UNIVERSIDADE FEDERAL DE CAMPINA GRANDE - UFCG
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA - COPIN

DISSERTAÇÃO DE MESTRADO

WEBS COMPOSER: UMA FERRAMENTA BASEADA EM ONTOLOGIAS PARA A DESCOBERTA E COMPOSIÇÃO DE SERVIÇOS NA WEB

Fabio Gomes de Andrade
(Mestrando)

Ulrich Schiel
(Orientador)

Cláudio de Souza Baptista
(Orientador)

CAMPINA GRANDE, JULHO DE 2006

WEBS COMPOSER:UMA FERRAMENTA BASEADA EM ONTOLOGIAS PARA A DESCOBERTA E COMPOSIÇÃO DE SERVIÇOS NA WEB

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para a obtenção de grau de Mestre em Informática.

Área de concentração: **Ciência da Computação**
Linha: **Sistemas de Informação e Banco de Dados**

Fabio Gomes de Andrade
(Mestrando)

Ulrich Schiel
(Orientador)

Cláudio de Souza Baptista
(Orientador)

CAMPINA GRANDE, JULHO DE 2006

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

A553w 2006

Andrade, Fábio Gomes de

Webs composer: uma ferramenta baseada em ontologias para a descoberta e composição de serviços na web / Fábio Gomes de Andrade. ─ Campina Grande, 2006.

115f.: il.

Referências.

Dissertação (Mestrado em Informática) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Orientador: Cláudio de Souza Baptista.

1─ Sistemas de Gerenciamento de Base de Dados 2─ Ontologia 3─ Web Services I─Título

CDU 004.65:111

"WEBS COMPOSER: UMA FERRAMENTA BASEADA EM ONTOLOGIAS PARA A DESCOBERTA E COMPOSIÇÃO DE SERVIÇOS NA WEB"

FABIO GOMES DE ANDRADE

DISSERTAÇÃO APROVADA EM 08.06.2006

PROF. ULRICH SCHIEL, Dr.
Orientador

PROF. CLÁUDIO DE SOUZA BAPTISTA, Ph.D
Orientador

PROF. MARCUS COSTA SAMPAIO, Dr.
Examinador

PROFª CLÁUDIA MARIA FERNANDES ARAÚJO RIBEIRO, D.Sc
Examinadora

CAMPINA GRANDE – PB

"Não é mérito o fato de nunca termos caído, mas o de termos nos levantado todas as vezes em que caímos"

Provérbio Árabe

Agradecimentos

Primeiramente a Deus, maior fonte de amor e misericórdia.

Ao meu pai José e à minha mãe Alzira, por todo o amor e carinho que me dedicaram por toda a minha vida, e por nunca terem me deixado desistir, às minhas irmãs Sheila e Fernanda por todo o carinho e apoio e ás minhas sobrinhas Gabriella e Lívia, fontes de doçura que me fazem esquecer de todos os meus problemas.

À minha noiva Ranielly, por todo o incentivo e carinho recebidos nos momentos mais difíceis.

Aos meus orientadores Ulrich e Cláudio, pela paciência, apoio, motivação, dedicação, conhecimentos transmitidos, pelas caronas para a rodoviária e pelos excelentes bate-papos.

Aos meus colegas de curso Elvis, Fábio Leite, Cláudio Campêlo e Damião, por todo o apoio e pelos ótimos momentos no LSI.

A todos os meus colegas de trabalho, especialmente aos meus grandes amigos Claudivan, Edemberg e Daladier, que sempre me incentivaram e permitiram a minha liberação.

Aos meu cunhado Gean, pela companhia durante as viagens para Campina Grande, e ao meu cunhado Camillo, pelo incentivo.

À Aninha, por sua simpatia e por resolver todos os meus problemas burocráticos.

À COPIN, pela oportunidade de cursar o mestrado.

A todas as pessoas que, de alguma forma, contribuíram para este trabalho.

Resumo

Atualmente, muitas empresas utilizam a plataforma de *web services* para oferecer os seus serviços na *web*. Estes serviços podem ainda ser compostos para resolver tarefas mais complexas. Isto fez surgir a necessidade do desenvolvimento de ferramentas de buscas capazes de localizar estes serviços com precisão e descobrir automaticamente composições de serviços que podem ser montadas para resolver uma requisição do usuário. Este trabalho apresenta WebS Composer, uma ferramenta que usa ontologias para localizar serviços e composições de serviços que atendem uma determinada solicitação. Além disso, ela usa características especificas do provedor do serviço para selecionar, dentre os resultados recuperados, aqueles que mais se adequam às necessidades e preferências do usuário.

Abstract

Recently, there are many companies using service oriented architectures in order to offer their services in the web. These services may still be composed in order to solve more complex tasks. This service composition demands for developing searching tools which are able to find those services with accuracy. This work presents WebS Composer, a tool that uses ontologies in order to find services and service compositions which satisfy a specific request. Furthermore, WebS Composer uses constraints based on the service provider's features to select, among the retrieved results, those that are more relevant to user request

Índice

Índice de Figuras

Índice de Tabelas

Lista de Abreviações

API (Application Programming Interface)

BPEL (Business Process Execution Language)

CDL (Contract Description Language)

CSSL (Composite Service Specification Language)

DAML (DARPA Agent Markup Language)

DAML - S (DARPA Agent Markup Language Services)

DAML + OIL (DARPA Agent Markup Language + Ontology Inference Layer)

HTML (HyperText Markup Language)

IRS (Internet Reasoning Service)

OIL (Ontology Inference Layer)

OWL (Ontology Web Language)

OWL-S (Ontology Web Language Services)

RAM (Random Access Memory)

RDF (Resource Description Logic)

SGBD (Sistema Gerenciador de Banco de Dados)

SQL (Structured Query Language)

SWRL (semantic Web Rule Language)

SWSF (Semantic Web Services Framework)

SWSL (Semantic Web Services Language)

UDDI (Universal Description, Discovery and Integration)

URL (Uniform Resource Locator)

XML (Extensible Marup Language)

WSDL (Web Services Definition Language)

WSFL (Web Services Flow Language)

WSMO (Web Service Modeling Ontology)

CAPÍTULO I - INTRODUÇÃO

Nos últimos anos, a tecnologia de *web services* (ALONSO et al, 2004) tem conquistado uma grande popularidade entre as empresas que utilizam a *web* para oferecer os seus serviços. Este crescimento fez surgir a necessidade de localizar estes serviços para os usuários finais da rede de uma forma rápida e precisa.

No entanto, as principais ferramentas de busca disponibilizadas para esta localização, correspondentes às máquinas de busca tradicionais, como *Google* (PAGE & BRIN, 1998) e Alta Vista, e os serviços de diretório UDDI, se mostraram ineficientes para a recuperação destes serviços. As máquinas de busca tradicionais se tornaram ineficientes porque suas buscas são baseadas apenas em palavras-chaves, o que torna difícil tanto a descrição quanto a recuperação de serviços. Alguns pesquisadores da área, como KLEIN & BERNSTEIN (2001) e DAVIES et al (2003), já destacavam esta deficiência em seus trabalhos. Já os serviços de diretório UDDI se mostraram deficientes porque os mesmos armazenam poucas informações sobre cada serviço registrado (nome, descrição textual, classificação, etc). Estas informações são insuficientes para que um sistema de busca possa recuperar serviços para os usuários de uma forma precisa.

Para superar estas deficiências, os pesquisadores da comunidade de recuperação da informação combinaram a tecnologia de *web services* com a tecnologia da *web* semântica (HENDLER et al, 2002), que foi idealizada por Tim Berners-Lee como uma extensão da *web* atual, onde os seus recursos são anotados com meta-dados que descrevem a sua semântica, tornando-a compreensível por homens e máquinas. Através dela pode-se criar relacionamentos entre os dados disponibilizados na rede e melhorar significativamente a localização de recursos, além de permitir a automatização de uma grande quantidade de serviços. A combinação entre a tecnologia de *web services* e a tecnologia de *web* semântica foi chamada de *web services* semânticos (McILRAITH et al, 2001). A tecnologia de *web services* semânticos tem conquistado uma grande aceitação entre os pesquisadores da área de recuperação da informação, e vem sendo aplicada em diversas áreas de pesquisa, como aplicações de turismo (DOGAC et al, 2004), médicas (LEE & PATEL, 2004), aplicações

cientes de contexto (VUKOVIC & ROBINSON, 2004) e grids computacionais (FOSTER et al, 2003).

Uma das características mais importantes para a implantação da *web* semântica corresponde ao uso de ontologias (FENSEL, 2004). A palavra ontologia vem da filosofia e representa a ciência de descrever as entidades do mundo e seus relacionamentos. Trazendo este termo para a forma como ele é aplicado na *web* semântica, podemos dizer que uma ontologia define os termos utilizados para descrever e representar uma determinada área de conhecimento. Assim, podemos ter ontologias para vários domínios diferentes, como a medicina, a física, a indústria automobilística, etc. As informações podem ser utilizadas por pessoas, bases de dados e aplicações que precisam compartilhar informações. Aplicações e agentes de *software* podem aplicar raciocínio sobre uma ontologia para inferir informações sobre o domínio que ela descreve, podendo assim, realizar buscas mais precisas e automatizar uma grande quantidade de tarefas.

As ontologias que descrevem uma área de conhecimento são chamadas de ontologias de domínio. Elas são responsáveis por descrever o domínio de aplicação do serviço. Para fazer a ligação entre um serviço e estas ontologias, foi desenvolvido um novo tipo de ontologia, chamado de ontologia de serviços. Este tipo de ontologia permite que as informações que descrevem a funcionalidade do serviço, como os seus parâmetros de entrada e saída, pré-condições e efeitos, sejam ligados a conceitos definidos em alguma ontologia de domínio.

A definição de *web services* semânticos permitiu o desenvolvimento de máquinas de busca mais poderosas, permitindo o desenvolvimento de ferramentas de busca mais precisas para a localização de serviços. Contudo, um novo problema foi encontrado com relação à descoberta de serviços. Este problema acontece quando uma determinada requisição do usuário não pode ser totalmente atendida por um único serviço, mas pode ser atendida se dois ou mais serviços cadastrados forem combinados. Este tipo de situação é chamado de composição de serviços. Uma composição de serviços pode ser definida tanto em tempo de desenvolvimento, pelo programador da aplicação, quanto em tempo de execução, pela ferramenta de busca. Este segundo tipo é chamado de composição automática de serviços.

Nos últimos anos, vários trabalhos foram propostos para resolver o problema da descoberta automática de composições de serviços. Alguns trabalhos, como os desenvolvidos por MILANOVIC & MALEK (2004) e por SRIVASTAVA & KOEHLER (2003), mostram uma discussão sobre as principais abordagens desenvolvidas. Várias destas soluções propõem algoritmos que permitem a descoberta destas composições baseadas nas características semânticas de cada serviço, principalmente àquelas ligadas as suas entradas, saídas, pré-condições e efeitos.

Um problema importante associado à descoberta de serviços e composições diz respeito a como selecionar para o usuário, dentre vários serviços recuperados, o serviço mais relevante para o mesmo. Para fazer esta seleção, a maior parte dos trabalhos propostos utiliza algumas propriedades não funcionais destes serviços, como atraso, confiança e custo. Alguns trabalhos propostos, como o de LUDWING (2003) e o de MENASCÉ (2004), mostram a exploração destas propriedades por ferramentas de localização de serviços.

No entanto, uma característica pouco explorada nestes trabalhos, e que nós consideramos muito importante, é a possibilidade de o usuário selecionar serviços de acordo com as características pertinentes ao seu provedor. Neste caso, não estamos nos referindo às propriedades não funcionais citadas acima, mas às características intrínsecas ao tipo de empresa que o provedor representa e ao domínio de aplicação no qual ele está inserido. Acreditamos que, em vários tipos de serviços, estas características são mais importantes para o usuário do que as propriedades não funcionais destacadas acima.

Por exemplo, se o usuário está procurando por um serviço de reserva de hotel, e a máquina de busca precisa escolher entre vários hotéis selecionados, o usuário pode preferir aquele que tenha classificação cinco estrelas ou o que tenha quadra de tênis, mesmo que o custo para acessar estes serviços seja superior. Ou, no caso de um serviço de compra de produtos, o usuário pode preferir aquele onde ele pode parcelar o pagamento ou o que tenha um prazo de entrega menor.

Estas características, embora sejam importantes e muito comuns na hora de se procurar um serviço, pouco foram exploradas até agora pelas ferramentas de localização e seleção de serviços. Uma característica importante que mostra a pouca atenção que os trabalhos relacionados têm direcionado a este tema é que nenhuma das principais

tecnologias propostas para a descrição semântica de serviços oferece suporte à descrição deste tipo de informação.

Esta dissertação descreve WebS Composer, uma ferramenta que usa marcações semânticas para permitir o registro, a descoberta de serviços e a execução de serviços na *web*. Além disso, ela permite que o usuário especifique restrições quanto ao provedor do serviço, oferecendo assim buscas precisas para os usuários.

Este trabalho está dividido em mais quatro capítulos. No capítulo II, são mostrados os principais trabalhos relacionados à descoberta de serviços. Os principais trabalhos abordados se referem à composição automática de *web services*. Para cada trabalho analisado, é feita uma análise crítica, que aborda seus aspectos positivos e negativos e o compara com as características adotada em WebS Composer. O objetivo deste capítulo é mostrar o estado da arte em relação à descoberta e composição automática de serviços e mostrar o relacionamento dos principais trabalhos com a ferramenta descrita por esta dissertação. No terceiro capítulo, é feita a apresentação de WebS Composer, uma nova ferramenta para a descoberta e composição de serviços na *web*. Nele, é oferecida uma descrição detalhada da ferramenta, mostrando, dentre outras coisas, as suas principais características, a sua arquitetura e os detalhes da sua implementação. No capitulo IV, é mostrado um estudo de caso, que mostra a aplicação de WebS Composer a um domínio real de aplicação, mostrando as principais funcionalidades oferecidas pela ferramenta. É mostrada a aplicação da ferramenta a serviços do domínio de viagens. Por fim, o capitulo V mostra as conclusões obtidas ao fim deste trabalho, abordando as suas principais contribuições e as suas limitações. Nesta seção, também são abordados os trabalhos futuros, sugeridos para melhorar a qualidade dos resultados obtidos com o desenvolvimento do trabalho e superar algumas das limitações impostas à versão atual da ferramenta.

CAPÍTULO II – TRABALHOS RELACIONADOS

Neste capítulo, são abordados alguns trabalhos relacionados à WebS Composer. Na primeira seção deste capítulo, são descritos alguns trabalhos que não definem uma metodologia para composição automática de serviços, mas que definem uma estrutura que permite a especificação de composições de serviços. Por fim, são mostrados alguns trabalhos relacionados à composição e seleção automática de *web services* semânticos. Para cada trabalho abordado, são destacados os seus pontos positivos e negativos, que confrontam as idéias utilizadas no desenvolvimento destes trabalhos com as idéias utilizadas no desenvolvimento de WebS Composer. Ao longo deste capítulo, uma maior ênfase é dada aos trabalhos julgados mais importantes, enquanto que os demais trabalhos são mostrados de uma forma mais breve.

2.1 A composição de serviços

Esta seção mostra a análise feita sobre alguns trabalhos desenvolvidos para a especificação de composições de serviços. O problema da composição de serviços já é alvo de pesquisa por parte da comunidade da *web* semântica há algum tempo. Os primeiros trabalhos desenvolvidos nesta área concentram-se na elaboração de modelos que permitem a especificação destas composições.

Desde então, vários trabalhos foram propostos como soluções para este problema, com o uso de diversas abordagens, mas nenhum deles conseguiu se tornar um padrão entre os usuários da área de pesquisa envolvida. Alguns trabalhos publicados fazem uma análise das principais abordagens desenvolvidas, como o trabalho desenvolvido por SRIVASTAVA & KOEHLER (2003) e, mais recentemente, por MILANOVIC & MALEK (2004).

2.1.1 Uma solução baseada em contratos

Ao longo dos anos, várias abordagens foram desenvolvidas para resolver o problema da composição de serviços, baseadas em tecnologias como BPEL (CURBERA et al, 2003), OWL-S (McILRAITH & SON, 2002), Redes de Petri (HAMADI & BENATALLAH, 2003), componentes *web* (YANG & PAPAZOGLOU, 2002), cálculo-Π (MEREDITH & BJORG, 2003), checagem de modelos (FU et al, 2002) e máquinas de estados finitos (BERARDI et al, 2003). Calculo de situação Dentre elas, duas tecnologias ganharam destaque na especificação de composições de serviços: BPEL (JURIC et al, 2006) e OWL-S (MARTIN et al, 2004). BPEL é uma linguagem baseada em XML, desenvolvida pela BEA, IBM, Microsoft, SAP e Siebel para a definição de composições de serviços, enquanto que OWL-S é uma ontologia de serviços que também permite esta especificação. As duas tecnologias ganharam aceitação devido ao seu grande poder de expressividade, com várias construções de controle que permitem a especificação de composições complexas. Devido ao grande poder de expressividade e, consequentemente, do alto grau de complexidade destas tecnologias, é difícil descrever formalismos para verificar a correção das composições descritas nestas tecnologias.

Uma abordagem para a definição de composições de serviços foi desenvolvida por MILANOVIC (2005). Esta abordagem é baseada em contratos e foi proposta como uma tentativa de padronizar a composição de serviços, já que, segundo o autor, os principais padrões "industriais" existentes para a composição de serviços (BPEL e OWL-S) não oferecem meios de verificar a correção de uma composição, enquanto que outras abordagens mais formais, como redes de Petri e máquinas de estados finitos, oferecem esta característica, mas são difíceis de serem aplicadas ao mundo real devido a grandes problemas de escalabilidade.

A descrição de serviços é baseada em estruturas chamadas de contratos, que descrevem o que o serviço faz e quais as condições necessárias para que o mesmo execute de forma correta. Além disso, eles podem oferecer informações sobre as características não funcionais da execução do serviço. A especificação dos contratos é feita através da linguagem CDL.

Embora a linguagem CDL seja usada para a especificação dos contratos, existe uma limitação quando a mesma é utilizada para a representação de composições de serviços. Por ser uma linguagem baseada em XML, ela não oferece meios para verificar a correção destas composições. No entanto, segundo o autor, esta é uma característica muito importante para a definição de composições.

Para superar esta limitação, os serviços foram modelados como máquinas abstratas. Assim, sempre que dois serviços são compostos, suas descrições CDL são mapeadas para máquinas abstratas, e uma terceira máquina é gerada para representar a composição. Depois que a correção da composição gerada é verificada, a máquina que representa a composição é mapeada para CDL e armazenada.

A representação de serviços como máquinas abstratas permite a anotação de muitas características dos mesmos. Dentre estas características podemos citar:

- Restrições, que descrevem as condições a serem mantidas pelos parâmetros de entrada;
- Conjuntos, que representam os conjuntos de dados usados pela máquina;
- Invariantes, que correspondem às características que devem ser mantidas antes e após a execução do serviço;
- Propriedades, que mostram as invariantes envolvendo conjuntos e constantes;
- Asserções, que devem ser dedutíveis a partir das propriedades da máquina; e
- Operações, que correspondem às operações permitidas pela máquina, com suas pré e pós-condições.

Estas características são representadas através de predicados lógicos, e o método da substituição é usado para testar e validar estes predicados.

A composição de serviços, formada a partir de máquinas abstratas, pode ser construída através de quatro operadores. A máquina gerada para a composição apresenta todas as características presentes na definição dos serviços que as constituem. O valor de cada propriedade é gerado a partir dos valores definidos para cada serviço, e varia de acordo com o tipo de composição gerada:

- ***Sequence:*** define composições onde os serviços devem ser executados em uma ordem pré-definida;
- ***Parallel:*** define composições onde os serviços podem ser executados concorrentemente. Neste tipo de composição, os serviços podem executar com ou sem comunicação entre eles;
- ***Choice:*** define composições onde um serviço é escolhido para ser executado dentre vários serviços similares. Esta escolha acontece de forma não determinística;
- ***Looping:*** define composições onde um ou mais serviços são executados várias vezes enquanto que uma determinada condição é satisfeita. Esta condição é verificada através do estado de uma das variáveis do serviço.

A realização de uma composição ocorre em de quatro etapas. A primeira etapa corresponde à especificação da composição de dois ou mais serviços através de uma máquina abstrata. A segunda etapa corresponde à verificação de tipos da composição, onde a ferramenta verifica a compatibilidade dos tipos de entrada e saída de cada serviço integrante da composição.

A terceira etapa consiste em verificar a corretude da máquina abstrata gerada para representar a composição. A máquina abstrata é considerada correta se três condições forem satisfeitas:

- A inicialização da composição deve garantir as suas invariantes;
- As asserções da composição devem ser dedutíveis a partir das propriedades e invariantes da composição;
- A execução da composição deve garantir as invariantes da mesma;

A última etapa consiste em verificar se a composição tem uma terminação correta. Nesta etapa, verifica-se, através do método da substituição, se a execução dos serviços que compõem a composição vai manter as pós-condições da mesma.

A ferramenta mostrada descreve uma abordagem para a especificação de composições de serviços. No entanto, ela não mostra uma solução para o problema da composição automática de serviços, que corresponde ao problema a que nos propusemos resolver. A anotação de serviços baseada em contratos tem uma característica interessante,

que é anotação de características não funcionais do serviço, como atraso, tempo de resposta, etc. Contudo, ela não fornece meios para a marcação do provedor de serviços, o que nós consideramos importante para a descoberta de serviços mais complexos baseados em restrições do usuário.

2.1.2 IRS-III

O *framework* IRS-III (DOMINGUE et al, 2005) foi desenvolvido como uma solução no suporte à criação de *web services* semânticos baseados em ontologias WSMO (ROMAN et al, 2005). Ele oferece uma estrutura que permite que o usuário crie *web services* com facilidade, a partir de um código java ou lisp, além da publicação, seleção, composição e invocação de serviços.

Para realizar buscas mais eficientes, a ferramenta usa uma anotação semântica para todos os serviços cadastrados em sua base de dados. Esta marcação é baseada numa ontologia que estende a ontologia WSMO, acrescentando algumas características à mesma, como, por exemplo, um mediador que liga serviços a objetivos. A ferramenta também oferece uma API escrita em java que implementa a ontologia WSMO.

A estrutura da ontologia utilizada é baseada em quatro elementos: os serviços, que representam os *web services* cadastrados no sistema, os objetivos, que representam as tarefas que o usuário espera que sejam oferecidas pelo serviço, as ontologias, utilizadas para fazer a anotação semântica dos serviços e dos objetivos cadastrados, e os mediadores, que resolvem questões relacionadas à interoperabilidade de serviços. Todos os serviços são agrupados em objetivos.

Sempre que um novo serviço é cadastrado, todas as suas entradas e saídas são ligadas a conceitos definidos em alguma ontologia da ferramenta. Este tipo de informação é chamada de tipo semântico. Além dessa informação, são cadastradas as informações de ligação para cada entrada e saída. Todos os serviços cadastrados são ligados a um objetivo pré-definido através de um mediador, que transforma o formato das entradas de um objetivo do usuário para um formato aceitável pelo serviço. Após a invocação, outro

mediador é usado para mapear o formato resultante da execução do serviço para o formato do objetivo do usuário.

Além disso, o serviço cadastrado é associado a uma interface que contém informações sobre a orquestração e a coreografia do serviço. A primeira característica especifica o fluxo de dados e as estruturas de controle do serviço, caso o mesmo represente uma composição de serviços. A segunda característica define como ocorre a comunicação com o serviço. Em IRS-III, quando um serviço é publicado, todas as informações de acesso ao mesmo, como o endereço e a porta de acesso, são armazenadas nessa coreografia.

Uma característica interessante da ferramenta é que a consulta é realizada sobre os objetivos. Para executar um serviço, o usuário pode, através de um *browser* oferecido pela ferramenta, selecionar diretamente um determinado serviço, ou selecionar um objetivo, que corresponde a uma tarefa pré-definida pela ferramenta. Quando o usuário seleciona um determinado objetivo ou serviço, a ferramenta mostra todas as entradas e saídas relacionadas ao mesmo. Se o usuário pede para invocar um objetivo, a ferramenta automaticamente seleciona o serviço que está ligado ao objetivo desejado. Se mais de um serviço estiver ligado ao objetivo, a ferramenta escolhe um destes serviços aleatoriamente e o invoca.

O usuário pode ainda selecionar objetivos que satisfaçam a uma determinada restrição. O usuário pode, por exemplo, descrever as entrada e saídas que ele deseja para o objetivo, e a ferramenta recupera todos os objetivos que satisfazem a essa restrição. Mostrado o resultado, o usuário pode selecionar um dos objetivos recuperados para a invocação.

Para procurar por serviços, o usuário define as entradas e saídas que ele deseja para o seu objetivo. Todas estas entradas e saídas são passadas através do tipo semântico, ou seja, o tipo definido em alguma ontologia. Quando o usuário entra com os dados e submete a consulta, o IRS-III recupera todos os objetivos que satisfazem à sua requisição.

O usuário seleciona então um dos objetivos selecionados. Quando isto acontece, o IRS-III seleciona um serviço que esteja associado ao objetivo. Se o objetivo tiver mais de um serviço associado a ele, um serviço é escolhido aleatoriamente. Selecionado o serviço, o usuário entra com os dados de entrada e o mesmo é invocado, através do mediador associado entre o objetivo e o serviço.

De uma forma geral, a ferramenta oferece características muito interessantes que a tornam atraente para os usuários da comunidade da w*eb* semântica. Dentre estas características, podemos citar a facilidade para a criação de novos serviços, a geração automática da marcação semântica e o servidor que permite o armazenamento e recuperação dos elementos da ontologia WSMO. Além disso, a estrutura implementada em IRS-II já permite o tratamento de pré-condições e efeitos, que enriquecem consideravelmente as buscas da ferramenta. Outra característica importante é a definição dos mediadores, principalmente para a invocação de serviços, que amenizam os efeitos da heterogeneidade dos serviços existentes.

Contudo, também notamos que a ferramenta falha em pontos que achamos importantes para a descoberta de serviços na *Web*. O primeiro ponto diz respeito a descoberta automática de composições. Sabemos que, com freqüência, os usuários podem desejar serviços (ou objetivos, como são abordados na ferramenta) que não são atendidos por um único serviço ou por um único objetivo. Neste tipo de situação, temos a necessidade de que a ferramenta de busca saiba, sem a interferência humana, como juntar serviços para formar composições em tempo de execução. Na abordagem da ferramenta, isto deveria ser oferecido através da composição de objetivos para a formação de objetivos mais complexos. Algumas ferramentas já começaram a utilizar a estrutura desenvolvida pela ferramenta para a montagem interativa de composições de serviços. Falaremos um pouco mais desta ferramenta na seção seguinte.

Outro ponto que destacamos é o fato de apenas os serviços serem anotados semanticamente. Este tipo de anotação permite facilmente a descoberta de serviços que ofereçam uma determinada funcionalidade, mas não permite que o usuário filtre os provedores de serviços de acordo com as suas restrições. Por exemplo, não adianta a ferramenta encontrar uma companhia aérea que ofereça um vôo entre duas cidades desejadas pelo usuário, se a mesma não aceita o cartão de crédito usado pelo mesmo. Nós acreditamos que esse tipo de restrição poderia ser facilmente resolvida se, além do serviço, o provedor que oferece esse serviço também fosse anotado semanticamente de acordo com uma ontologia.

Desta forma, achamos que a ferramenta IRS-III é uma ferramenta de grande valia, pois oferece muitos serviços relativos à criação, publicação, invocação e seleção de

serviços, e facilita o rápido desenvolvimento de aplicações relacionadas a *web services* semânticos. Mesmo assim, concluímos que ela, em sua versão atual, não resolve o problema que nos propusemos a resolver em nosso projeto, devido às razoes mostradas acima.

Uma ferramenta baseada em IRS-III para composição interativa de serviços foi desenvolvida por SELL et al. (2004). Neste trabalho, os autores definem uma ferramenta gráfica em Java que permite que um usuário monte uma composição de serviços de forma interativa, definindo em cada passo qual o novo objetivo a ser adicionado à composição.

A composição é definida de uma forma bastante simples. Primeiro, o usuário seleciona qual o seu primeiro objetivo. Para que o usuário faca essa seleção, é exibida uma lista contendo todos os objetivos cadastrados no servidor IRS-III. O usuário pode também selecionar vários objetivos que satisfaçam um determinado critério de seleção e, depois, escolher um dos serviços encontrados.

Nos passos subseqüentes, o usuário seleciona e adiciona objetivos cuja entrada possa receber o resultado do objetivo anterior, ou um de seus elementos. O usuário pode conectar a saída de um objetivo a vários outros, desde que a entrada destes serviços seja compatível com o resultado do objetivo anterior.

Uma característica importante da abordagem usada por esta ferramenta é que o usuário pode selecionar mediadores para transformar dados quando a saída do serviço anterior não for compatível com a entrada do serviço subseqüente. Isto é interessante porque amplia o número de objetivos que podem ser adicionados à composição do usuário, que pode também usar estruturas de controle como *if-then-else*, que permitem a definição de composição de serviços mais complexas.

Uma vez definida a composição, o usuário pode executá-la. Esta execução é feita através de uma API java desenvolvida para a orquestração de serviços. Esta API instancia os serviços, os mediadores e as estruturas definidas na composição e invoca os objetivos definidos na composição. Além disso, a composição gerada pode ser armazenada no servidor IRS, e, posteriormente, pode ser recuperada e alterada.

Esta ferramenta complementa a estrutura definida pelo IRS-III, permitindo que o usuário agrupe objetivos para a formação de objetivos maiores, através de composições. No entanto, toda a montagem da composição é feita manualmente pelo usuário, que tem que

selecionar, em cada passo, manualmente, os objetivos que devem ser adicionados à composição. Existem situações em que a composição de serviços deve ser montada não pela vontade explícita do usuário, mas sim pela falta de existência de um serviço que atenda totalmente à requisição solicitada por ele. Estas composições devem ser montadas de forma totalmente transparente para o usuário, ou seja, a requisição deve ser resolvida sem que o usuário saiba que uma composição de serviços foi montada.

Além do mais, WSMO é uma linguagem complexa, tornando o seu uso difícil para usuários que não tenham um grande conhecimento na área de informática. Uma ferramenta de busca deste tipo deve ter uma interface que interaja de forma simples com o usuário, permitindo o seu uso por qualquer pessoa que usa uma máquina de busca nos dias atuais.

