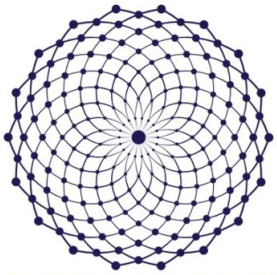
UNIVERSIDADE FEDERAL
DE ALAGOAS



MODELAGEM
COMPUTACIONAL
DE CONHECIMENTO

Dissertação de Mestrado

Um Middleware adaptável para descoberta, composição e invocação automática de Serviços Web Semânticos

Heitor José dos Santos Barros
heitorjsbarros@gmail.com

Orientador:

Evandro de Barros Costa

Coorientador:

Ig Ibert Bittencourt

Maceió, Março de 2011

Heitor José dos Santos Barros

Um Middleware adaptável para descoberta, composição e invocação automática de Serviços Web Semânticos

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Curso de Mestrado em Modelagem Computacional de Conhecimento do Instituto de Computação da Universidade Federal de Alagoas.

Orientador:

Evandro de Barros Costa

Coorientador:

Ig Ibert Bittencourt

Maceió, Março de 2011

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária Responsável: Maria Auxiliadora G. da Cunha

B277u Barros, Heitor José dos Santos.
Um Middleware adaptável para descoberta, composição e invocação automática de serviços web semânticos / Heitor José dos Santos Barros. – 2011.
64 f. : il.

Orientador: Evandro de Barros Costa.
Co-Orientador: Ig Ibert Bittencourt..
Dissertação (mestrado em Modelagem Computacional de Conhecimento) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2011.

Bibliografia: f. 59-64.

1. Serviços web semânticos. 2. Web semântica. 3. Descoberta e composição de serviços. I. Título.

CDU: 004.738.52

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa Multidisciplinar de Pós-Graduação em Modelagem Computacional de Conhecimento
Avenida Lourival Melo Mota, Km 14, Bloco 09, Cidade Universitária
CEP 57.072-900 – Maceió – AL – Brasil
Telefone: (082) 3214-1364

Membros da Comissão Julgadora da Dissertação de Mestrado de Heitor José dos Santos Barros, intitulada: "Um *Middleware* adaptável para descoberta, composição e invocação automática de Serviços Web Semânticos", apresentada ao Programa de Pós-Graduação em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas em 25 de março de 2011, às 17h00min, na sala de aula do Mestrado em Modelagem Computacional de Conhecimento.

COMISSÃO JULGADORA

Prof. Dr. Evandro de Barros Costa
UFAL – Instituto de Computação
Orientador

Prof. Dr. Ig Ibert Bittencourt Santana Pinto
UFAL – Instituto de Computação
Examinador

Prof. Dr. Patrick Henrique da Silva Brito
UFAL – Instituto de Computação
Examinador

Profa. Dra. Daniela Barreiro Claro
UFBA – Departamento de Ciência da Computação
Examinadora

Prof. Dr. Hyggo Oliveira de Almeida
UFCG – Centro de Engenharia Elétrica e Informática
Examinador

Maceió, março de 2011.

Resumo

Os Serviços Web Semânticos têm ganhado uma atenção especial pela academia e indústria. Eles têm sido utilizados como uma promessa para possibilitar a automação de todos os aspectos da provisão e uso de Serviços Web, tais como criação, seleção, descoberta, composição e invocação de serviços. Para isso, a comunidade tem se dedicado a criação de ferramentas e técnicas para explorar as informações semânticas destes serviços. Entretanto, como apontado pela comunidade especializada no tema, aplicações baseadas em Serviços Semânticos possuem suas próprias características, interesses e prioridades. Esta diversidade influencia diretamente na escolha das técnicas e tecnologias utilizadas para manipulação de serviços, ou seja, uma mesma ferramenta pode ter resultados satisfatórios em uma determinada aplicação e não ser adequada para outras. Além disso, essas aplicações podem evoluir, o que implica na necessidade de mudança nestas ferramentas. Com o objetivo de contribuir na solução deste problema, propõe-se na pesquisa em pauta um Middleware adaptável para gerenciamento de descoberta e invocação de serviços capaz de integrar diferentes técnicas e ferramentas de acordo com as necessidades da aplicação. Como forma de avaliar o trabalho, realizou-se um estudo de caso envolvendo o uso de Serviços Web Semânticos no domínio de educação, com isso, verificou-se que o Middleware proposto se mostrou eficiente na realização dos processos de descoberta, composição e invocação de serviços de maneira adaptável.

Palavras-chave: Serviços Web Semânticos; Web Semântica; Descoberta e Composição de Serviços.

Abstract

Semantic Web Services domain has gained special attention in academia and industry. It has been adopted as a promise to enable automation of all aspects of Web services provision and use, such as service creation, selection, discovery, composition, invocation. For that, the Semantic Web Services community has been devoted to creating tools and techniques that explore the semantic information of these services. However, the state of the art shows that the aplications based on Semantic Services have their own characteristics, interests and priorities. This diversity directly influences the choice of techniques and technologies for handling services, ie, a single tool can have satisfactory results in a particular application and is not appropriate for others. Moreover, these applications can evolve, which implies the need of changing these tools. Aiming to solve this problem, this work proposes an adaptive middleware for managing discovery and invocation of services capable of integrating different tools and techniques according to application needs. In order to validate the work, a case study with Semantic Web Services of education domain is presented, with this, it was noted that the proposed Middleware is efficient for performing the processes of discovery, composition and invocation of services in an adaptable manner.

Key-words: Semantic Web Services; Semantic Web; Service Discovery and Composition.

Agradecimentos

Agradeço sempre a Deus por todas as portas abertas em minha vida e por sempre me iluminar diante das escolhas difíceis, que não foram poucas.

Agradeço aos meus pais (José Geraldo e Mariluce) e meus irmãos (Pedro e Marília) pelo constante incentivo e apoio. Pelos conselhos dados e pela paciência em relação as minhas ausências. Também agradeço por sempre me mostrar o valor do estudo. Agradeço também ao restante da minha enorme família: avós, tios, tias, primos e agregados pelo carinho e preocupação comigo.

Agradeço aos meus orientadores professor doutor Evandro de Barros Costa e professor doutor Ig Ibert Bittencourt pelos conselhos dados e por me convencerem a ficar na UFAL e realizar este trabalho. Agradeço também a todos os meus professores que, desde a graduação, compartilharam comigo seu conhecimento e ajudaram a me tornar a pessoa que sou.

Agradeço também aos meus amigos do GrOW Old: Diego, Douglas, João Pedro, Lucas, Marlos, Pedro e Rafael Ligeiro pelos dias compartilhados em nossos laboratórios buscando conhecimento e crescimento como pessoas e pesquisadores, além das 'meia horinha's que duravam várias horas. Também aos demais membros do GrOW: Endhe, Jean, Olavo, Thyago, Társis e outros que me ajudaram no dia-a-dia do desenvolvimento deste trabalho. Aos meus amigos da UFAL: Adolfo, Arthur, Bruno, Clesivaldo, Gérdson, Levi, Rafael, Ronaldo, Sávio que fizeram parte da minha turma de graduação e também aos tantos outros que me ajudaram a sobreviver na UFAL, tanto na graduação quanto no mestrado.

Por fim, agradeço também a todos que me ajudaram a relaxar me tirando um pouco do trabalho e compartilhando momentos de diversão, entre eles meus amigos do CEFET, do Rui Palmeira e da Central.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Capítulo 1

Introdução

A *World Wide Web* tem mudado a forma como as pessoas interagem e como as organizações conduzem os seus negócios. Ela pode ser considerada como o núcleo de uma revolução tecnológica, em que, a partir do computador, é possível, por exemplo, comunicar de maneira mais interativa, fechar negócios de maneira mais ágil e obter uma fonte cada vez maior de conhecimento (Antoniou & van Harmelen (2004)).

Entretanto, no modelo clássico da Web, baseado em descrições puramente sintáticas dos recursos, a tarefa de encontrar e utilizar os recursos presentes na Web fica a cargo de seus usuários (agentes humanos). Contudo, a Web tem apresentado um grande crescimento, aumentando o volume de dados presentes nas páginas Web, o que torna a busca por recursos na Web um processo cada vez mais complexo (Berners-Lee et al. (2001)).

Neste contexto, Shadbolt et al. (2006) propôs que o modelo original da Web fosse estendido de forma a adicionar metadados aos seus recursos, permitindo aos agentes de software efetuar tarefas utilizando estes recursos. Tais metadados são baseados na utilização de ontologias de forma a descrever o domínio relacionado a cada recurso e a partir daí permitir que agentes de software realizem inferência sobre esses recursos, surgindo então o conceito de Web Semântica.

De forma mais detalhada, a Web Semântica busca utilizar recursos provenientes de áreas diferentes, tais como: Inteligência Artificial (agentes inteligentes e representação de conhecimento), Engenharia de Software (frameworks), Computação Distribuída (serviços Web), entre outras, para executar atividades na Web que antes só eram possíveis por agentes humanos.

A Web Semântica permitirá um maior acesso não só ao conteúdo, mas também para serviços presentes na Web. Os usuários e agentes de software poderão ser capazes de descobrir, invocar, compor e monitorar os recursos

da Web que oferecem serviços específicos e com propriedades particulares, e devem ser capazes de fazê-lo com um alto grau de automação, se desejarem (Martin et al. (2004)).

Para prover estas funcionalidades, a Web Semântica necessita da declaração explícita do conhecimento através do uso de ontologias para representar adequadamente as informações de maneira que máquinas possam interpretar. Em algumas aplicações, também é necessário fornecer serviços que máquinas e agentes inteligentes possam entender, selecionar, compor e invocar estes serviços automaticamente. Isto se torna possível através do uso de Serviços Web Semânticos (McIlraith et al. (2001)). A Figura 1.1 ilustra a união das tecnologias que são utilizadas em aplicações baseadas na Web Semântica.

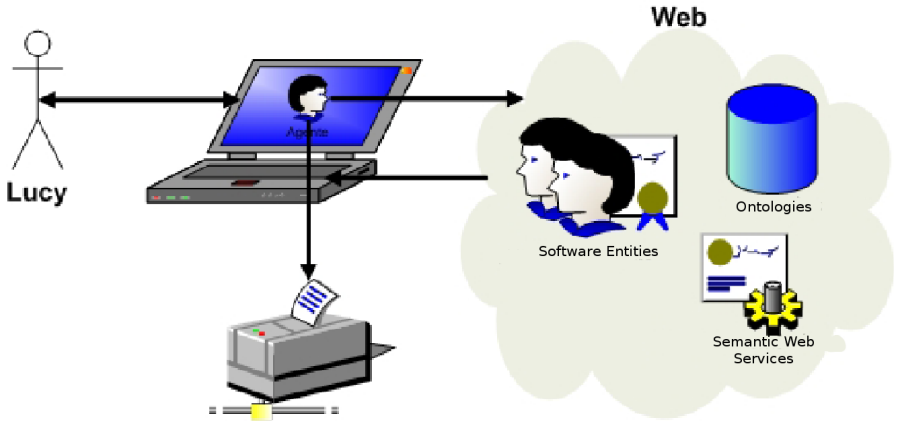


Figura 1.1: Tecnologias presentes na Web Semântica (adaptada de Berners-Lee et al. (2001))

Neste contexto, os Serviços Web Semânticos surgem como um ponto de intersecção entre a Web Semântica e os *Web Services*, por ter como objetivo a utilização da interoperabilidade semântica de um e a dinamicidade de recursos do outro, através da aplicação de semântica às descrições dos serviços.

De acordo com Nandigam et al. (2005), os *Web Services* são definidos como unidades de aplicação lógica que transcendem linguagens de programação, sistemas operacionais e protocolos de comunicação em rede. São aplicações projetadas para serem utilizadas por outras aplicações, em contraste com as páginas Web tradicionais, diretamente direcionadas para usuários humanos.

