

            System.out.println("Pre-condicoes: ");
            List cond2 = servSelec2.getProfile().getConditions();
            Iterator k2 = cond2.iterator();
            while( k2.hasNext() ){
                  Condition con = (Condition) k2.next();
                  System.out.println(con.getLocalName());
            }
```

```
            System.out.println("Efeitos: ");
            List co2 = servSelec.getProfile().getResults();
            Iterator y2 = co2.iterator();
            while( y2.hasNext() ){
                  Condition con = (Condition) y2.next();
                  System.out.println(con.getLocalName());
            }

      //selecao das saidas da composicao.
            //abstraido.

      //criacao da composicao.
            List services = new ArrayList();
            services.add(servSelec);
            services.add(servSelec2);
            Service composto =
Compositor.getInstance().createSequence(services);

      //mostrar OWL-S da composicao.
            System.out.println(composto.toRDF());
            System.out.println(composto.getProfile().toRDF());
            System.out.println(composto.getProcess().toRDF());

      //executar composicao (com passagem de parametros).
            List entradas = new ArrayList();
            entradas.add("love");
            //entradas.add("French");
            //entradas.add("City");
            //entradas.add("English");
            Executor.getInstance().executarServico(composto, entradas);

      } catch (Exception e) {
            e.printStackTrace();
      }

    }
}
```

APÊNDICE II – DESCRIÇÕES DE WEB SERVICES

Neste apêndice, serão mostradas as descrições em OWL-S dos serviços utilizados no estudo de caso da ferramenta COMPOWS.

1. BookFinder

OWL-S

```
<rdf:RDF xml:base="http://www.mindswap.org/2004/owl-s/1.1/BookFinder.owl">
–
<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Service.owl"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Profile.owl"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Process.owl"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Grounding.owl"/>
<!-- use the cached version for bibtex ontology -->
<owl:imports rdf:resource="http://www.mindswap.org/ontologies/bibtex.owl"/>
</owl:Ontology>
<!-- Service description -->
–
<service:Service rdf:ID="BookFinderService">
<service:presents rdf:resource="#BookFinderProfile"/>
<service:describedBy rdf:resource="#BookFinderProcess"/>
<service:supports rdf:resource="#BookFinderGrounding"/>
</service:Service>
<!-- Profile description -->
–
<mind:BookInformationService rdf:ID="BookFinderProfile">
<service:presentedBy rdf:resource="#BookFinderService"/>
<profile:serviceName xml:lang="en">Book Finder</profile:serviceName>
–
<profile:textDescription xml:lang="en">
This service returns the information of a book whose title best matches the
given string.
</profile:textDescription>
<profile:hasInput rdf:resource="#BookName"/>
<profile:hasOutput rdf:resource="#BookInfo"/>
</mind:BookInformationService>
<!-- Process description -->
–
<process:AtomicProcess rdf:ID="BookFinderProcess">
<service:describes rdf:resource="#BookFinderService"/>
```

<process:hasInput rdf:resource="#BookName"/>
<process:hasOutput rdf:resource="#BookInfo"/>
</process:AtomicProcess>
–
<process:Input rdf:ID="BookName">
<process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/20
01/XMLSchema#string</process:parameterType>
<rdfs:label>Book Name</rdfs:label>
</process:Input>
–
<process:Output rdf:ID="BookInfo">
<process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://purl.org/net/
nknouf/ns/bibtex#Book</process:parameterType>
<rdfs:label>Book Info</rdfs:label>
</process:Output>
<!-- Grounding description -->
–
<grounding:WsdlGrounding rdf:ID="BookFinderGrounding">
<service:supportedBy rdf:resource="#BookFinderService"/>
<grounding:hasAtomicProcessGrounding
rdf:resource="#BookFinderProcessGrounding"/>
</grounding:WsdlGrounding>
–
<grounding:WsdlAtomicProcessGrounding rdf:ID="BookFinderProcessGrounding">
<grounding:owlsProcess rdf:resource="#BookFinderProcess"/>
–
<grounding:wsdlDocument
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://cheeso.members.winisp.net/books/books.asmx?WSDL
</grounding:wsdlDocument>
–
<grounding:wsdlOperation>
–
<grounding:WsdlOperationRef>
–
<grounding:portType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://cheeso.members.winisp.net/books/LookyBookServiceSoap
</grounding:portType>
–
<grounding:operation
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://cheeso.members.winisp.net/books/DoKeywordSearch
</grounding:operation>
</grounding:WsdlOperationRef>
</grounding:wsdlOperation>
–
<grounding:wsdlInputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">