2.2 A descoberta e seleção de composição de serviços

Nesta seção, são abordados alguns trabalhos relacionados à descoberta e seleção de serviços e de composições de serviços. Nos últimos anos, várias soluções foram propostas para resolver o problema da seleção de *web services* semânticos. Assim como as soluções propostas para a especificação de composições de serviços, nenhuma destas soluções conseguiu se tornar um padrão na solução deste problema. Apesar de muitos trabalhos interessantes terem sido propostos, acreditamos que ainda existem algumas questões importantes ainda não totalmente resolvidas. Nas próximas subseções, são mostrados alguns dos trabalhos que se destacam na tentativa de resolver este problema. Para cada trabalho apresentado, são destacadas as suas características mais fortes e as questões que julgamos não resolvidas ou resolvidas de uma maneira insatisfatória.

2.2.1 Compat

Uma abordagem para descoberta de serviços anotados semanticamente é mostrada por BIANCHINI et al. (2005). Eles mostram o desenvolvimento de uma ferramenta chamada COMPAT (COMPATibility), que permite, entre outras coisas, a publicação e a seleção de serviços.

A ferramenta utiliza marcações semânticas baseadas em ontologias. Para isso, os autores usam dois tipos de ontologia. O primeiro tipo corresponde às ontologias de domínio, chamadas de *DomOnt*. Elas descrevem os conceitos aos quais são associadas às entradas e saídas dos serviços, o nome do serviço e as categorias de serviços existentes. Nestas ontologias, os conceitos são relacionados através de três relacionamentos semânticos: generalização, equivalência e disjunção.

O segundo tipo corresponde às ontologias de serviço, que são usadas para a descoberta e seleção de serviços. Elas são organizadas hierarquicamente em três camadas: serviços concretos, serviços abstratos e categorias. Os serviços concretos podem ser executados diretamente através de sua interface WSDL (CHRISTENSEN et al, 2001). Eles são agrupados em clusters compostos por serviços similares. Esta similaridade é calculada através da avaliação do nome e dos parâmetros de entrada e saída de cada serviço. Estes clusters, por sua vez, são associados a algum serviço abstrato, que corresponde a um serviço que não pode ser invocado diretamente e é usado para representar a funcionalidade de um conjunto de serviços. Um serviço abstrato pode ser relacionado a outros serviços abstratos através de dois relacionamentos semânticos: generalização e composição. As categorias são usadas como uma taxonomia que permite a classificação dos serviços abstratos. A Figura 1, extraída do trabalho de BIANCHINI et al (2005), mostra um exemplo de uma taxonomia de serviços usada pelo Compat. Esta taxonomia representa serviços associados ao domínio de turismo. Todos os serviços representados são empresas de vôos fictícias.

Na anotação semântica, todos os serviços são representados em função de suas entradas, saídas, nome de sua operação e categorias. Todas estas características são representadas como construções escritas em *Description Logic* (BAADER et al, 2002). Para a realização das buscas, a ferramenta usa um algoritmo de casamento semântico entre serviços e requisições definidas pelo usuário. A requisição do usuário é representada internamente como um serviço, onde a categoria pode ser omitida. Dada uma requisição e um serviço abstrato, podem existir entre os mesmos os seguintes tipos de relacionamento:

- **Pré-filtragem:** se uma categoria for definida na requisição do usuário, um processo de pré-filtragem é realizado sobre os serviços cadastrados na base

de dados. Nesta etapa, são selecionados apenas os serviços que possuem uma categoria igual ou relacionada à categoria definida na requisição;

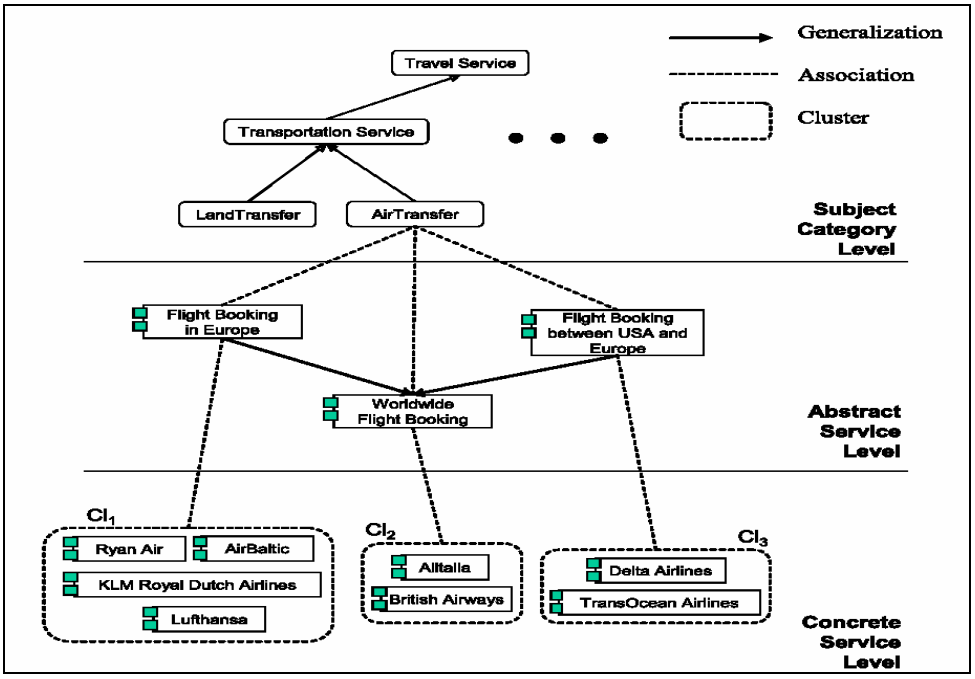


Figura 1: Taxonomia de serviços do Compat

- **Exato:** um serviço é dito ser exatamente igual a uma requisição quando ambos têm o mesmo nome de operação, e os parâmetros de entrada e de saída dos mesmos são similares;
- ***Plug-in:*** este tipo de relacionamento existe quando o serviço oferece pelo menos as capacidades definidas na requisição, ou seja, quando a operação definida na requisição pode ser mapeada para operações definidas no serviço;
- ***Subsume:*** este tipo de relacionamento ocorre quando o serviço oferece apenas algumas das capacidades desejadas pela requisição;
- **Interseção:** este tipo de relacionamento acontece quando o serviço e a requisição têm algumas operações ou parâmetros de entrada e saída em comum.

- **Incompatível:** ocorre quando nenhum dos relacionamentos acima pode ser aplicado entre a requisição e o serviço.

Cada tipo de relacionamento é representado internamente como um número. Esta informação é usada para definir um ranking entre os serviços recuperados. Os relacionamentos mais fortes são, em ordem decrescente, exato, *plug-in*, *subsume*, interseção e incompatível. Apenas os serviços abstratos cujo grau de similaridade for superior a um determinado *threshold* são recuperados como serviços candidatos. Os serviços são ordenados em ordem decrescente de similaridade.

Depois que o sistema verifica que um serviço abstrato Si é um serviço candidato, ele usa a organização ontológica dos serviços abstratos para acelerar a verificação dos serviços abstratos relacionados a Si, sem a necessidade de aplicar o algoritmo de verificação de similaridade mostrado acima. Por exemplo, se um serviço abstrato Sj corresponde a uma generalização de Si, o sistema pode tomar várias ações.

Primeiro, se Si tem um relacionamento exato com a requisição do usuário, o serviço Sj pode ter um relacionamento *plug-in* com a requisição, desde que o mesmo ofereça pelo menos as funcionalidades oferecidas pelo serviço Si. Quando o serviço Si tem um relacionamento do tipo *plug-in* com a requisição do usuário, então o serviço Sj tem o mesmo tipo de relacionamento com a mesma. Se o relacionamento de Si com a requisição for do tipo *subsume*, então Sj pode ter um relacionamento do tipo *subsume*, *plug-in* ou *exato* com a mesma. Neste caso, o algoritmo de verificação de similaridade deve ser executado novamente. Quando Si e a requisição apresentam um relacionamento do tipo interseção, o relacionamento entre Sj e a requisição do usuário pode ser do tipo *subsume*, *plug-in*, exato ou interseção. Neste caso, o algoritmo de verificação de similaridade deve ser aplicado entre Sj e a requisição. Se o relacionamento entre Si e a requisição for do tipo incompatível, então o relacionamento entre Sj e a requisição também é incompatível. Neste caso, não há a necessidade da reaplicação do algoritmo de verificação de similaridade.

Uma vez que os serviços candidatos são selecionados, o sistema pode selecionar alguns dos serviços concretos ligados ao serviço abstrato. Nesta etapa, pode haver um novo refinamento baseado em algumas características, como requisitos de qualidade, disponibilidade, etc.

A principal contribuição do Compat é o desenvolvimento de um *framework* que permite que usuários e aplicações possam registrar e recuperar serviços. A anotação semântica permite uma maior precisão na recuperação de serviços. O agrupamento de serviços concretos similares em clusters oferecem uma certa indexação para os mesmos, já que, no lugar de comparar uma requisição do usuário com cada serviço concreto cadastrado, a mesma é comparada apenas aos serviços abstratos. Uma vez que um serviço abstrato similar é encontrado, podemos recuperar os serviços concretos associados a ele. Esta "indexação" garante um melhor desempenho ao algoritmo de busca. No entanto, achamos que isto impõe certas limitações ao processo de descoberta e seleção de serviços. Como os serviços oferecidos na *Web* têm como uma de suas principais características a heterogeneidade, acreditamos que mesmo os serviços similares, que desempenham basicamente a mesma tarefa, dificilmente serão exatamente iguais, com, rigorosamente, os mesmos parâmetros de entrada e saída. Desta forma, acreditamos que há uma grande tendência dos parâmetros do serviço concreto diferir um pouco dos parâmetros do cluster. Por exemplo, um serviço concreto pode oferecer algumas saídas a mais do que as oferecidas no serviço abstrato. Como numa consulta, a requisição é casada com os serviços abstratos, esses parâmetros de saída a mais que são oferecidos pelo serviço serão perdidos durante o processo de busca. Esta perda pode fazer a diferença na hora da descoberta de serviços e, principalmente, de composições.

Outra contribuição do Compat é a apresentação de um algoritmo de verificação de similaridade que contém vários níveis de relacionamento. Isto proporciona uma maior flexibilidade e a recuperação de serviços com casamento parcial, que achamos indispensável para um algoritmo de busca. No entanto, o algoritmo mostrado pela ferramenta tem uma limitação. O algoritmo de verificação de similaridade é aplicado para o serviço como um todo. Devido à heterogeneidade dos serviços publicados, poucos serviços podem casar totalmente com a requisição do usuário, e assim, a maioria dos serviços pode apresentar um relacionamento do tipo interseção, mesmo contend0o a maior parte das capacidades definidas na requisição. Como este tipo de relacionamento é um dos mais baixos considerados pela ferramenta, muitos serviços relevantes podem não ser recuperados. Isto também impõe limitação para a descoberta de composições paralelas, onde cada serviço desempenha um subconjunto das tarefas solicitadas pelo usuário. No

lugar de uma comparação feita diretamente para o serviço como um todo, uma comparação feita para cada conceito definido nos conjuntos de entrada e saída de cada serviço, com um nível de similaridade sendo calculado posteriormente com base nestes valores, poderia facilmente resolver estas limitações.

2.2.2 O trabalho de Kvaloy

Outra abordagem para resolver o problema de composição automática de serviços foi desenvolvida por KVALOY et al. (2005). Os autores desenvolveram um algoritmo para a descoberta automática de *workflows*. O sistema desenvolvido foi testado com serviços na área de aplicações biomédicas.

Neste trabalho, cada serviço é anotado semanticamente através do uso de ontologias de domínio e ontologias de serviço. As ontologias de domínio foram descritas através de *Description Logic*. As anotações semânticas de serviços são especificadas através de descrições OWL-S. Cada parâmetro de entrada e de saída do serviço é associado a um conceito definido na ontologia de domínio.

O casamento de conceitos é feito seguindo o algoritmo definido por PAOLUCCI et al. (2002). Ele usa dois tipos de níveis de similaridade definidos no citado algoritmo. O primeiro deles corresponde ao nível de similaridade exato, que ocorre quando dois conceitos são similares. O segundo nível de similaridade usado é chamado de *plug-in*, que ocorre quando o serviço oferece, além da funcionalidade desejada pela requisição do usuário, outras funcionalidades, ou seja, quando a funcionalidade definida pela requisição pode ser *"plugada"* na funcionalidade oferecida pelo serviço. O relacionamento do tipo *plug-in* é mais fraco do que o relacionamento exato, pois, como oferece funcionalidades indesejadas pelo usuário, pode apresentar um tempo de resposta maior do que um serviço com nível de similaridade exato.

A composição de serviços é descoberta através de um algoritmo bastante simples. Primeiro, o *matchmaker* tenta encontrar todos os serviços cuja entrada é similar à entrada definida na requisição do usuário. Depois, de forma progressiva, para cada serviço selecionado, adiciona novos serviços cuja entrada seja similar à entrada do último serviço do *workflow*. Todo este procedimento gera um grafo onde cada ramo representa um novo

workflow. O processo é repetido até que nenhum casamento possa ser encontrado, ou que as saídas do último serviço sejam idênticas às saídas definidas pelo usuário em sua requisição de serviço.

O algoritmo acima pode encontrar várias composições de serviços que satisfaçam à requisição do usuário. Assim, a máquina de busca deve selecionar uma destas composições para exibi-la para o usuário final. Para isso, foi desenvolvido um algoritmo que avalia cada composição gerada, associando um peso para cada uma, para que as mesmas possam ser comparadas e selecionadas. O algoritmo é baseado no tipo de relacionamento encontrado em cada composição, e leva em consideração as seguintes observações:

- As composições com o menor número de serviços devem ter prioridade, pois podem reduzir o *overhead* gasto com a comunicação entre os serviços encontrados;
- Composições com relacionamento exato devem ter prioridade sobre as composições com relacionamento do tipo *plug-in*, devido às limitações de performance já citadas;
- Caso haja a necessidade de selecionar uma composição com o nível de relacionamento do tipo *plug-in*, é preferível selecionar aquelas em que o relacionamento ocorre no meio ou no fim da composição. Composições que apresentam um relacionamento deste tipo no início devem ser evitadas, pois é preferível que o *workflow* comece a ser executado com entradas compatíveis com as entradas definidas pelo usuário em sua requisição de serviço;

A equação usada para calcular o custo de uma composição de serviços é definida da seguinte forma:

$$Custo = entradas_1(relacionamento) + \sum_{n=1}^{N} saídas_n(relacionamento)$$

A função *entradas*, que recebe como entrada um determinado relacionamento, retorna o valor 1, caso o relacionamento seja do tipo exato, ou 2, caso o relacionamento seja do tipo *plug-in*. Já a função *saídas*, que também tem como entrada um tipo de

relacionamento, retorna o valor 1, quando o relacionamento é do tipo exato, e 1.5, quando o mesmo é do tipo *plug-in*. A composição que obtiver o menor custo calculado pela equação acima é selecionada e mostrada para o usuário.

As principais contribuições deste trabalho são a proposição de um algoritmo capaz de descobrir composições automáticas de serviços e o desenvolvimento de uma equação capaz de calcular os custos das composições descobertas. O algoritmo mostrado para a seleção de serviços consegue descobrir composições seqüenciais, mas não leva em consideração o fato de que, muitas vezes, composições precisam ser montadas não de forma seqüencial, mas sim, em paralelo. Este tipo de situação pode acontecer quando a requisição do usuário pode ser dividida em subtarefas que podem ser paralelizadas e executadas de forma independente. Como este tipo de composição é indispensável para trabalhos relacionados à descoberta automática de serviços, julgamos que a ausência desta característica representa uma séria limitação do trabalho proposto. Além disso, o trabalho não considera o problema da restrição de provedores, que também julgamos uma característica importante para algoritmos de seleção e composição de serviços. Desta forma, embora o trabalho proposto seja muito interessante, ele não resolve completamente o problema da composição automática de serviços.

2.2.3 A proposta de Aversano para a descoberta de composições

Uma abordagem baseada em ontologias para a descoberta de serviços e composições de serviços na *web* foi proposta AVERSANO et al (2004). Eles especificaram um algoritmo usado para a descoberta de composições.

Nesta abordagem, as ontologias de domínio são especificadas através da linguagem DAML. A marcação semântica de cada serviço é feita através de uma ontologia DAML-S (ANKOLEKAR et al, 2001), através de uma estrutura chamada *Service Profile*. Nesta estrutura, todos os parâmetros de entrada e saída do serviço são associados a algum conceito definido numa ontologia de domínio. Além dessa informação, são anotadas algumas informações sobre as características não funcionais do serviço, como o custo, a confiabilidade, tempo de resposta, etc. A busca por serviços é baseada em duas medidas: a

similaridade entre o serviço recuperado e a requisição do usuário e a qualidade do serviço recuperado.

Quando um usuário entra com uma nova requisição, a mesma é representada como um objetivo. A máquina de busca tenta encontrar um único serviço que atenda às necessidades do usuário, o que acontece se o mesmo tiver um nível de similaridade mínimo com o objetivo do usuário. Caso não exista nenhum serviço que atenda às suas necessidades, a máquina tenta encontrar uma composição de serviços para realizar a tarefa.

O algoritmo de descoberta de composições de serviços, que tem como entrada um objetivo consistindo das saídas da requisição do usuário, é dividido em três fases. Na primeira, são selecionados todos os serviços cuja saída é suficientemente similar à requisição. Esses serviços representam as folhas da árvore que representa a composição de serviços. Existem ainda casos onde mais de um serviço pode ser agrupado para atender este objetivo. Por exemplo, se o objetivo possui três saídas, podemos ter três serviços, um associado a cada saída.

Na segunda etapa, o algoritmo visita cada folha gerada na etapa anterior, e, para cada uma, cria um novo objetivo correspondendo às entradas do serviço associado à folha. Para cada folha, o algoritmo tenta encontrar um serviço cuja saída seja similar a este objetivo, desde que a entrada do objetivo não seja oferecida por nenhum dos serviços presentes na árvore.

Por fim, este procedimento é repetido até que o algoritmo encontre um caminho entre o objetivo e as folhas, ou até que um *time out* seja atingido. A melhor composição encontrada é exibida para o usuário. Se ela não for selecionada, a segunda melhor composição será proposta, e assim sucessivamente.

O algoritmo descrito acima permite a descoberta de dois tipos de composições de serviços. A primeira delas é chamada de composição horizontal e acontece quando os serviços compostos são executados em paralelo. A segunda é chamada de composição vertical, onde os serviços compostos são executados em seqüência.

Para efetuar o cálculo da similaridade entre serviços, um algoritmo de casamento de entidades é utilizado. O algoritmo permite verificar a similaridade entre conceitos definidos em ontologias diferentes. Esta similaridade é verificada através de três medidas. A primeira medida corresponde à similaridade entre os nomes dos conceitos. O *WordNet* é

utilizado para a verificação de sinônimos. A segunda medida é calculada através da similaridade das propriedades associadas a cada conceito, enquanto que a terceira medida é calculada através do relacionamento semântico existente entre os dois serviços (herança, disjunção, equivalência, etc.). O cálculo da similaridade entre os dois conceitos é realizado através da soma de cada uma destas medidas de similaridade multiplicadas por seu respectivo peso. Um exemplo de composição descoberta através deste algoritmo é mostrado na Figura 2, extraída da mesma fonte citada acima (AVERSANO et al, 2004).

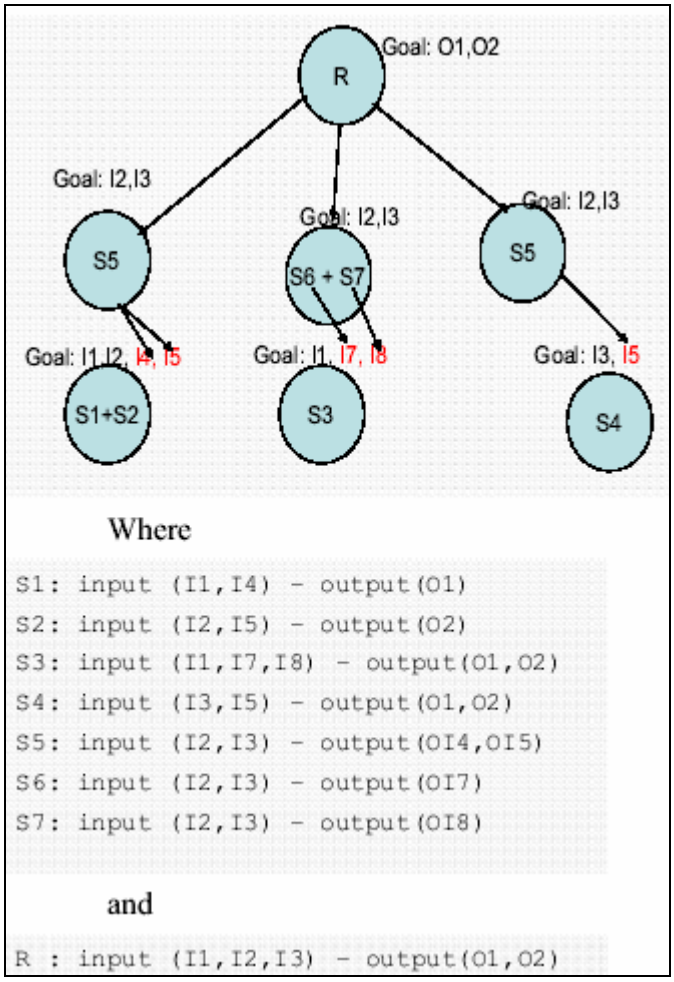


Figura 2: Exemplo de composição de serviços descoberta

Uma vez que os serviços e composições são descobertos, o algoritmo usa ainda meta valores para selecionar o melhor serviço para o usuário. Para isso, foi definido um conjunto de qualidades para o serviço, como disponibilidade, tempo de resposta, custo, etc.

Para cada uma destas propriedades é atribuído um peso, que vai depender do perfil do usuário. Numa composição de serviços, cada atributo deve ser composto de forma adequada, de acordo com o tipo de composição gerada. Por exemplo, o atributo *custo* deve ser somado em ambos os tipos de composição. Já o atributo *tempo de resposta* corresponde ao maior tempo, em uma composição horizontal, e à soma dos tempos, em uma composição vertical. Essas informações também são anotadas no *Service Profile*.

O algoritmo mostrado por este trabalho permite a descoberta de serviços e composições de serviços. Ele apresenta uma flexibilidade, que é o alinhamento de ontologias em tempo de execução. Para isso, sempre que dois serviços são comparados, um mapeamento deve ser feito durante a análise de cada parâmetro de entrada e saída do serviço comparado e do objetivo. Em nossa opinião, esta característica pode impor ao algoritmo um grande problema de desempenho, considerando que, para a descoberta de composições, o algoritmo pode percorrer várias vezes todos os serviços da base de dados, e que o número de parâmetros de entrada e saída de serviços tende a ser grandes. O resultado ainda piora se considerarmos que todos os serviços são descritos como *profiles* da ontologia DAML-S, e que a comparação de arquivos baseados na linguagem XML é uma operação custosa.

Além disso, o algoritmo não permite que o usuário imponha restrições sobre o provedor do serviço. Nós acreditamos que esta característica é muito importante e, algumas vezes, mais importante do que algumas medidas usadas pelo algoritmo para medir a qualidade de um serviço. Por exemplo, se um usuário está tentando reservar um quarto de hotel em uma cidade, pode ser muito mais importante para ele que o hotel tenha piscina e aceite um determinado cartão de crédito do que o tempo que o mesmo vai gastar para acessar o serviço do mesmo. Devido a estas limitações, acreditamos que esta solução ainda não resolve totalmente o problema da composição de serviços.

2.2.4 Uma solução baseada em planos de contingência

COSTA et al (2004) desenvolveram uma abordagem para resolver o problema da descoberta de composições automáticas de serviços. A principal característica deste trabalho é a geração de planos de contingência para a composição, que permite que, se um

erro acontecer durante a execução da composição, um outro plano seja imediatamente executado. Isto oferece para as composições de serviços uma maior robustez e uma maior confiabilidade.

Todos os serviços cadastrados são anotados semanticamente através de uma descrição OWL-S. Cada serviço tem parâmetros de entrada e saída, e implementa uma determinada atividade, que pode ter pré-condições, efeitos e subclasses.

O algoritmo de descoberta de composições de serviços é dividido em cinco passos. A sua entrada consiste de uma descrição de *workflow* em alto nível, descrevendo as entradas e pré-condições disponíveis, e o conjunto de atividades que devem ser executadas. A sua saída consiste de um grafo cujos nós são estruturas de controle de fluxo ou chamadas a serviços específicos.

O primeiro passo do algoritmo de composição corresponde à geração de um grafo inicial. Este grafo é criado a partir do *workflow* passado pelo usuário como a entrada do algoritmo. Os nós do grafo representam uma estrutura de controle de fluxo ou uma atividade a ser executada. Os nós que representam uma estrutura de controle são associados a uma condição relacionada a um ou mais conceitos definidos numa ontologia. As estruturas de controle permitidas pela ferramenta são: *if/else*, *while*, *do while*, *do until*, *try/catch exception*, *split* e *split and join*. Os nós filhos representam caminhos alternativos. Cada caminho gerado representa uma execução possível que atende à requisição do usuário.

A segunda etapa corresponde à seleção das operações proveitosas. Uma operação é dita proveitosa, quando a mesma está disponível e gera uma saída ou um efeito desejado pela requisição, ou implementa uma atividade desejada pelo usuário. Ela também pode ser proveitosa quando gera uma entrada ou uma pré-condição de uma outra atividade proveitosa.

A seleção destas operações é dividida em duas fases. Na primeira, são criados oito conjuntos, para armazenar os seguintes dados:

- Entradas disponíveis;
- Pré-condições disponíveis;
- Atividades disponíveis;
- Saídas desejadas;

- Efeitos desejados;
- Atividades desejadas;
- Operações proveitosas, caso disponíveis;
- Operações proveitosas;

Inicialmente, todas as entradas do usuário são adicionadas ao conjunto das entradas disponíveis. O mesmo acontece com as suas pré-condições, que são adicionadas ao seu respectivo conjunto. As saídas e efeitos gerados por cada nó encontrado a partir do nó raiz do grafo são adicionadas aos conjuntos de entradas e pré-condições disponíveis, respectivamente. Finalmente, as atividades, saídas e efeitos requeridos pelo usuário são adicionados aos seus respectivos conjuntos.

A segunda etapa corresponde à aplicação de cinco regras, até que nenhuma regra possa ser aplicada ou um plano seja encontrado. As regras usadas são:

- **Regra 1:** se houver novos elementos nos conjuntos de entrada ou pré-condições disponíveis, então as operações proveitosas disponíveis que precisam apenas destas entradas e pré-condições devem ser adicionadas ao conjunto de operações proveitosas;
- **Regra 2:** se houver novos elementos no conjunto de operações proveitosas, então suas saídas, efeitos e atividades devem ser adicionadas aos seus respectivos conjuntos de disponibilidade;
- **Regra 3:** se houver novos elementos nos conjuntos de saídas, efeitos ou atividades desejadas então as operações que produzem pelo menos um destes elementos devem ser adicionadas ao conjunto de operações proveitosas disponíveis;
- **Regra 4:** se houver novos elementos no conjunto de operações proveitosas disponíveis, então suas entradas e pré-condições devem ser adicionadas aos conjuntos de saídas e efeitos desejados, respectivamente;
- **Regra 5:** se todas as saídas, efeitos e atividades desejadas pelo usuário estão disponíveis, então é possível gerar um plano.

Se uma operação produz uma saída ou um efeito desejado, ou implementa uma atividade que representa uma subclasse de uma saída, efeito ou atividade desejada pelo usuário então ela é selecionada pela regra 3, e considerada proveitosa. Da mesma forma, se uma entrada ou pré-condição disponível é uma subclasse de uma entrada ou pré-condição de uma operação, ou a tem como uma propriedade, a operação é selecionada pela regra 1 e considerada disponível.

Depois que as operações proveitosas são selecionadas, ocorre a geração de todos os caminhos de execução possíveis. Para cada nó que representa uma operação, todos os caminhos possíveis compostos por *web services* são gerados a partir das operações proveitosas selecionadas na etapa anterior do algoritmo. Para cada saída, efeito ou atividade desejada pelo usuário, um vetor é criado para armazenar as operações do conjunto de operações proveitosas que produzem tal entrada, efeito ou atividade. Desta forma, todos os caminhos de execução possíveis, compostos de um elemento de cada vetor, são gerados.

Na próxima etapa, todos os caminhos gerados são adicionados a um grafo. Este grafo vai conter todos os planos de contingência possíveis. Se a execução de um determinado nó falha, a máquina de execução tenta executar um caminho que inicia a partir de um nó irmão do nó onde ocorreu a falha. A execução do *workflow* só falha por completo se houver uma falha em um determinado nó e não haja nenhum caminho iniciado em um nó irmão ou tio (do nó onde aconteceu a falha) que possa ser executado. Ainda nesta etapa, as operações repetidas e redundantes são removidas do grafo. No grafo gerado, os estados correspondem às operações a serem executadas. Se o resultado de uma operação corresponde à entrada de outra operação, uma transição entre os dois estados é criada. O rótulo da transição é a saída produzida pela operação representada pelo estado de origem da transição, que servirá como entrada da operação representada pelo estado de destino da mesma.

A última etapa do algoritmo de descoberta de composições de serviços consiste na otimização do grafo encontrado. Esta otimização é feita através de um conjunto de heurísticas. Atualmente, a heurística utilizada é a do menor caminho, para a diminuição dos custos de comunicação.

A principal contribuição deste trabalho é, sem duvida, a descoberta de composições de serviços com planos de contingência, que oferecem a elas uma maior robustez e uma

maior confiabilidade. O algoritmo proposto é capaz de descobrir composições complexas, onde serviços podem ser compostos e executados em paralelo ou de forma seqüencial. Outra característica interessante é o tratamento de pré-condições e efeitos.

Uma característica importante do trabalho é que, durante a fase da seleção das operações proveitosas, as regras de seleção de serviços (especificamente as regras 1 e 3), são capazes de recuperar serviços cujo conceito que define uma saída, efeito ou atividade da requisição represente uma subclasse do conceito definido pela saída, efeito ou atividade do serviço a ser analisado. Da mesma forma, os serviços que contêm o conceito definido na requisição do usuário (para uma saída, efeito ou atividade) como uma propriedade também são recuperados. Isto é importante porque permite o casamento parcial de informações, ou seja, operações cujos conceitos não casam exatamente com os conceitos definidos na requisição do usuário, mas que possuem termos relacionados, podem ser recuperados. Considerando a diversidade e a heterogeneidade dos serviços encontrados na *Web*, esta é uma característica indispensável para a recuperação e descoberta de serviços. Contudo, a ferramenta armazena todos estes elementos num único vetor, sem fazer distinção entre os mesmos. Por exemplo, se uma operação tem como saída um conceito exatamente igual ao conceito definido pela requisição do usuário, e uma outra operação tem como saída um conceito que tem o conceito da requisição do usuário como uma propriedade, ambas serão recuperadas e tratadas de forma similar. Na hora da geração do grafo de execução, a segunda operação pode ser colocada no plano original, enquanto que a primeira, que é mais similar, poderia ser colocada em um plano de contingência. A recuperação de serviços com similaridade parcial é muito importante mas, é preciso uma forma de classificar e quantificar esta similaridade. Desta forma, pode-se aplicar um algoritmo de *ranking* para garantir que os serviços com maior grau de similaridade sejam exibidos antes para o usuário. Em algumas situações, isto pode significar que a composição tenha um caminho mais longo do que o caminho gerado com a seleção de outro serviço, mas consideramos que a heurística "nível de similaridade" deve ser mais importante do que a heurística que considera a menor quantidade de serviços da composição.