Na perspectiva do uso de Serviços Web Semânticos, Martin et al. (2007*a*) cita algumas destas funcionalidades que podem ser alcançadas através de seu uso, são elas:

- **Descoberta Automática de Serviços**: É um processo automático para localização de serviços que possam atender a uma determinada necessidade, baseado em restrições especificadas por usuários. Por exemplo, o

usuário pode necessitar de um serviço que venda passagens aéreas entre duas cidades especificas e aceite um determinado cartão de crédito. Atualmente, para que esta tarefa seja executada, um usuário deve utilizar uma ferramenta de busca para encontrar um serviço, ler a página Web deste serviço, e executar o serviço manualmente, para determinar se ele satisfaz suas restrições. Com Serviços Web Semânticos, a informação necessária para descoberta do serviço poderia ser especificada através de marcações semânticas, interpretáveis por computador, presentes na página do serviço, e uma ferramenta de busca baseada em ontologias poderia ser usada para localizar os serviços automaticamente.

- **Invocação Automática de Serviços**: É a invocação automática de um Serviço Web por um programa de computador ou agente de software, dada apenas uma descrição declarativa deste serviço, diferente do caso em que agentes são pré-programados para invocar um serviço em particular. Isto é necessário, por exemplo, para que um usuário possa fazer uma compra, em um site encontrado por este usuário, de uma passagem aérea de um vôo particular. Serviços Web Semânticos possuem descrições semânticas dos parâmetros usados na execução desse serviço, bem como dos resultados dessa execução, tendo ela obtido sucesso ou falha. Um agente de software poderá interpretar estas descrições semânticas dos parâmetros de entrada necessários para invocar o serviço, bem como da informação que será retornada, desta forma sendo possível invocar este serviço de forma transparente ou semi-transparente ao usuário.

- **Composição e interação automática de serviços**. Esta tarefa envolve a seleção, composição e interação de serviços Web para realizar uma tarefa complexa, a partir de uma descrição em alto nível de um objetivo. Por exemplo, o usuário pode querer fazer todos os arranjos envolvidos na realização de uma viagem a uma conferência. Na maneira tradicional, o usuário deve selecionar os serviços, especificar a composição manualmente, e ter certeza de que qualquer software necessário para a interação entre estes serviços, que deverão trocar informações, está bem configurado. Com Serviços Web Semânticos, a informação necessária para selecionar e compor serviços pode ser encontrada nas páginas dos provedores dos serviços. Um programa pode ser construído para manipular estas informações, junto com a especificação dos objetivos da tarefa, para executar a tarefa automaticamente.

Dentro deste contexto, este trabalho busca solucionar problemas relacio-

nados ao uso de Serviços Web Semânticos. Tais problemas serão discutidas a seguir.

1.1 Problemática

Como visto na seção anterior, o uso de Serviços Web Semânticos possibilitará a automatização de diversos processos, hoje em dia realizados manualmente, através do uso de descrições semânticas presentes nos Serviços Web Semânticos. Deste modo, a comunidade tem se dedicado ao desenvolvimento de tecnologias que possibilitem a utilização das descrições semânticas destes serviços. Ou seja, ferramentas que realizem as tarefas citadas na seção anterior.

Dentro deste contexto, diversos grupos têm efetuado pesquisas objetivando a construção de algoritmos capazes de efetuar a descoberta e composição automática de serviços, chamados *matchmaking*. Várias arquiteturas e algoritmos têm sido desenvolvidos e publicados constantemente, como Paolucci et al. (2002*b*); Kuster et al. (2007); Klusch et al. (2006); Calado et al. (2009). No entanto, por se tratar de um campo de pesquisa ainda recente, como citado por Martin et al. (2007*a*) e Klusch (2008), a pesquisa em descoberta e composição de serviços semânticos ainda está em desenvolvimento, principalmente no que diz respeito ao planejamento automático de processos para composição de serviços.

Além disso, não há uma padronização entre estes algoritmos, ou seja, um usuário terá que adaptar seu sistema a um determinado algoritmo para utilizá-lo, e caso necessite usar outro algoritmo, ele terá que readaptar seu sistema.

Ademais, os resultados obtidos por Weise et al. (2008) indicam que a escolha das técnicas para descoberta e composição de serviços estão relacionadas ao cenário em que estas técnicas serão utilizadas. Além disso, em ambientes dinâmicos estes cenários podem mudar (Por exemplo, um sistema pode crescer e aumentar o número de serviços em seu repositório), o que implica na necessidade do uso de diferentes técnicas no mesmo sistema para aumentar a qualidade do processo de descoberta e composição.

1.2 Objetivo

Tendo em vista a problemática apresentada, este trabalho tem como principal objetivo conceber e desenvolver um *Middleware* que ofereça a descoberta e composição de serviços de maneira **automática** e **flexível**. Possibilitando que

o usuário do Middleware possa integrar diferentes ferramentas de acordo com as necessidades da aplicação que irá utilizar o *Middleware*.

Além disso, o *Middleware* deve levar em consideração outros requisitos como:

- Mecanismos para **invocação** de serviços;
- Mecanismo de **tolerância** a falhas para possibilitar que o *Middleware* possa encontrar um substituto para um serviço que não consiga ser invocado.

1.3 Relevância

A partir do desenvolvimento deste trabalho, espera-se a obtenção dos seguintes ganhos:

- O desenvolvimento de uma ferramenta capaz de descobrir, compor e invocar Serviços Web Semânticos de forma automática e transparente ao usuário;
- A capacidade de integração de ferramentas e técnicas ao *Middleware* proposto neste trabalho, com a finalidade de melhorar a qualidade do processo de descoberta, composição e invocação de serviços.

Ressalta-se, portanto, que os ganhos citados são relevantes para a pesquisa no domínio de Serviços Web Semânticos, pois o Middleware proposto apresenta funcionalidades necessárias em aplicações baseadas em Serviços Semânticos.

1.4 Estrutura do Trabalho

Este trabalho está organizado como segue:

- No Capítulo 2, encontra-se a fundamentação teórica deste trabalho. São apresentados conceitos relacionados às noções de *Service Oriented Architecture* (Arquitetura Orientada a Serviços), Web Service e Web Semântica.
- No Capítulo 3, é apresentada uma visão geral do Middleware proposto neste trabalho, mostrando detalhes da sua arquitetura e funcionalidades, envolvendo requisitos funcionais e não-funcionais.

- O Capítulo 4 aborda aspectos de implementação do Middleware e as características relacionadas a sua adaptabilidade e configuração.
- No Capiítulo 5 é contruído um cenário de aplicação para este trabalho, mostrando as etapas envolvidas no uso do Middleware, desde a sua personalização e configuração, até seu funcionamento.
- O Capítulo 6 apresenta uma análise dos principais resultados em relação a este trabalho e alguns outros propostos pela comunidade.
- No Capítulo 7 encontram-se as considerações finais e consclusões deste trabalho, fornecendo apontamentos para os desdobramentos imediatos do presente trabalho.

Capítulo 2

Fundamentação Teórica

O objetivo deste capítulo é apresentar a fundamentação teórica referente ao foco deste trabalho. Será apresentada uma pequena introdução sobre as tecnologias e conceitos relacionados aos termos principais que embasam esta pesquisa.

Inicialmente serão mostrados conceitos relacionados à Arquitetura Orientada a Serviços (SOA) e Web Services. Em seguida serão abordados os conceitos da Web Semântica e suas camadas. Após isso, os Serviços Web Semânticos (SWS) serão discutidos, mostrando seus conceitos básicos e principais modelos conceituais. Por fim, será discutida a necessidade e os conceitos relacionados à confiabilidade no uso de serviços.

2.1 Arquitetura Orientada a Serviços – SOA

Apesar de haver várias definições um tanto divergentes sobre o termo SOA, em geral ele é entendido como uma evolução da computação distribuída, na qual as lógicas de negócio são encapsuladas na forma de serviços de maneira modular, de forma que possam ser invocados por clientes.

A seguir são apresentadas algumas das principais características da Arquitetura Orientada a Serviços (Singh & Huhns (2004)):

1. Independência de implementação: serviços implementados em diferentes tecnologias podem se comunicar, visto que em SOA, o que se considera é a interface do serviço. Por exemplo, uma aplicação desenvolvida na plataforma JEE pode invocar um serviço que foi desenvolvido na plataforma .NET;

2. Fraco acoplamento: em SOA cada serviço, deve ser entendido como um

elemento independente, da mesma forma que um componente é independente na Arquitetura Orientada a Componentes;

3. Granularidade: os serviços em SOA devem ser tomados como elementos de baixa granularidade, pois se deve ter uma visão mais abstrata quanto possível, ocultando detalhes como modelagens de ações e interações;

4. Configuração flexível: Os sistemas devem dar suporte à configuração tardia, isto é, em tempo de execução, devido ao fato que os próprios serviços podem ser conectados entre si.

Devido a tais características, a Arquitetura Orientada a Serviços, tem sido bastante disseminada, visto que possibilita benefícios como um alto nível de abstração na construção de sistemas de grande porte, desenvolvimento de ferramentas capazes de gerenciar todo o ciclo de vida da aplicação etc.

Pelo fato de SOA ser uma arquitetura, não existe uma implementação padrão. Ela pode ser implementada utilizando diversas tecnologias, como *Web Services* (Booth et al. (2004)), DCOM (Microsoft (1996)), etc. Dentre estas, a tecnologia de *Web Services* aparenta ser a mais promissora e sobre a qual mais se tem trabalhado para se tentar agregar semântica às descrições dos serviços.

2.2 *Serviços Web*

Um *Web service* é um componente de software que pode ser acessado pela Internet e usado remotamente. A comunicação dos *Web services* é construída sobre tecnologias padrões da Internet, como XML, HTTP, e outros protocolos que dão suporte a interoperabilidade. Usando protocolos padronizados, *Web services* permitem que desenvolvedores criem aplicações que são compatíveis com diferentes linguagens de programação, sistemas operacionais, plataformas de hardware e que são acessíveis de qualquer localização geográfica.

Como resultado, qualquer sistema capaz de efetuar comunicação utilizando os protocolos já bem estabelecidos da Internet, como citados acima, pode comunicar-se com um *Web service*. A única informação que o provedor do serviço e o consumidor precisam compartilhar são as entradas, saídas e sua localização, ou seja, disponibilizando sua interface e abstraindo totalmente a maneira como foram implementadas.

Segundo a visão de Kreger (2001), a arquitetura dos *Web Services* está baseada na interação dos seguintes papéis: provedor do serviço, registrador de serviços e consumidor de serviços. As interações envolvem a publicação, a

busca e a invocação do serviço. Neste esquema, o provedor do serviço desenvolve o serviço e publica sua descrição em um registrador de serviços. Já o consumidor de serviço efetua busca nos registros de serviços para descobrir o serviço que atende as suas necessidades, através das descrições adicionadas pelos provedores de serviços. Após descobrir o serviço ideal o consumidor utiliza a descrição para interagir diretamente com o provedor e invocar o serviço desejado. Esta interação pode ser vista na figura 2.1.