```
http://cheeso.members.winisp.net/books/DoKeywordSearchSoapIn
</grounding:wsdlInputMessage>
-
<grounding:wsdlInput>
-
<grounding:WsdlInputMessageMap>
<grounding:owlsParameter rdf:resource="#BookName"/>
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://cheeso.member
s.winisp.net/books/keyword</grounding:wsdlMessagePart>
</grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
-
<grounding:wsdlOutputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://cheeso.members.winisp.net/books/DoKeywordSearchSoapOut
</grounding:wsdlOutputMessage>
-
<grounding:wsdlOutput>
-
<grounding:WsdlOutputMessageMap>
<grounding:owlsParameter rdf:resource="#BookInfo"/>
-
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://cheeso.members.winisp.net/books/DoKeywordSearchResult
</grounding:wsdlMessagePart>
-
<grounding:xsltTransformationString>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ns1="http://dinoch.dyndns.org/webservices/books">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/ ">
      <xsl:variable name="pubdate"
select="ns1:DoKeywordSearchResult/ns1:bookInfo/ns1:pubdate"/>
      <xsl:variable name="month_day" select="substring-before($pubdate,',
')"/>
      <xsl:variable name="year" select="substring-after($pubdate,', ')"/>
      <rdf:RDF
            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
            xmlns:bibtex="http://purl.org/net/nknouf/ns/bibtex#">
            <bibtex:Book>
                  <bibtex:hasISBN>
                        <xsl:value-of
select="ns1:DoKeywordSearchResult/ns1:bookInfo/ns1:isbn"/>
                  </bibtex:hasISBN>
                  <bibtex:hasTitle>
```

```
                        <xsl:value-of
select="ns1:DoKeywordSearchResult/ns1:bookInfo/ns1:title"/>
                  </bibtex:hasTitle>
                  <bibtex:hasAuthor>
                        <xsl:value-of
select="ns1:DoKeywordSearchResult/ns1:bookInfo/ns1:author"/>
                  </bibtex:hasAuthor>
                  <bibtex:hasPublisher>
                        <xsl:value-of
select="ns1:DoKeywordSearchResult/ns1:bookInfo/ns1:publisher"/>
                  </bibtex:hasPublisher>
                  <bibtex:hasMonth>
                        <xsl:choose>
                              <xsl:when test="contains($month_day,' ')">
                                    <xsl:value-of select="substring-
after($month_day,' ')"/>
                              </xsl:when>
                              <xsl:otherwise>
                                    <xsl:value-of select="$month_day"/>
                              </xsl:otherwise>
                        </xsl:choose>
                  </bibtex:hasMonth>
                  <bibtex:hasYear
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
                        <xsl:value-of select="$year"/>
                  </bibtex:hasYear>
            </bibtex:Book>
      </rdf:RDF>
    </xsl:template>
</xsl:stylesheet>

</grounding:xsltTransformationString>
</grounding:WsdlOutputMessageMap>
</grounding:wsdlOutput>
</grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