Uma característica não abordada neste trabalho é a restrição a provedores de serviços. Muitas vezes, as características não funcionais do provedor de serviço (forma de pagamento, desconto, localização, etc) podem ser muito importantes na hora da seleção de

um serviço. Em algumas situações, estas características são mais relevantes para o usuário do que as características funcionais do serviço (custo, atraso, etc), embora estas últimas características possam ser decisivas para a seleção de um serviço em detrimento de outro de mesmo nível de similaridade.

2.2.5 O trabalho de Gekas & Masli

Outra solução para o problema da composição automática de *web services* foi oferecida por GEKAS & MASLI (2005) Este trabalho focaliza a extração automática de informações sobre a estrutura da conectividade de um registro de serviços. Esta informação é usada para guiar o processo de composição automática de *web services*. A pesquisa é baseada em dois tipos de serviços: serviços informativos (como um serviço de novidades financeiras) e serviços de comércio eletrônico (como o serviço da *Amazon Books*).

O *framework* desenvolvido tem uma base de dados contendo anotações semânticas sobre todos os serviços cadastrados. Um provedor de serviços pode usar o *framework* para registrar novos serviços, enquanto que usuários podem usá-lo para encontrar serviços (ou composições de serviços) que satisfaçam às suas necessidades.

Ao registrar um novo serviço, o *framework* gera automaticamente uma anotação semântica para o novo serviço cadastrado. Esta informação é armazenada em um arquivo DAML-S, e baseada em uma das ontologias de domínio. Estas ontologias de domínio foram criadas pelos próprios desenvolvedores do trabalho, e são armazenadas no formato DAML+OIL.

Para a descoberta de composições de serviços, os autores usam heurísticas com relação à estrutura de conectividade dos serviços armazenados na base de dados. Através desta informação, a máquina de busca verifica como os serviços cadastrados estão interligados entre si. Basicamente, os autores tentam empregar técnicas relacionadas ao *ranking* de páginas *Web* à recuperação de serviços. O trabalho não é ligado a um algoritmo de composição de serviços particular. A intenção dos autores é testar e avaliar o desempenho de vários algoritmos de composição. A versão atual usa um algoritmo recursivo chamado de *depth-first*.

Mais especificamente, eles usam um algoritmo de *ranking* de páginas utilizadas na ferramenta de busca *Google* (PAGE & BRIN, 1998), chamado de *PageRank*. O valor do *page rank* de um serviço A em relação a um serviço B é calculado através da probabilidade do serviço B ser recuperado em uma busca uma vez que o serviço A é acessado. De forma análoga a estrutura de *links* existente entre páginas *Web*, o *framework* considera que um serviço pode ser "ligado" a outros serviços. Ele considera que, se um serviço B aceita como entrada uma saída produzida por um serviço A, então existe um link partindo do serviço A para o serviço B. Todos estes cálculos são feitos de forma "off-line", já que estas operações são custosas e poderiam comprometer o tempo de resposta do algoritmo, se calculado em tempo de execução.

O processo de descoberta de composição de serviços utiliza esta informação sobre o *page rank* dos serviços. Em cada etapa do processo de busca, o algoritmo examina o valor do *page rank* das operações que estão ligadas ao serviço que está sendo pesquisado. Através desta medida, o serviço mais "importante" é recuperado, enquanto que os demais serviços são colocados em uma fila de prioridade.

A representação da conectividade de serviços é interessante e facilita o descobrimento de composições de serviços que ocorrem de forma seqüencial. Como toda a análise de conectividade é feita de forma "off-line", estas composições podem ser descobertas de uma forma mais rápida. No entanto, existem situações onde a composição de serviços criada para atender à requisição do usuário não ocorre de forma seqüencial, mas de forma paralela. Infelizmente, o trabalho não mostra nenhum suporte à descoberta deste tipo de composição. Outra característica que também não é abordada pelo mesmo, é a noção de casamento parcial de informações, já que dois serviços são ligados apenas se a saída do primeiro serviço for igual à entrada do segundo serviço. Além disso, o trabalho não permite a recuperação de serviços baseados nas restrições de provedores de serviços.

2.2.6 O trabalho de Medjahed et al.

Foi proposto por MEDJAHED et al. (2003) um modelo para verificar a possibilidade de composição de serviços e uma técnica de composição de serviços, baseada neste modelo.

Para fazer a anotação semântica de serviços, os autores desenvolveram uma ontologia para serviços usando a linguagem DAML+OIL. Esta ontologia armazena tanto as características referentes ao acesso do serviço, que são encontradas numa descrição WSDL, quanto as informações semânticas do mesmo. A representação desta ontologia desenvolvida é mostrada na Figura 3 (MEDJAHED et al, 2003).

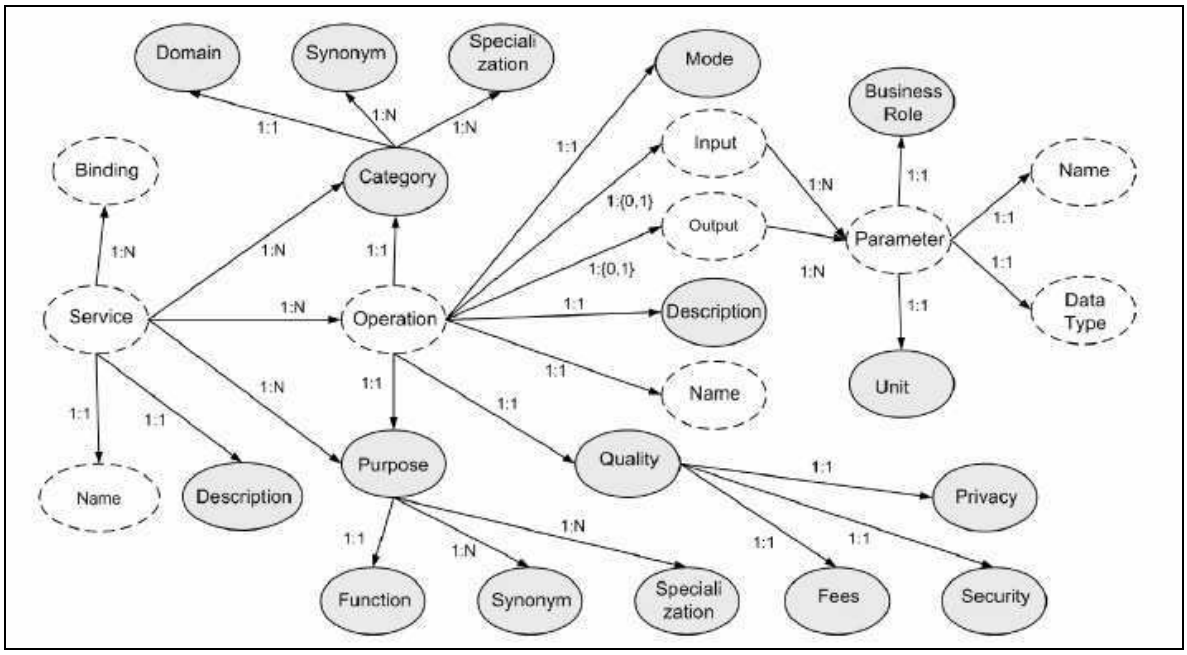


Figura 3: Ontologia para a descrição de serviços

Os *web services*, ou provedores de serviços, são descritos através das seguintes características:

- **Descrição:** corresponde a uma descrição textual das características do serviço publicado;
- **Operações**: representa o conjunto de operações oferecidas pelo serviço;
- ***Bindings:*** representa o conjunto de protocolos de ligação que são aceitos pelo serviço;
- **Propósito:** representa o conjunto de propósitos oferecidos pelo serviço. Um propósito se refere à classificação do serviço, de acordo com alguma taxonomia;

- **Categoria:** representa o conjunto de categorias das operações que compõem o serviço. Uma categoria se refere ao domínio de aplicação a que o serviço pertence;

De acordo com a definição de um *web service*, ele pode oferecer várias operações. Uma operação também é definida através de um conjunto de características. A primeira destas características é a descrição, que corresponde a uma descrição textual sobre as suas características. A segunda característica corresponde ao modo. Quatro tipos de modos de operação são definidos: *one-way*, notificação, *solicite-response* e *request-response*. Além disso, cada operação tem um conjunto de entradas e outro de saídas, além de um propósito, uma categoria, e uma medida de qualidade, que provê as propriedades qualitativas da operação (custo, confiança, etc).

Uma grande contribuição deste trabalho é a definição de um modelo de composição de serviços, que verifica se dois serviços podem ou não ser compostos. Para especificar este modelo, foram definidos vários níveis de composição de serviços. Durante a geração de uma composição de serviços, todos estes níveis são verificados.

O primeiro nível de composição é chamado de composição de modo e *binding*. Este nível verifica se os modos das operações definidas pelos dois serviços são compatíveis. Além disso, ele verifica se as duas operações possuem pelo menos um protocolo de comunicação em comum.

O segundo nível é chamado de composição de mensagem. Este tipo de compatibilidade verifica se os tipos de dados que compõem as mensagens são compatíveis. Esta comparação é feita através do tipo sintático dos mesmos.

O terceiro nível é chamado de composição semântica. Este nível verifica se o propósito e a categoria das duas operações envolvidas no processo de composição são compatíveis.

O quarto nível é chamado de composição qualitativa, e avalia se os serviços envolvidos são compatíveis com relação ao nível de segurança, privacidade, etc.

O último nível de composição é chamado de solidez, e verifica se a composição de dois serviços pode ou não adicionar valor aos mesmos. Por exemplo, dois serviços de reserva de hotel e de reserva de passagens podem ser combinados para formar um serviço

de viagem. A idéia dos autores foi a definição de regras para verificar se esta condição ocorre numa composição de serviços. Esta verificação é feita através da noção de *templates* de composição e de grafos direcionados.

A composição de serviços é dividida em quatro etapas. A primeira etapa corresponde à especificação da composição. Nela, o compositor define uma descrição de alto nível da composição desejada, através de uma linguagem chamada CSSL. A linguagem permite a especificação de planos de composição onde serviços podem ser executados em paralelo ou de forma seqüencial. Ela ainda permite o uso de estruturas de controle de fluxo. Nesta etapa, o compositor precisa apenas ter uma idéia do serviço em que ele está interessado, sem a necessidade de especificar saber detalhes técnicos da composição, como a descrição de cada componente, suas características e como os mesmos são interligados.

Depois que a composição é especificada, a segunda etapa corresponde à geração dos planos de execução, com o uso de um *matchmaker*. O usuário pode controlar o número máximo de planos a serem gerados. A premissa básica do algoritmo de busca é mapear cada operação descrita na composição criada na fase de especificação para uma ou mais operações de serviços existentes. O algoritmo procura por serviços cujo propósito e comunidade sejam compatíveis com pelo menos um propósito ou comunidade descrita na requisição. Depois, o algoritmo verifica quais serviços podem ser compostos. O algoritmo de busca da ferramenta é mostrado na Figura 4 (MEDJAHED et al, 2003).

Observando o algoritmo, notamos que o mesmo usa algumas funções para verificar se dois serviços podem ser compostos. A primeira função chama-se *purpose_compatible*, e verifica se o propósito da composição é compatível com o propósito da operação. A função *category_compatibile* verifica a compatibilidade entre a categoria dos mesmos. A função *quality_composable* verifica se a operação da composição pode ser qualitativamente composta com a operação. A função me*ssage_compatible* verifica se as mensagens das duas operações são compatíveis, enquanto que a função s*ound* verifica a solidez da composição gerada.

Figura 4: Algoritmo de busca da ferramenta (MEDJAHED et al, 2003)

A terceira etapa do processo de composição corresponde à seleção de um plano de composição, caso mais de um plano tenha sido gerado durante a fase anterior. Para facilitar essa seleção, três parâmetros de qualidade são especificados: *ranking*, relevância e completude. O ranking dá uma aproximação da importância da composição, enquanto que a relevância estima a sua solidez e a completude indica a proporção entre o número de serviços que podem ser compostos e o numero de componentes da composição.

A última etapa corresponde à geração de uma descrição detalhada da composição de serviços. Esta descrição tem informações sobre a lista de serviços que a compõem, mapeamento entre mensagens e parâmetros, e o fluxo de controle e dados entre os componentes. As principais características desta fase são: customização e expansão. A customização permite que a composição gerada possa ser descrita em várias linguagens de composição de serviços, como WSFL (LEYMAN, 2001) e BPEL. A expansão permite que outras linguagens de composição de serviços sejam adicionadas.

O trabalho apresenta algumas contribuições interessantes. O modelo de composição proposto consegue verificar vários níveis de composição. No entanto, ele apresenta uma deficiência que pode lhe impor uma grande limitação. A anotação semântica dos serviços é feita sem a noção de tipo semântico. Na ontologia de serviços desenvolvida pelos autores, cada parâmetro de entrada e saída tem como atributo um nome, um tipo de dados (do ponto de vista sintático), uma unidade e uma regra de negócio. Esta característica limita muito a recuperação de serviços, pois a máquina de busca perde a capacidade de fazer inferências sobre estes termos, que corresponde ao principal beneficio do uso de ontologias de domínio.

Além disso, o processo de composição só considera composições onde os serviços são organizados de forma seqüencial. Outra limitação decorre do fato que ele não permite a seleção de serviços baseada em restrições ao provedor, que também pode ser importante na hora de realizar a seleção de serviços.

2.3 Considerações Finais

Este capítulo apresentou alguns trabalhos desenvolvidos nos últimos anos para resolver os problemas da especificação e descoberta de composições de serviços. Ele mostrou também que todos os trabalhos apresentam vários pontos fortes e algumas deficiências, e que nenhum deles conseguiu se tornar uma solução padrão para a comunidade de pesquisa da área. A Tabela 1 mostra um quadro comparativo entre os trabalhos apresentados.

O quadro mostra que alguns trabalhos permitem a especificação de composição de serviços, mas não oferecem meios para a descoberta automática destas composições. Outros trabalhos realizam a descoberta de composições automáticas, mas tem deficiências de flexibilidade, por não permitirem a recuperação de serviços com casamento parcial. A restrição a provedores de serviços, que permite que a seleção de serviços baseados nas características de seus provedores, não é explorada em nenhum dos trabalhos apresentados.

Trabalho	Composição Automática	Composição Paralela	Composição Seqüencial	P.Condições e Efeitos	Ranking	Restrição Provedor
Aversano et al	sim	sim	sim	não	sim	não
Costa et al.	sim	sim	sim	sim	não	não
Gekas & Fasli	sim	não	sim	não	sim	não
Kvaloy	sim	não	sim	não	sim	não
Medjahed et al.	sim	não	sim	não	sim	não
Milanovic	não	sim	sim	sim	não	não
IRS-III	não	sim	sim	sim	não	não
Sell et al.	não	sim	sim	sim	não	não
Compat	não	não	não	não	sim	não

Tabela 1: Quadro comparativo entre os trabalhos apresentados

CAPÍTULO III – WEBS COMPOSER

Este capítulo descreve WebS Composer, um sistema desenvolvido para a descoberta e composição de *web services*. Ele descreve todo o processo de concepção da ferramenta, desde a fase de levantamento de requisitos até a fase de implementação, abordando, dentre outras características, a arquitetura do sistema e de todos os seus módulos Ele também descreve a especificação dos algoritmos usados tanto para a descoberta de serviços simples quanto para a descoberta de composições de serviços.

3.1 Os requisitos funcionais

O objetivo principal desta dissertação é o desenvolvimento de uma ferramenta capaz de descobrir serviços e composições automáticas de serviços na *web* de uma forma simples, rápida e eficiente. Devido a esse objetivo, foi dada a essa ferramenta o nome de *WebS Composer*.

Os principais requisitos funcionais levantados para o desenvolvimento do trabalho foram:

RF1 – O cadastro de provedores de serviços

O sistema deve permitir que empresas cadastrem as suas informações na base de dados da ferramenta. A empresa pode efetuar o seu cadastro no sistema se ela oferece serviços *on-line* na rede na forma de *web services*, serviços *off-line* que o usuário pode solicitar por outros meios, como telefone, correio eletrônico ou pessoalmente, ou os dois tipos de serviços.

RF2 – O cadastro de serviços.

O sistema deve permitir que as empresas provedoras de serviços cadastradas no sistema possam cadastrar os serviços oferecidos de forma *on-line*.

RF3 – Descoberta de serviços

Quando o usuário entrar com uma requisição por um determinado serviço, o sistema deve localizar, da forma mais precisa possível, os serviços cadastrados no sistema que satisfazem a esta requisição. Para que essa busca tenha resultados relevantes, a cobertura e a precisão dos resultados devem ser altas.

RF4 – Descoberta de composições de serviços

Quando o usuário entrar com uma requisição de serviços que não puder ser totalmente resolvida por um dos serviços da base de dados, o sistema deve ser capaz, quando possível, de encontrar uma composição de serviços que satisfaça esta solicitação. Esta composição deve ser descoberta em tempo de execução, no ato da consulta, de forma totalmente automatizada e transparente para o usuário. Além disso, o sistema deve permitir que o plano de execução que descreve a composição de serviços seja representado em uma linguagem que permita a descrição dos serviços que compõem a composição e a ordem em que eles devem ser executados para atingir os resultados desejados.

RF5 – Localização de Provedores de Serviços

O sistema também deve ser capaz de localizar, de forma precisa, os provedores de serviços que satisfaçam certas restrições impostas pelo usuário final numa consulta.

RF6 – Execução de serviços e composições

O sistema deve permitir a execução de todos os serviços e composições de serviços descobertos em uma consulta.

3.2 Os requisitos não funcionais

Os requisitos não funcionais levantados foram:

RNF1 – Marcações Semânticas

A ferramenta deve armazenar uma descrição semântica para cada serviço cadastrado, bem como para a empresa que o oferece. A anotação dos serviços é importante para facilitar a descoberta de serviços e de composições, enquanto que a anotação semântica do provedor do serviço auxilia na recuperação e seleção de provedores de serviços. Todas estas informações semânticas devem ser providas por meio de ontologias. Para isso, seria necessária a utilização de linguagens apropriadas para descrever as ontologias de domínio, os serviços cadastrados e os seus provedores. Também foi definido que a preferência na escolha destas linguagens deve ocorrer por tecnologias que representam um padrão internacional ou que tenham uma grande aceitação pela comunidade relacionada à *web* semântica.

RNF2 – Flexibilidade

A ferramenta deve saber relacionar e manipular dados anotados em várias ontologias de domínio. Mais ainda, ela deve permitir que novas ontologias de domínio sejam acrescentadas ou retiradas de sua base de conhecimento, sem a necessidade de fazer nenhuma alteração em seu código fonte.

RNF3 – Casamento Parcial

Sempre que uma nova requisição do usuário for submetida à máquina de busca, ela deve percorrer todos os serviços cadastrados em sua base de dados e recuperar todos aqueles que tenham um certo grau de similaridade com esta requisição. Além desses serviços, ela deve recuperar também serviços que se pareçam parcialmente com a requisição, ou seja, os serviços que oferecem parte da funcionalidade desejada, e os que oferecem a mesma com alguns parâmetros diferentes. Para isso, uma medida de similaridade é necessária para calcular a similaridade entre a requisição do usuário e um serviço. Além disso, um *threshold* mínimo é necessário para indicar quando um serviço com casamento parcial deve ou não ser recuperado, e um algoritmo de *ranking* deve ser

utilizado, de forma a garantir que os serviços com maior grau de similaridade com a requisição sejam mostrados primeiro para o usuário.

3.3 Os atores do sistema

Após a identificação dos requisitos da ferramenta e das suas características desejáveis, foi feito um levantamento dos atores do sistema, ou seja, a identificação de todas as pessoas que interagem diretamente com o mesmo. De acordo com os seus papéis, os atores identificados foram:

- **Administrador do sistema:** este ator usa o sistema para acrescentar, alterar ou remover alguma ontologia de domínio;
- **Provedor de serviços:** registra as informações de sua empresa e cadastra as informações referentes a cada serviço que ela oferece;
- **Cliente:** usa o sistema para localizar serviços ou provedores de serviços;

3.4 A arquitetura de WebS Composer

WebS Composer foi implementado utilizando uma arquitetura distribuída baseada em três camadas. As camadas são: a camada de apresentação, a camada da lógica do negócio, e a camada de armazenamento de dados. A arquitetura geral da ferramenta é mostrada na Figura 5.

A camada de apresentação é responsável por oferecer aos usuários os meios para que o mesmo possa interagir diretamente, e de forma simples e fácil, com a ferramenta. Ela permite que o usuário selecione a funcionalidade que ele deseja executar, envie as informações requeridas por esta funcionalidade e observe o resultado final obtido para a sua ação. Esta camada foi implementada com a tecnologia de páginas *web*. Foram usadas três tecnologias:

- ***HTML:*** a linguagem HTML (RAGGET et al, 1999) foi usada para o desenvolvimento do conteúdo estático das páginas da interface;

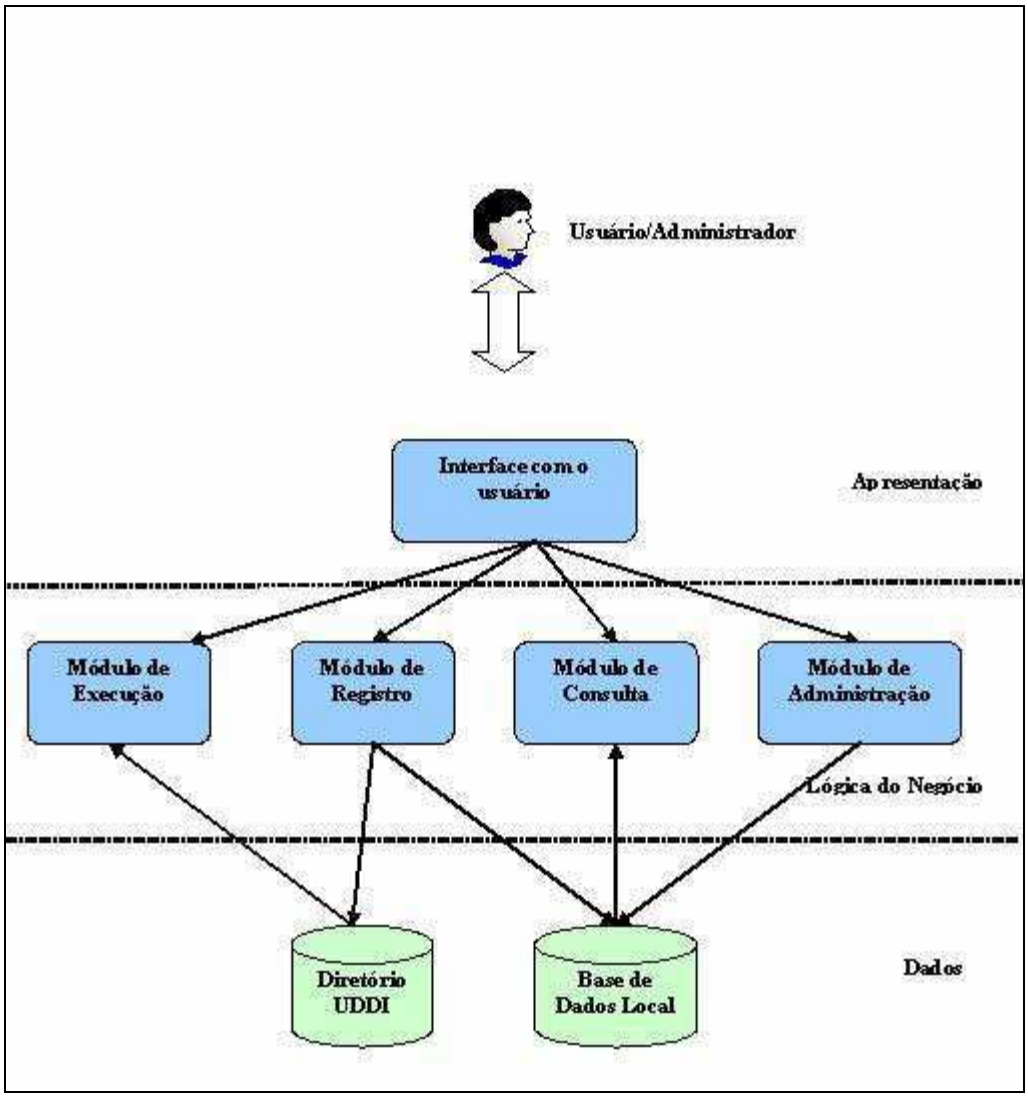


Figura 5: Arquitetura Geral de WebS Composer

- ***Java Server Pages:*** usada para o desenvolvimento do conteúdo dinâmico das páginas da interface *web*, ou seja, o conteúdo cujo valor depende das ações do usuário e dos dados decorrentes destas ações;
- ***Java Servlets:*** usada para fazer a comunicação das páginas da interface *web* com os módulos da camada da lógica do negócio;

A camada da lógica do negócio corresponde ao *WebS Composer* propriamente dito. Esta camada corresponde à ferramenta propriamente dita, que executa todas as ações solicitadas pelo usuário através da camada de interface. Esta camada, também chamada de

camada de aplicação, é subdividida em quatro módulos. Cada um destes módulos desempenha uma função bem definida.

O primeiro módulo é o de registro, que permite a inserção de novos serviços e provedores de serviço no sistema. Sempre que o usuário registra um novo serviço, a ferramenta cadastra este serviço num serviço de diretório UDDI (CLEMENT et al, 2004), e gera uma marcação semântica para o mesmo. Esta marcação é armazenada numa base de dados local da ferramenta. Além disso, a ferramenta também gera uma marcação semântica para o provedor do serviço, de forma a permitir a realização de buscas mais complexas e melhorar a seleção de serviços.

O segundo módulo realiza o processamento de uma consulta. Ele é responsável pela descoberta de serviços e de composições de serviços. Sempre que o usuário entra com uma nova requisição, ele gera uma marcação semântica para a mesma. Depois, ele consulta todas as marcações semânticas dos serviços armazenados na base de dados local, e tenta encontrar serviços cujas marcações sejam similares às da requisição. Quando nenhum serviço é recuperado, ele tenta então encontrar uma composição de serviços que satisfaça esta requisição. Neste caso, é gerado um plano de execução para a composição descoberta, indicando quais os serviços que devem ser executados e a ordem em que os mesmos devem ser executados.

O terceiro módulo da camada de aplicação é o de execução, que permite que o usuário execute os serviços e as composições de serviços descobertas pelo módulo de consulta. Para executar um serviço atômico, ele acessa o arquivo WSDL do serviço para verificar as informações de acesso ao mesmo, como a sua URI, os parâmetros de entrada e saída, os tipos destes parâmetros, o formato das mensagens, etc. Para executar uma composição de serviços, ele recebe um objeto que representa a composição de serviços descoberta pelo módulo de consulta, juntamente com os dados de entrada passados pelo usuário, e invoca todos os serviços que compõem o mesmo. O próprio módulo se encarrega de gerenciar o fluxo de informações entre os serviços durante todo o processo de execução da composição.

O último módulo desta camada é o de administração, usado pelo administrador do sistema para fazer alterações nos domínios de conhecimento utilizados pela mesma. Este

módulo permite, mediante a uma autenticação, que o administrador do sistema adicione, exclua ou atualize um domínio de conhecimento da ferramenta.

A camada de dados corresponde ao armazenamento dos dados. No WebS Composer, esta camada é composta de dois elementos:

- um serviço de diretório UDDI, que armazena a descrição UDDI das empresas e dos serviços cadastrados;
- um sistema gerenciador de banco de dados. Este componente armazena todas as ontologias do sistema e as descrições semânticas de todos os serviços;

3.5 A camada da lógica do negócio

Nesta seção, é descrita a camada da lógica do negócio, aplicação, que corresponde ao núcleo do WebS Composer. Esta camada está dividida em diversos módulos: registro,consulta, execução e administração.

3.5.1 O módulo de registro

O módulo de registro permite a inserção de novos serviços e provedores de serviços na base de dados da ferramenta. Para aumentar a eficiência dos algoritmos de descoberta de serviços, uma marcação semântica é gerada para cada serviço cadastrado, para tornar suas informações inteligíveis tanto por pessoas quanto por máquinas. Para atingir este objetivo, são definidas ontologias de domínio, e cada parâmetro de entrada e de saída do serviço é associado a um conceito definido nestas ontologias. Para melhorar a seleção de serviços para o usuário, e permitir que o usuário localize provedores de serviços de uma forma apropriada, também é gerada uma descrição semântica para o provedor do serviço. Esta descrição também é gerada através dos conceitos descritos nas ontologias de domínio.

A marcação semântica é uma das características mais importantes da ferramenta, pois descreve as características semânticas das informações cadastradas, permitindo assim que a máquina de busca do sistema possa fazer inferências sobre estas informações para

realizar consultas mais exatas. Toda esta semântica é acrescentada às informações por meios de ontologias de domínios, que descrevem os conceitos e os relacionamentos entre os conceitos que compõem um determinado domínio de conhecimento.

Desta forma, a primeira decisão a ser tomada a respeito da implementação do módulo de registro foi decidir qual a linguagem que seria usada para a representação destas ontologias. Nos últimos anos, várias linguagens foram desenvolvidas com o intuito de descrever este tipo de informação. Uma das primeiras linguagens usadas para a descrição de ontologias foi a linguagem RDF (KLYNE & CARROLL, 2004). Seguidas a ela, surgiram outras linguagens importantes, como *RDF Schema* (BRICKLEY & GUHA, 2004) DAML+OIL (CONNOLY et al, 2001), além da linguagem OWL (McGUINNESS & VAN HARMELEN, 2004). Dentre estas várias linguagens disponíveis, optou-se pelo uso da linguagem OWL, devido ao seu grande poder de expressividade e, principalmente, por ela ser uma recomendação da *W3C*, que é a comunidade que idealizou a w*eb* semântica.