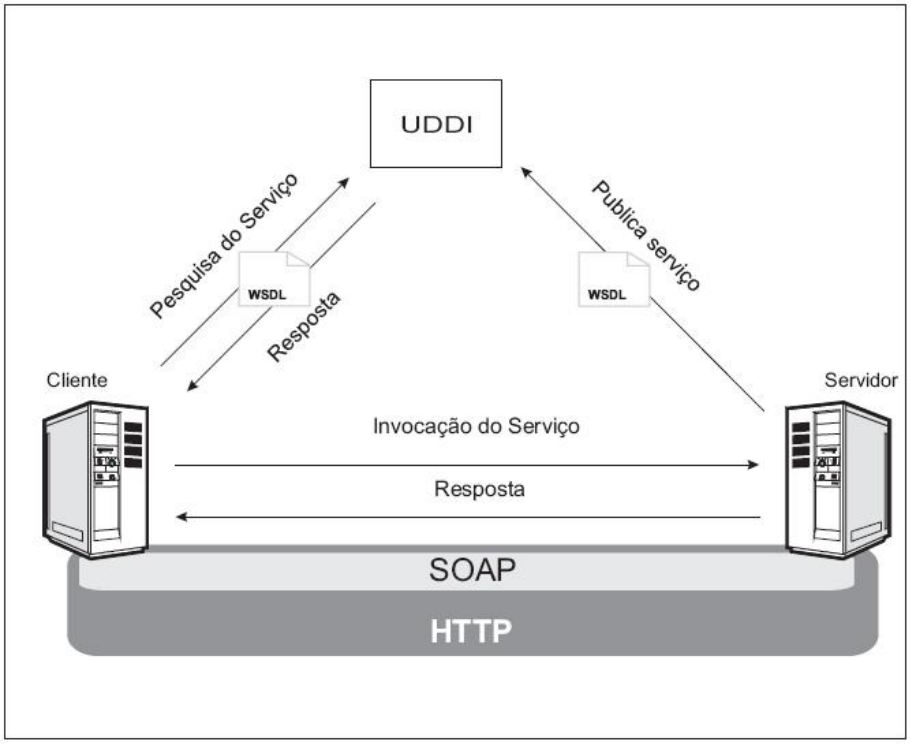


Figura 2.1: Modelo da interação entre cliente, servidor e repositório de descrições de serviços

Diante do exposto, algumas tecnologias foram desenvolvidas como forma de permitir tanto a publicação quanto a descoberta e invocação de serviços, como: registros UDDI[1] (repositórios de descrições de serviços que permitem a busca), WSDL[2] (linguagem para descrição de interfaces, protocolos de invocação e detalhes de distribuição de Web Services) e o protocolo de troca de mensagens SOAP[3].

[1] *Universal Description, Discovery and Integration*
[2] *Web Service Description Language*
[3] *Simple Object Access Protocol*

2.3 Web semântica

A Web Semântica estende a Web clássica, provendo uma estrutura semântica para páginas Web, a qual permite que tanto agentes humanos quanto agentes de software possam entender o conteúdo presente em páginas Web. Dessa forma, a Web Semântica provê um ambiente no qual agentes de software podem navegar através de páginas Web e executar tarefas sofisticadas. Em outras palavras, a Web Semântica é necessária para expressar informações de forma precisa e que possam ser interpretadas por máquinas e dessa forma permitir que agentes de software possam processar, compartilhar, reusar, além de poder entender os termos que estão sendo descritos pelos dados (Devedzic (2004)).

A seguir será apresentada uma descrição do modelo proposto por Berners-Lee para estruturação da Web Semântica, baseado em camadas de informações. Além disso, serão discutido aspectos relacionados a linguagem para representação de ontologias para a Web Semântica (OWL), dada sua importância no entendimento de outros conceitos relacionados a este trabalho, como ontologia de serviços.

2.3.1 Camadas

As camadas da Web Semântica foram inicialmente propostas por Tim Berners-Lee. A figura 2.2 ilustra as camadas da Web Semântica, que são:

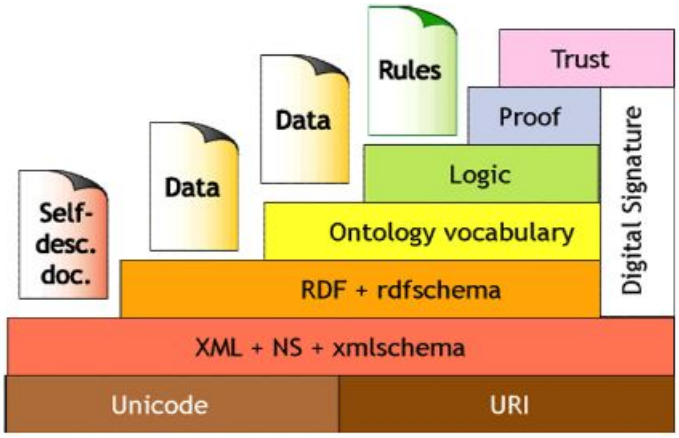


Figura 2.2: Camadas da Web Semântica proposta inicialmente em Berners-Lee et al. (2001).

- URI e Unicode: uma das principais características da Web é sua flexibilidade para atribuir *coisas* ou *recursos*. Esssa flexibilidade é provida pelo <URI>(*Unified Resource Identifiers*) e *Unicode*. URI é um identificador

que consiste em uma sequência de caracteres que obedecem a uma regra de sintaxe. Além disso, tal identificador possibilita a identificação única de um recurso através de um conjunto extensível de esquemas de nome URI (Berners-Lee et al. (2005));

- XML e XML *Schema*: estes estão acima da camada URI e Unicode. XML (*eXtensible Markup Language*) é utilizado para prover estruturas que possam ser processadas por aplicações computacionais. XML oferece aos usuários uma descrição de dados estruturados. XML *Schema* é utilizado para descrever a estrutura de um documento XML. Apesar de aplicações computacionais processarem documentos XML, esta linguagem descreve a estrutura sintática apenas. Por essa razão, uma nova camada é indispensável para definir a estrutura semântica;

- RDF e RDF *Schema*: o objetivo do RDF (*Resource Description Framework*) é permitir que recursos da Web Semântica possam ser acessados por máquinas Breitman (2005). Em outras palavras, o objetivo do RDF é prover uma representação minimalista do conhecimento da Web Shadbolt et al. (2006). Além disso, o RDF descreve qualquer recurso na Web, como imagens, páginas Web, vídeos, pessoas e assim por diante. RDF *Schema* é uma extensão semântica ao RDF e é responsável por prover um conjunto de recursos inter-relacionados. Dessa forma, RDF *Schema* é útil para descrever a estrutura de um documento RDF. No entanto, RDF *Schema* não suporta inferência nem expressividade. Portanto, uma nova camada é necessária para garantir expressividade;

- Ontologia: de acordo com Gruber (1993), uma ontologia é a especificação de um vocabulário representacional para um domínio de discurso compartilhado. Tal termo foi emprestado pela filosofia, onde ontologia é um estudo sistemático dos princípios de um determinado domínio;

- Lógica: esta camada permite a criação de regras. Dessa forma, inferência explícita é possibilitada por linguagens de regras;

- Prova e Confiança: a camada de prova executa as regras e verifica junto com os mecanismos da camada de confiança se uma dada prova é confiável ou não (Koivunen & Muller (2002)). No entanto, essas camadas estão atualmente sendo pesquisadas e pequenas aplicações demonstrativas estão sendo construídas;

- Assinatura Digital: essa camada provê integridade, garantia e não repudiabilidade sobre os dados da Web. Em outras palavras, assinatura

digital pode ser utilizada para garantir a autoria de um documento.

2.3.2 OWL

OWL (*Ontology Web language*) (OWL (2007)) é uma linguagem para definição de ontologias para Web semântica. Ela permite um alto grau de expressividade e inferência implícita. A OWL foi dividida em três sub-linguagens, ou dialetos, de acordo com suas capacidades de expressividade e garantia de computabilidade:

- *OWL Lite*: pode ser definida como a mais simples sub-linguagem da OWL, dado que, sua expressividade é muito baixa se comparada com as outras sub-linguagens. A OWL Lite é recomendada para definição de simples hierarquias ou restrições;
- *OWL-DL*[4]: mais expressiva que a OWL Lite, visto que é baseada em Lógica de Descrição Baader & Nutt (2003). Igualmente importante, a OWL-DL permite a verificação de satisfatibilidade de conceitos e classificação de hierarquias;
- *OWL Full*: representa a mais expressiva sub-linguagem da OWL. Esta linguagem deve ser utilizada quando a expressividade do conhecimento é mais importante que a garantia de computabilidade.

2.4 *Serviços Web Semânticos*

Serviços Web Semânticos foram inicialmente propostos por McIlraith et al. (2001) como uma extensão dos serviços Web com descrições semânticas a fim de fornecer definições declarativas formais de suas interfaces, bem como a captura declarativa que os serviços fazem.

Os *Serviços Web Semânticos* podem ser vistos como um ponto de intersecção entre a tecnologia dos *Web Services* e a Web semântica. Isto se deve ao fato dos *Serviços Web Semânticos* proverem a interoperabilidade semântica encontrada na Web semântica juntamente com a dinamicidade de recursos presentes nos *Web Services* C.Daconta et al. (2003); Martin et al. (2007*a*) (figura 2.3).

Segundo Payne & Lassila (2004), os *Serviços Web Semânticos* podem ser definidos como um aprimoramento das descrições dos *Web Services* através da utilização de anotações semânticas, de forma que possa ser conseguido

[4]DL significa *Description Logic* ou, em Português, Lógica de Descrição

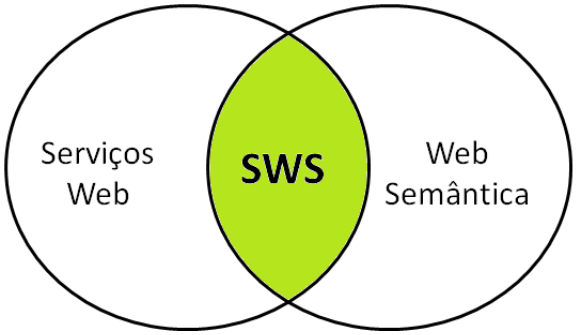


Figura 2.3: Os Serviços Web Semânticos surgem como integração entre os Serviços Web e a Web Semântica

um alto grau de automação, tanto na descoberta de serviços, quanto na composição, monitoramento e invocação de *Web Services*.

A característica essencial dos SWS é o uso de linguagens com semântica bem definida que possibilitam o raciocínio automatizado sobre as características dos serviços. Várias linguagens tem sido propostas e utilizadas, incluindo aquelas derivadas da Web Semântica, como RDF e OWL, linguagens específicas para Serviços Semânticos como a Web Service Modeling Language (WSML), ou outras derivadas da pesquisa em Sistemas Baseados em conhecimento como F-Logic e OCML (Pedrinaci et al. (2010)).

Apesar da grande diversidade, todos os *frameworks* para descrição de serviços, como McIlraith et al. (2001); Roman et al. (2005); Preist (2004), tomam como ponto de partida os princípios da orientação a serviços e são fortemente baseados na Arquitetura Orientada a Serviços (SOA). Assim, eles vêem a construção de sistemas como um processo que envolve o encapsulamento de componentes reutilizáveis como serviços, sua publicação em registros compartilhados, a localização destes serviços nestes repositórios, sua composição em um fluxo de processos executáveis, e sua eventual execução. *Frameworks* para SWS essencialmente propõe o uso de semântica para alcançar um alto nível de automação na execução destas atividades.

As tarefas e conceitos tipicamente utilizados no campo dos SWS foram identificados em Pedrinaci et al. (2010). Estas definições estão de acordo com as definições do *Web Services Glossary* da W3C (Haas & Brown (n.d.)). Estes conceitos são discutidos a seguir:

- **Crawling** é a tarefa de navegação na Web para localizar os serviços existentes. Esta tarefa é essencialmente idêntica a de qualquer outro Web Crawling com a diferença de que as informações solicitadas não são páginas da Web, mas sim arquivos WSDL ou, mais recentemente páginas HTML, descrevendo Web APIs.

- A **Descoberta** envolve a localização de serviços que são capazes de cumprir as exigências de um usuário. Esta tarefa envolve, a partir de definições abstratas das necessidades do usuário, interpretar estas exigências de tal forma que possam ser usadas para identificar os serviços da Web que podem posteriormente ser utilizados para fornecer as necessidades do usuário.

- **Matching**, também conhecido como **Matchmaking**, é a tarefa que, dada a um pedido de algum tipo de serviço, tenta identificar os serviços Web que tem um certo degrau de semelhança com o pedido. A pesquisa em SWS Matching tem dedicado esforços substanciais para formalizar a funcionalidade dos serviços Web de forma que possa suportar a correspondência automática usando raciocinadores. Em geral, a funcionalidade do serviço da Web é especificado em termos de entradas, saídas, précondições e efeitos.

- **Classificação** (Ranking) é a atividade que dado um conjunto de Serviços obtidos no processo de **Matching**, classifica os serviços de acordo com um conjunto de preferências. Estas preferências são freqüentemente dadas em tempo de invocação e são especificadas em termos de propriedades não-funcionais dos Serviços Web como: preço, qualidade do serviços.