2. BNPrice

OWL-S

```
<rdf:RDF xml:base="http://www.mindswap.org/2004/owl-s/1.1/BNPrice.owl">
–
<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Service.owl"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Profile.owl"/>
```

```
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Process.owl"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Grounding.owl"/>
<owl:imports rdf:resource="http://www.mindswap.org/2004/owl-
s/concepts.owl"/>
<!-- use the cached version for bibtex ontology -->
<owl:imports rdf:resource="http://www.mindswap.org/ontologies/bibtex.owl"/>
</owl:Ontology>
<!-- Service description -->
–
<service:Service rdf:ID="BNPriceService">
<service:presents rdf:resource="#BNPriceProfile"/>
<service:describedBy rdf:resource="#BNPriceProcess"/>
<service:supports rdf:resource="#BNPriceGrounding"/>
</service:Service>
<!-- Profile description -->
–
<mind:BookInformationService rdf:ID="BNPriceProfile">
<service:presentedBy rdf:resource="#BNPriceService"/>
<profile:serviceName xml:lang="en">BN Price Check</profile:serviceName>
–
<profile:textDescription xml:lang="en">
This service returns the price of a book as advertised in Barnes and Nobles
web site given the ISBN Number.
</profile:textDescription>
<profile:hasInput rdf:resource="#BookInfo"/>
<profile:hasOutput rdf:resource="#BookPrice"/>
</mind:BookInformationService>
<!-- Process Model description -->
–
<process:AtomicProcess rdf:ID="BNPriceProcess">
<service:describes rdf:resource="#BNPriceService"/>
<process:hasInput rdf:resource="#BookInfo"/>
<process:hasOutput rdf:resource="#BookPrice"/>
</process:AtomicProcess>
–
<process:Input rdf:ID="BookInfo">
<process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://purl.org/net/
nknouf/ns/bibtex#Book</process:parameterType>
<rdfs:label>Book Info</rdfs:label>
</process:Input>
–
<process:Output rdf:ID="BookPrice">
–
<process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://www.mindswap.org/2004/owl-s/concepts.owl#Price
</process:parameterType>
```

```
<rdfs:label>Book Price</rdfs:label>
</process:Output>
<!-- Grounding description -->
–
<grounding:WsdlGrounding rdf:ID="BNPriceGrounding">
<service:supportedBy rdf:resource="#BNPriceService"/>
<grounding:hasAtomicProcessGrounding
rdf:resource="#BNPriceProcessGrounding"/>
</grounding:WsdlGrounding>
–
<grounding:WsdlAtomicProcessGrounding rdf:ID="BNPriceProcessGrounding">
<grounding:owlsProcess rdf:resource="#BNPriceProcess"/>
–
<grounding:wsdlDocument
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl
</grounding:wsdlDocument>
–
<grounding:wsdlOperation>
–
<grounding:WsdlOperationRef>
–
<grounding:portType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#BNPriceSoap
</grounding:portType>
–
<grounding:operation
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#GetBNQuote
</grounding:operation>
</grounding:WsdlOperationRef>
</grounding:wsdlOperation>
–
<grounding:wsdlInputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#GetBNQuoteSoap
In
</grounding:wsdlInputMessage>
–
<grounding:wsdlInput>
–
<grounding:WsdlInputMessageMap>
<grounding:owlsParameter rdf:resource="#BookInfo"/>
–
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#sISBN
</grounding:wsdlMessagePart>
–
<grounding:xsltTransformationString>
```

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:bibtex="http://purl.org/net/nknouf/ns/bibtex#">
    <xsl:template match="/ ">
      <xsl:value-of select="rdf:RDF/bibtex:Book/bibtex:hasISBN"/>
    </xsl:template>
</xsl:stylesheet>

</grounding:xsltTransformationString>
</grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
-
<grounding:wsdlOutputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#GetBNQuoteSoap
Out
</grounding:wsdlOutputMessage>
-
<grounding:wsdlOutput>
-
<grounding:WsdlOutputMessageMap>
<grounding:owlsParameter rdf:resource="#BookPrice"/>
-
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl#GetBNQuoteResu
lt
</grounding:wsdlMessagePart>
-
<grounding:xsltTransformationString>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <xsl:variable name="X1" select="/"/>
      <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:concepts="http://www.mindswap.org/2004/owl-s/concepts.owl#">
            <concepts:Price>
                  <concepts:currency
rdf:resource="http://www.daml.ecs.soton.ac.uk/ont/currency.owl#USD"/>
                  <concepts:amount
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
                        <xsl:value-of select="$X1"/>
                  </concepts:amount>
            </concepts:Price>
      </rdf:RDF>
    </xsl:template>
</xsl:stylesheet>
```

```
</grounding:xsltTransformationString>
</grounding:WsdlOutputMessageMap>
</grounding:wsdlOutput>
</grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