Para descrever a marcação semântica de *web services*, foram desenvolvidas as ontologias de serviços. Uma ontologia de serviço é utilizada para descrever as características semânticas de um serviço. Isso é feito fazendo uma associação entre algumas informações que o descrevem e conceitos definidos em ontologias de domínio. Nesta área, duas ontologias se destacam. A primeira delas é DAML-S, que permite descrever ontologias de serviços baseada na linguagem DAML. A outra é OWL-S, que é baseada na linguagem OWL. Como a linguagem OWL foi a escolhida para definir as ontologias, optou-se pela ontologia OWL-S para fazer a marcação semântica dos serviços.

Para melhorar o processo de seleção de serviços, WebS Composer usa uma descrição semântica também para os provedores de serviços que se cadastram em sua base de dados. Esta marcação semântica é feita representando o provedor do serviço como uma instância de um conceito definido em uma ontologia de domínio.

De forma a oferecer as funcionalidades descritas acima, o módulo de registro de serviços e provedores é baseado na arquitetura mostrada na Figura 6.

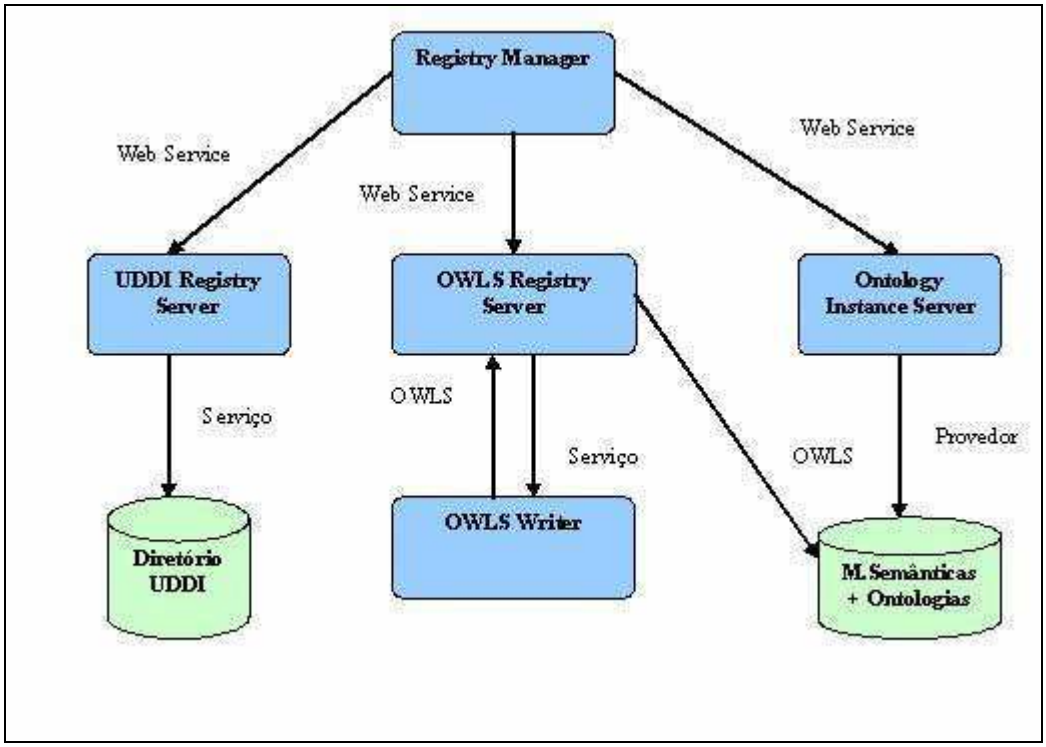


Figura 6: Visão geral da arquitetura do módulo de registro

Para cadastrar seus serviços no sistema, o provedor do serviços precisa primeiro passar algumas informações referentes à sua empresa. Este processo se divide em duas etapas. Na primeira etapa, o provedor fornece informações gerais sobre a empresa, como o nome da organização, sua descrição, telefone de contato, e-mail, *home page,* entre outras. Na segunda etapa, ele seleciona na ontologia de domínio o conceito que mais se assemelha a sua empresa. Por exemplo, num domínio de aplicação de hospedagens ele pode indicar que sua empresa é um hotel, uma pousada, um motel, etc. Em seguida, deve fornecer algumas informações que o sistema usa para descrever empresas do tipo selecionado. Estas informações correspondem às propriedades que a ontologia define para o conceito que representa a empresa. Por exemplo, se a empresa representa um hotel, esta deve passar informações como o nome, o número de estrelas, os cartões de crédito aceitos e as facilidades oferecidas, como piscina, boate, quadra de tênis e estacionamento.

Após cadastrar as informações da sua empresa, o provedor pode cadastrar os serviços *on-line* oferecidos por ele. Para cada serviço cadastrado, o usuário deve informar o nome, a descrição, e os parâmetros de entrada e saída do serviço. As informações passadas sobre cada parâmetro se referem às características semânticas do mesmo. O usuário fornece

estas informações associando cada parâmetro de entrada e de saída a um dos conceitos definidos nas ontologias de domínio do sistema. Através desta informação, a máquina de busca pode entender o significado de cada um destes parâmetros, independentemente da notação sintática como eles foram definidos.

Após receber estes dados, o sistema cria um objeto que representa um *web service*. Este objeto encapsula todas as informações passadas pelo provedor do serviço. O sistema então repassa este objeto para um objeto chamado de *Registry Manager*, que corresponde à porta de entrada do modulo de registro. Ao receber este objeto, o *Registry Manager* o repassa para três outros componentes: o *UDDI Registry Server*, o *OWLS Registry Server* e o *Ontology Instance Server*.

O componente *UDDI Registry Server* é responsável por armazenar o *web service* em um serviço de diretório UDDI. Este tipo de publicação permite que o serviço seja descoberto por clientes que realizam buscas em serviços de diretório tradicionais. Ele abre o objeto que representa o *web service*, extrai as informações necessárias para cadastrar o serviço no diretório (nome da empresa, descrição, telefone, e-mail, serviços oferecidos, etc) e efetiva o cadastro do serviço. Para a implementação desta funcionalidade, foi usado o *framework Java Web Services Development Kit*, versão 1.5, disponibilizado gratuitamente pela empresa *Sun Mycrosystems*, que provê, dentre várias funcionalidades, API's que permitem o armazenamento e a recuperação de serviços em um serviço diretório UDDI. Além disso, ele oferece um serviço chamado *Registry Server*. Este serviço, que deve ser implantado dentro de um servidor de aplicações, simula um serviço de diretório UDDI, permitindo o registro, a consulta e a recuperação de serviços. Ele foi utilizado como o serviço de diretório UDDI que armazena todos os serviços cadastrados na ferramenta. Para implantá-lo, foi utilizado um dos servidores de aplicações disponibilizados pela mesma empresa que disponibiliza o *framework* acima, que é o *Sun Java Application Server*, versão 8.1.

Outro componente que recebe os dados passados pelo *RegistryManag*er é o *OWLS Registry Server*. Ao receber esta informação, ele extrai os serviços oferecidos pela organização e, para cada serviço, aciona um componente chamado de *OWL Writer*. Este componente recebe os dados do serviço (nome, descrição, parâmetros de entrada e saída, etc) e gera uma marcação semântica para o mesmo. A marcação semântica gerada por ele

corresponde a uma instância de um serviço da ontologia OWL-S. Esta descrição é utilizada pelo módulo de consulta para fazer a descoberta de serviços e composições. O arquivo OWL-S gerado é devolvido para o *OWLS Registry Server*, que o armazena na base de dados local da ferramenta. Como a ontologia OWL-S é baseada na linguagem XML, o sistema gerenciador de banco de dados *Oracle 9i* foi escolhido para implantar a base de dados da ferramenta, devido ao suporte que o mesmo oferece ao armazenamento e à recuperação de documentos XML. Além da marcação semântica, o componente armazena na base de dados local uma cópia das informações publicadas no serviço de diretório. Estas informações são publicadas em tabelas que usam um esquema relacional. As marcações semânticas dos serviços são armazenadas em uma coluna do tipo *XMLType,* como um documento XML. O esquema utilizado para o armazenamento destas informações é mostrado mais adiante.

O último componente do módulo de registro corresponde ao *Ontology Instance Server*. Ele tem a função de armazenar na base de dados as informações semânticas referentes ao provedor do serviço. Ele extrai do *web service* que está sendo cadastrado as informações sobre as características semânticas da empresa. Estas características se referem às propriedades do conceito ao qual a empresa foi associada durante ao seu cadastro. Ao extrair estas informações, este componente gera uma instância deste conceito. As informações passadas pelo provedor na hora do cadastro são usadas para atribuir o valor das propriedades desta instância. Esta instância é cadastrada na ontologia a que este conceito se refere. Para fazer a conversão de objetos Java para ontologias OWL e vice-versa, foi usado o *framework* Jena. Dentre várias funcionalidades, ele permite a criação e manipulação de ontologias descritas em várias linguagens.

3.5.2 O módulo de consulta

O módulo de consulta é responsável por fazer toda a descoberta de serviços e de composições de serviços. Para obter uma máquina de busca mais eficiente, todas as buscas da ferramenta são apoiadas nas ontologias de domínio definidas pela mesma. Além disso, um dos modelos clássicos de recuperação da informação, que é o modelo vetorial

(SALTON & McGILL, 1983), foi usado para determinar o casamento entre a requisição do usuário e os serviços e provedores de serviços cadastrados no sistema.

A ferramenta oferece basicamente dois tipos de consultas para o usuário: buscas simples e buscas combinadas. As buscas simples se referem aos serviços ou aos provedores de serviços. Já as buscas combinadas utilizam os dois tipos de buscas simples em uma única consulta.

O primeiro tipo de busca simples oferecido por WebS Composer é a descoberta de serviços e composições. Quando o usuário solicita uma consulta deste tipo, ele deve informar todas as entradas que ele tem disponível para o serviço e todas as saídas que ele deseja obter dele. Assim como os parâmetros de entrada e saída dos serviços cadastrados, cada conceito de entrada e saída da requisição também é associado a um conceito definido em uma das ontologias de domínio do sistema.

Ao receber a solicitação, o sistema a compara com todos os serviços cadastrados em sua base de dados, na tentativa de encontrar um ou mais serviços que a satisfaçam. Um serviço satisfaz uma requisição quando o nível de similaridade entre ambos é superior ao nível de similaridade mínimo definido para a consulta. Caso não encontre nenhum serviço que a satisfaça, ele tenta automaticamente encontrar uma composição de serviços que satisfaça a solicitação. O sistema consegue encontrar composições de dois tipos: paralela e seqüencial. Estes dois tipos de composição são mostrados com mais detalhes mais adiante.

A requisição de serviço solicitada pelo usuário através da interface do sistema é recebida por um componente chamado *Query Manager*, que representa a interface do módulo de consulta com os demais módulos da ferramenta. Ao receber esta solicitação, o *Query Manager* a repassa para os componentes que realizam a descoberta de serviços.

Neste módulo, três componentes são responsáveis pela descoberta de serviços. São eles:

- ***Simple Service Discoverer:*** responsável pela descoberta de serviços simples;
- ***Parallel Composition Discoverer:*** responsável pela descoberta de composições paralelas;
- ***Sequence Composition Discoverer:*** responsável pela descoberta de composições seqüenciais.

Ao receber a requisição do usuário, o *Query Manager* a repassa para o *Simple Service Discoverer,* que tenta encontrar um serviço que atenda sozinho a esta requisição. Caso nenhum serviço seja encontrado, e a saída desejada pelo usuário contenha pelo menos dois componentes, o *Query Manager* repassa a consulta para o *Parallel Composition Discoverer*, que tenta encontrar uma composição paralela para a mesma. Caso ele também não encontre nenhuma composição que a satisfaça, o *Query Manager* encaminha a consulta para o *Sequence Composition Discoverer,* que tenta encontrar uma composição seqüencial para resolver a consulta. Para acelerar o processo de busca, cada componente gera um relatório com os resultados de sua descoberta. Este relatório é reaproveitado pelos demais componentes. Caso nenhum componente consiga encontrar um serviço ou uma composição de serviços que satisfaça, o sistema retorna um relatório contendo as informações descobertas dos dois primeiros componentes.

O diagrama de seqüência que ilustra o processo de descoberta e composição de serviços pela ferramenta é mostrado na Figura 7:

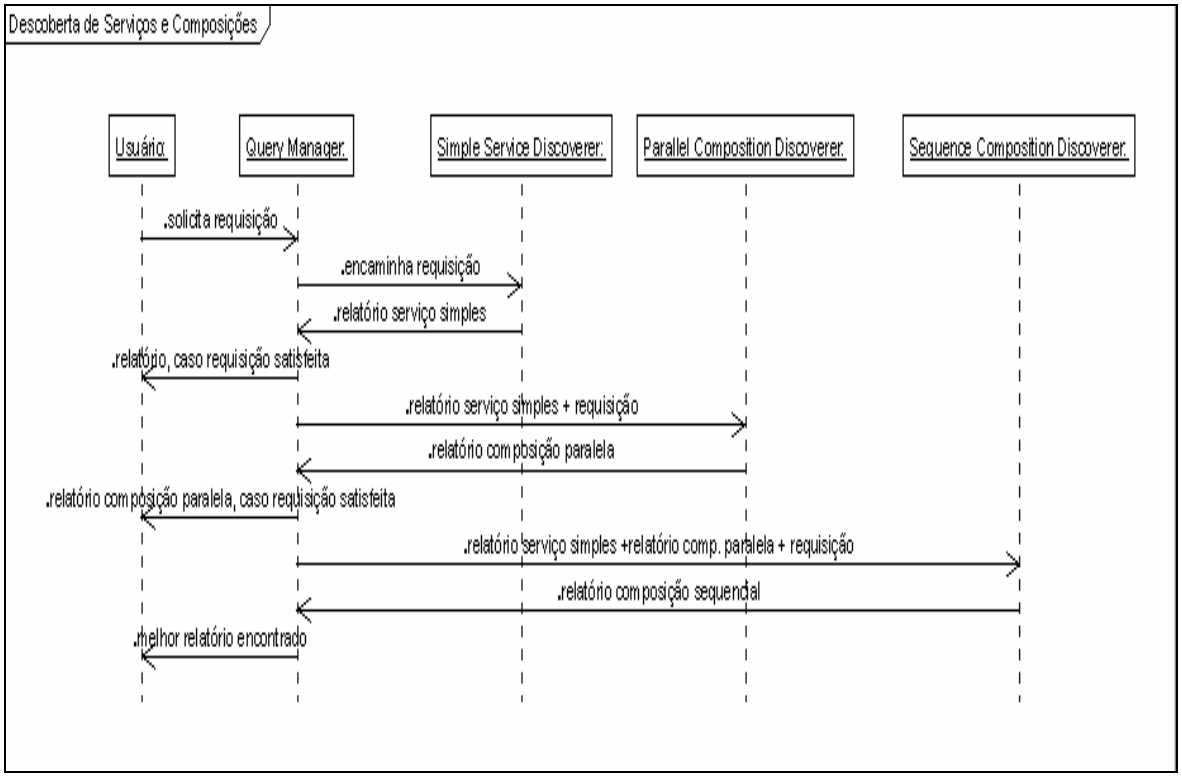


Figura 7: Diagrama de seqüência para a descoberta de serviços e composições

A arquitetura geral deste módulo é mostrada na Figura 8:

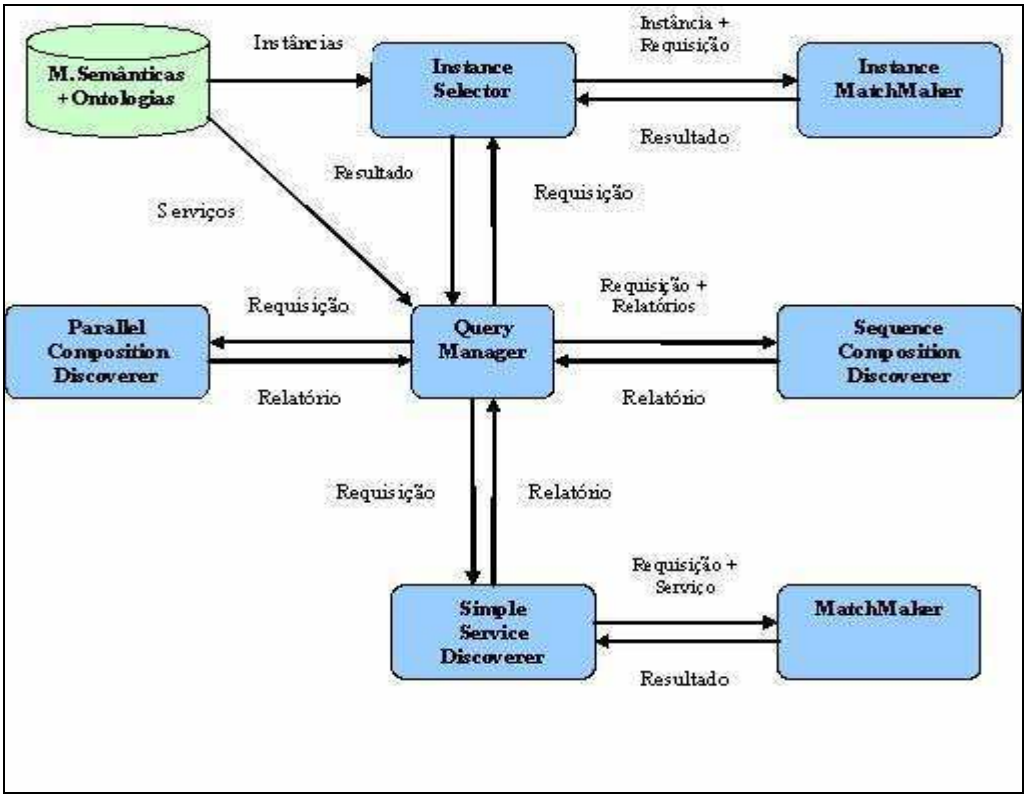


Figura 8: Arquitetura Geral do módulo de consulta

3.5.2.1 A descoberta de serviços simples

O primeiro componente que recebe a requisição do usuário do *Query Manager* é o *Simple Service Discoverer*, que tenta encontrar serviços que possam atender sozinhos a requisição do usuário. Para isso, ele compara a solicitação com cada serviço cadastrado na base de dados. Para cada serviço, ele verifica se o mesmo tem uma similaridade mínima com a mesma. Para isso, ele ativa o *matchmaker* de serviços, que faz o casamento da requisição do usuário com um determinado serviço da ferramenta. Para cada casamento, ele calcula três medidas de similaridade, que mostram o quanto este serviço atende à requisição. Estas medidas são:

- **Grau de similaridade da entrada:** esta medida avalia o grau de similaridade entre as entradas da requisição e as entradas do serviço. Esta medida é calculada verificando-se o grau de existência de cada entrada do serviço no conjunto de entradas da requisição do usuário;
- **Grau de similaridade da saída:** esta medida avalia o grau de similaridade entre as saídas da requisição e as saídas do serviço. Esta é medida é calculada verificando-se o grau de presença de cada saída desejada pela requisição do usuário no conjunto de saídas do serviço cadastrado;
- **Grau de similaridade global:** esta medida usa as duas medidas acima para calcular o grau de similaridade global entre a requisição e o serviço, ou seja, a similaridade em função das entradas e saídas dos serviços;

Para facilitar a descoberta de composições de serviços, o modelo vetorial de recuperação da informação foi utilizado para representar o casamento entre a requisição do usuário e os serviços. Quando o usuário entra com uma nova requisição, *WebS Composer* a representa como se ela fosse um serviço. Depois, ele gera dois vetores para representá-la. O primeiro vetor representa as entradas desta solicitação, e armazena todos os conceitos presentes no seu conjunto de entradas, enquanto que o segundo vetor representa as saídas da mesma, e armazena todos os conceitos presentes no seu conjunto de saídas.

A seguir, a ferramenta compara este serviço com todos os serviços armazenados em sua base de dados. Para cada um destes serviços, são gerados dois vetores: um que representa a similaridade entre as entradas da requisição e as entradas do serviço, e outro que representa a similaridade entre as saídas dos mesmos.

O peso das dimensões destes vetores é calculado através da comparação dos conceitos definidos na requisição e no serviço. Para fazer esse cálculo, o sistema usa o algoritmo de casamento de conceitos definido por PAOLUCCI et al (2002), com uma pequena adaptação. Como já foi dito, todos os conceitos envolvidos na requisição e no serviço são definidos em alguma ontologia de domínio. Se o conceito R for definido na requisição do usuário e S for o conceito definido no serviço, os seguintes casamentos podem ocorrer entre estes dois conceitos:

- **Exato:** este tipo de similaridade ocorre quando os dois conceitos são similares. Ele ocorre em quatro tipos de situação: quando R e S representam exatamente o mesmo conceito, quando R e S definem conceitos semelhantes (sinônimos), quando R representa uma subclasse ou uma subpropriedade de S ou quando R é uma propriedade cujo intervalo é S, um sinônimo de S ou uma subclasse de S;
- ***Plug-in*:** este tipo de relacionamento ocorre quando o conceito S contém o conceito R, ou seja, o conceito definido por R pode ser "plugado" no conceito S. Neste caso, o conceito R é uma propriedade do conceito S;
- ***Subsumes:*** este tipo de relacionamento ocorre quando R é um conceito que contém S. Este relacionamento representa o inverso do relacionamento do tipo *plug-in*;
- **Incompatível:** este tipo de relacionamento ocorre quando nenhum dos outros relacionamentos acima se aplica aos dois conceitos.

Para cada caso acima um valor numérico entre 0 e 1 é associado à dimensão do vetor, dependendo do tipo de relacionamento. Quanto maior for o grau de relacionamento entre os conceitos, maior será este valor. O relacionamento exato corresponde ao maior grau de associação entre dois conceitos, sendo seguido, respectivamente, pelos relacionamentos *plug-in*, *subsumes* e incompatível, que é o tipo de relacionamento mais fraco.

O relacionamento exato, que é o mais forte, é representado pelo valor 1. O relacionamento do tipo *plug-in* é representado pelo valor 0,8. O relacionamento do tipo *subsumes* é representado pelo valor 0,4. Já o relacionamento incompatível, que é o grau de relacionamento mais fraco, é representado pelo valor 0.

Para calcular o vetor que representa a similaridade das entradas da requisição do usuário e as entradas de um serviço do sistema, são utilizados os algoritmos mostrados abaixo. Antes de descrevermos os algoritmos, vamos supor que:

- R é o serviço que representa a requisição do usuário;
- S é um serviço cadastrado na base de dados;

- A função *similaridade* corresponde à implementação do algoritmo de similaridade de conceitos mostrado na seção anterior.

```
encontraVetorEntrada(R,S): Vetor;
Início
   Vetor entrada := criaVetor(S.entradas.size);
   for k:=1 to S.entradas.size do
       entrada[k] := calculaGrauDeSimilaridade(S.entradas[k], R.entradas);
   return entrada;
Fim.

calculaGrauDeSimilaridade(Conceito c, Conjunto s): real;
 Início
      max := similaridade(c,s[1]);
       for k:=2 to s.size do
            if similaridade(c,s[k])>max then max = similaridade(c,s[k]);
       return max;
Fim.
```

Para calcular o grau de similaridade de entrada, o algoritmo usa um vetor com o mesmo número de dimensões do vetor encontrado acima, onde para cada uma das dimensões é atribuído o valor máximo, que é 1. Este vetor, que será chamado de I, representa um vetor cujo casamento seria totalmente compatível com as entradas do serviço.

Caso o vetor que representa o casamento entre as entradas da requisição e as entradas do serviço tenha alguma dimensão cujo valor seja zero, o grau de similaridade de entrada para este serviço é igual a zero. Caso contrário, este grau é calculado através do cosseno do ângulo entre o vetor encontrado pelo algoritmo acima e o vetor I. Quanto mais o vetor encontrado se parecer com o vetor I, menor será o ângulo formado por eles e maior será o valor deste cosseno. Quando ambos são totalmente similares, eles formam um ângulo de 0^o, que tem como cosseno o valor 1. Nestes casos, a similaridade entre as entradas será de 100%.

Para calcular o vetor que representa a similaridade da saída de um serviço com a saída da requisição o sistema usa um algoritmo semelhante ao utilizado para encontrar o

vetor de casamento das entradas. Neste caso, o sistema verifica se todas as saídas que o usuário solicitou em sua requisição estão presentes no conjunto de saídas do serviço: Antes de descrevermos os algoritmos, suponhamos que:

- R é o serviço que representa a requisição do usuário;
- S é um serviço cadastrado na base de dados;
- A função *calculaGrauDeSimilaridade* corresponde à mesma função utilizada pelo algoritmo anterior.

```
encontraVetorEntrada(R,S): Vetor;
Início
   Vetor saida := criaVetor(R.saidas.size);
   for k:=1 to R.saidas.size to
       saida [k] := calculaGrauDeSimilaridade(R.saidas[k], S.saidas);
   return saida;
Fim.
```

Para calcular a similaridade entre as saídas da requisição do usuário e as saídas do serviço, o algoritmo usa um vetor semelhante ao vetor usado para encontrar o grau de similaridade entre as saídas. O número de dimensões deste vetor, que será chamado de O, é igual ao número de saídas contidas na requisição. Para cada uma de suas dimensões é atribuído o valor máximo, que é 1. O grau de similaridade de saída é calculado através do cosseno do ângulo formado pelo vetor de casamento de saída encontrado pelo algoritmo e o vetor O.

Depois de calcular o grau de similaridade de entrada e de saída, o algoritmo calcula o grau de similaridade global entre o serviço e a requisição. Esta medida indica a similaridade entre dois serviços em função da similaridade da entrada e saída dos mesmos. O cálculo desta medida de similaridade é feito da seguinte forma:

- Se o grau de similaridade de entrada for igual a zero, a similaridade global entre os mesmos é igual a zero;
- Se o grau de similaridade de entrada for diferente de zero, a similaridade global é igual ao grau de similaridade de saída.

Após fazer o casamento entre a solicitação do usuário e o serviço, o *Matchmaker* retorna para o *SimpleServiceDiscoverer* um resultado parcial contendo as seguintes informações:

- O serviço que foi comparado à solicitação;
- O vetor de casamento das entradas;
- O vetor de casamento das saídas;
- O grau de casamento das entradas;
- O grau de casamento das saídas;
- O grau de casamento global entre os dois serviços.

O *Simple Service Discoverer* repete este procedimento para todos os serviços cadastrados no sistema. Ele armazena todos os resultados parciais gerados pelo *matchmaker*. Depois que todos os serviços são "casados" com a requisição, ele gera um relatório sobre os dados obtidos com o casamento de cada serviço. O relatório gerado contém cinco conjuntos de dados. Cada resultado parcial pode ser incluído em mais de um conjunto. O relatório gerado para a consulta é formado pelos seguintes conjuntos:

- Os serviços cujas entradas casaram parcialmente com as entradas da requisição;
- Os serviços cujas saídas casaram parcialmente com as saídas da requisição;
- Os serviços cujas entradas casaram totalmente com as entradas da requisição;
- Os serviços cujas saídas casaram totalmente com as saídas da requisição;
- Os serviços que casaram totalmente com a requisição do usuário, em relação ao grau de similaridade global.

Depois de gerar o relatório, o *Simple Service Discoverer* o repassa para o *Query Manager*, que, ao recebê-lo, verifica se foi encontrado pelo menos um serviço que atende sozinho à solicitação do usuário. Ele examina esta informação analisando o último conjunto gerado. Se o resultado for positivo, ele encaminha o relatório para o módulo de interface com o usuário. Caso contrário, se o conjunto de saídas da requisição contiver mais de um elemento, ele repassa a requisição para o componente que faz a descoberta de composições

paralelas. Caso contrário, ele a repassa para o componente que faz a descoberta de composições seqüenciais.

3.5.2.2 A descoberta de composições paralelas

Existem situações onde nenhum serviço, sozinho, consegue atender à solicitação do usuário. Neste caso, *WebS Composer* verifica se existem serviços que, juntos, conseguem satisfazê-la. A esta tarefa de combinar dois ou mais serviços para atender uma requisição do usuário dá-se o nome de composição de serviços. Neste trabalho, foram identificados dois tipos de composições: paralela e seqüencial.

A composição paralela ocorre quando não temos nenhum serviço capaz de atender sozinho a todas as requisições do usuário, mas temos dois ou mais serviços que, juntos, em paralelo, podem realizar toda a tarefa. Neste caso, cada serviço se responsabiliza por resolver uma parte da solicitação do usuário.

Serviço	Entradas	Saídas
S_1	C_1	C_2
S_2	C_1	C_3
S_3	C_2	C_4
S_4	C_4	C_5

Tabela 2: Exemplos de serviços

Para explicar melhor este tipo de composição, considerem-se os serviços descritos na Tabela 2. Supõe-se que o usuário entrou com uma requisição onde a entrada corresponde ao conceito C_1 e as saídas desejadas correspondem aos conceitos C_2 e C_3. Observando a Tabela 2, pode-se perceber facilmente que não existe nenhum serviço capaz de atender sozinho a esta requisição. Por outro lado, se os serviços S_1 e S_2 forem executados em paralelo, a requisição do usuário pode ser totalmente satisfeita. Neste caso, o primeiro serviço fica responsável por produzir a primeira saída desejada, enquanto que o segundo serviço fica responsável por produzir a segunda saída.

A descoberta deste tipo de composição no sistema é feita por um componente chamado *Parallel Composition Discoverer*. Para acelerar o processo de descoberta destas

composições, este componente utiliza o relatório produzido pelo *Simple Service Discoverer*. Desta forma, ele pode descobrir as composições paralelas, caso existam, sem a necessidade de uma nova consulta aos serviços cadastrados na base de dados. A representação do casamento de serviços através de vetores facilita muito a descoberta de composições deste tipo.

O algoritmo desenvolvido neste trabalho para a descoberta de composições paralelas é bastante simples. Na sua primeira etapa, o componente responsável pela descoberta deste tipo de composição extrai do relatório gerado pelo *Simple Service Discoverer* os resultados de serviços cuja saída casou parcialmente com a saída da requisição do usuário. Estes resultados são adicionados a um conjunto de resultados parcialmente compatíveis, que será chamado de SPC.

Na segunda etapa, ele cria um vetor V cujo número de dimensões corresponde ao número de saídas da requisição do usuário. Para cada dimensão deste vetor, é associada uma coleção de serviços. Para cada dimensão, observa-se o vetor de casamento de saída de cada resultado presente no conjunto SPC. Se o vetor apresentar para a dimensão que está sendo analisada um valor de similaridade diferente de incompatível, que corresponde ao valor 0, o serviço associado a este resultado é acrescentado à coleção de serviços da dimensão.

Por exemplo, suponha que os serviços associados a um conjunto SPC tenham apresentado os seguintes vetores de casamento de saída para uma requisição do usuário, que solicita três saídas.