- **Seleção** é a tarefa em que tendo obtido uma lista com possíveis serviços, um destes é selecionado para ser invocado. Esta atividade é freqüentemente realizada por humanos mas existem sistemas que realizam este passo automaticamente baseando-se na classificação realizada anteriormente.

- A **Composição** é a tarefa responsável de combinar serviços Web, a fim de realizar uma tarefa complexa. Normalmente essa tarefa é acionada sempre que o sistema é incapaz de encontrar um serviço Web que preencha todos os requisitos. O resultado de uma tarefa de Composição é uma Orquestração de serviços que, dadas algumas condições iniciais e um conjunto de serviços Web, atenderia os requisitos do usuário quando executado. A maioria do trabalho na composição automática de Serviços Web Semânticos tem sido abordada como uma tarefa de planejamento, que acontece a partir da especificação formal de entradas, saídas, précondições e efeitos de serviços para gerar orquestrações adequadas.

- **Orquestração** define a seqüência e as condições para a execução de serviços Web, a fim de realizar um objetivo complexo através da combinação

das funcionalidades fornecidas por serviços Web existentes. Orquestração inclui definições de fluxo de dados (ou seja, como os dados são propagados e utilizados em todo o processo) e de controle de fluxo (ie, quando uma determinada atividade deve ser executada).

- **Coreografia** descreve as interações de serviços com seus usuários, segundo a qual qualquer cliente de um serviço Web, pode ser uma máquina ou um humano, é considerado um usuário. Uma coreografia define o comportamento de um serviço Web, ou seja, as mensagens trocadas na sua execução, do ponto de vista do cliente.

- **Invocação** diz respeito com a chamada real de uma operação de um serviço web. Invocação é, portanto, intimamente relacionada com coreografias que especificam uma ordem para a realização de um conjunto de invocações.

Ainda nesta perspectiva, estas etapas podem variar de acordo com as preferências de cada autor. Neste trabalho, o processo de Descoberta de serviços engloba as etapas de *descoberta, matching, classificação e seleção* definidos por Pedrinaci et al. (2010). Esta escolha se dá devido necessidade de integração destas etapas com o objetivo de automatizar o uso de Serviços Semânticos.

Ainda em relação ao processo de descoberta e composição de serviços, alguns autores utilizam o termo *matching direto* para o processo de descoberta que obtém como resultado um único serviço, enquanto o termo *matching indireto* caracteriza os processos de descoberta que utilizam composições de serviços para atender as necessidades do usuário (Giv et al. (2004)).

Ainda neste contexto, a criação de modelos de processos para composição de serviços pode ocorrer através do uso de planejadores dotados de técnicas de Inteligência Artificial (planejamento dinâmico), como em Klusch & Gerber (2005), ou através da especificação criada por agente humanos (planejamento estático).

2.5 Modelos Conceituais

O foco do trabalho em SWS são as descrições ou anotações semânticas de serviços que oferecem suporte a automatização para uma série de atividades, como a sua descoberta, seleção e composição. Consequentemente, a comunidade tem se dedicado ao longo dos anos na criação de modelos conceituais que consigam oferecer suporte a criação destas descrições semânticas para serviços.

Assim, esta seção se dedica em detalhar as três principais abordagens propostas pela comunidade. Inicialmente, serão apresentadas as abordagens OWL-S de Martin et al. (2007*a*) e WSMO deRoman et al. (2005) que são consideradas abordagens $Top-down$ pelo fato de utilizarem de ontologias de alto nível para descrição de serviços. Em seguida será mostrada a abordagem SAWSDL/WSDL-S, proposta em Farrell & Lausen (2007), que é considerada um modelo $Bottom-up$ por se tratar de uma abordagem incremental que propõe a adição de semântica aos padrões de Web Services já existentes (WSDL), através de extensões específicas que conectam as definições sintáticas as suas descrições semânticas.

2.5.1 OWL-S: *Ontology Web Language for Services*

Embora algumas vezes a OWL-S seja referida como uma linguagem, porque provê um vocabulário padrão para representação de serviços, trata-se de uma ontologia de topo (*Upper Ontology*) lançada em meados de 2004 em continuação ao projeto DAML-S (Ankolekar et al. (2001)).

Por ser uma submissão do W3C, a OWL-S tem atraído o interesse de grande parte da comunidade científica relacionada a *Serviços Web Semânticos* e possui uma grande quantidade de ferramentas desenvolvidas. Diferentemente da linguagem de descrição de serviços WSDL, a OWL-S efetiva a descrição de serviços semanticamente, o que permite uma maior interoperação. Segundo Martin et al. (2004) a estruturação de uma ontologia de serviços é motivada pela necessidade de três essenciais tipos de conhecimento básicos sobre um serviço, o que poderia ser referenciado pela resposta às três perguntas abaixo:

- *O que o serviço faz?*
- *Como o serviço trabalha?*
- *Como acessar o serviço?*

A resposta às três perguntas acima habilita um agente de software, a saber: qual a interface e o objetivo do serviço, quais são as etapas realizadas em sua execução e quais os protocolos necessários a sua invocação.

Diante do exposto, a OWL-S foi desenvolvida tendo em mente três componentes básicos:

1. Um perfil (*Profile*) que apresenta as “intenções” do serviço, isto é, qual a funcionalidade do serviço, o que ele provê;

2. Um modelo do Serviço (*ServiceModel*) que apresenta um maior detalhamento do funcionamento do serviço;

3. Um *grounding*[5] que apresenta detalhes relacionado a maneira como o serviço pode ser invocado.

O modelo abstrato de um *Serviços Web Semânticos* em OWL-S com todos os relacionamentos entre as entidades presentes em um *Serviços Web Semânticos* pode ser visto na figura 2.4.

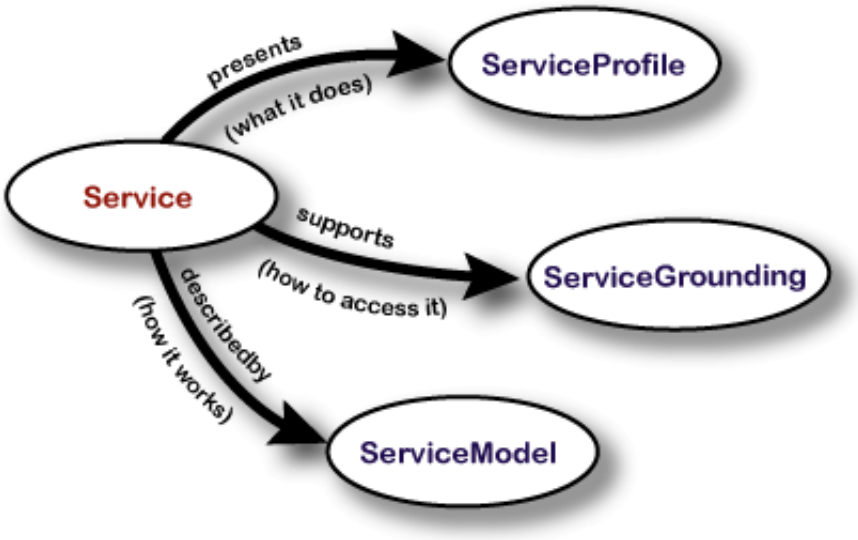


Figura 2.4: Estrutura proposta de um serviço semântico em OWL-S
Imagem extraída de Martin et al. (2004).

2.5.2 WSMO

WSMO é uma submissão da W3C de uma ontologia que tem por objetivo descrever todos os aspectos relevantes para a automação completa ou parcial de descoberta, seleção, composição, mediação, execução e acompanhamento dos serviços Web, proposta em Roman et al. (2005). WSMO tem suas raízes na Modelagem Web Service Framework, ver Fensel & Bussler (2002).

Assim, WSMO fornece uma ontologia para descrever serviços Web baseados em WSMF. WSMO propõe quatro elementos de alto nível como os principais conceitos, são eles: *Ontologias(Ontologies)*, *WebServices*, *Objetivos(Goals)* e *Mediadores(Mediators)* (Pedrinaci et al. (2010)), como pode ser visto na Figura 2.5.

[5]Obs: Não foi encontrado em nenhum texto, relacionado a OWL-S, em língua Portuguesa um termo equivalente, portanto, preferiu-se utilizar o termo no original em Inglês

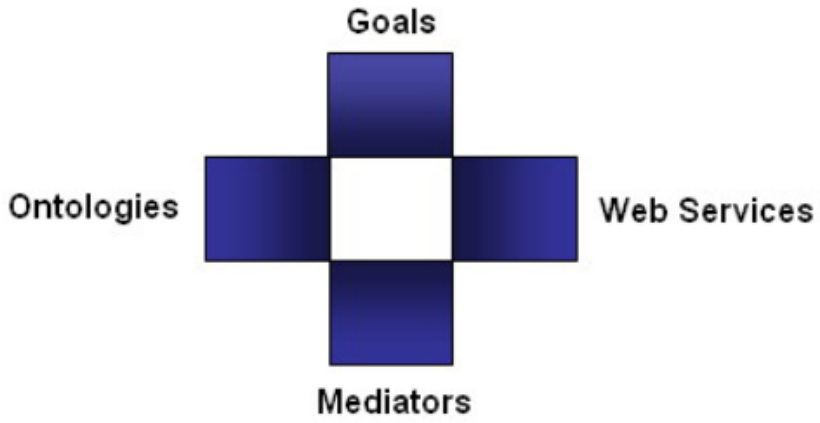


Figura 2.5: Os elementos de alto nível do WSMO (Retirado de Roman et al. (2005))

- **Ontologias** provêem a terminologia usada pelos outros elementos WSMO para descrever os aspectos relevantes do domínio em questão.
- **Web services** descrevem a entidade computacional permitindo acesso para serviços que provêem algum valor ao domínio. Estas descrições compreendem as capacidades, interfaces e funcionamento interno dos Serviços Web. Todos estes aspectos são descritos usando as terminologias definidas pelas *ontologias*.
- **Goals** representam as perspectivas dos clientes, apoiando a representação dos desejos dos usuários para uma determinada funcionalidade.
- Finalmente, **Mediadores** representam elementos que lidam com problemas de interoperabilidade entre quaisquer dois elementos WSMO. Na verdade, uma princípio fundamental por trás do WSMO, é a centralidade da mediação como forma de reduzir o acoplamento e lidar com a heterogeneidade que caracteriza a web

2.5.3 WSDL-S e SAWSDL

WSDL-S é uma abordagem de baixa complexidade para associar anotações semânticas a serviços Web. A principal inovação do WSDL-S está no uso da extensibilidade dos elementos e atributos suportados pela especificação WSDL (Sivashanmugam et al. (2003)). Usando a extensibilidade do WSDL, anotações semânticas em forma de referências URI para modelos externos (que pode ser ontologias, e foram amplamente denominados modelos conceituais) podem ser adicionados as interfaces, operações e uma mensagens dos serviços. WSDL-S é independente da linguagem utilizada para a definição dos modelos semânti-

cos e contempla expressamente a possibilidade de utilizar WSML, OWL e UML como potenciais candidatos (Akkiraju et al. (2005)).

WSDL-S oferece um conjunto de atributos de extensão e elementos para associar as anotações semânticas. O atributo de extensão *modelReference* permite que se especifique a associação entre uma entidade WSDL e um conceito em um modelo semântico. Esta extensão pode ser usado para anotar os tipos complexos elementos de um esquema XML, operações WSDL e os elementos de extensão pré-condição (*precondition*) e efeito (*effect*).

WSDL-S define dois novos elementos filhos do elemento *operation* da WSDL, são eles *precondition* e *effect*. Esses elementos facilitam a definição das condições que devem ser consideradas antes de executar uma operação e os efeitos que a execução teria. Esta informação é normalmente utilizado para descobrir serviços adequados.