3. AmazonBookPrice

OWL-S

```
<rdf:RDF xml:base="http://www.mindswap.org/2004/owl-
s/1.1/AmazonBookPrice.owl">
–
<owl:Ontology>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Service.owl#"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Profile.owl#"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Process.owl#"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Grounding.owl#"/>
<!-- use the cached version for bibtex ontology -->
<owl:imports rdf:resource="http://www.mindswap.org/ontologies/bibtex.owl"/>
</owl:Ontology>
–
<service:Service rdf:ID="AmazonPriceService">
<service:presents rdf:resource="#AmazonPriceProfile"/>
<service:describedBy rdf:resource="#AmazonPriceProcess"/>
<service:supports rdf:resource="#AmazonPriceGrounding"/>
</service:Service>
–
<mind:BookInformationService rdf:ID="AmazonPriceProfile">
<service:isPresentedBy rdf:resource="#AmazonPriceService"/>
<profile:serviceName xml:lang="en">Amazon Book Price</profile:serviceName>
<profile:hasInput rdf:resource="#BookInfo"/>
<profile:hasOutput rdf:resource="#BookPrice"/>
</mind:BookInformationService>
–
<process:AtomicProcess rdf:ID="AmazonPriceProcess">
<service:describes rdf:resource="#AmazonPriceService"/>
<process:hasInput rdf:resource="#BookInfo"/>
<process:hasOutput rdf:resource="#BookPrice"/>
</process:AtomicProcess>
–
<process:Input rdf:ID="BookInfo">
<rdfs:label>Book</rdfs:label>
```