$S_1 = (0, 1, 0)$
$S_2 = (1, 0, 0)$
$S_3 = (0, 0, 1)$
$S_4 = (0, 0.8, 0.8)$

Ao aplicar o passo acima do algoritmo a estes resultados, pode-se observar facilmente que apenas o serviço S_2 apresentou um valor diferente de zero para a primeira dimensão, logo apenas ele é associado ao conjunto de serviços da mesma. Já para a segunda dimensão, dois serviços foram encontrados, S_1 e S_4, que são adicionados aos serviços da

mesma. O mesmo procedimento é aplicado à terceira dimensão. Ao fim deste passo, o vetor V encontrado terá a seguinte forma:

$V = (\{S_2\}, \{S_1, S_4\}, \{S_3,S_4\})$

O último passo do algoritmo consiste em gerar todas as combinações de resultados possíveis para o vetor, associando-se apenas um serviço para cada dimensão. Aplicando este passo para o vetor acima, as seguintes combinações são encontradas:

$C_1 = (S_2, S_1,S_3)$
$C_2 = (S_2, S_1, S_4)$
$C_3 = (S_2, S_4, S_3)$
$C_4 = (S_2, S_4, S_4)$

Para cada combinação encontrada, o *Parallel Composition Discoverer* cria um objeto chamado de *Split*, que representa uma composição de serviços que devem ser executados em paralelo. Para cada s*plit* gerado, ele adiciona os serviços que compõem a combinação. Uma vantagem deste tipo de objeto é que, apesar de representar uma composição de serviços, ele pode ser visto "atomicamente" como um conjunto de entradas e saídas, assim como os serviços simples. Desta forma, o mesmo algoritmo de *matchmaking* que é aplicado para casar serviços atômicos pode ser aplicado com objetos deste tipo. No caso de um s*plit*, o seu conjunto de entradas corresponde à união das entradas de todos os seus componentes, enquanto que o seu conjunto de saídas corresponde à união das saídas de cada um dos seus componentes.

Para cada *split* gerado, ele invoca o algoritmo de *matchmaking*, casando cada uma destas composições com a requisição do usuário. Depois, ele coleta os resultados parciais e cria um relatório semelhante ao gerado pelo *Simple Service Discoverer*, contendo as características do casamento de cada composição paralela descoberta com a requisição do usuário. Este relatório é enviado para o *QueryManager*.

O algoritmo que mostra a descoberta deste tipo de composição de serviços é descrito abaixo. Ele produz como resposta um relatório contendo os resultados encontrados para este tipo de composição. As informações que compõem a sua entrada são:

- A requisição de serviço passada pelo usuário, representada pela variável R;
- O relatório contendo os resultados encontrados pelo componente que faz a descoberta de serviços simples, representado pela variável Rel;
- O nível de similaridade mínimo definido para a consulta, representado pela variável t;

O algoritmo que descreve a descoberta destas composições é o seguinte:

```
encontrarComposicaoParalela (R, Rel, t): Relatorio;
SPC := rel.saidasParcialmenteSimilares;
V := criaVetor (R.numeroDeSaidas);
for k:=1 to SPC.tamanho do
        vetorDeSaida := SPC[k].vetorDeCasamentoDeDaida;
        for i:= 1 to V.tamanho do
                if vetorDeSaida[i] >0 then V[i].add(SPC[k].serviço);
splits = gerarTodasAsCombinacoes(V);
for k:= 1 to splits.tamanho do
      resultadoParcial := MatchMaker.match(R, splits[k]);
      resultadoFinal.add(resultadoParcial);
return gerarRelatorio(resultadoFinal, threshold);
```

3.5.2.3 A descoberta de composições seqüenciais

A composição seqüencial de serviços ocorre quando não temos um serviço que atenda sozinho à requisição do usuário, mas temos dois ou mais serviços que, executados numa certa ordem, permitem que a solicitação do usuário seja totalmente satisfeita.

Para entender melhor este tipo de composição, suponha os serviços descritos na Tabela 2. Vamos supor que o usuário solicitou um serviço que receba como entrada o conceito C_1 e ofereça como saída o conceito C_5. Observando a tabela, pode-se perceber que nenhum serviço atende sozinho a esta requisição, mas, se o os serviços S_1, S_3 e S_4, forem executados, exatamente nesta ordem, a solicitação do usuário será totalmente satisfeita.

Para descobrir este tipo de composição, alguns algoritmos propostos usam uma abordagem chamada de *forward chaining* (KVALOY et al, 2005). Esta técnica consiste em descobrir o primeiro serviço da seqüência que casa com a entrada da requisição e, a cada passo, descobrir os serviços que podem ser "plugados" nela, até que seja adicionado a ela um serviço cuja saída seja compatível com a saída desejada pelo usuário. Outros trabalhos, no entanto, usam uma técnica chamada de *backward chaining* (KVALOY et al, 2005), que consiste em descobrir primeiro o último serviço integrante da composição e, a cada passo, descobrir os serviços anteriores que podem ser adicionados a ela, até que seja adicionado à seqüência um serviço cuja entrada seja compatível com a entrada da requisição do usuário. A solução proposta por este trabalho para a descoberta deste tipo de composição consiste em um algoritmo que combina estas duas técnicas, e começa a descobrir esta composição pelas pontas. Em cada interação, o algoritmo vai descobrindo os serviços que podem ser adicionados tanto ao inicio quanto ao final da seqüência até elas se encontrarem. Por esta característica, ele foi chamado de *back–forward chaining*.

O componente responsável por descobrir as composições seqüências é chamado de *Sequence Composition Discoverer*. Para acelerar o processo de descoberta, ele utiliza os relatórios produzidos pelo *Simple Service Discverer* e pelo *Parallel Composition Discoverer*. Para facilitar, o *QueryManager* junta estes dois relatórios produzidos num único relatório e o repassa para este componente.

A idéia do algoritmo de composição seqüencial é baseada em uma seqüência de coleções de serviços. Ao receber o relatório dos outros componentes, ele cria uma seqüência, que será chamada de G, e adiciona a ela um elemento chamado de ponte. Esta ponte representa a requisição de serviços que está sendo procurada. Depois, ele invoca uma função que tenta encontrar uma ou mais seqüências de serviços que satisfaçam a requisição. Encontradas as seqüências, ele invoca o algoritmo de *mathcmaking* para casar cada seqüência encontrada com a solicitação. Os resultados destes casamentos são usados para a produção do relatório que é repassado como resposta ao *Query Manager*.

O algoritmo de descoberta de composições seqüências é mostrado abaixo. Este algoritmo retorna um relatório contendo as informações descobertas para este tipo de composição. As informações que ele recebe como entrada são:

- A requisição de serviço passada pelo usuário, representada pela variável R;

- O nível de similaridade mínima exigida pelo usuário, representado pela variável t;
- Um relatório contendo os resultados obtidos pelo componente de descoberta de serviços simples e pelo componente de descoberta de composições paralelas, representado pela variável Rel;

```
encontrarComposiçãoSequencial(R, t, Rel): Relatório;
G := criarSequencia();
G.add("ponte");
seqüências : = encontrarSequencias (R, t, Rel, G);
resultado := criarColecao();
for k:=1 to seqüências.tamanho do
        resultadoParcial := MatchMaker.match(R, sequencias[k]);
       resultado.add(resultadoParcial);
return geraRelatorio (resultados, t);
```

O algoritmo responsável por encontrar as seqüências de serviços que satisfazem a requisição do usuário é dividido em quatro etapas. Em sua primeira etapa, ele verifica se alguma das suas condições de saída é satisfeita. Estas condições indicam quando o algoritmo deve parar de procurar seqüências e retornar uma coleção vazia, indicando que nenhuma seqüência pode ser encontrada. Ele toma uma ação deste tipo quando uma das três condições é satisfeita:

- O relatório mostra que não foi encontrado nenhum serviço que tem a entrada compatível com a entrada da requisição;
- O relatório indica que nenhum serviço produz a saída desejada pela requisição;
- A seqüência principal, que contém a ponte e as coleções de serviços, atingiu o seu tamanho máximo. Nesta ferramenta, foi decidido que o algoritmo deve parar de procurar serviços quando a seqüência principal tiver pelo menos dez componentes;

O pseudocódigo que representa esta etapa do algoritmo é o seguinte:

```
encontrarSequencia (R, Rel, t, G): Coleção;
if (Rel.entradasSimilares= vazio) or (Rel.saidasSimilares = vazio) or (G.tamanho >= 10)
      then  return empty;
```

Caso nenhuma condição de saída seja satisfeita, o algoritmo passa para a segunda etapa. Nesta etapa, ele verifica se o relatório recebido descobriu pelo menos um serviço que satisfaz à solicitação. Se o resultado for positivo, ele substitui a ponte pelos serviços encontrados.

Após substituir a ponte, o algoritmo gera um grafo a partir da seqüência de coleções que foi encontrada. Durante a geração do grafo, um arco que parte de um serviço S_1 para um serviço S_2 é inserido quando a saída do primeiro serviço é compatível com a entrada do segundo. As seqüências que satisfazem à requisição do usuário são obtidas através de todos os caminhos possíveis entre os nós iniciais do grafo e cada uma de suas folhas. O pseudocódigo desta etapa é o seguinte:

```
SPC := Rel.serviçosSimilares;
if  SPC <> vazio
      indice := G.indice("ponte");
      G[indice] := SPC;
      grafo := gerarGrafo(G);
      sequencias = grafo.gerarTodosOsCaminhos();
      return sequencias;
```

Caso nenhum serviço com estas características seja encontrado, ele gera dois conjuntos de serviços. No primeiro, que será chamado de EC (entrada compatível), ele adiciona todos os serviços cuja entrada seja compatível com a entrada da requisição. No segundo, chamado de SC (saída compatível), ele adiciona todos os serviços cuja saída seja compatível com a saída da requisição.

Ao encontrar estes dois conjuntos, ele gera todas as combinações de seqüência possíveis entre os serviços destes dois conjuntos, para verificar se existe alguma seqüência que satisfaça à requisição. Estas seqüências são representadas por um objeto chamado de *Sequence*. Assim como um *split*, um objeto *Sequence* também corresponde a uma composição de serviços. A diferença entre estes objetos é que em um *Sequence* os

componentes são executados em seqüência. Uma seqüência também pode ser vista como um conjunto de entradas e saídas e, assim, pode ser casada com outros serviços através do mesmo algoritmo de *matchmaking* usado para o casamento de serviços simples. No caso de uma seqüência, as entradas correspondem ao conjunto de entradas do primeiro componente, enquanto que as saídas correspondem ao conjunto de saídas do último componente.

Ao gerar estas seqüências, o *Sequence Composition Discoverer* testa se as mesmas são consistentes. Ele considera que uma seqüência de serviços é consistente quando todas as saídas de um serviço são compatíveis com a entrada do serviço subseqüente. Caso ele encontre alguma seqüência consistente entre as combinações geradas, ele casa cada seqüência consistente encontrada com a requisição do usuário e, com o resultado destes casamentos, gera um relatório e o envia como resposta para o *Query Manager*.

O algoritmo formal que mostra esta etapa é o seguinte:

```
EC := Rel.entradasSimilares;
SC := Rel.saídasSimilares;
sequencias := gerarTodasAsSequencias(EC, SC);
sequenciasConsistentes := sequenciasConsistentes(seqüências);
if sequenciasConsistentes <> vazio
        indice := G.indexOf("bridge");
        G[indice] := sequenciasCosnistentes;
        grafo := gerarGrafo(G);
        sequencias = grafo.gerarTodosOsCaminhos();
        return sequencias;
```

Se nenhuma seqüência consistente for encontrada, o algoritmo avança então para a sua quarta etapa, onde ele expande a seqüência principal. Ele adiciona à esquerda da ponte os serviços do conjunto EC. Já os serviços do conjunto SC são adicionados à direita da ponte.

Feita a expansão da seqüência, o objetivo do algoritmo agora é tentar encontrar um ou mais serviços que possam substituir a ponte. Para isso, ele gera novas requisições. Para cada nova requisição, o conjunto de entradas disponíveis consiste da união das saídas de todos os serviços do lado esquerdo que estão diretamente ligados à ponte. A saída da requisição corresponde à entrada de um dos serviços do lado direito da ponte.

Depois, as novas requisições são enviadas para o *Simple Service Discoverer*, que produz um relatório indicando os resultados encontrados para cada requisição. O algoritmo combina todos estes relatórios num único relatório final, e verifica se foi encontrado pelo menos um serviço que atende a uma das requisições. Se a resposta for afirmativa, ele chama recursivamente a função que descobre uma seqüência, passando como informações as novas requisições que foram montadas, o relatório final encontrado, o *threshold* mínimo e o estado atual da seqüência.

Caso o relatório final mostre que nenhum serviço pode ser encontrado, o algoritmo encaminha as requisições para o *Parallel Composition Discoverer,* que tenta encontrar uma composição paralela para alguma das requisições. Ao concluir esta consulta, o algoritmo combina num único os relatórios produzidos pelo *Simple Service Discoverer* e pelo *Parallel Composition Discoverer.* Feito isso, ele chama recursivamente a função que descobre a seqüência de serviços, passando como parâmetro as novas requisições criadas, o relatório que combina os resultados encontrados pelos dois componentes, o *threshold* mínimo e o estado atual da seqüência. Este procedimento se repete até que o algoritmo encontre pelo menos um serviço que possa substituir a ponte, ou até que uma condição de saída seja satisfeita. O pseudocódigo que mostra este trecho do algoritmo é o seguinte:

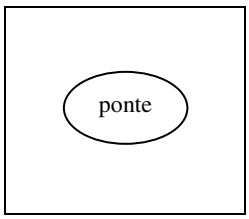
```
indice := G.indexOf(“ponte”);
G.add(SIS, indice);
G.add(SOS, indice+2);
novasRequisicoes := criarNovasRequisicoes(G);
for k:= 1 to novasRequisicoes.tamanho do
      relatorioSimples[k] := encontrarServicosSimples(novasRequisicoes[k], t);
relatorioSimplesCompleto : = combinarRelatorios(relatoriosSimples);
if relatorioSimplesCompleto.servicosSimilares <> vazio
      return encontrarComposicaoSequencial (novasRequisicoes, relatorioSimplesCompleto,
t, G);
for k:= 1 to novasRequisicoes.tamanho do
      relatorioParalelo[k] := encontrarComposicaoParalela (relatorioSimples[k],
novasRequisicoes[k]);
relatorioParaleloCompleto : = combinarRelatorios(relatorioParalelo);
relatorioCompleto := combinarRelatorios (relatorioSimplesCompleto,
relatorioParalelocompleto);
return encontrarComposicaoSequencial (novasRequisicoes, relatorioCompleto, t, G);
```

Para ilustrar a descrição do algoritmo, considere os serviços descritos na Tabela 3. Suponha que o usuário solicitou um serviço cuja entrada é o conceito C_1 e a saída desejada é o conceito C_7.

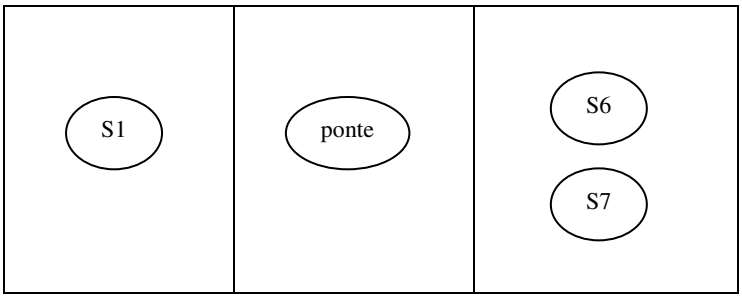
Serviço	Entradas	Saídas
S_1	C_1	C_2
S_2	C_2	C_3
S_3	C_3	C_4
S_4	C_4	C_5
S_5	C_4	C_6
S_6	C_5	C_7
S_7	C_6	C_7

Tabela 3: Exemplo de Serviços

No primeiro passo do algoritmo de descoberta de composição de serviços ocorre a criação da seqüência principal. O primeiro elemento a ser inserido nela é a ponte. Depois de criar a seqüência, o algoritmo de descoberta de seqüências é invocado. Quando o algoritmo é invocado, a seqüência tem a seguinte forma:



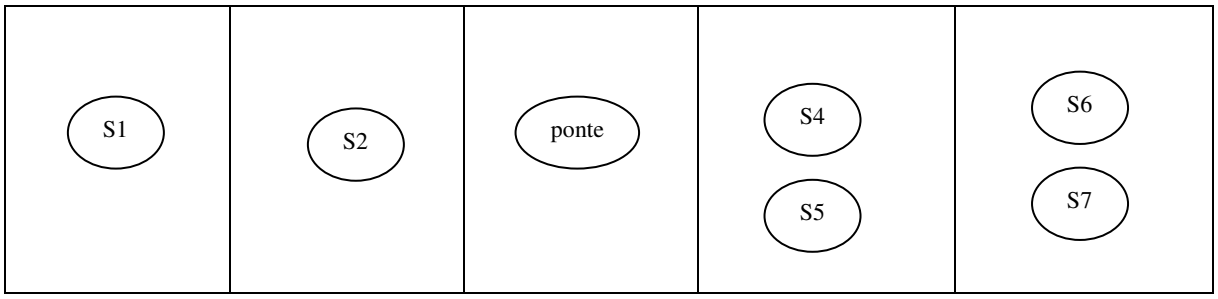
No primeiro passo do algoritmo, ele verifica que nenhuma das condições de saída foi satisfeita, pois existe pelo menos um serviço com entrada compatível com a requisição (S_1), dois serviços que produzem a saída desejada pelo usuário (S_6 e S_7) e que a seqüência principal não atingiu o tamanho máximo permitido. Então, ele avança para a segunda etapa e verifica se existe algum serviço que atenda sozinho a solicitação do usuário. Como nenhum serviço atende à requisição, ele vai para a próxima etapa e gera os conjuntos EC, que contém o serviço S_1, e o conjunto SC, que contém os serviços S_6 e S_7. Ao gerar estes conjuntos, ele verifica a consistência das seqüências $\{S_1, S_6\}$ e $\{S_1, S_7\}$, e percebe que as duas seqüências são inconsistentes. Desta forma, ele vai para a última etapa e expande a lista, adicionando à esquerda da ponte os serviços do conjunto EC, e à direita os serviços do conjunto SC. Ao fim desta iteração, a lista terá a seguinte forma:



Depois, o algoritmo cria novas requisições e consulta por serviços que tenham como entrada a saída do serviço S_1 e que tenham como saída as entradas dos serviço S_6 ou S_7. A primeira requisição tem como entrada a saída do serviço S_1 (C_2) e como saída a entrada do serviço S_6, que corresponde ao conceito C_5. A segunda também tem como entrada o conceito C_2, mas tem como saída a entrada do serviço S_7, que é o conceito C_6.

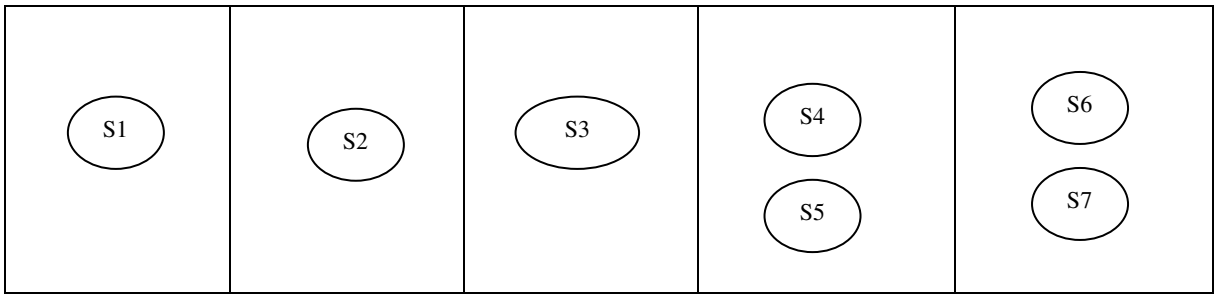
Casando os serviços com as duas novas requisições, ele não encontra nenhum serviço que case totalmente com nenhuma das duas. Então ele aplica recursivamente o algoritmo de descoberta de seqüências passando como informação as novas requisições, o relatório gerado para o casamento das mesmas, o grau de similaridade mínimo e o estado atual da seqüência principal.

Repetindo os mesmos passos da primeira interação a estas requisições, ele percebe que há serviços tanto para o conjunto EC (S_2) quanto para o conjunto SC (S_4, S_5), mas que não existe nenhuma seqüência consistente formada entre os elementos destes conjuntos. Desta forma, ele expande novamente a lista, adicionando à esquerda da ponte os serviços do conjunto EC e à direita os serviços do conjunto SC. Ao fim desta interação, a lista terá a seguinte forma:



Como o algoritmo ainda não encontrou nenhum serviço para substituir a ponte e não chegou a nenhuma condição de saída, o algoritmo cria novas requisições e faz uma

nova consulta. Ele agora procura por serviços que tenham como entrada o conceito C_3 (saída do serviço S_2) e que tenham como saída o conceito C_4 (entrada dos serviços S_4 e S_5). Ao executar esta nova consulta, ele descobre que existe um serviço S_3, que pode ligar a saída do serviço S_2 às entradas dos serviços S_4 e S_5. Desta forma, o serviço substitui a ponte e esta etapa do algoritmo de composição é encerrada. Ao fim da etapa, a lista terá a seguinte forma:



Terminada esta etapa, o algoritmo gera um grafo conectando os serviços encontrados na lista. Ao aplicar esta etapa para os resultados encontrados acima, ele vai encontrar o seguinte grafo mostrado na Figura 9.

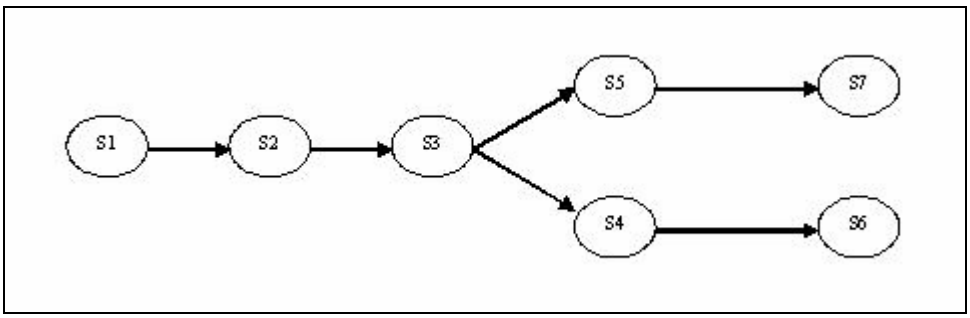


Figura 9: Grafo de conexão dos serviços encontrados

Encontrado o grafo de conexão, as seqüências que satisfazem à requisição do usuário correspondem a todos os caminhos existentes no grafo entre seu ponto inicial (S_1) e as suas folhas (S_6 e S_7). Desta forma, o *Sequence Composition Discoverer* vai retornar para

o *Query Manager* um relatório indicando que duas seqüências satisfazem a requisição do usuário. Estas seqüências são Seq1 = {S_1, S_2, S_3, S_5, S_7} e Seq2 = {S_1, S_2, S_3, S_4, S_6}.

A vantagem obtida com um algoritmo de composição deste tipo é a diminuição da quantidade de varreduras a serem feitas nos serviços. No exemplo acima, para descobrir as duas seqüências de cinco componentes, o algoritmo só precisou percorrer os serviços cadastrados na base de dados por três vezes. Um algoritmo que usa as abordagens *forward chaining* ou *backward chaining*, para descobrir estas duas seqüências, precisaria realizar cinco acessos à base de dados da ferramenta. Como o número de serviços cadastrados nas bases de dados de sistemas de recuperação de serviços tende a crescer, a diminuição do número de acessos aos serviços e, consequentemente do número de casamentos a serem realizados, pode significar uma grande melhora no desempenho do algoritmo de busca.

3.5.2.4 A consulta a provedores de serviços

Uma das características que diferenciam o *WebS Composer* de outros trabalhos existentes na área da recuperação e composição de serviços, é que, além de gerar uma marcação semântica para cada serviço, ele também gera outra descrição para os provedores de serviços. Esta característica é importante porque permite que a ferramenta ofereça para o usuário uma nova modalidade de busca, permitindo a recuperação de provedores de serviços que atendem a determinadas restrições, e também que a mesma faça uma seleção de serviços mais qualificada, baseada nas restrições e preferências do usuário em relação a esses provedores.

Uma vez que a ferramenta armazena cada provedor de serviço como uma instância de um conceito de uma ontologia de domínio, a máquina de busca do WebS Composer pode fazer inferências quanto às características destes provedores. Um tipo de busca simples oferecido pelo módulo de consulta permite que o usuário verifique quais provedores de serviços atendem a determinadas restrições. Neste tipo de consulta, o sistema leva em consideração apenas o tipo de estabelecimento que o cliente deseja e as características deste estabelecimento, desconsiderando os serviços oferecidos por estas empresas. Embora este tipo de consulta seja bastante simples, ele é importante por alguns motivos:

- Permite que usuários que, por algum motivo, queiram descobrir certos estabelecimentos independentemente deles oferecem ou não serviços on-line. Por exemplo, suponha que um cliente deseja viajar para uma determinada cidade e quer saber quais hotéis que se localizam nesta cidade oferecem piscina, boate e aceitam o seu cartão de crédito. Por questões de segurança, ele prefere entrar em contato com o hotel e fazer a reserva por e-mail ou telefone. Neste tipo de consulta, o sistema verifica todos os hotéis que atendem a estas características e os recupera para o usuário. Ao recuperar esta informação, o sistema mostra para o usuário as informações de contato de cada hotel recuperado. O cliente pode usar esta informação para entrar em contato com o estabelecimento escolhido e agendar a sua reserva. Para realizar uma consulta deste tipo numa ferramenta de busca tradicional, baseada em palavras-chaves, o cliente pode demorar muito tempo e ainda não encontrar exatamente a informação que deseja;
- Permite que um estabelecimento possa se cadastrar e ser descoberto por requisições do usuário mesmo que não ofereçam nenhum serviço on-line para os seus clientes;
- Permite um melhor refinamento dos serviços recuperados, pois permite que os serviços oferecidos por empresas que atendem às restrições impostas pelo usuário possam ser mostradas primeiro quando o resultado final for exibido.

Este tipo de consulta é feito de uma forma bastante simples. Ao realizar a consulta, o usuário indica qual o tipo de estabelecimento que ele está procurando e quais as restrições que ele deseja que esta empresa atenda. Para que a consulta seja mais flexível, para cada restrição imposta ele pode associar um nível de relevância, que indica o quanto ela é importante para ele. Os três tipos de relevância permitidos pela ferramenta são:

- **Indispensável:** corresponde ao maior nível de relevância e é excludente, já que as empresas que não atendem a esta restrição são descartadas do resultado final;

- **Importante:** corresponde a uma restrição que tem um certo peso para o usuário, mas que não é excludente, ou seja, serviços que não atendam esta restrição podem ser recuperados;
- **Preferível:** corresponde a características que o usuário prefere, mas que tem pouca relevância para ele. Este é o menor nível de relevância permitido pela ferramenta e, assim como o nível importante, não é excludente.

Como já foi dito, ao realizar uma busca por provedores de serviços o usuário indica qual o tipo de estabelecimento desejado e quais as suas restrições, com seus respectivos graus de relevância. Recebida estas informações, o *Query Manager* cria uma requisição e a repassa para um componente chamado de *Instance Selector*. Este componente consulta a ontologia de domínio ao qual o tipo de estabelecimento pertence e recupera todas as instâncias do conceito que representa o tipo de organização desejado. Além disso, ele recupera instâncias de conceitos similares, que correspondem aos sinônimos e às subclasses do conceito desejado. Para cada instância recuperada, ele invoca um componente chamado *Instance Match Maker*, que faz o casamento entre a empresa recuperada e a requisição do usuário.

O *matchmaking* entre a requisição do usuário e uma instância do sistema também é feita com o auxílio do modelo vetorial. Ao receber os dados da requisição, o *Instance Match Maker* cria um vetor para representá-la. O número de dimensões deste vetor corresponde ao número de restrições impostas pelo usuário àquele conceito. Cada uma destas dimensões representa uma das restrições do usuário. O valor de cada dimensão deste vetor corresponde ao nível de relevância da restrição. Na ferramenta, cada grau de relevância é representado por um valor numérico. O nível indispensável tem peso 1, o nível importante tem peso 0.8 e o nível preferível tem peso 0.4.

Desta forma, se o usuário emite uma consulta onde ele escolhe um certo tipo de estabelecimento e impõe 4 restrições R_1, R_2, R_3 e R_4, cujos graus de relevância são, respectivamente, indispensável, importante, indispensável e desejável, a sua requisição será representada pelo seguinte vetor R = (1, 0.8, 1, 0.4), onde a primeira dimensão se refere à restrição R_1 e as demais se referem, respectivamente, às restrições R_2, R_3 e R_4.

Para cada instância, ele cria um vetor com o mesmo número de dimensões do vetor que representa a requisição. Depois, para cada dimensão, ele verifica se a instância atende ou não a restrição referente àquela dimensão. Caso a instância atenda à restrição avaliada, ele atribui à dimensão o valor do grau de relevância da mesma. Caso contrário, ele atribui à dimensão o valor zero. Para facilitar o entendimento do algoritmo, suponha que o valor das instâncias recuperadas corresponde aos valores mostrados na Tabela 4:

Instância	**R_1**	**R_2**	**R_3**	**R_4**
I_1	sim	sim	sim	não
I_2	sim	sim	não	sim
I_3	sim	não	sim	sim
I_4	não	sim	sim	sim

Tabela 4: Exemplos de instâncias de provedores de serviços

Aplicando esta etapa do algoritmo às instâncias acima, o algoritmo vai obter o seguinte vetor para cada instância:

$I_1 = (1, 0.8, 1, 0)$
$I_2 = (1, 0.8, 0, 0.4)$
$I_3 = (1, 0, 1, 0.4)$
$I_4 = (0, 0.8, 1, 0.4)$

Uma vez que o vetor de casamento entre a requisição e a instância é encontrado, o *Instance Match Maker* calcula o grau de similaridade entra as duas. Neste tipo de casamento, esta medida é calculada através da divisão da soma das dimensões do vetor que representa a instância pela soma das dimensões do vetor que representa a requisição. Depois ele encapsula estas informações num resultado parcial e encaminha o resultado de volta ao *Instance Selector.* O resultado enviado para este componente encapsula as seguintes informações:

- A instância ao qual o algoritmo foi aplicado;
- O vetor resultante do casamento da instancia com a requisição;

- O grau de similaridade obtido com o casamento;

Após receber o resultado do casamento de cada instância, o *Instance Selector* filtra os resultados encontrados, removendo os resultados que apresentaram o valor zero para alguma das dimensões cujo peso era "indispensável". Os resultados restantes são encaminhados como resposta ao *QueryManager*.