O atributo de extensão *schemaMapping* pode ser usado para especificar mecanismos para lidar com as diferenças estruturais entre os elementos do XML Schema e tipos complexos e seus respectivos conceitos do modelo semântico. Estas anotações podem ser usados para o que é referido como o *lifting* e *lowering* dos dados de execução, ou seja, transformar os dados sintáticos em sua contraparte semântica e vice-versa) (Verma & Sheth (2007)).

Finalmente, WSDL-S inclui o atributo de extensão da categoria sobre o elemento de interface para definir as informações de categorização para a publicação de serviços Web em registros.

SAWSDL (Semantic Annotations for WSDL and XML Schema) é uma versão restrita e homogeneizadas de WSDL-S, incluindo algumas alterações tentando dar um maior nível de genericidade para as anotações e desconsiderando as questões para as quais não existia qualquer acordo entre a comunidade no momento da especificação foi criada. Ela foi adotada como recomendação da W3C em Agosto de 2007 (Farrell & Lausen (2007)).

Existem essencialmente três principais diferenças entre SAWSDL e WSDL-S. O primeiro é o fato de que os atributos *precondition* e *effect* não estão diretamente contemplados, pois não houve acordo sobre como modelá-los dentro da comunidade da Web Semântica e Serviços Web Semânticos. Vale notar, contudo, que SAWSDL não impede a inclusão deste tipo de anotações, como ilustrado no manual de uso gerado pelo grupo de trabalho da SAWSDL (Farrell & Lausen (2007)). Em segundo lugar, a anotação da *category* foi substituído pelo atributo mais geral *modelReference*, que pode ser usado para anotar definições de tipos complexos, definições de tipo simples, declarações de elementos e declarações de atributos em XML Schema, bem como interfaces WSDL, operações e falhas. Finalmente, a anotação WSDL-S *schemaMapping* foi de-

composta em dois atributos diferentes, chamados *liftingSchemaMapping* e *loweringSchemaMapping*, para identificar especificamente o tipo de transformação realizada (Pedrinaci et al. (2010)).

2.6 Confiabilidade na Computação Orientada a Serviços

Confiabilidade de software pode ser definida como a probabilidade de uma operação livre de falhas de um programa de computador para um determinado momento em um ambiente especificado (Musa et al. (1987)). Em sistemas baseados em Web Services, é necessário o uso de mecanismos que proporcionam a confiabilidade no uso desses serviços, caso contrário o sistema torna-se vulnerável.

Na utilização de serviços, podemos identificar dois tipos de erros que podem causar problemas no sistema:

- **Serviço não pôde ser executado**: Na utilização de Web Services, existe a possibilidade de falha na execução de um serviço, devido a problemas de rede ou indisponibilidade de um serviço. Se essa falha não for tratada, o sistema pode parar de funcionar ou funcionar incorretamente.

- **Serviço fornece resultados incorretos**: Como todos os tipos de software, os serviços podem conter bugs que fazem com que a prestação de resultados incorretos em determinados cenários. Além disso, devido à reutilização de serviços, o utilizador do serviço nem sempre é o seu desenvolvedor, o que implica um grau de incerteza sobre os reais objetivos e funcionamento destes serviços. Assim, um aplicativo pode usar um serviço que não atende às suas expectativas. Este tipo de falha é mais grave porque, se não for tratada, pode levar o sistema a trabalhar com dados inconsistentes que poderiam causar problemas significativos.

Zhang & Zhang (2005) define os requisitos para garantir a confiabilidade em sistemas baseados em Web Services, como mostrado na Figura 2.6. Este requisitos são divididos em funcionais e não funcionais, como pode ser visto abaixo:

Requisitos Funcionais:

- **Resultados Corretos**: implica que o serviço Web gera uma saída razoável a partir da entrada derivada do domínio do problema.

- **Tolerância a Falhas**: implica que o serviço Web é capaz de produzir resultados aceitáveis, embora esteja com defeito.
- **Capacidade de Teste**: implica que o serviço Web permite que as falhas existentes a ser detectado no momento do teste.

Requisitos não-funcionais:

- **Interoperabilidade**: implica que o serviço Web coexiste com outros componentes do sistema no contexto de um ambiente de sistema específico.
- **Acesso ao Serviço**: levar em conta a disponibilidade e desempenho.
  - **Disponibilidade**: implica que o serviço Web está disponível no momento de invocação.
  - **Desempenho**: implica que o serviço Web oferece a uma velocidade aceitável no momento de invocação.

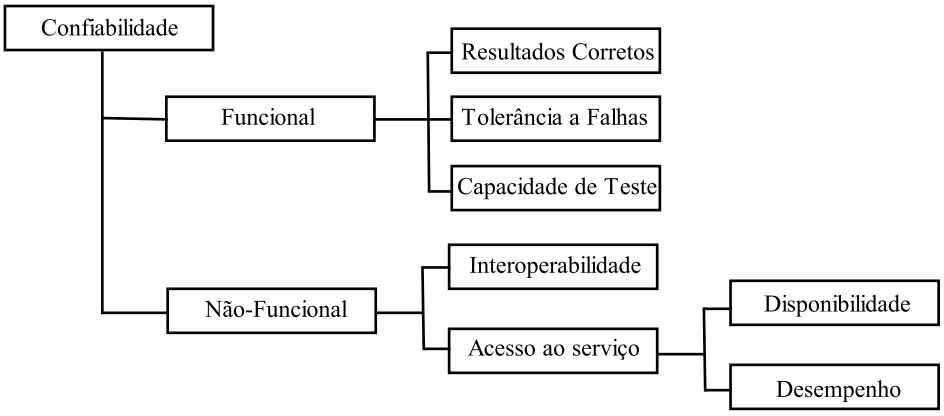


Figura 2.6: Requisitos de confiabilidade para Web Services (adaptado de Zhang & Zhang (2005))

Analisando esta definição, podemos ver que o **Resultados Corretos** e **Capacidade de Teste** requisitos estão diretamente relacionados. Para verificar a regularidade da invocação de um serviço é necessário para testar este serviço.

Em relação aos tipos de erros mencionados anteriormente, a falha **serviço não pode ser invocado** está relacionada com a *exigência de tolerância a falhas*, enquanto o erro **Serviço fornece resultados incorretos** está relacionada com os requisitos *Resultados Corretos* e *Capacidade de Teste*. Os requisitos não funcionais estão relacionados a questões de QoS (Qualidade de Software).

Capítulo 3

Visão Geral e Arquitetura do Middleware

Neste capítulo apresenta-se a concepção e o desenvolvimento do Middleware proposto no presente trabalho, mostrando sua arquitetura de alto nível e funcionalidades. O nome dado a solução proposta foi Grinv (***Gr**ow **Inv**oker*).

3.1 Visão Geral

Em resumo, o Middleware proposto neste trabalho[1] visa proporcionar a descoberta, composição e invocação automática de Serviços Web Semânticos de uma maneira extensível e também transparente para a aplicação. Desta forma, desenvolvedores podem modificar as técnicas e algoritmos utilizados nas etapas de descoberta e composição de serviços sem causar maiores impactos ao restante do sistema.

Além disso, o Grinv deve oferecer suporte a diferentes abordagens de repositórios de serviços, como repositórios baseados em Ontologias, variações da UDDI tradicional (OASIS (2006)), Banco de Dados, arquivos, entre outros. O Middleware também deve possibilitar a utilização de técnicas que venham a melhorar o desempenho na execução dos processos de descoberta e composição.

Em outra perspectiva, aplicações baseadas em Serviços Web Semânticos, como aplicações que necessitem de recursos presentes na Web, precisam tratar questões referentes à confiabilidade (Musa et al. (1987)). Assim, o Middleware deve prover mecanismos que ofereçam confiabilidade às aplicações que utilizam esta ferramenta.

[1] http://www.grow.ic.ufal.br:8080/grow/pesquisa-1/projetos/grinv-middleware-project

3.2 Representação Semântica de Serviços

Em relação à abordagem para descrição semântica de serviços, a SAWSDL proposta em Farrell & Lausen (2007) é a atual recomendação da W3C. Esta abordagem representa de forma conservadora e incremental a introdução de características semânticas de Serviços Web ao cenário prático do uso de Serviços Web. Seus objetivos são relativamente modestos. Por exemplo, esta abordagem busca prover descrições semânticas aos parâmetros de entrada e saída de serviços, o que pode ser bastante útil para evitar ambigüidades no processamento destes parâmetros em abordagens simples para descoberta de serviços. Entretanto, a SAWSDL não visa oferecer um abrangente arcabouço para suportar abordagens mais sofisticadas para descoberta, composição, ou outras atividades relacionadas ao uso de serviços que a pesquisa em Serviços Web Semânticos espera automatizar, como, por exemplo, a invocação destes serviços (Martin et al. (2007*b*)).

Neste contexto, a SAWSDL não especifica um arcabouço semântico específico para descrever o contexto semântico de um Serviço Web. Ao invés disso, ela define um pequeno conjunto de atributos para estender a WSDL [2]. Esta extensão pode ser utilizada como referência para algum outro arcabouço semântico. Assim, SAWSDL é completamente evasiva em relação à escolha de um arcabouço semântico mais sofisticado (Martin et al. (2007*b*)). Desta forma, o uso de arcabouços semânticos mais sofisticados, como WSMO, proposta por Roman et al. (2005), ou OWL-S, proposta em Martin et al. (2004), são mais apropriados para permitir a execução de atividades mais complexas. Tanto a OWL-S quanto a WSMO são submissões da W3C, o que, de certa forma, mostra sua qualidade.

WSMO e OWL-S visam representar Serviços Web que façam uso de ontologias da Web Semântica, e os objetivos de ambas as abordagens é oferecer suporte mais eficiente aos processos de descoberta, composição e interoperabilidade entre serviços. Entretanto, as duas abordagens utilizam conceitos bem diferentes. WSMO foca no papel de mediação com o objetivo de oferecer interoperação entre Serviços Web, enquanto a OWL-S mantém foco na representação de ações para oferecer suporte a planejamento de processos que possibilite composição automática de serviços (Paolucci et al. (2004)).

Entretanto, WSMO utiliza a linguagem WSML (Web Services Modeling Language) para construção de descrições semânticas para conceitos e serviços (Roman et al. (2005)). WSML é uma linguagem para criar ontologias que não é derivada de XML ou OWL (Recomendação da W3C para construção de ontolo-

[2] WebServices Description Language (http://www.w3.org/TR/wsdl)

gias na Web Semântica). A conseqüência desta escolha é a incompatibilidade entre as ferramentas que manipulam descrições em WSML e ferramentas baseadas em OWL. Assim, a aplicação necessita de ferramentas que sejam capazes de interpretar ontologias descritas em OWL e WSML, o que obriga ao desenvolvedor da aplicação conduzir uma série de mapeamentos entre estas duas linguagens.

Em contrapartida, OWL-S é baseada em OWL o que permite uma integração mais fácil entre ferramentas para OWL-S e as ferramentas para manipulação de ontologias em OWL, como a API Jena [3] (Jena é uma API Java para manipular RDF e OWL, e está entre as mais usadas pela comunidade da Web Semântica). Além disso, a comunidade propôs um considerável número de ferramentas para descoberta e composição automática de serviços baseados em OWL-S, como Paolucci et al. (2002*b*), Srinivasan et al. (2006), Klusch et al. (2006), Calado et al. (2009).

Levando estes fatos em consideração, a OWL-S foi escolhida para descrever os serviços semanticamente dentro do Middleware. O Middleware foi desenvolvido utilizando a linguagem Java e a OWL-S API [4], baseada em Jena, foi utilizada para manipular as descrições semânticas dos serviços.

Apesar da escolha da OWL-S, o desenvolvedor pode estender o Middleware com o objetivo de oferecer suporte para outras abordagens, como a SAWSDL. Neste caso, a técnica proposta em Martin et al. (2007*b*) pode ser utilizada para integrar SAWSDL e OWL-S.