<process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://purl.org/net/
nknouf/ns/bibtex#Book</process:parameterType>
</process:Input>
–
<process:Output rdf:ID="BookPrice">
<rdfs:label>Price</rdfs:label>
–
<process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://www.mindswap.org/2004/owl-s/concepts.owl#Price
</process:parameterType>
</process:Output>
–
<grounding:WsdlGrounding rdf:ID="AmazonPriceGrounding">
<service:supportedBy rdf:resource="#AmazonPriceService"/>
<grounding:hasAtomicProcessGrounding
rdf:resource="#AmazonPriceProcessGrounding"/>
</grounding:WsdlGrounding>
–
<grounding:WsdlAtomicProcessGrounding rdf:ID="AmazonPriceProcessGrounding">
<grounding:owlsProcess rdf:resource="#AmazonPriceProcess"/>
–
<grounding:wsdlOperation>
–
<grounding:WsdlOperationRef>
<grounding:portType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">/AWSECommerceService
Port</grounding:portType>
–
<grounding:operation
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://webservices.amazon.com/AWSECommerceService/2005-10-13/ItemLookup
</grounding:operation>
</grounding:WsdlOperationRef>
</grounding:wsdlOperation>
–
<grounding:wsdlInputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://webservices.amazon.com/AWSECommerceService/2005-10-
13/ItemLookupRequestMsg
</grounding:wsdlInputMessage>
–
<grounding:wsdlOutputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://webservices.amazon.com/AWSECommerceService/2005-10-
13/ItemLookupResponseMsg
</grounding:wsdlOutputMessage>
–
<grounding:wsdlInput>

```
–
<grounding:WsdlInputMessageMap>
<grounding:owlsParameter rdf:resource="#BookInfo"/>
–
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://webservices.amazon.com/AWSECommerceService/2005-10-13/SubscriptionId
</grounding:wsdlMessagePart>
–
<grounding:xsltTransformationString>

            <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:amazon="http://webservices.amazon.com/AWSECommerceService/2005-10-
13">
              <xsl:template match="/">

<amazon:SubscriptionId>0YR717KBKX6R0GPTS082</amazon:SubscriptionId>
              </xsl:template>
            </xsl:stylesheet>

</grounding:xsltTransformationString>
</grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
–
<grounding:wsdlInput>
–
<grounding:WsdlInputMessageMap>
<grounding:owlsParameter rdf:resource="#BookInfo"/>
–
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://webservices.amazon.com/AWSECommerceService/2005-10-13/AssociateTag
</grounding:wsdlMessagePart>
–
<grounding:xsltTransformationString>

            <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
              <xsl:template match="/"/>
            </xsl:stylesheet>

</grounding:xsltTransformationString>
</grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
–
<grounding:wsdlInput>
–
<grounding:WsdlInputMessageMap>
```

```
<grounding:owlsParameter rdf:resource="#BookInfo"/>
–
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://webservices.amazon.com/AWSECommerceService/2005-10-13/Validate
</grounding:wsdlMessagePart>
–
<grounding:xsltTransformationString>

            <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
              <xsl:template match="/"/>
            </xsl:stylesheet>

</grounding:xsltTransformationString>
</grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
–
<grounding:wsdlInput>
–
<grounding:WsdlInputMessageMap>
<grounding:owlsParameter rdf:resource="#BookInfo"/>
–
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://webservices.amazon.com/AWSECommerceService/2005-10-13/XMLEscaping
</grounding:wsdlMessagePart>
–
<grounding:xsltTransformationString>

            <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
                              <xsl:template match="/"></xsl:template>
                        </xsl:stylesheet>

</grounding:xsltTransformationString>
</grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
–
<grounding:wsdlInput>
–
<grounding:WsdlInputMessageMap>
<grounding:owlsParameter rdf:resource="#BookInfo"/>
–
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://webservices.amazon.com/AWSECommerceService/2005-10-13/Shared
</grounding:wsdlMessagePart>
–
<grounding:xsltTransformationString>
```

```
            <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
                                <xsl:template match="/"></xsl:template>
                    </xsl:stylesheet>

</grounding:xsltTransformationString>
</grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
-
<grounding:wsdlInput>
-
<grounding:WsdlInputMessageMap>
<grounding:owlsParameter rdf:resource="#BookInfo"/>
-
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://webservices.amazon.com/AWSECommerceService/2005-10-13/Request
</grounding:wsdlMessagePart>
-
<grounding:xsltTransformationString>

            <xsl:stylesheet version="1.0"
              xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
              xmlns:bibtex="http://purl.org/net/nknouf/ns/bibtex#"

xmlns:amazon="http://webservices.amazon.com/AWSECommerceService/2005-10-
13">
              <xsl:template match="/">
                <amazon:Request>
                  <ItemId>
                    <xsl:value-of
select="rdf:RDF/bibtex:Book/bibtex:hasISBN"/>
                  </ItemId>
                  <ResponseGroup>Medium</ResponseGroup>
                </amazon:Request>
              </xsl:template>
            </xsl:stylesheet>

</grounding:xsltTransformationString>
</grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
-
<grounding:wsdlOutput>
-
<grounding:WsdlOutputMessageMap>
<grounding:owlsParameter rdf:resource="#BookPrice"/>
-
<grounding:wsdlMessagePart
```

```
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://webservices.amazon.com/AWSECommerceService/2005-10-13/Items
</grounding:wsdlMessagePart>
–
<grounding:xsltTransformationString>

            <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:amazon="http://webservices.amazon.com/AWSECommerceService/2005-10-
13">
              <xsl:template match="/">
                <xsl:variable name="price"
select="//amazon:Items/amazon:Item/amazon:ItemAttributes/amazon:ListPrice/a
mazon:Amount"/>
                <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
                  xmlns:concepts="http://www.mindswap.org/2004/owl-
s/concepts.owl#">
                  <concepts:Price>
                    <concepts:currency
rdf:resource="http://www.daml.ecs.soton.ac.uk/ont/currency.owl#USD"/>
                    <concepts:amount
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
                      <xsl:value-of select="$price div 100"/>
                    </concepts:amount>
                  </concepts:Price>
                </rdf:RDF>
              </xsl:template>
            </xsl:stylesheet>

</grounding:xsltTransformationString>
</grounding:WsdlOutputMessageMap>
</grounding:wsdlOutput>
–
<grounding:wsdlDocument
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl
</grounding:wsdlDocument>
</grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

_____

Prof. Frederico Luiz Gonçalves de Freitas

Orientador

_____

Filipe Luiz Mélo da Costa Monteiro

Aluno