3.5.2.5 A busca combinada

A busca combinada permite que o usuário combine os dois tipos de buscas simples, ou seja, permite que ele escolha serviços e ainda imponha restrições quanto às organizações que oferecem estes serviços.

Nesta tipo de busca, o usuário deve passar para a máquina de busca do sistema as seguintes informações:

- As entradas que ele tem disponível para o serviço;
- As saídas que ele deseja para o serviço;
- O tipo de provedor que ele deseja para o serviço;
- As restrições impostas a este provedor;
- A similaridade mínima desejada;

Este tipo de consulta é realizado em duas partes. Primeiro, o sistema resolve a parte relacionada às restrições aos provedores de serviços. Esta parte é resolvida primeiro para reduzir o espaço de busca da segunda etapa e, consequentemente, melhorar o desempenho do algoritmo. Depois, o sistema soluciona a parte relacionada à descoberta de serviços.

Neste tipo de consulta, ao receber a requisição do usuário, o *QueryManage*r ativa o componente *Instance Filter* para filtrar todas as instâncias do conceito relacionado ao tipo de provedor desejado. Todas estas instâncias são armazenadas em um conjunto de instâncias similares. Logo depois, ele repassa ao *Instance Selector* a parte da requisição referente aos provedores de serviço. Com vimos, este componente verifica quais instâncias atendem às restrições do usuário. Estas instâncias são armazenadas em um conjunto de

instâncias de interesse. Depois, o *QueryManager* realiza uma subtração de conjuntos e gera um conjunto de provedores de serviços indesejáveis.

Depois de gerar esta lista, o *Query Manager* realiza uma busca simples por serviços, sendo que os serviços oferecidos por provedores indesejáveis são desconsiderados pelo sistema e não são passados para os componentes de descoberta de serviços e composições de serviços.

3.5.3 O módulo de execução

O módulo de execução de serviços é responsável pela execução dos serviços e das composições de serviços descobertas no módulo de consulta. Ele foi implementado de acordo com a arquitetura mostrada na Figura 10.

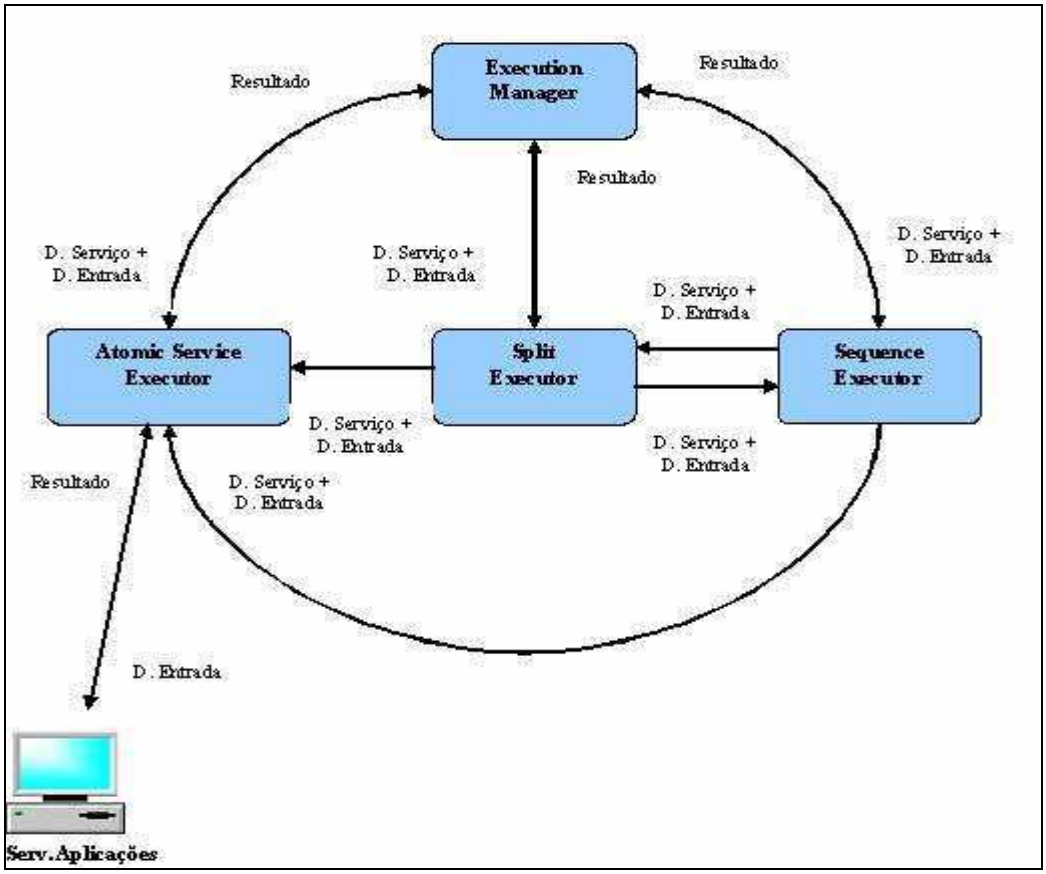


Figura 10: Visão Geral da Arquitetura do Módulo de Execução

Através da interface *web* o usuário do sistema seleciona qual o serviço (simples ou composto) que ele deseja executar. É através dela que o usuário entra com os valores de entrada para a invocação dos serviços. A interface recebe estas duas informações e as repassa para um componente chamado de *ExecutionManage*r.

O principal componente deste módulo é o *Execution Manager*, que faz a comunicação deste módulo com os demais módulos da camada de aplicação. Ele é responsável por receber uma ordem de execução de serviços e encaminhá-la para o executor de serviços apropriados.

Além do *Execution Manager*, este módulo conta com três outros componentes, que tem a função de acessar e invocar serviços. Cada um destes componentes é especialista em executar um tipo especial de serviço. São eles:

- ***Atomic Service Executor:*** este componente tem a função de invocar serviços simples (atômicos);
- ***Split Executor:*** este componente tem a função de executar uma composição paralela de serviços;
- ***Sequence Executor:*** este componente tem a função de executar uma composição seqüencial de serviços;

Para executar um serviço, ele deve informar ao *Execution Manager* o serviço que deve ser executado e o valor dos parâmetros de entrada. Quando o *Execution Manager* recebe algum serviço para ser invocado, ele analisa o tipo do serviço (se ele é um serviço simples, uma composição paralela ou uma composição seqüencial) e chama o executor apropriado para o serviço. Ao chamar o executor, ele passa para o mesmo as informações passadas pelo usuário.

O *Atomic Service Executor* é responsável por executar um serviço atômico. Quando ele recebe uma ordem de execução, ele acessa a URL que armazena o arquivo WSDL que descreve o serviço a ser executado, para pegar as informações necessárias para a execução do mesmo (parâmetros de entrada e saída, número da porta, etc). Ao verificar estas informações, ele invoca o serviço com os parâmetros de entrada recebidos. O resultado da invocação do serviço é repassado para o componente que solicitou a execução.

Quando o *Split Executor* recebe uma composição de serviços para executar, ele percorre cada componente da composição (que pode ser um serviço simples, um outro *split* ou uma seqüência) e solicita a execução do mesmo através do seu executor apropriado. Todos os componentes são executados em paralelo com o mesmo conjunto de dados de entrada. Depois que todos os componentes são executados, ele combina o resultado de cada serviço num único resultado e o devolve para o componente que solicitou a execução.

Quando o usuário solicita a execução de uma seqüência de serviços, ou o *Split Executor* precisa executar um componente que representa uma seqüência, a ordem de execução é repassada para o *Sequence Executor*. Ao receber estas informações, ele pega o primeiro elemento da seqüência a ser executado (que pode ser um serviço simples, um outro *split* ou uma subseqüência) e delega a sua execução para o executor apropriado. Depois que o componente é executado, ele usa os resultados de sua execução como a informação de entrada da execução do próximo componente. O procedimento se repete até que todos os componentes da seqüência tenham sido executados. O resultado da execução do último componente é repassado para o componente que solicitou a execução.

Quando o executor encerra a execução do serviço ou da composição, ele repassa o resultado de volta para o *Execution Manager.*

3.5.2.3 O módulo de administração do sistema

O módulo de administração do sistema permite que o administrador faça atualizações nos domínios de conhecimento da ferramenta. Dentre as funcionalidades que Webs Composer oferece para o mesmo estão:

- **A adição de novos domínios de conhecimento:** ele pode acrescentar novos domínios através de novas ontologias. Para adicionar uma ontologia, ele deve informar o nome do domínio descrito pela mesma e sua URL de acesso;
- **A atualização de um domínio de conhecimento já cadastrado:** ele pode atualizar um domínio já cadastrado quando uma nova ontologia for usada

para descrevê-lo. Para isso, ele deve informar o nome do domínio a ser atualizado e a URL da nova ontologia que será utilizada;

- **A exclusão de um domínio de conhecimento:** ele pode excluir um domínio cadastrado na ferramenta. Para isto, ele deve informar o nome do domínio a ser excluído;

Todas estas funcionalidades são oferecidas por um componente chamado de *System Manager*. Para usar cada uma das funcionalidades oferecidas pelo componente, o usuário deve ser autenticado para comprovar a sua situação de administrador.

3.6 A camada de dados

A camada de dados é responsável pelo armazenamento de todos os dados persistentes do sistema. Esta camada é composta por dois componentes. São eles:

- Um sistema gerenciador de banco de dados;
- Um serviço de diretório UDDI;

O sistema gerenciador de banco de dados é responsável pelo armazenamento das ontologias de domínio do sistema e das informações referentes aos serviços e provedores de serviços.

As ontologias de domínio, responsáveis por descrever os domínios de conhecimento da ferramenta, são armazenadas através do uso de um modelo de persistência de dados, que faz com que o *parsing* da mesma seja armazenado no SGBD. Esta forma de armazenamento é importante para oferecer uma melhorar a performance do sistema, pois evita que um novo *parsing* seja feito a cada vez que a ontologia é consultada. O *framework* Jena oferece a estrutura de persistência de dados usada pela ferramenta.

As informações referentes aos serviços e provedores de serviços são armazenadas em tabelas, usando o modelo relacional. O esquema usado para armazenar estas informações é bastante simples, sendo constituído apenas de duas tabelas. A primeira tabela é responsável por armazenar as informações referentes aos provedores de serviços,

enquanto que a segunda é responsável pelo armazenamento das informações dos serviços oferecidos por estes provedores.

O esquema destas duas tabelas é mostrado nas tabelas abaixo. A Tabela 5 mostra o esquema da tabela que armazena as informações referentes aos provedores de serviços, enquanto que a Tabela 6 mostra o esquema da tabela que armazena os serviços cadastrados no sistema.

Atributo	Tipo	Significado
Id	VARCHAR2	Identificação única do provedor de serviços na tabela
Nome	VARCHAR2	Nome da empresa
Descrição	VARCHAR2	Descrição das características da empresa
HomePage	VARCHAR2	HomePage oficial da empresa
Telefones	VARCHAR2	Telefones de contato da empresa
Emails	VARCHAR2	E-mails de contato da empresa
WSDL	VARCHAR2	URL do arquivo WSDL deste provedor

Tabela 5: Esquema da Tabela de Provedores de Serviços

Atributo	Tipo	Significado
Id	VARCHAR2	Identificação única do serviço na tabela
Nome	VARCHAR2	Nome do serviço
Descrição	VARCHAR2	Descrição textual do serviço
Provedor	VARCHAR2	Referência para o provedor do serviço
Marcação Semântica	VARCHAR2	Arquivo OWLS que descreve as características semânticas do serviço

Tabela 6: Esquema da Tabela de Serviços

Os *scripts* SQL usados para a criação das duas tabelas são os seguintes:

```
create table provedores(
    id VARCHAR2(100),
    nome VARCHAR2(50),
    descricao VARCHAR2(150),
    homepage VARCHAR2(50),
    telefones VARCHAR2(50),
    emails VARCHAR2(50),
    wsdl VARCHAR2(50),
    PRIMARY KEY(id)
)
```

```
create table services (
     id NUMBER,
   provedor VARCHAR2(50),
   nome VARCHAR2(50),
   descricao VARCHAR2(150),
   marcacaoSemantica XMLTYPE,
   PRIMARY KEY(id),
   FOREIGN KEY(provider) REFERENCES
providers(id)
)
```

O outro componente responsável pelo armazenamento de dados do sistema é o serviço de diretório UDDI. Ele armazena as informações tradicionais dos serviços e provedores de serviços armazenados nestes serviços. Dentre as informações armazenadas por este tipo de serviço, pode-se citar: o nome do provedor, sua descrição, suas informações de contato (e-mail e telefone), o nome dos serviços oferecidos e uma descrição textual dos mesmos.

Analisando os esquemas das tabelas armazenadas na base de dados do sistema, pode-se perceber que ele armazena uma boa parte das informações que já são armazenadas por este serviço de diretório. Esta característica permite que o sistema recupere estas informações para o usuário sem a necessidade de consultar este serviço. Para a implementação deste serviço de diretório, foi utilizado o *framework Java Web Services Development Pack*, versão 1.*5,* que disponibiliza um software chamado de *Registry Server*, que simula este serviço de diretório.

3.7 Analisando o desempenho de WebS Composer

Uma das características oferecidas por WebS Composer corresponde à descoberta de composições seqüenciais de serviços, onde uma série de serviços devem ser executados em uma determinada ordem para satisfazer à solicitação de um usuário. Para descobrir este tipo de composição de serviços, ele usa um algoritmo que foi chamado *back-and-forward chaining*, que tem como principal característica a descoberta, em cada iteração com os serviços da base de dados, dos serviços que podem ser adicionados tanto no início quanto

no fim da seqüência. Esta abordagem tem como objetivo reduzir o número de acessos feitos a base de dados e, assim, melhorar o desempenho do algoritmo.

Para analisar o desempenho desta abordagem, foi feito um teste para comparar o seu desempenho com o desempenho de algoritmos similares que usam as abordagens tradicionais (*backward chaining e forward chaining),* onde a descoberta da seqüência ocorre linearmente em apenas um sentido. Para isso, implementamos estes algoritmos tradicionais usando a mesma abordagem de WebS Composer, baseada no modelo vetorial.

No primeiro teste feito, foi comparado o tempo dos dois algoritmos para a descoberta de uma seqüência de dez serviços, a medida que o número de serviços cadastrados na base de dados aumentava.

Como os dois algoritmos tradicionais são similares, já que os dois realizam a descoberta de seqüências em um único sentido, diferindo apenas o sentido adotado pelos mesmos, eles apresentaram desempenhos semelhantes, cuja variação do tempo é desprezível.

Os resultados colhidos da execução dos algoritmos para este teste são mostrados no gráfico apresentado na Figura 11. A série que mostra o desempenho dos algoritmos tradicionais foi gerada através da média do tempo gasto pelos dois algoritmos. Os testes foram realizados em uma máquina *desktop*, com processador Athlon de 2.4 GHz e com 1 GB de memória RAM. O desempenho dos algoritmos foi medido em segundos:

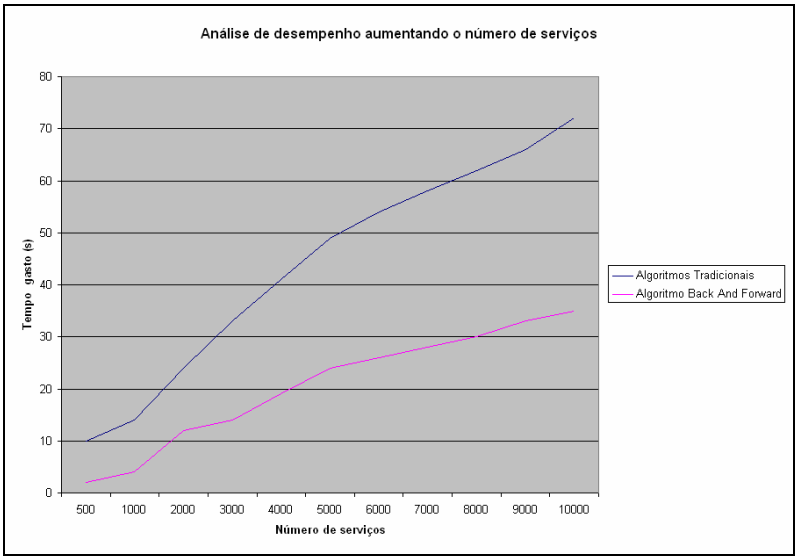


Figura 11: Resultado da execução dos algoritmos com seqüências de tamanho fixo

O gráfico mostra que, a medida que o número de serviços aumenta, o tempo gasto pelo algoritmo *back-forward* de WebS Composer tende a ser a metade do tempo gasto pelo outro algoritmo, já que, como ele descobre dois componentes da seqüência em cada iteração, o número de acessos feitos aos serviços da base de dados cai pela metade.

No segundo teste realizado, foi comparado o tempo gasto pelos dois algoritmos a medida que o tamanho da seqüência descoberta ia aumentando, mantendo fixo o número de serviços da base de dados. Executando os algoritmos para descobrir estas seqüências numa base de dados com dez mil serviços, chegou-se ao seguinte resultado, mostrado no gráfico da Figura 12. A série que mostra o desempenho dos algoritmos tradicionais foi gerada através da média do tempo gasto pelos dois algoritmos.

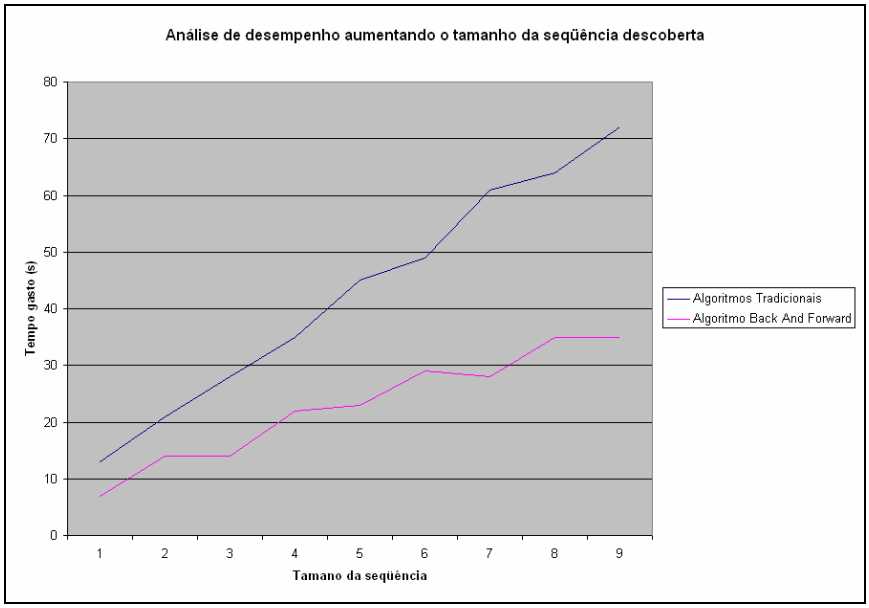


Figura 12: Resultado da execução dos algoritmos variando o tamanho das seqüências

Analisando o gráfico acima, percebemos que, a medida que o tamanho da seqüência aumenta, o tempo gasto pelo algoritmo de WebS Composer também é muito inferior em relação ao outro algoritmo, pelos mesmos motivos citados acima. Como o número de serviços publicados tende a crescer cada vez mais, este algoritmo surge como uma alternativa muito interessante para a descoberta de composições deste tipo.

3.8 Considerações Finais

Este capítulo apresentou WebS Composer, uma ferramenta utilizada para a descoberta de serviços e composições de serviços na *web*. Ela pode ser utilizada por usuários e por provedores de serviços. Os provedores de serviços podem utilizar esta ferramenta para registrar as informações referentes à sua empresa e para cadastrar os serviços que eles oferecem. Os usuários, que podem ser pessoas ou uma outra aplicação, podem usá-la para localizar e executar os serviços cadastrados na sua base de dados. Uma característica importante deste sistema é que ele gera uma descrição semântica para cada provedor de serviços, oferecendo uma nova modalidade de busca para estes usuários, onde eles podem localizar empresas que satisfazem a certas restrições, e oferecendo uma seleção de serviços mais flexível. Todas estas descobertas de serviços são feitas com o uso do modelo vetorial, um dos modelos clássicos da recuperação de informação.

WebS Composer permite a descoberta de dois tipos de composições de serviços: paralela e seqüencial. A composição paralela ocorre quando os seus componentes são serviços que podem se executados ao mesmo tempo. Neste tipo de composição, cada serviço é responsável por desempenhar uma parte da tarefa solicitada pelo usuário. Alguns trabalhos propostos para resolver o problema da composição automática de serviços não levam em consideração este tipo de composição, o que representa uma fragilidade para os mesmos. A abordagem proposta por este trabalho permite que este tipo de composição seja descoberto de uma forma bastante simples. Além disso, ele oferece meios para verificar qual combinação de serviços oferece uma melhor cobertura para as necessidades do usuário. Já a composição seqüencial acontece quando os seus componentes precisam ser executados em uma determinada ordem para satisfazer as necessidades do usuário. Este trabalho apresenta um novo tipo de algoritmo para descobrir este tipo de composição, cujo desempenho é superior aos algoritmos usados nas abordagens tradicionais. A consulta baseada nas características dos provedores de serviços oferecida pela ferramenta não é explorada por nenhum dos trabalhos apresentados. A Tabela 7 mostra um quadro comparativo entre os trabalhos apresentados no segundo capitulo e WebS Composer.

Trabalho	Composição Automática	Composição Paralela	Composição Seqüencial	P.Condições e Efeitos	Ranking	Restrição Provedor
Aversano et al	sim	sim	sim	não	sim	não
Costa et al.	sim	sim	sim	sim	não	não
Gekas & Fasli	sim	não	sim	não	sim	não
Kvaloy	sim	não	sim	não	sim	não
Medjahed et al.	sim	não	sim	não	sim	não
Milanovic	não	sim	sim	sim	não	não
IRS-III	não	sim	sim	sim	não	não
Sell et al.	não	sim	sim	sim	não	não
Compat	não	não	não	não	sim	não
WebS Composer	sim	sim	sim	não	sim	sim

Tabela 7: Quadro comparativo entre os trabalhos apresentados e WebS Composer

CAPÍTULO IV – UM ESTUDO DE CASO

Agora que já descrevemos *WebS Composer*, e todas as suas funcionalidades, vamos apresentar um estudo de caso usado para a ilustrar seu funcionamento. Como a principal contribuição deste trabalho é a descoberta e composição de serviços baseadas em restrições do usuário, nossos exemplos vão se limitar a demonstração dos algoritmos de busca mostrados no capítulo anterior.

O estudo de caso utilizado se refere ao domínio de viagens. Neste caso, a ferramenta permite que um cliente, que vai fazer uma viagem, descubra e invoque serviços referentes aos locais que podem ser utilizados para a sua hospedagem e para a sua alimentação. O estudo de caso é composto de serviços que permitem que um usuário reserve um local para dormir, reserve um local para comer, verifique os preços destes locais e verifique se eles têm ou não disponibilidade em uma determinada data.

4.1 Ontologias para viagens

Para demonstrarmos os algoritmos de busca por serviços e composições, definimos algumas ontologias, que descrevem parcialmente os domínios de hospedagem, alimentação e clientes. É importante ressaltar que as ontologias usadas neste estudo de caso são bastante simples, tendo em vista que elas servirão apenas para demonstrar a nossa abordagem para resolver o problema da descoberta e composições de serviços. Para uma verdadeira implantação da ferramenta precisamos de ontologias mais completas, que descrevam as características destes domínios de aplicação de uma forma detalhada.

As ontologias que utilizamos neste estudo de caso são mostradas nas figuras abaixo. A Figura 13 mostra uma ontologia para o domínio de hospedagens, enquanto que a Figura 14 e a Figura 15 mostram, respectivamente, as ontologias para os domínios de alimentação e de cliente. Estas ontologias serão chamadas, respectivamente, de *Hotel*, *Restaurant* e *Client*.

Nestas ontologias, os retângulos representam os conceitos (classes e atributos), os relacionamentos do tipo *"Datatype Property"* ligam um conceito a um atributo atômico (simples) e os relacionamentos do tipo *"Object Property"* associam um conceito a um

atributo representado por uma classe. O relacionamento de herança é definido através de um relacionamento do tipo *"is a"*, partindo do conceito mais específico para o conceito mais geral. O relacionamento *"equals"* indica que dois conceitos são sinônimos.

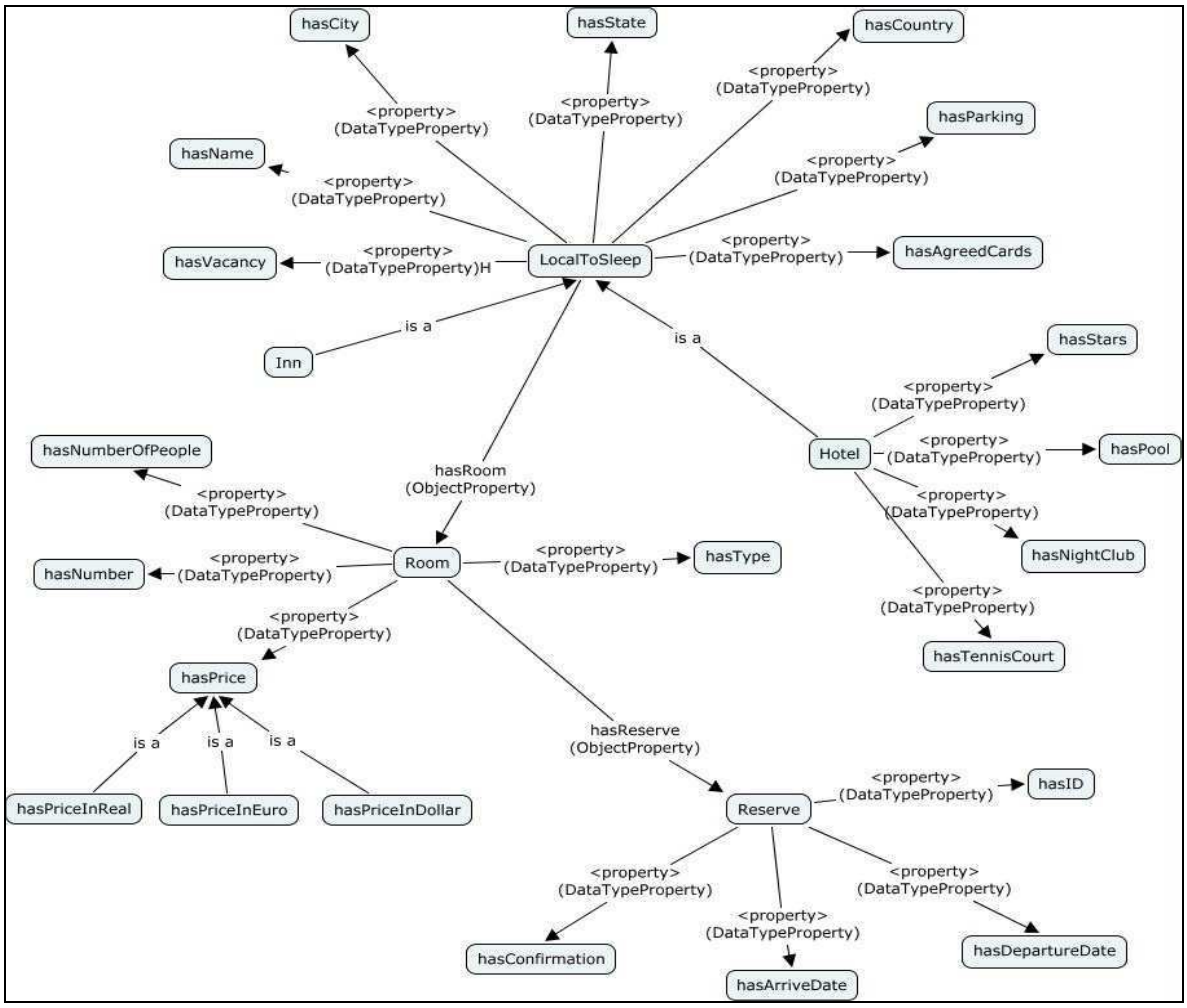


Figura 13: Representação parcial de uma ontologia para hospedagens

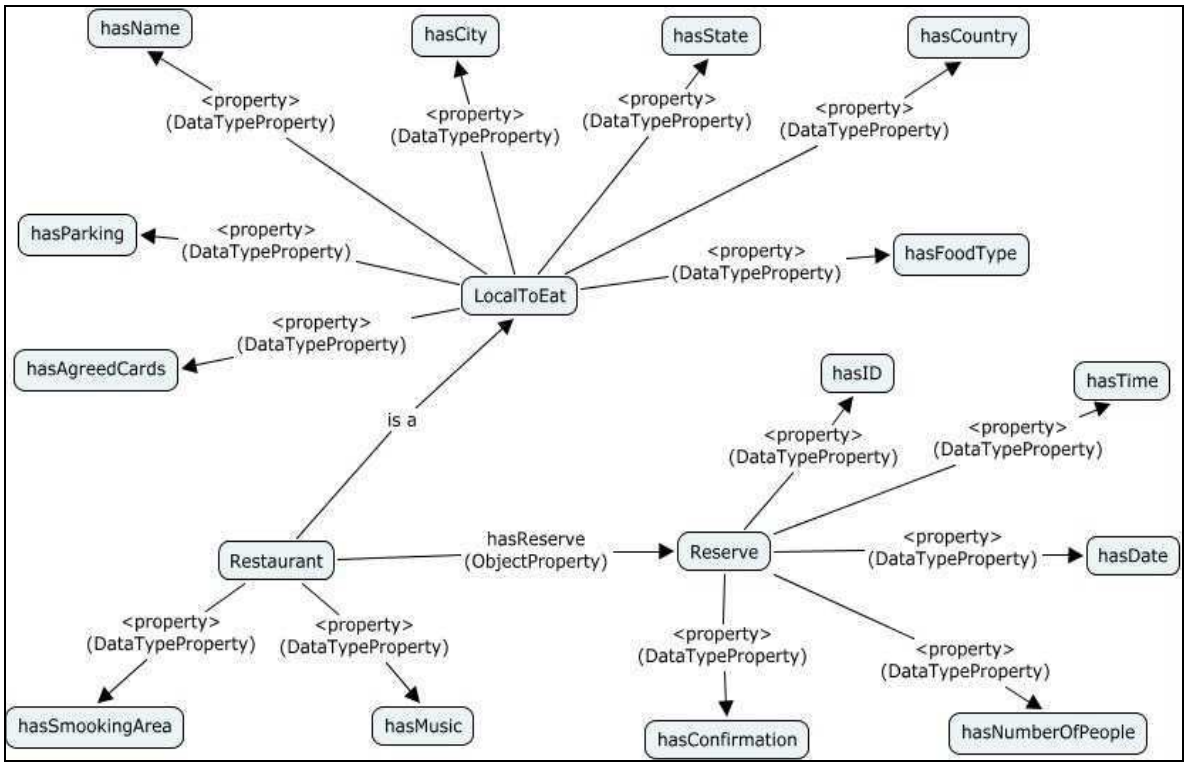


Figura 14: Representação parcial de uma ontologia para alimentação

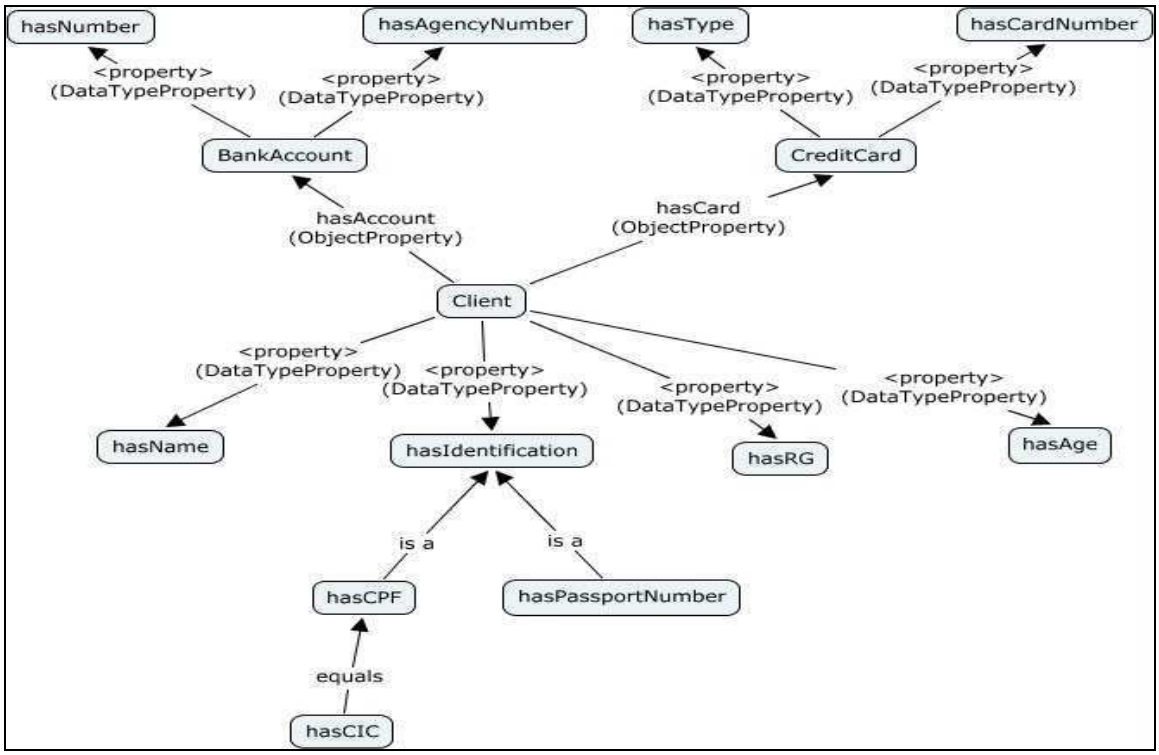


Figura 15: Representação parcial de uma ontologia para clientes

4.2 Os serviços

Ainda para ilustrar a nossa abordagem para a descoberta de serviços e composições, definimos uma relação de serviços que vão representar a base de dados local do sistema. Para cada serviço, associamos todas as suas entradas e saídas a conceitos definidos nas ontologias mostradas na seção anterior. Vale lembrar que cada organização pode disponibilizar qualquer quantidade de serviços. Todos os serviços e provedores usados no exemplo são hipotéticos. Os serviços que vão servir de exemplo para este estudo de caso são mostrados na Tabela 7.