3.3 Arquitetura

A Figura 3.1 mostra a arquitetura conceitual do Middleware. Os módulos desta arquitetura são comentadas a seguir:

- **Request Manager**: É o núcleo do Middleware. Ele é responsável por indicar o fluxo dos processos que devem ser realizados para atender à requisição do usuário;
- **Discovery Controller**: Este módulo é responsável por encontrar o serviço presente no repositório que é mais similar às descrições enviadas pelo usuário. Este módulo implementa um conjunto de técnicas para descoberta de serviços que são utilizados para realizar a atividade de

[3] jena.sourceforge.net/
[4] disponível em http://on.cs.unibas.ch/owls-api/

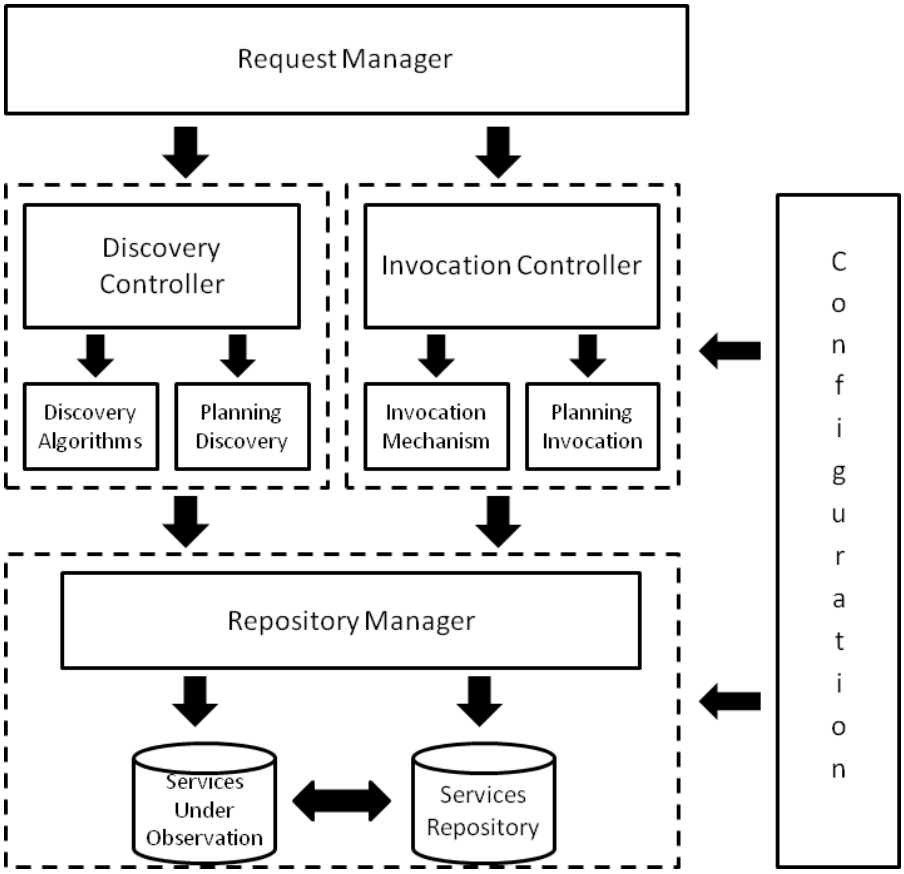


Figura 3.1: Arquitetura conceitual do Middleware

descoberta. Este módulo também é responsável por encontrar composições de serviços de acordo com a necessidade da aplicação, através de planejamento estático e dinâmico.

- **Invocation Controller**: Este módulo é responsável pela execução dos serviços, sejam eles providos pela aplicação ou descobertos pelo *Discovery Controller*. Este módulo deve ser capaz de invocar serviços simples (um único serviço) e processos compostos de uma maneira transparente à aplicação.

- **Repository Manager**: Este módulo tem a função de armazenar as descrições dos serviços que serão utilizados pelos demais módulos do Middleware. Além disso, ele deve oferecer suporte a diferentes tecnologias para armazenamento das descrições dos serviços. Para garantir o bom funcionamento do sistema, este módulo deve conter rotinas para verificar se um serviço continua disponível e atualizar suas descrições semânticas.

  O Gerenciador de Repositório tem acesso a dois tipos de repositórios: *Services Repository* (SR) e *Services Under Observation* (SUO). SR é res-

ponsável por armazenar os serviços que serão utilizados pelo sistema, enquanto o SUO armazena os serviços que apresentam problemas durante o processo de descoberta ou invocação. Por exemplo, um serviço cuja descrição semântica não está disponibilizada na Web ou um serviço que não pode ser invocado devem ser levados para este repositório.

Os serviços que se encontram no repositório SUO são isolados para evitar que seus problemas afetem o desempenho do sistema. Estes serviços defeituosos são submetidos a avaliações periódicas para verificar se eles retornaram ao seu comportamento normal e, quando isto ocorrer, estes serviços são reintegrados ao sistema;

- **Configuration**: Este módulo é responsável por ajustar o sistema de acordo com as configurações feitas pelo desenvolvedor da aplicação. Assim, este módulo carrega os arquivos de configuração que indicam as técnicas para descoberta de serviços que serão utilizadas, o mecanismo de invocação de serviços e as políticas do repositório de serviços.

3.4 Mecanismo de Tolerância a Falhas

Como citado anteriormente, sistemas baseados em Serviços Web Semânticos devem prover mecanismos de tolerância a falhas para oferecer confiabilidade no uso de serviços. O Middleware proposto oferece tolerância a falhas através da detecção de erros nos mecanismos de invocação de serviços (*Invocation Controller*). A partir desta falha, o serviço é removido do *Services Repository* (SR) e adicionado ao repositório *Services Under Observation* (SUO). Assim, este serviço não será mais utilizado pelo sistema enquanto estiver com problemas. Com a remoção do serviço defeituoso, um novo processo de descoberta é criado com o objetivo de buscar um serviço que seja similar ao serviço defeituoso, ou seja, que possua os mesmos objetivos.

Nesta busca pelo serviço substituto, o módulo *Discovery Controller* pode utilizar diferentes técnicas de descoberta e composição presentes no módulo *Discovery Algorithms*. Com o uso de diferentes técnicas, aumenta a possibilidade de se encontrar um serviço ou composição de serviços similar ao serviço defeituoso. Se nenhuma das técnicas for capaz de encontrar um novo serviço com uma similaridade aceitável em relação ao serviço defeituoso, a requisição feita pela aplicação não poderá ser atendida pelo Middleware e o módulo *Requisition Manager* envia para a aplicação uma mensagem de erro que contém informações acerca da execução do serviço defeituoso e o erro gerado.

3.5 Avaliação de Resultados através de Restrições

Como mostrado anteriormente, sistemas baseados em Serviços Semânticos necessitam de mecanismos que garantam que a execução dos serviços está produzindo resultados consistentes em relação às expectativas do sistema. Para resolver este problema, o Middleware utiliza o conceito de *Restrições* com o objetivo de avaliar a invocação dos serviços.

Para tornar possível a avaliação da invocação de um serviço particular, o desenvolvedor da aplicação deve criar *Restrições* que serão checadas após a invocação destes serviços. Estas *Restrições* são baseadas nas descrições dos serviços e nos resultados obtidos após o processo de invocação.

Por exemplo, para invocar um serviço que calcula a raiz de uma função matemática f. Uma possível *Restrição* para este serviço seria verificar se os valores obtidos na invocação do serviço são realmente raízes da função f. Ou seja, dado o conjunto de valores Cx, resultados da invocação do serviços S, S estará de acordo com as restrições se para todo valor de $x \in Cx$, teremos $f(x) = 0$. (Ou $f(x) \approx 0$ dentro de um limite aceitável).

3.6 Diagrama de Atividades

A Figura 3.2 exibe um diagrama de atividades que ilustra os processos envolvidos na execução de requisições no Middleware. Este diagrama mostra o processo completo, desde o recebimento da requisição, a descoberta do serviço que irá atender esta requisição, a invocação deste serviços, a validação dos resultados e o envio da resposta para a aplicação. Os passos deste diagrama serão discutidos a seguir:

- **Recebe Requisição**: Nesta etapa, o Middleware recebe a descrição dos serviços que a aplicação deseja executar, estas descrições serão utilizadas no processo de descoberta do serviço. A aplicação também informa os parâmetros de entrada que o serviço desejado deverá utilizar em sua invocação.

- **Seleciona técnicas de Descoberta**: Como citado anteriormente, o Middleware possui um conjunto de mecanismos para descoberta e composição de serviços. Nesta etapa, o módulo *Configuration* é responsável por carregar as configurações presentes no arquivo de configuração da aplicação.

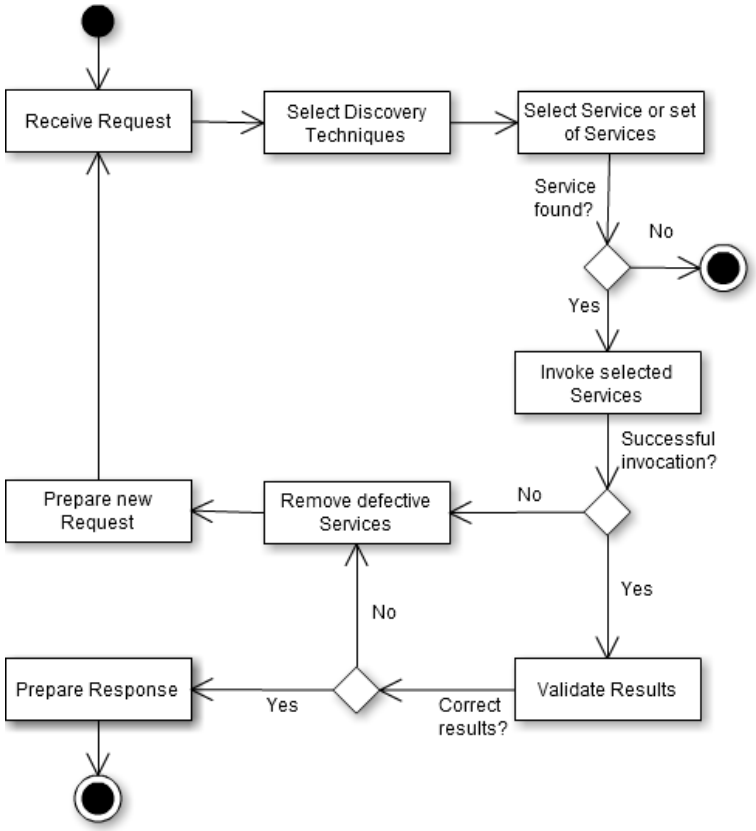


Figura 3.2: Diagramas de atividades do Middleware

- **Seleciona serviço ou conjunto de serviços**: Nesta etapa as técnicas escolhidas no passo anterior são utilizadas para buscar o serviço ou conjunto de serviços que melhor se adéqua às descrições enviadas pela aplicação. A execução do Middleware é interrompida se os mecanismos de descoberta falharem na tentativa de encontrar um serviço que ofereça as funcionalidades requisitadas pela aplicação.

- **Executa Serviços selecionados**: Com o processo de descoberta completo e o serviço configurado, o módulo *Invocation Controller* é responsável por invocar o serviço ou composição de serviços utilizando os parâmetros de entrada especificados pela aplicação. Se a invocação não for bem sucedida, o mecanismo de tolerância a falhas será ativado e se iniciará a etapa de remoção de serviços defeituosos. Se a invocação for efetuada com sucesso, começará a validação dos resultados.

- **Valida os Resultados**: Como mostrado na subseção 3.5, a validação dos resultados é feita através da avaliação de restrições. Para isso, o mecanismo de validação carrega as restrições escolhidas pela aplicação. Assim, este mecanismo irá checar quais as restrições que devem ser verificadas para o serviço que foi executado, evitando que sejam avaliadas

restrições que não façam sentido. Por exemplo, restrições que foram criadas para avaliar a execução de um serviço que calcule as raízes de uma função não devem ser aplicadas para um serviço que provê recomendação de conteúdo.

- **Remove serviços defeituosos**: Dado que ocorreu um problema na execução de um serviço (serviço não pôde ser invocado ou produziu resultados inconsistentes), o mecanismo de tolerância a falhas é responsável por remover o serviço defeituoso do sistema, como especificado na subseção 3.4, o serviço será retirado do repositório padrão (SR) e será adicionado ao repositório de serviços em observação (SUO), que possui serviços que não apresentam execução correta.

- **Prepara nova Requisição**: Com o serviço defeituoso removido do repositório, o mecanismo de tolerância a falhas cria uma nova requisição para que o Middleware possa encontrar um serviço que substitua o serviço defeituoso. Esta requisição pode ser idêntica à requisição original ou pode conter descrições de um serviço em particular que é parte de uma composição onde apenas este serviço apresenta problema.

- **Prepara Resposta**: Esta é a última etapa na execução do Middleware, ela é responsável por criar a resposta que será enviada à aplicação, com os resultados validados.

Por fim, este Capítulo buscou apresentar uma visão geral do Middleware desenvolvido no presente trabalho, descrevendo as funcionalidades providas e o funcionamento da ferramenta. Detalhes acerca da implementação destas funcionalidades serão descritos no próximo Capítulo.

Capítulo 4

Implementação do Middleware

Este Capítulo faz consideração acerca da implementação do Middleware proposto neste trabalho, apresentando as características de cada funcionalidade do Middleware e como estas funcionalidades podem ser estendidas e adaptadas de acordo com as necessidades do sistema.

4.1 Representação interna de serviços

A representação dos serviços dentro do Middleware é baseada na proposta da linguagem OWL-S (Martin et al. (2007*a*)). Assim, foi utilizado o padrão de projeto *Composite*, ver Gamma et al. (1994), com o objetivo de permitir a representação de serviços simples e compostos, de maneira semelhante aos processos *Atomic Process* e *Composite Process* da linguagem OWL-S. Os serviços estão representados no diagrama apresentado na Figura 4.1.

Neste contexto, a classe abstrata *Service* é a classe base da representação dos serviços, ela define o método *invoke* que deve conter as instruções necessárias para se invocar o serviço. Esta classe é estendida pelas classes *SimpleService* e *CompositeService*, o primeiro representa a implementação de um serviço simples e o segundo a implementação de uma composição de serviços.

Sendo assim, a classe *CompositeService* possui um conjunto de serviços que fazem parte desta composição. Entretanto, estes serviços podem estar organizados em diferentes estruturas de acordo com as características desta composição. Assim, esta classe deve ser estendida com o objetivo de representar as diferentes estruturas de composição, como as classes *SequenceComposition* e *SplitComposition* mostradas na Figura 4.1, onde a primeira representa uma composição de serviços em seqüência e a segunda uma composição em paralelo.

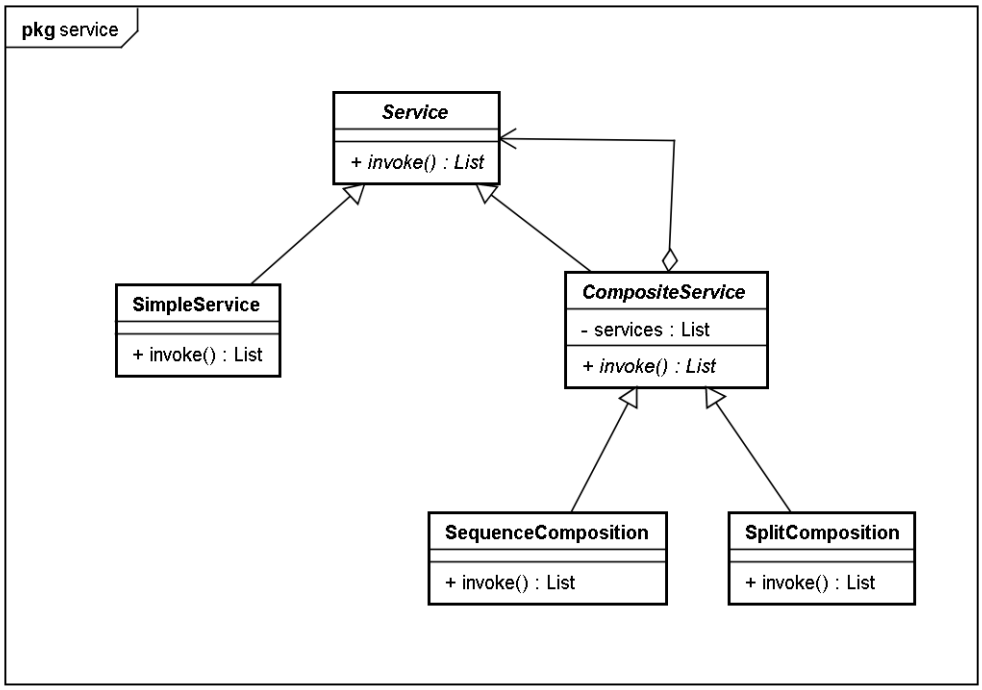


Figura 4.1: Diagrama de classes que representam os serviços no sistema

Devido à adoção do padrão *Composite* se torna possível criar composições de serviços heterogêneas em relação à sua estrutura, ou seja, composições que utilizem mais de um tipo de estrutura. Por exemplo, é possível a criação de uma composição que utilize tanto a estrutura em seqüência quanto a estrutura em paralelo.

Além disso, o usuário pode estender o Middleware com o objetivo de criar serviços modificados, tanto serviços simples como novas estruturas de serviços compostos. Por exemplo, o desenvolvedor do sistema pode necessitar que haja uma análise em relação ao desempenho na invocação de cada serviço, para isso ele pode estender a classe *SimpleService* e modificar o método *invoke* para que ele realize esta análise. Da mesma forma, o desenvolver pode desejar uma composição de serviços que seja executada com certa frequência, em um ciclo, para isso ele deve estender a classe *CompositeService* e fazer as alterações necessárias para que o método *invoke* realize esta invocação cíclica. A Figura 4.2 mostra o diagrama de classes com estas novas funcionalidades.

4.2 Gerenciador de Requisições

A Figura 4.3 mostra o diagrama de classes do módulo *Request Manager*. Como dito anteriormente, este módulo é responsável por manipular os processos no sistema através da classe *RequestManager*. A avaliação da invocação dos

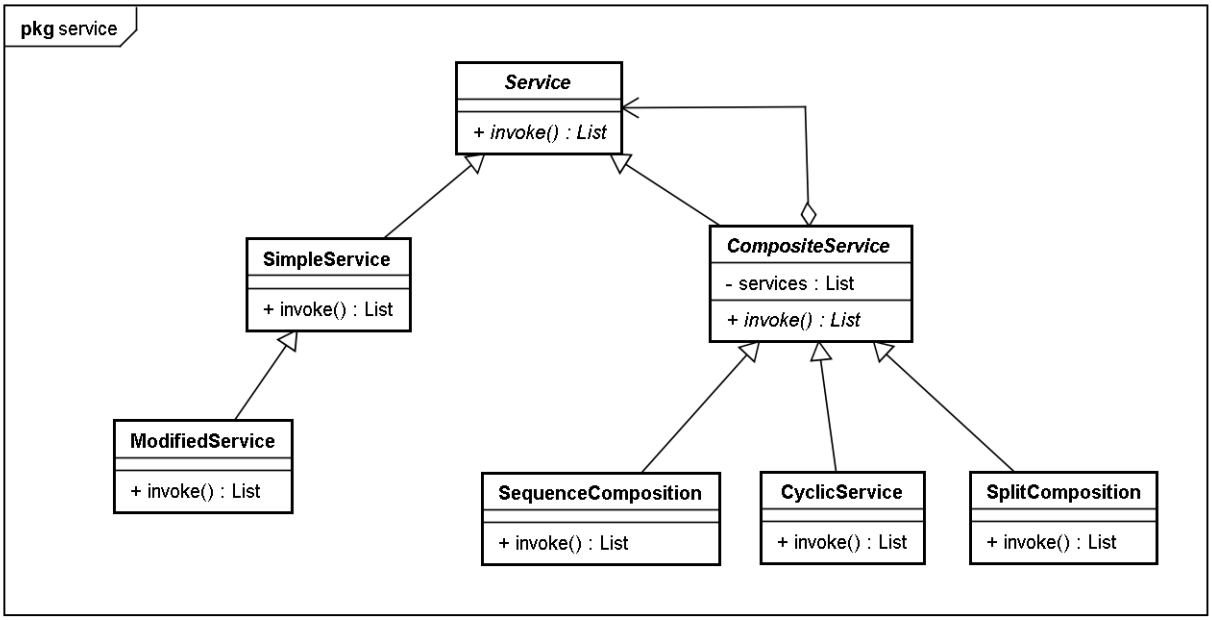


Figura 4.2: Diagrama de classes estendido

serviços é feita pela classe *ExecutionValidator*. Esta classe é responsável por verificar se os resultados da invocação são consistentes de acordo com as *Restrições* definidas pelo desenvolvedor da aplicação.

A classe *RequestManager* é responsável por gerenciar o atendimento de requisições e conduzir o funcionamento do sistema baseado nas requisições do usuário, mais detalhes sobre estas restrições na Seção 4.2.1. O mecanismo de tolerância a falhas é implementado pela classe *RecoveryMechanism*, de acordo com a Seção 3.4.

Para personalizar sua aplicação e adicionar novas restrições ao sistema, o desenvolvedor da aplicação dever criar uma nova classe que implemente esta restrição através do método *validate(Service s, Map parameters, List results)*, aonde `s` é o serviços que foi invocado, `parameters` correspondem aos parâmetros definidos pela aplicação e `results` o resultado da invocação. Além disso, esta classe deve estender a classe *Constraint*.

Com esta classe implementada, o desenvolvedor do sistema deve adicioná-la à configuração do sistema. Para isso, o arquivo de configuração do Middleware deve ser alterado para que o módulo *Configuration* possa identificar esta mudança e aplicá-la ao restante do sistema, ver Seção 4.6.

4.2.1 Requisições e Respostas

As requisições são a forma do usuário especificar que funcionalidades ele deseja obter do sistema (descoberta/invocação de serviços), as requisições tam-

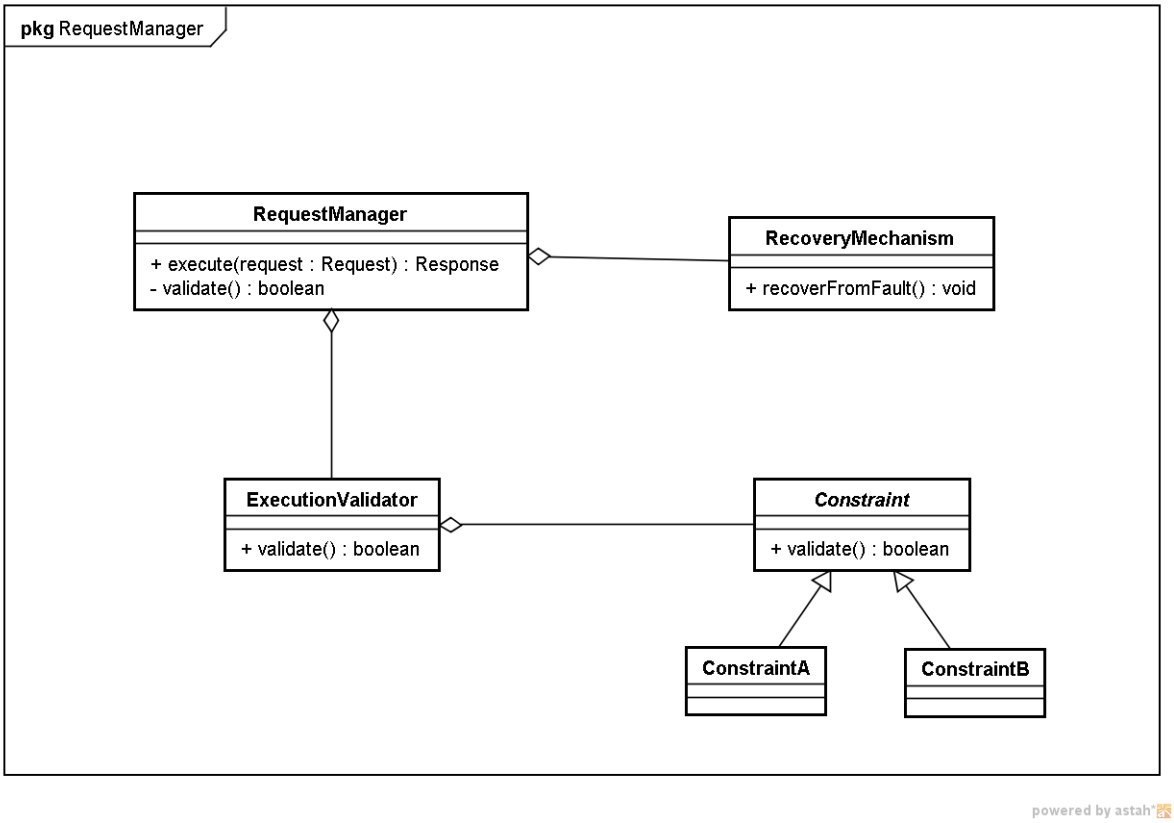


Figura 4.3: Diagrama de classes do Gerenciador de Requisições

bém contêm os parâmetros que serão utilizados para que sejam realizadas estas funcionalidades. O Middleware possui três tipos de requisição:

- **Requisições de Descoberta e Invocação**: este tipo de requisição, também chamado de DaIRequest, envia ao sistema uma lista de descrições acerca do serviço que se deseja invocar e os parâmetros de entrada para a invocação deste serviço. A partir desta requisição, o sistema deve descobrir qual serviço melhor atende as descrições do serviço, se necessário for, fazendo composição, e deve invocá-lo automaticamente para o usuário, retornando os resultados dessa invocação.