4.3 Inserindo um novo provedor de serviços

A primeira funcionalidade de WebS Composer que vamos abordar neste estudo de caso é o cadastro de novos provedores de serviços. Nesta funcionalidade, o usuário do sistema é uma empresa que está querendo registrar as suas informações. Um provedor oferece *web services* que podem ser descobertos e invocados através da *web*. Cada provedor pode oferecer vários serviços.

O cadastro de novos serviços é feito em quatro etapas. Na primeira etapa, o provedor informa qual o tipo de estabelecimento corresponde a sua empresa. Por exemplo, uma empresa da área de hospedagem pode ser um hotel, uma pousada, uma pensão, etc. Cada tipo de estabelecimento é representado como uma classe na ontologia de domínio do sistema.

Na segunda etapa, ele oferece as informações que são tradicionalmente armazenadas em serviço de diretório UDDI convencional. Dentre estas informações, podemos citar o nome da empresa, sua descrição, telefone de contato e e-mail. Além dessas informações, outras características de contato devem ser passadas, como a sua *homepage* e a URL do seu arquivo WSDL.

Serviço	Nome	Provedor	Entradas	Saídas
S_1	bookRoom	Hotel Ouro Branco	Client#hasName Client#hasCPF Hotel#hasArriveDate Hotel#hasType	Hotel#hasConfirmation
S_2	getRoomPrice	Hotel Ouro Branco	Hotel#hasType	Hotel#hasPriceInReal
S_3	hasVacancy	Hotel Ouro Branco	Hotel#hasArriveDate Hotel#hasNnumberOfPeople	Hotel#hasVacancy
S_4	soliciteReserve	Garden Hotel	Client#hasName Client#hasCIC Hotel#hasArriveDate Hotel#hasType	Hotel#Reserve
S_5	assertPrice	Garden Hotel	Hotel#hasType	Hotel#hasPriceInReal
S_6	hasEnoughVacancy	Garden Hotel	Hotel#hasArriveDate	Hotel#hasVacant
S_7	effectReserve	Hotel Ritz	Client#hasIdentification Hotel#hasArriveDate Hotel#hasType	Hotel#hasConfirmation
S_8	hasVacancy	Hotel Ritz	Hotel#hasArriveDate Hotel#hasNumberOfPeople	Hotel#hasVacancy
S_9	soliciteReserve	Pousada Good Dreams	Client#hasIdentification Client#hasNname Hotel#hasArriveDate Hotel#hasType	Hotel#hasConfirmation
S_{10}	getPrice	Pousada Good Dreams	Hotel#hasType	Hotel#hasPriceInReal
S_{11}	hasVacancy	Pousada Good Dreams	Hotel#hasArriveDate Hotel#hasNumberOfPeople	Hotel#hasVacancy
S_{12}	bookTable	Restaurante Terra Chinesa	Client#hasName Restaurant# hasDate Restaurant#hasTime Restaurant#hasNumber OfPeople	Restaurant#hasConfirmation
S_{13}	bookTable	Restaurante Sertanejo	Client#hasName Restaurant#hasDate Restaurant#hasTime Restaurant#hasNumber OfPeople	Restaurant#reserveConfirmation
S_{14}	dollarToEuro	Cambio	Hotel#hasPriceInDollar	Hotel#hasPriceInEuro
S_{15}	realToDollar	Cambio	Hotel#hasPriceInReal	Hotel#hasPriceInDollar
S_{16}	euroToReal	Cambio	Hotel#hasPriceInEuro	Hotel#hasPriceInReal

Tabela 8: Exemplo de serviços usados no estudo de caso

Durante a terceira etapa deste processo, ele deve passar informações adicionais sobre sua empresa. O conjunto de informações a serem passadas diz respeito ao tipo de estabelecimento selecionado pelo usuário na primeira etapa. Por exemplo, se o estabelecimento corresponde a um hotel, ele deve informar características como o número

de estrelas, a cidade onde ele se localiza, os cartões de crédito aceitos e as facilidades oferecidas pelo mesmo (piscina, boate, estacionamento, etc). Estas informações correspondem às propriedades que a ontologia define para o tipo de estabelecimento que representa a empresa.

Na última etapa do registro de serviços, o provedor deve informar os serviços que a sua empresa oferece. Para cada serviço oferecido, ele deve informar o seu nome, a sua descrição, e os seus parâmetros de entrada e saída. Como mostramos no capítulo anterior, todos os parâmetros passados pelo mesmo correspondem a conceitos definidos na ontologia de domínio do sistema. Durante esta fase, ele pode cadastrar qualquer quantidade de serviços.

Ao fim desta etapa, o módulo de interface com o usuário encapsula estas informações em um objeto do tipo *Web Service* e o repassa para o módulo de registro. Ao receber esta informação, o módulo de registro gera duas marcações semânticas para a empresa cadastrada.

A primeira marcação gerada se refere à empresa. Como vimos anteriormente, ela é armazenada na ontologia de acomodações como uma nova instância do conceito que representa o tipo de estabelecimento da empresa.

Além da descrição semântica do provedor do serviço, o sistema gera uma marcação semântica para cada serviço oferecido pela empresa. Esta marcação semântica corresponde a um arquivo OWL-S, que descreve as características semânticas do serviço. A Figura 16 mostra a descrição OWL-S gerada pelo sistema para o serviço *bookRoom* oferecido pelo Hotel Ouro Branco, descrito na Tabela 7.

4.4 Realizando buscas em WebS Composer

Agora que já mostramos a inclusão de um novo *web service* no sistema, vamos mostrar como o sistema realiza as buscas do usuário. Primeiro, vamos mostrar exemplos da aplicação das buscas simples, onde o usuário seleciona serviços ou provedores de serviços. Depois, mostraremos como o sistema resolve buscas combinadas, que envolve os dois tipos de buscas simples numa única consulta.

```
<rdf:RDF   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<AtomicProcess rdf:id="bookRoom">
   <hasInput rdf:resource="reservaQuarto-i1"/>
   <hasInput rdf:resource="reservaQuarto-i2"/>
   <hasInput rdf:resource="reservaQuarto-i3"/>
   <hasInput rdf:resource="reservaQuarto-i4"/>
   <hasOutput rdf:resource="reservaQuarto-o1"/>
</AtomicProcess>
<Input rdf:id="reservaQuarto-i1">
    <serviceParameterName>i1</serviceParameterName>
    <sParameter>Client#hasName</sParameter>
</Input>
<Input rdf:id="reservaQuarto-i2">
    <serviceParameterName>i1</serviceParameterName>
    <sParameter>Client#hasCPF</sParameter>
</Input>
<Input rdf:id="reservaQuarto-i3">
    <serviceParameterName>i1</serviceParameterName>
    <sParameter>Hotel#hasArriveDate</sParameter>
</Input>
<Input rdf:id="reservaQuarto-i4">
    <serviceParameterName>i1</serviceParameterName>
    <sParameter>Hotel#hasType</sParameter>
</Input>
<Output rdf:id="reservaQuarto-o1">
    <serviceParameterName>o1</serviceParameterName>
    <sParameter>Hotel#hasConfirmation</sParameter>
</Output>

</rdf:RDF>
```

Figura 16: Exemplo de uma marcação semântica gerada por WebS Composer

4.4.1 Consultando por serviços

Nesta seção, vamos mostrar como WebS Composer auxilia o usuário a encontrar serviços que ofereçam uma determinada funcionalidade. Ao submeter uma consulta deste tipo, o usuário deve informar ao sistema quais as entradas que ele dispõe para o serviço e as saídas que ele deseja receber do serviço. Estes dois conjuntos de informações são representados por conceitos definidos na ontologia de domínio. Por exemplo, se ele deseja reservar um quarto para a sua viagem, ele pode informar que dispõe de todas as suas informações pessoais (nome, cpf, rg, etc) e do tipo de quarto que ele deseja reservar e que deseja um serviço que como saída ofereça uma confirmação da sua reserva.

4.4.1.1 Descoberta de serviços simples

Vamos supor que o usuário, em sua requisição, solicitou um serviço de reserva de quarto. Vamos supor que, para este serviço, ele dispõe das seguintes informações: o seu nome, o número do seu RG, o número do seu CPF, a data de sua chegada e o tipo de quarto desejado. Como saída, ele deseja uma confirmação da sua reserva. Vamos considerar que a similaridade mínima é de 80 %. Para realizar esta consulta, o sistema gera a seguinte requisição mostrada na Figura 17:

Figura 17: Requisição de serviços para uma busca por serviços simples

Ao receber esta requisição de serviço, o sistema vai compará-la com todos os serviços cadastrados na base de dados. Para cada serviço, ele analisa as entradas e saídas seguindo os algoritmos mostrados no capítulo anterior. Ao realizar o casamento entre esta requisição e cada serviço do sistema, ele vai perceber que, com o conjunto de entradas disponibilizadas pelo usuário, ele pode executar os serviços S_1, S_2, S_4, S_5, S_6, S_7, S_9 e S_{10}. Os serviços S_1, S_2, S_5, S_6 e S_{10} podem ser executados porque todas as suas entradas estão presentes na entrada disponibilizada do usuário. Os serviços S_7 e S_9 disponibilizam em seu conjunto de entradas um conceito chamado *hasIdentification*, que também tem um relacionamento exato com a requisição porque a mesma apresenta um conceito (*hasCPF*) que é uma subpropiedade deste conceito. Já o serviço S_4 apresenta em suas entradas um

conceito *hasCIC*, que também casa exatamente com a solicitação do usuário, que tem em suas entradas o conceito *hasCPF*, que é um sinônimo deste conceito. Além disso, ele percebe que os serviços que apresentam em sua saída um conceito relacionado ao conceito definido na saída da requisição do usuário são S_1, S_4, S_7 e S_9. Ao exibir o resultado final para o usuário, os serviços S_1, S_7 e S_9 são mostrados primeiro, pois suas saídas correspondem exatamente ao mesmo conceito solicitado pelo usuário, tendo assim, um relacionamento exato. Já o serviço S_4 também é recuperado, mas é mostrado por último, pois a sua saída representa um conceito que contém a saída desejada pelo usuário, apresentando, assim, um relacionamento do tipo *plug-in*. O resultado produzido por WebS Composer para esta consulta é mostrado na Figura 18:

Figura 18: Resultado para a descoberta de serviços simples

4.4.1.2 Descoberta de composições paralelas

Vamos supor que o usuário, agora, fez uma nova requisição. Agora, ele deseja encontrar um serviço que reserve um local para a sua hospedagem e um local para jantar em um dos dias de sua estadia. Para isso, ele dispõe das seguintes informações: o seu nome, o número do seu RG, o número do seu CPF, a data de sua chegada, o tipo de quarto

desejado, a data e a hora do jantar, e o número de pessoas que vão compor a mesa. Como saída, ele deseja a confirmação das duas reservas. Vamos considerar que a similaridade mínima desejada é de 80 %. Desta forma, a requisição do usuário terá a seguinte forma mostrada na Figura 19.

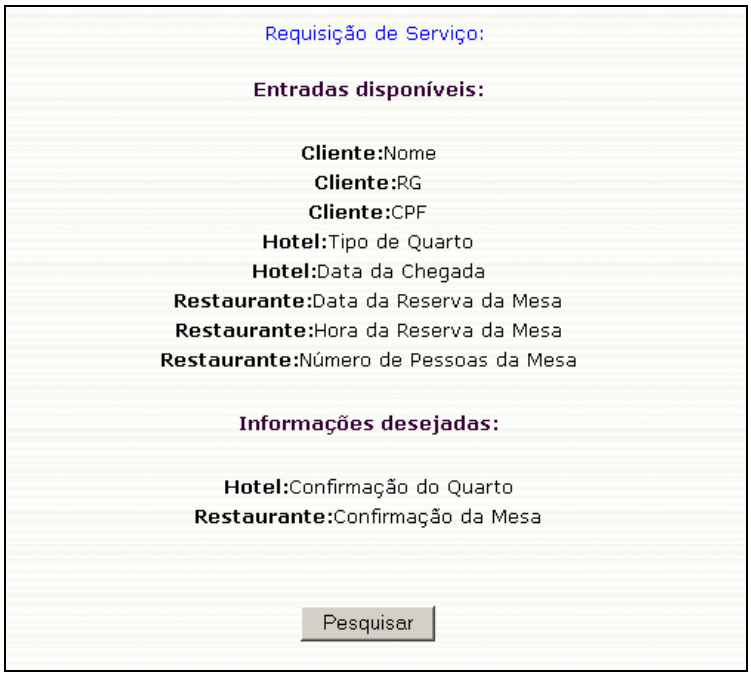


Figura 19: Requisição de serviço para uma composição paralela

Ao aplicar o algoritmo de descoberta de serviços simples para esta requisição, o sistema vai encontrar os seguintes resultados mostrados na Tabela 9. Ao analisar os resultados obtidos, o algoritmo percebe que nenhum dos serviços cadastrados na base de dados do sistema consegue atender sozinho à requisição do usuário. Desta forma, ele tenta montar uma composição paralela para satisfazer esta requisição.

Na primeira etapa do algoritmo de descoberta de composições paralelas, ele seleciona os serviços que apresentam uma saída parcialmente similar com a saída da requisição, ou seja, aqueles serviços que apresentam em seu vetor de casamento de saída pelo menos uma dimensão cujo valor seja diferente de zero. Aplicando esta etapa aos resultados mostrados na tabela, o algoritmo seleciona os serviços S_1, S_4, S_7, S_{11} e S_{12}.

Serviço	Vetor de entrada	Similaridade de entrada	Vetor de saída	Similaridade de saída	Similaridade global
S_1	(1, 1, 1, 1)	100 %	(1, 0)	70 %	70 %
S_2	(1)	100 %	(0, 0)	0 %	0 %
S_3	(1, 1)	100 %	(0, 0)	0 %	0 %
S_4	(1, 1, 1, 1)	100 %	(0.8, 0)	70 %	70 %
S_5	(1)	100 %	(0, 0)	0 %	0 %
S_6	(1)	100 %	(0, 0)	0 %	0 %
S_7	(1, 1, 1)	100 %	(1, 0)	70 %	70 %
S_8	(1, 1)	100 %	(0, 0)	0 %	0 %
S_9	(1, 1, 1, 1)	100 %	(1, 0)	0 %	0 %
S_{10}	(1)	100 %	(0, 0)	70 %	0 %
S_{11}	(1, 1)	100 %	(0, 0)	70 %	70 %
S_{12}	(1, 1, 1, 1)	100 %	(0, 1)	70 %	70 %
S_{13}	(1, 1, 1, 1)	100 %	(0, 1)	0 %	0 %
S_{14}	(0)	0 %	(0, 0)	0 %	0 %
S_{15}	(0)	0 %	(0, 0)	0 %	0 %
S_{16}	(0)	0%	(0, 0)	0 %	0 %

Tabela 9: Resultados encontrados com a descoberta de serviços simples

Selecionados os serviços, o algoritmo cria um vetor cujo número de dimensões é igual ao número de saídas desejadas pelo usuário. Como, nesta requisição, o usuário requisitou duas saídas, este vetor terá tamanho dois. Na etapa seguinte, o algoritmo associa para cada dimensão deste vetor os serviços que apresentaram em seu vetor de casamento de saída um valor diferente de zero para a mesma. Por exemplo, neste caso, ele associa à primeira dimensão os serviços S_1, S_4 e S_7, que apresentaram, em seus respectivos vetores, um valor diferente de zero para esta dimensão. Aplicando o mesmo passo para a segunda dimensão, o algoritmo associa a ela os serviços S_{11} e S_{12}.

Seguindo ainda o algoritmo de descoberta de composições paralelas, o sistema gera todas as combinações de serviços ligados às duas dimensões. Como vimos no capítulo anterior, o sistema cria um s*plit*, que representa uma composição de serviços que devem ser executados em paralelo, para armazenar cada uma destas combinações. Como temos três serviços associados à primeira dimensão e dois serviços associados à segunda, teremos,

neste exemplo, seis combinações geradas, que são $\{S_1, S_{11}\}$, $\{S_1, S_{12}\}$, $\{S_4, S_{11}\}$, $\{S_4, S_{12}\}$, $\{S_7, S_{11}\}$, $\{S_7, S_{12}\}$.

Após gerar todas as combinações, o sistema aplica o algoritmo de casamento entre a requisição do usuário e cada *split* gerado. Isto pode ser feito facilmente porque um split pode ser visto, assim como um serviço simples, como um conjunto de entradas e saídas. Por fim, o algoritmo de descoberta de composições paralelas gera um relatório a partir dos resultados encontrados para cada combinação e retorna-o para o *Query Manager*.

Neste exemplo, ao analisar o relatório produzido, o sistema percebe que as seis combinações geradas satisfazem à solicitação e exibe as combinações encontradas, em ordem decrescente de similaridade, para o usuário. O resultado encontrado pela ferramenta para esta consulta é mostrado na Figura 20.

Figura 20: Resultado para a descoberta de composições paralelas

4.4.1.3 Descoberta de composições seqüenciais

Vamos supor agora a seguinte consulta. O usuário vai viajar e deseja saber quais os preços das diárias de um quarto de solteiro em vários hotéis, mas, como o mesmo é europeu, ele deseja saber este preço em euros, que é a moeda corrente do seu país. Desta vez, a requisição, que tem similaridade mínima de 80%, é mostrada na Figura 21:

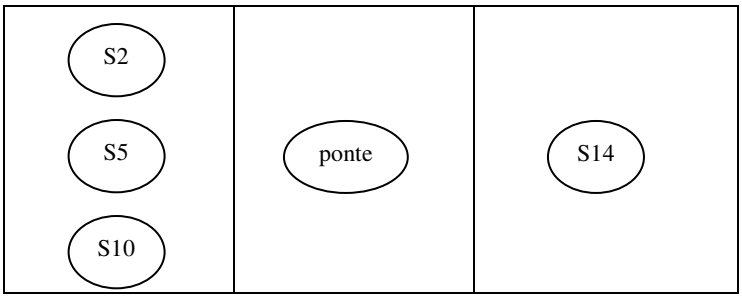
Figura 21: Requisição de serviços para uma composição seqüencial

Seguindo o algoritmo aplicado a todas as consultas, o sistema tenta primeiro encontrar um serviço simples que atenda à requisição do usuário. Ao aplicar o algoritmo de busca simples, ele percebe que o único serviço que produz a saída desejada pelo usuário é o serviço S_{14}, cuja entrada não é compatível com as informações oferecidas pelo usuário em sua requisição. O sistema tampouco pode encontrar uma composição paralela para resolver o serviço, já que a saída do mesmo só contém um elemento. Neste caso, como o sistema não encontra nenhum serviço simples e nenhuma composição paralela que satisfaça à requisição do usuário, ele tenta encontrar uma composição seqüencial.

Para isso, o sistema usa as informações obtidas no relatório gerado quando ele tentou descobrir um serviço simples. Ao analisar este resultado, ele percebe que existe pelo menos um serviço cuja entrada é similar à entrada da requisição (S_2, S_5, S_{10}) e que existe pelo menos um serviço cuja saída é similar à saída da requisição (S_{14}), e conclui que existe a possibilidade de encontrar uma seqüência. Feita essa descoberta, ele passa para a segunda parte do algoritmo e forma todas as combinações de seqüências possíveis entre os serviços cuja entrada é similar à da requisição e os serviços cuja saída é similar a da requisição. Cada uma destas combinações é armazenada em uma seqüência, que representa uma composição de serviços onde os serviços devem ser executados de forma seqüencial na ordem em que foram inseridos. Como temos três serviços que casam com a entrada da requisição e um serviço que casa com a saída, ele gera três seqüências {S_2, S_{14}}, {S_5, S_{14}},

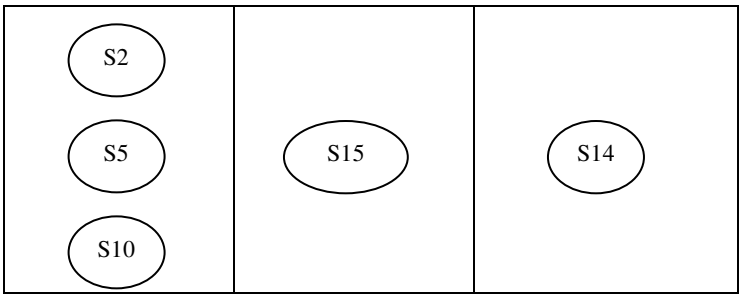
{S_{10}, S_{14}}, mas percebe que nenhuma delas é consistente, pois a saída do primeiro serviço não é compatível com a entrada do segundo serviço.

Desta forma, ele passa para a etapa seguinte. Ele gera uma seqüência e adiciona a ela um componente chamado de ponte. Além dela, ele gera uma outra camada contendo os serviços com entrada compatível com a entrada da requisição e outra contendo os serviços com saída compatível com a saída da requisição. A primeira camada é adicionada à esquerda da ponte, enquanto que a segunda é adicionada à direita da ponte. Desta forma, nesta etapa, temos o grafo tem a seguinte forma:



Na próxima etapa, ele gera uma nova requisição, onde as entradas correspondem à união das saídas dos serviços à esquerda da ponte (*hasPriceInRea*l) e as saídas correspondem à entrada do serviço à direita da ponte (*hasPriceInDollar*). O sistema então invoca o algoritmo de descoberta de serviços simples para a nova requisição, e descobre que o serviço S_{15} atende totalmente a requisição e, pode, assim, substituir a ponte. Feita a substituição, a seqüência principal do algoritmo de composição seqüencial terá a seguinte forma:

Ao executar o algoritmo para a nova requisição, ele percebe que o serviço S_{15} atende totalmente esta requisição e pode substituir a ponte. Ao fazermos a substituição a seqüência terá a seguinte forma:



Uma vez que a seqüência é finalizada, o algoritmo gera então o grafo que conecta cada serviço aos serviços cuja entrada é compatível com a sua saída. Aplicando este procedimento para a seqüência descoberta, encontramos o seguinte grafo mostrado na Figura 22:

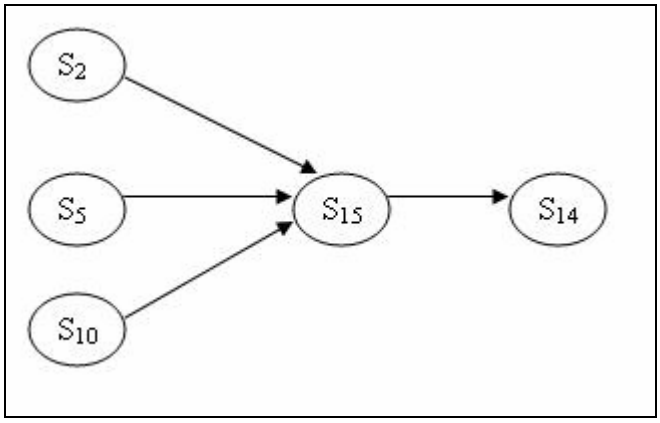


Figura 22: Grafo gerado pelo algoritmo de descoberta de composições seqüenciais

Finalizado o grafo, o sistema gera todas as seqüências possíveis e as retorna para o usuário. Neste caso, ele retorna para o usuário as seguintes seqüências $\{S_2, S_{15}, S_{14}\}$, $\{S_5, S_{15}, S_{14}\}$ e $\{S_{10}, S_{15}, S_{14}\}$. O resultado encontrado é mostrado Figura 23.

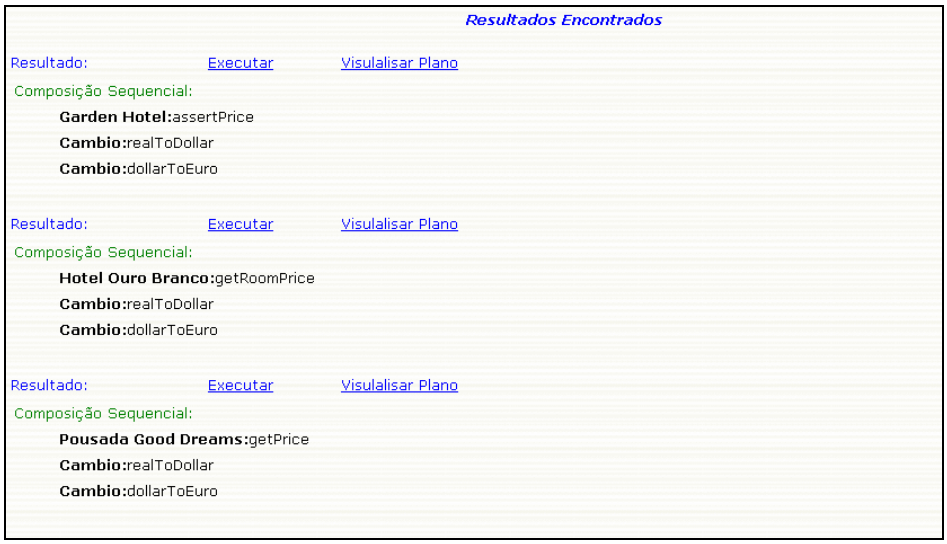


Figura 23: Resultado da descoberta de composições seqüenciais

4.4.1.4 Localizando provedores de serviços

Seguindo o nosso estudo de caso, vamos mostrar um exemplo de aplicação que aborda o segundo tipo de busca simples oferecido por WebS Composer, que corresponde à consulta por provedores de serviços. Este tipo de busca permite que o usuário encontre, de forma simples, rápida e precisa, estabelecimentos comerciais que satisfaçam algumas restrições impostas por ele. Além disso, ele permite que a ferramenta aplique a capacidade de inferência oferecida por ontologias para melhorar a seleção de provedores de serviços.

Neste tipo de busca, o usuário escolhe o tipo de estabelecimento que ele quer recuperar e as restrições impostas a este estabelecimento. Estas restrições são definidas sobre os valores dos atributos que a ontologia de domínio define para este tipo de estabelecimento. Para ilustrar este tipo de consulta, vamos supor que os hotéis cadastrados na ontologia de domínio da ferramenta são os descritos na Figura 24.

```
<Hotel>
    <hasName>Hotel Ouro Branco</hasName>
    <hasStars>4</hasStars>
    <hasCity>Campina Grande</hasCity>
    <hasState>Paraiba</hasState>
    <hasCountry>Brasil</hasCountry>
    <hasPool>true</hasPool>
    <hasNightClub>true</hasNightClub>
    <hasTennisCourt>false</hasTennisCourt>
    <hasParking>true</hasParking>
    <hasAgreedCards>[Total Card, Golden Card]</hasAgreedCards>

</Hotel>
<Hotel>
    <hasName>Garden Hotel</hasName>
    <hasStars>5</hasStars>
    <hasCity>Campina Grande</hasCity>
    <hasState>Paraiba</hasState>
    <hasCountry>Brasil</hasCountry>
    <hasPool>false</hasPool>
    <hasNightClub>true</hasNightClub>
    <hasTennisCourt>true</hasTennisCourt>
    <hasParking>true</hasParking>
    <hasAgreedCards>[Golden Card]</hasAgreedCards>
</Hotel>
<Hotel>
    <hasName>Hotel Ritz</hasName>
    <hasStars>3</hasStars>
    <hasCity>Campina Grande</hasCity>
    <hasState>Paraiba</hasState>
    <hasCountry>Brasil</hasCountry>
    <hasPool>true</hasPool>
    <hasNightClub>true</hasNightClub>
    <hasTennisCourt>false</hasTennisCourt>
    <hasParking>true</hasParking>
    <hasAgreedCards>[Total Card, Golden Card]</hasAgreedCards>

</Hotel>
<Inn>
    <hasName>Pousada Good Dreams</hasName>
    <hasCity>Campina Grande</hasCity>
    <hasState>Paraiba</hasState>
    <hasCountry>Brasil</hasCountry>
    <hasParking>false</hasParking>
    <hasAgreedCards>[Total Card, Golden Card]</hasAgreedCards>
</Inn>
```

Figura 24: Exemplos de provedores de serviços

Vamos supor agora, que o usuário está viajando para uma determinada cidade e deseja localizar um hotel para se hospedar. Ele deseja que o hotel recuperado, além de ser

localizado na cidade de destino da viagem, que é Campina Grande, ofereça piscina, estacionamento, quadra de tênis e, de preferência, aceite o seu cartão de crédito *Golden Card*. Vamos admitir também que a cidade, a piscina e o estacionamento sejam restrições indispensáveis, e que as demais restrições são preferíveis. A requisição é mostrada na Figura 25.

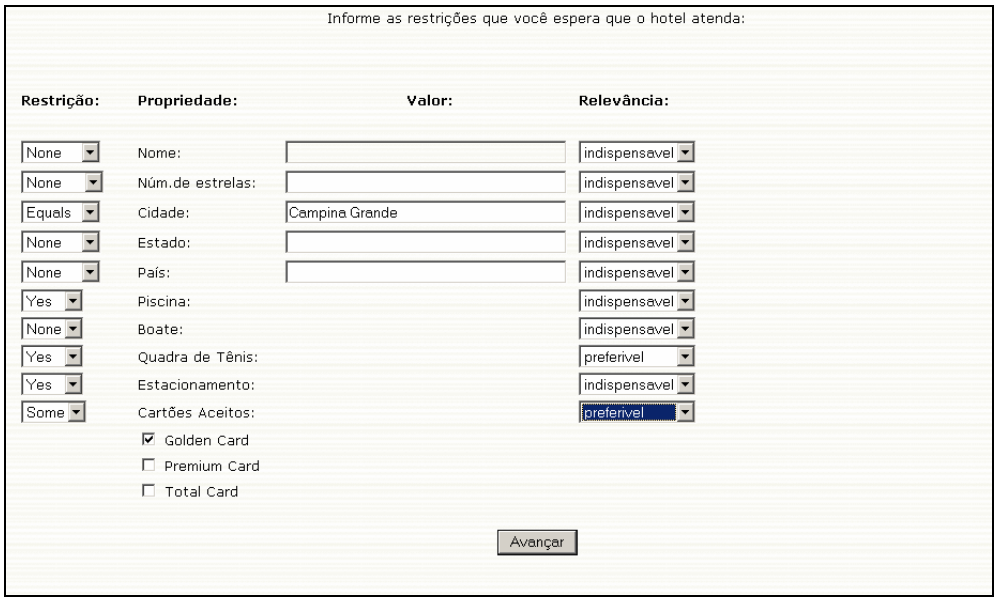


Figura 25: Requisição de serviços para uma busca por provedores de serviços

Ao receber esta requisição, o sistema gera o vetor de relevância, que neste caso, tem a forma (1, 1, 1, 0.4, 0.4). Estas dimensões se referem, respectivamente, às restrições impostas à cidade, à existência de piscina, à existência de estacionamento, à existência de quadra de tênis e à aceitação do seu cartão de crédito. Após selecionar todos os hotéis cadastrados em sua base de dados, o sistema aplica o algoritmo de casamento de instâncias para cada item selecionado. Aplicando este algoritmo para cada provedor de serviço cadastrado, encontramos os seguintes resultados mostrados na Tabela 10.

Hotel	Vetor de casamento	Grau de similaridade
Hotel Ouro Branco	(1, 1, 1, 0, 0.4)	89 %
Garden Hotel	(1, 1, 1, 0.4, 0.4)	100 %
Hotel Ritz	(1, 0, 1, 0.4, 0.4)	73 %

Tabela 10: Resultado encontrado para o casamento de provedores de serviços

O Hotel Ritz é descartado pelo algoritmo de busca, já que ele não satisfaz uma das restrições indispensáveis para o usuário, que é a presença de estacionamento. Desta forma, o sistema vai recuperar e exibir para o usuário os Hotéis Ouro Branco e Garden. O resultado produzido por WebS Composer para esta busca é mostrado na Figura 26.

Figura 26: Resultado da busca por provedores de serviços

4.4.1.5 Consultando por serviços e provedores ao mesmo tempo

Outra opção de busca oferecida pela ferramenta, combina a consulta por serviços e por provedores de serviços em uma única consulta. Neste caso, o usuário pode desejar escolher entre os serviços que ofereçam uma certa funcionalidade, aqueles que satisfaçam certas características. Vamos supor que um cliente, agora, deseja encontrar um serviço de

reserva de quarto on-line. As entradas que ele tem disponíveis e as saídas que ele deseja obter do serviço são as mesmas usadas no exemplo que mostramos na seção 4.1.1.1. A diferença deste novo exemplo é que agora ele não deseja qualquer local, mas um local para dormir que se localize em Campina Grande, tenha estacionamento e aceite o cartão *Total Card*. Vamos assumir ainda que as duas primeiras restrições são indispensáveis para o usuário, enquanto que a terceira restrição é preferível..

Neste tipo de busca, o sistema resolve primeiro a parte da seleção dos provedores de serviços que possuam as características desejadas. Como já dissemos antes, isto é feito para limitar o escopo dos serviços a serem descobertos, aumentando assim a eficiência dos algoritmos de busca. No primeiro passo desta seleção, o algoritmo seleciona todas as instâncias que representam locais para dormir. Assim, o sistema recupera todas as instancias de hotéis e pousadas cadastradas no sistema. Estas instâncias são armazenadas em um conjunto de instâncias similares. Aplicando esta etapa do algoritmo ao conjunto de provedores mostrados na Figura 4.4, este conjunto terá a seguinte forma:

Instâncias similares = {Hotel Ouro Branco, Garden Hotel, Hotel Ritz, Pousada Good Dreams}

A próxima etapa do algoritmo consiste em selecionar os provedores que satisfazem às restrições do usuário. Isto é feito aplicando o mesmo algoritmo usado no exemplo da seção anterior. Ao aplicarmos o algoritmo para esta requisição, obtemos o seguinte conjunto, que chamamos de instâncias de interesse:

Instâncias de Interesse = {Hotel Ouro Branco, Garden Hotel, Hotel Ritz}

Ao fim deste passo, o algoritmo encontra um conjunto de provedores indesejáveis, através da subtração do conjunto das instâncias similares e o conjunto das instâncias de interesse. Neste exemplo, este conjunto terá a seguinte forma:

Provedores Indesejáveis = {Pousada Good Dreams}

Encontrados estes conjuntos, o sistema chama os algoritmos de descoberta de serviços. Os serviços oferecidos pelos provedores indesejáveis são automaticamente descartados pela ferramenta. Ao aplicar o algoritmo de busca para os serviços restantes, o sistema usa a medida de similaridade encontrada para o provedor do serviço como critério de seleção para escolher entre dois serviços que apresentam o mesmo grau de similaridade. Aplicando o algoritmo de busca para este exemplo, a ferramenta vai retornar para o usuário o resultado mostrado na Figura 27.

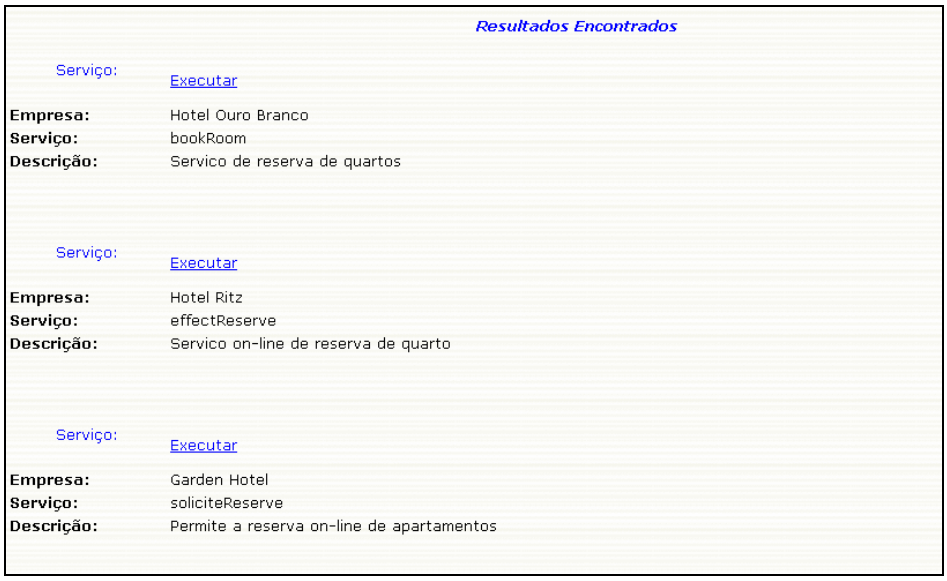


Figura 27: Resultado encontrado para uma busca combinada

4.5 Considerações Finais

Este capítulo apresentou um estudo de caso que ilustrou a aplicação de WebS Composer em um domínio de conhecimento particular. Ele mostrou a aplicação da ferramenta para descobrir serviços relacionados à hospedagem e à alimentação de um cliente que vai fazer uma viagem. Mais especificamente, ele mostrou a aplicação dos algoritmos utilizados pelo sistema para a descoberta de serviços e composições de serviços para resolver diversos tipos de consulta. Além disso, ele mostrou como o sistema usa a recuperação de provedores de serviços para melhorar a seleção de serviços, permitindo a

recuperação de serviços cujas empresas atendem às restrições impostas pelo usuário. Lembramos ainda que todos os exemplos usados neste capítulo são hipotéticos.

CAPÍTULO V – CONCLUSÃO

A tecnologia de *web services*, que surgiu como uma forma de resolver os problemas da heterogeneidade existente entre os serviços publicados na *web,* vem sendo cada vez mais utilizada pelas empresas que oferecem os seus serviços neste ambiente. Esse crescimento, aliado ao fato da ineficiência das ferramentas de busca atuais na recuperação destes serviços, fez surgir a necessidade do desenvolvimento de ferramentas capazes de localizar serviços de forma precisa e eficiente. Mais ainda, surgiu a necessidade da criação de sistemas de busca capazes de descobrir composições de serviços de forma automática, em tempo de execução, quando dois ou mais serviços puderem ser compostos para atender uma tarefa que não pode ser atendida por um único serviço.

Esta dissertação apresentou uma nova ferramenta, chamada de WebS Composer, que usa marcações semânticas baseadas em ontologias de serviços descritas em OWL-S para permitir a descoberta automática de serviços e composições de serviços. A ferramenta usa ainda a marcação semântica de provedores de serviços para melhorar a descoberta e seleção de serviços, e uma nova abordagem, baseada em um dos modelos clássicos da recuperação da informação, que é o modelo vetorial, que facilita a descoberta de um dos tipos de composições. Além disso, a ferramenta propõe algoritmos de composição de serviços, baseados na idéia de relatórios, que permitem que os componentes de busca do sistema colaborem entre si, e um melhor desempenho dos algoritmos que descobrem as composições de serviços.

5.1 Contribuições oferecidas pelo trabalho

WebS Composer é um trabalho que oferece as seguintes contribuições:

- A proposição de uma nova técnica, baseada no modelo vetorial, para descoberta automática, seleção e composição de serviços. Os algoritmos apresentados permitem que a máquina de busca do sistema descubra composições onde os serviços devem ser executados em paralelo, composições onde os serviços devem ser executados de forma seqüencial e

composições mistas que combinam estes dois tipos. No caso da descoberta de composições seqüenciais, o sistema propõe um novo tipo de algoritmo, chamado de *"back–forward chaining"*, que supera o desempenho dos algoritmos tradicionais para este tipo de composição. Outra vantagem obtida com a abordagem baseada no modelo vetorial é a obtenção de uma medida de similaridade, que pode ser calculada para cada serviço, e que permite que serviços com casamento parcial possam ser recuperados, aumentando a flexibilidade da ferramenta, e que os serviços recuperados possam ser *rankeados* e ordenados por um algoritmo de ordenação de forma que os resultados mais relevantes sejam exibidos primeiro para o usuário final.

- Uma nova abordagem para a seleção de serviços, que consiste na seleção de serviços baseada nas características de seus provedores, que corresponde a uma área pouco explorada até então pelos trabalhos da mesma área. Esta seleção é permitida através da anotação semântica do provedor do serviço, onde o mesmo é armazenado como uma instância de um conceito definido numa ontologia de domínio. Esta característica permite que a capacidade de inferência oferecida pelas ontologias possa ser aplicada não só para a descoberta de uma determinada funcionalidade, mas também para selecionar serviços cujos provedores mais se adequam às características e restrições desejadas pelo usuário.

Uma característica importante de WebS Composer é a sua facilidade de integração com outras ferramentas, que permite que outros sistemas possam interagir com ela de forma bastante simples. Isto pode ser oferecido devido às seguintes características:

- WebS Composer é uma ferramenta "genérica", ou seja, não está amarrada a nenhum domínio de aplicação em particular. Para acrescentar ao sistema um novo domínio de conhecimento, o administrador precisa apenas acrescentar uma ontologia que descreva a área de conhecimento a ser incluída, não havendo a necessidade de alterações no código fonte da ferramenta;

- O cliente pode interagir com cada módulo da camada de aplicação através de um único componente. Através deste componente, o cliente pode invocar qualquer funcionalidade oferecida pelo módulo. Esta característica facilita a integração com outros sistemas, que só precisam conhecer um componente para interagir com o módulo, e diminui o acoplamento entre WebS Composer e os outros sistemas, pois mudanças feitas nos demais componentes do módulo não afetam em nada a interação com os clientes.

Estas duas características permitem que WebS Composer possa ser integrada a vários sistemas que necessitam de buscas precisas em diversos tipos de aplicações, como turismo, sistemas multimídia, aplicações cientes de contexto, aplicações de roteamento, aplicações médicas, etc. No momento ela está sendo utilizada em um projeto de construção de roteiros turísticos, denominado SEI-Tur[1].

5.2 Trabalhos Futuros

Alguns trabalhos ainda podem ser desenvolvidos para continuar a pesquisa relativa a esta dissertação. Estes trabalhos podem incorporar novas características à ferramenta, aumentando a sua qualidade e superando algumas das limitações existentes na versão atual. Os principais trabalhos que podem ser realizados são:

- **Melhoramento da interface com o usuário:** a interface com o usuário deve ser melhorada para facilitar a interação do mesmo com a ferramenta;
- **Inclusão de pré-condições e efeitos:** As pré-condições definem as condições que devem ser satisfeitas para que o serviço possa ser chamado, enquanto que os efeitos definem as conseqüências que a execução do serviço de forma apropriada provoca no mundo real. Estas duas características vão permitir que a ferramenta realize buscas ainda mais exatas. Para a implementação destas características, é necessária uma linguagem adequada para a definição de regras, como SWRL (HORROCKS et al, 2004). Uma das

[1] Projeto CNPQ 507225/2004-0

mais fortes candidatas para a implementação desta funcionalidade é a linguagem SWSL, que compõe o *framework* SWSF (BATTLE et al, 2005), recém proposto pela *W3C* para a descrição semântica de *web services*. Restrições de tempo impediram que esta característica fosse incorporada à versão atual da ferramenta;

- **Implementação de um modelo de *workflow* para execução de serviços**: esta funcionalidade vai permitir uma execução mais eficiente e segura dos serviços e composições descobertos pela ferramenta. A versão atual permite a execução dos serviços e composições descobertos, mas não trata questões importantes, como o comportamento transacional de composições de serviços. Alguns trabalhos, como o de PIRES et al (2002) e MIKALSEN et al (2002), já propõem soluções para o comportamento transacional de composições de serviços e podem ser utilizados como referência;
- **O casamento entre ontologias:** esta é uma característica importante para aumentar a flexibilidade da ferramenta. Na versão atual do sistema, o sistema define ontologias para alguns domínios de conhecimento. Depois, todos os serviços e provedores cadastrados na base de dados da ferramenta e todas as consultas do usuário são feitas com base nestas ontologias de domínio. Esta característica impõe uma limitação ao sistema, pois não permite que empresas cadastrem os seus serviços de acordo com uma ontologia própria ou uma outra ontologia definida para o mesmo domínio. Alguns trabalhos foram propostos para resolver o casamento entre ontologias, como os de DOAN et al (2003), LACHER & GROH (2001) e SYEDA-MAHMOOD et al (2005). Alguns trabalhos dentro do próprio grupo de pesquisa responsável pelo desenvolvimento deste trabalho também estão em andamento e futuramente poderão ser integrados ao sistema;
- **Tratamento de propriedades não funcionais:** esta característica consiste em utilizar as propriedades não funcionais utilizadas pelos demais trabalhos de composição de serviços (custo, confiança, etc). Estes critérios, aliados com as características semânticas dos provedores de serviços, podem oferecer uma maior qualidade para a seleção de serviços;

- **Verificação da correção das composições:** um modelo formal para a verificação da correção de composições de serviços pode ser implementado para garantir uma maior consistência para as composições de serviços descobertas.

REFERÊNCIAS BIBLIOGRÁFICAS

(ALONSO et al, 2004) ALONSO G.; CASATI F.; KUNO H.; MACHIRAJU V. **Web Services: concepts, architectures and applications**. 1.ed. Berlim: Springer-Verlag, 2004.

(ANKOLEKAR et al, 2001) ANKOLEKAR A., et al. DAML-S: semantic markup for web services. In: INTERNATIONAL SEMANTIC WEB WORKING SYMPOSIUM, 1., 2001, Stanford. **Proceedings…** Stanford:2001. p. 411-430.

(AVERSANO et al, 2004) AVERSANO L.; CANFORMA G.; CIAMPI A. An algorithm for web service discovery through their composition. In: INTERNATIONAL CONFERENCE ON WEB SERVICES, 2., 2004, San Diego**. Proceedings…**San Diego:2004, p. 332-339.

(BAADER et al, 2002) BAADER F., et al. **The description logic handbook: Theory, Implementation and Applications.** Cambridge: Cambridge University Press, 2002.

(BATTLE et al, 2005) Battle S., et al. **Semantic Web Services Framework (SWSF) Overview.** Disponível em: <http://www.daml.org/services/swsl/1.0/overview/>. Acesso em: 28 jun. 2006.

(BERARDI et al, 2003) BERARDI D., et al. Automatic composition of e-services that export their behavior. In: INTERNATIONAL CONFERENCE ON SERVICE ORIENTED COMPUTING, 1., 2003, Trento. **Proceedings…**Trento:2003, p. 43-58.

(BIANCHINI et al., 2005) BIANCHINI D.; ANTONELLIS V. DE; MELCHIORI M. An ontology-based architecture for service discovery and advice system. In: INTERNATIONAL WORKSHOP ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 16., 2005, Copenhague. **Proceedings…**Copenhague:2005, p. 551-556.

(BRICKLEY & GUHA, 2004) BRICKLEY D.; GUHA R. V. **RDF Vocabulary Description Language 1.0 RDF Schema**. Disponível em: <http://www.w3.org/TR/rdf-schema/> Acesso em: 28 jun. 2006.

(CHRISTENSEN et al, 2001) CHRISTENSEN E., et al. **Web Services Description Language (WSDL) 1.1**. Disponível em: <http://www.w3.org/TR/wsdl/> Acesso em: 28 jun. 2006.

(CLEMENT et al, 2004) CLEMENT L., et al. **UDDI Version 3.0.2**. Disponível em: <http://uddi.org/pubs/uddi_v3.htm> Acesso em: 28 jun. 2006.

(CONNOLY et al, 2001) CONNOLY D., et al. **DAML + OIL Reference Description.** Disponível em: <http://www.w3.org/TR/daml+oil-reference> Acesso em: 28 jun. 2006.

(COSTA et al, 2004) COSTA L. A. G.; PIRES P. F.; MATTOSO M. Automatic composition of web services with contingency plans. In: INTERNATIONAL CONFERENCE ON WEB SERVICES, 2., 2004, San Diego. **Proceedings...**San Diego:2004, p. 454-461.

(CURBERA et al, 2003) CURBERA F., et al. The next step in web services. **Communications of ACM**, NovaYork, v. 46, n. 10, p. 29-34, out. 2003.

(DAVIES et al, 2003) DAVIES J.; FENSEL, D.; VAN HERMELEN F. **Towards the semantic web: Ontology-Driven Knowledge Management**. 1.ed. Chichester:John Wiley & Sons, 2003.

(DOAN et al, 2003) DOAN A., et al. Learning to match ontologies on the semantic web. **The VLDB Journal**, Berlim, v. 12, n.4, p. 303-319, nov. 2003.

(DOGAC et al, 2004) DOGAC A., et al. Semantically enriched web services for the travel industry. **ACM Sigmod Record**, Nova York, v. 33, n. 3, p.21-27, set. 2004.

(DOMINGUE et al, 2005) DOMINGUE J., et al. IRS-III: A platform and infrastructure for creating WSMO-based semantic web service. In: WSMO Implementation Workshop, 1., 2005, Frankfurt. **Proceedings...**Frankfurt:2005, p. 454-461.

(FENSEL, 2004) FENSEL D. **Ontologies: A silver bullet for knowledge management and electronic commerce**. 2 ed. Berlim:Springer-Verlag, 2004.

(FOSTER et al, 2003) FOSTER I., et al. The virtual data grid: A new model and architecture for data-intensive collaboration. In: CONFERENCE ON INNOVATIVE DATA SYSTEMS RESEARCH, 2., 2003, Asilomar. **Proceedings…**Asilomar:2003.

(FU et al, 2002) FU X.; BULTAN T.; SU J. Formal verification of e-services and workflow*s*. In: WORKSHOP ON WEB SERVICES, E-BUSINESS AND THE SEMANTIC WEB: FOUNDATIONS MODELS, ARCHITECTURE, ENGINEERING AND APPLICATIONS, 1., 2002, Toronto. **Proceedings…**Toronto:2002, p. 188-202.

(GEKAS & FASLI, 2005) GEKAS J.; FASLI M. Automatic web service composition using web connectivity analysis techniques. In: W3C WORKSHOP ON FRAMEWORKS FOR SEMANTICS IN WEB SERVICES, 1., 2005, Innsbruck. **Proceedings…**Innsbruck:2005.

(HAMADI & BENATALLAH, 2003) HAMADI R., BENATALLAH B*.,* A Petri Net-based model for web service composition. In: AUSTRALIAN DATABASE CONFERENCE ON DATABASE TECHNOLOGIES, 14., 2003, Adelaide. **Proceedings…**Adelaide:2003, p. 191-200.

(HENDLER et al, 2002) HENDLER J.; BERNERS-LEE T.; MILLER E. Integrating Applications on the Semantic Web. **Journal of the Institute of Electrical Engineers of Japan**, Tóquio, v. 122, n. 10, p. 676-680, out. 2002.

(HORROCKS et al, 2004) HORROCKS I., et al. **SWRL: A Semantic Web Rule Language Combining OWL and Rule ML**. Disponível em: <http://www.w3.org/Submission/SWRL/> Acesso em 28 jun. 2006.

(JURIC et al, 2006) JURIC M.; SARANGA P.; MATHEW B. **Business Process Execution Language for Web Services.** 2. Birmingham: Packt Publishing, 2006.

(KLEIN & BERNSTEIN, 2001) KLEIN M.; BERNSTEIN A. Searching for services on the semantic web using process ontologies. In: INTERNATIONAL SEMANTIC WEB WORKING SYMPOSIUM, 1., 2001, Stanford. **Proceedings…**Stanford:2001.

(KLYNE & CARROLL, 2004) KLYNE G.; CARROLL J. J. **Resource Description Framework (RDF): Concepts and Abstract Syntax**. Disponível em: <http://www.w3.org/TR/rdf-concepts> Acesso em 28 jun. 2006

(KVALOY et al, 2005) KVALOY T. A., et al. Automatic composition and selection of semantic web services. In: ADVANCES IN GRID COMPUTING, 1., 2005, Amsterdam. **Proceedings…**Amsterdam:2005, p. 184-192.

(LACHER & GROH, 2001) LACHER M.; GROH G. Facilitating the exchange of explicit knowledge through ontology mappings. In: INTERNATIONAL FLAIRS CONFERENCE, 14., 2001, Key West. **Proceedings…**Key West:2001, p. 305-309.

(LEE & PATEL, 2004) LEE Y.; PATEL C. Towards intelligent web services for automating medical service composition. In: INTERNATIONAL CONFERENCE ON WEB SERVICES, 2., 2004, San Diego. **Proceedings...**San Diego:2004, p. 384-391.

(LEYMAN, 2001) LEYMAN F. **Web Services Flow Language (WSFL 1.0)**. Disponível em: <http://www-3ibm.com/software/solutions/webservices/pdf/WSFL.pdf> Acesso em 28 jun. 2006.

(LUDWING, 2003) LUDWING H. Web Services QoS: external SLAs and internal policies: Or, how do we deliver what we promise? In: INTERNATIONAL CONFERENCE WEB INFORMATION SYSTEMS ENGINEERING, 4., 2003, Roma. **Proceedings…**Roma:2003, p. 115-120.

(MARTIN et al, 2004) MARTIN D.L., et al. Bringing Semantics to Web Services: The OWL-S Approach. In: WORKSHOP ON SEMANTIC WEB SERVICES AND WEB PROCESS COMPOSITION, 1., 2004, San Diego. **Proceedings…**San Diego:2004. p. 26-42.

(McGUINNESS & VAN HARMELEN, 2004) McGUINNESS D. L.; VAN HARMELEN F. **OWL Web Ontology Language Overview**. Disponível em: <http:// www.w3.org/TR/owl-features/> Acesso em 28 jun. 2006.

(McILRAITH & SON, 2002) McILRAITH S.; SON T. C. Adapting Golog for Composition of Semantic Web Services. In: INTERNATIONAL CONFERENCE ON THE PRINCIPLES OF KNOWLEDGE REPRESENTATION AND REASONING, 8., 2002, Toulouse. **Proceedings…**:Toulouse, 2002. p. 482-496.

(McILRAITH et al, 2001) McILRAITH S.; SON T. C.; ZENG H. Semantic Web Services. **IEEE Intelligent Systems**, Los Alamitos, v.16, n.2, p. 46-53, mar./abr. 2001.

(MEDJAHED et al,2003) MEDJAHED B.; BOUGUETTAYA A.; ELMAGARMID A. K., Composing web services on the semantic web. **The VLDB Journal**, Berlim, v. 12, n. 4, p. 333-351, dez. 2003.

(MENASCÉ, 2004) MENASCÉ D.A. Composing web services: a QoS View. **IEEE Internet Computing 2004**, Los Alamitos, v. 8, n.6, p. 88-90, nov/dez 2004.

(MEREDITH & BJORG, 2003) MEREDITH L. G.; BJORG S. Contract and Types. **Communications of the ACM**, Nova York, v. 46, n. 10, p. 41-47, out. 2003.

(MIKALSEN et al, 2002) MIKALSEN T.; TAI S.; ROUVELLOU I. Transactional Attitudes: reliable composition of autonomous web services. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 3., 2002, . Washington. **Proceedings…**:Washington: 2002.

(MILANOVIC, 2005) MILANOVIC N. Contract-based web service composition framework with correctness guarantees. In: INTERNATIONAL SERVICE AVAILABILITY SYMPOSIUM, 2., 2005, Berlim. **Proceedings…**Berlim:2005, p. 52-67.

(MILANOVIC & MALEK, 2004) MILANOVIC N.; MALEK M. Current solutions for web service composition. **IEEE Internet Computing**, Los Alamitos, v. 8, n. 1, p. 51-59, jan/fev 2004.

(PAGE & BRIN, 1998) PAGE L.; BRIN S. The anatomy of a large-scale hypertextual web search engine. **Computer Networks**, v.7, n. 1.-7. p. 107-117, abr. 1998.

(PAOLUCCI et al, 2002) PAOLUCCI M., et al. Semantic matching of web service capabilities. In: INTERNATIONAL SEMANTIC WEB CONFERENCE, 1., 2002, Sardinia. **Proceedings…**Sardinia:2002, p. 333-347.

(PIRES et al, 2002) PIRES P.; BENEVIDES M. R. F.; MATTOSO M. Building Reliable Web Service Compositions. In: WEB, WEB SERVICES AND DATABASE SYSTEMS, 1., 2002, Erfut. **Proceedings…**Erfut:2002, p. 59-72.

(RAGGET et al, 1999) RAGGET D.; HORS A. L.; JACOBS I., **HTML 4.01 Specification**. Disponível em:
<http://www.w3.org/TR/REC-html40> Acesso em 28 jun 2006.

(ROMAN et al, 2005) ROMAN D., et al. Web Service Modeling Ontology. **Applied Ontology**, Amsterdam, v.1, n.1, p. 77 – 106.

(SALTON & McGILL, 1983) SALTON G.; McGILL M. **An Introduction to Modern Information Retrieval**. 1. ed. Nova York:McGraw-Hill, 1983.

(SELL et al, 2004) SELL D., et al. Interactive Composition of WSMO-based Semantic Web Services in IRS-III. In: AKT WORKSHOP ON SEMANTIC WEB SERVICES, 1., 2004, Milton Keynes. **Proceedings…**Milton Keynes:2004.

(SRIVASTAVA & KOEHLER, 2003) SRIVASTAVA B.; KOEHLER H. Web Service Composition – Current Solutions and Open Problems. In: WORKSHOP ON PLANNING FOR WEB SERVICES, 1., 2003, Trento. **Proceedings…**Trento:2003.

(SYEDA-MAHMOOD et al, 2005) SYEDA-MAHMOOD T., et al. Searching service repositories by combining semantic and ontological matching. In: INERNATIONAL CONFERENCE ON WEB SERVICES, 3., 2005, Orlando. **Proceedings…**Orlando:2005, p.13-20.

(VUKOVIC & ROBINSON, 2004) VUKOVIC M., ROBINSON P., Adaptive, planning based, web service composition for context awareness. **Advances in Pervasive Computing**,Viena, v. 176, n. 1, p. 247-252, 2004.

(YANG & PAPAZOUGLOU, 2002) YANG J.; PAPAZOUGLOU M. P. Web Component: a substrate for web services reuse and composition. In: CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, 14., 2002. Toronto. **Proceedings…**Toronto: 2002.