- **Requisições de Descoberta**: este tipo de requisição, chamado de DiscoveryRequest, envia ao sistema apenas as descrições do serviço que se deseja invocar. Com posse desta requisição, o sistema busca o serviço que melhor a atende, fazendo composições se necessário, e retorna ao usuário o serviço encontrado para que este possa analisar, se desejar, e invocá-lo futuramente.

- **Requisições de Invocação**: este tipo de requisição, chamado de Invo-

cationRequest, envia ao sistema a URI[1] do serviço que se deseja invocar e os parâmetros de entrada para a invocação deste serviços. Automaticamente, o sistema invoca este serviço e retorna os resultados desta invocação ao usuário.

- **Requisições de execução de Processos**: Estas requisições estão diretamente ligadas à invocação de composições de serviços através de planejamento estático, ver Seção 4.3.2.

A Figura 4.4 mostra o diagrama de classes que representam as requisições e respostas do Middleware.

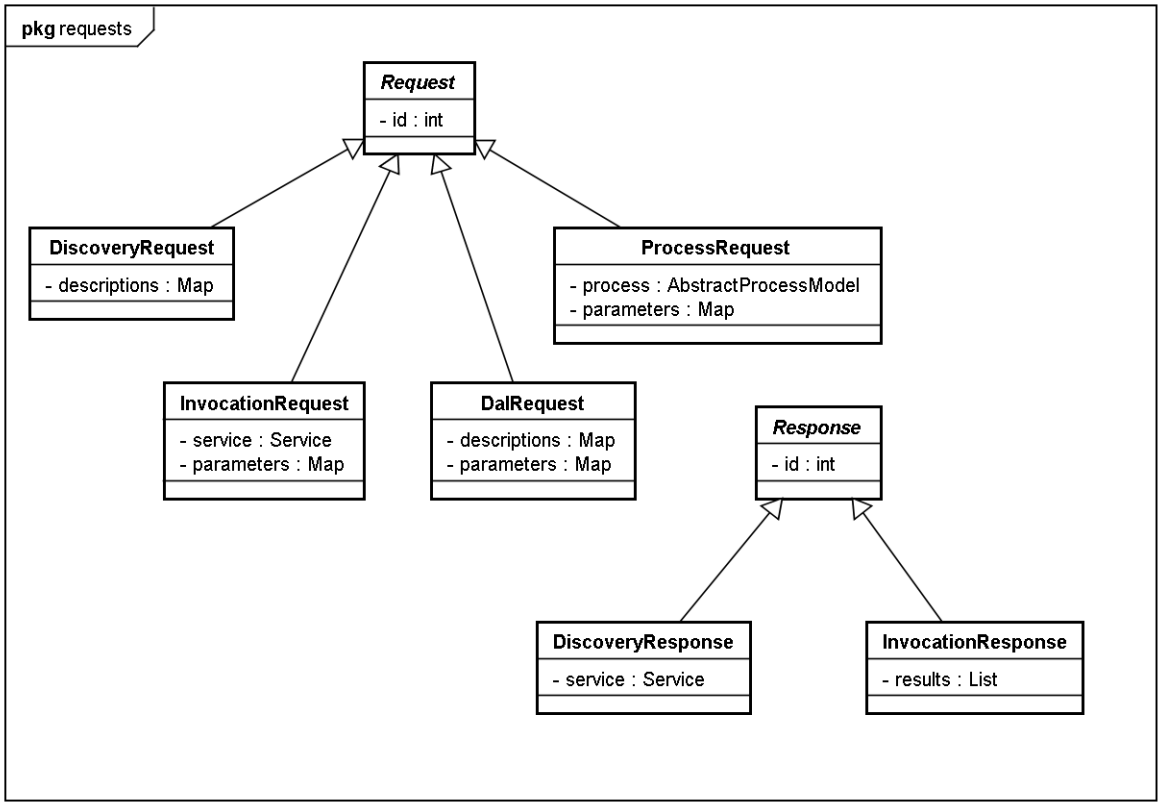


Figura 4.4: Requisições e Resposta do Middleware

4.3 Descoberta e Composição de Serviços

Este módulo é responsável pela descoberta, seleção e composição de serviços no Middleware, ou seja, ele é responsável por encontrar os serviços que atendam as necessidades do sistema, especificadas através das requisições. O diagrama de classes deste módulo é apresentado na Figura 4.5.

[1] *http://www.w3.org/TR/uri-clarification/*

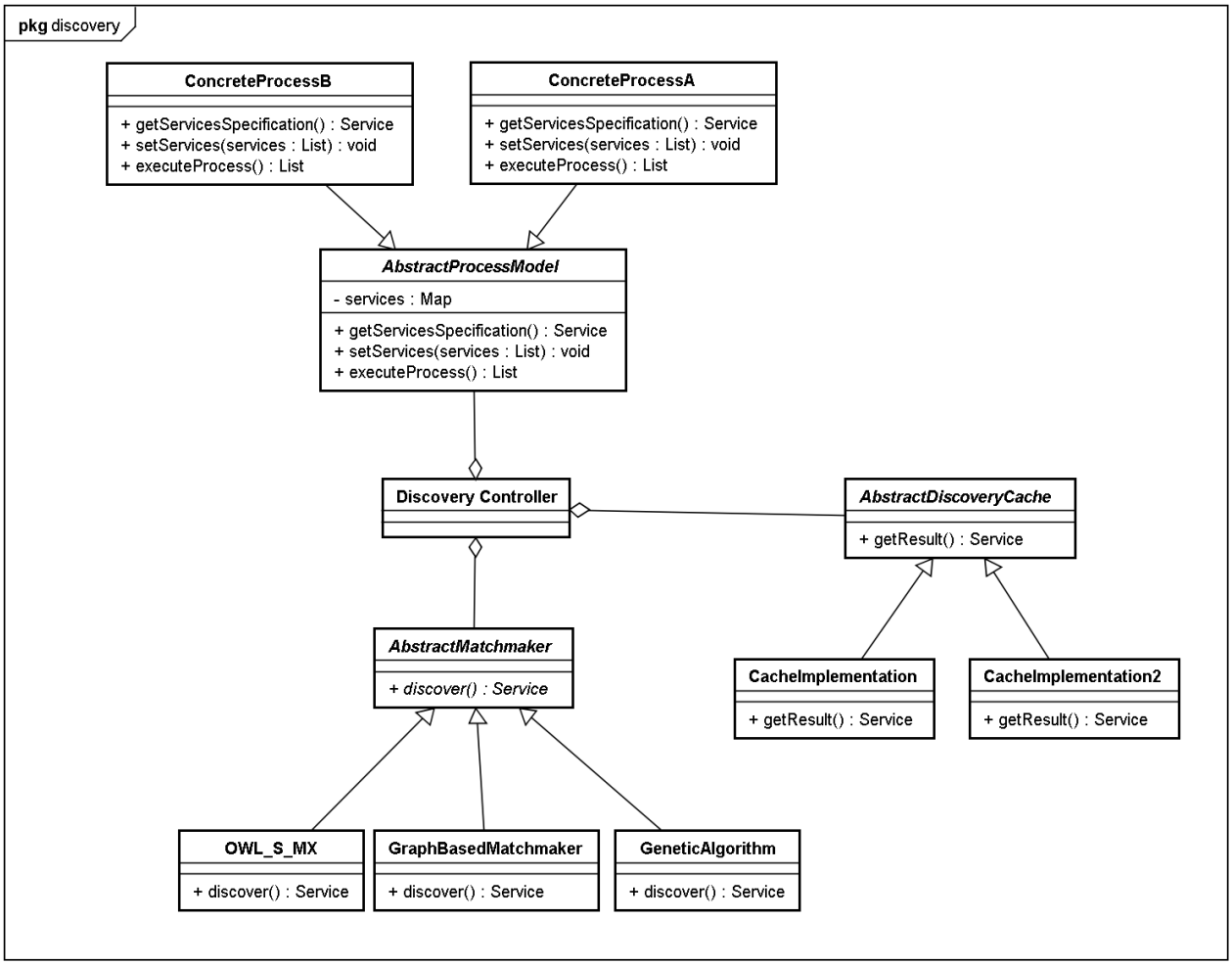


Figura 4.5: Diagrama de classes do módulo de Descoberta e Composição de Serviços

Para isso, o módulo de Descoberta e Composição de serviços possui a classe *DiscoveryController* que é responsável por gerenciar o processo de descoberta de serviços. Este controlador de descoberta possui um conjunto de técnicas para descoberta de serviços (*AbstractMatchmaker*) e as utiliza de acordo com a necessidade no processo de descoberta e seguindo uma ordem de prioridade estabelecida pelo desenvolvedor do sistema.

Para a personalização das técnicas para descoberta e composição de serviços, o desenvolvedor do sistema deve estender a classe *AbstractMatchmaker* com a implementação da nova técnica através do método *discover*. Após isso, ele também deve alterar o arquivo de configuração do Middleware como será mostrado na seção 4.6.

4.3.1 Mecanismo de Cache

Com o objetivo de melhorar o desempenho do processo de descoberta, foi desenvolvido o mecanismo de *Cache* que guarda as informações dos processos de descoberta e cria um índice, para que em futuras requisições não seja necessário utilizar as técnicas de descoberta, agilizando o processo de descoberta. Por exemplo, este mecanismo de Cache pode identificar que uma requisição R sempre tem como resultado o serviço S, assim ele pode guardar essa informação e quando o sistema buscar a requisição R novamente, o mecanismo de Cache recomenda o serviço S, sem que seja necessário analisar os demais serviços.

Além disso, o desenvolvedor pode criar seus próprios critérios para geração dos índices. Para isso, ele deve implementar seus próprios mecanismos de *Cache* estendendo a classe *AbstractDiscoveryCache*.

4.3.2 Planejamento Estático (Manual)

No planejamento estático o desenvolvedor especifica um processo, e os serviços que fazem parte deste processo, com o objetivo de que o Middleware possa descobrir os serviços que compõem este processo e possa invocá-los do modo em que eles estão descritos no processo.

Existem diversas maneiras para especificação de processos, como a ontologia ProcessModel da Linguagem OWL-S, proposta por Martin et al. (2007*a*), e linguagens baseadas em BPEL, ver Andrews et al. (2003). Nesta perspectiva, o Middleware também é flexível em questão ao formalismo para especificação dos processos no Planejamento Estático.

Nesta perspectiva, o desenvolvedor deve estender a classe *AbstractProcessModel* e implementar os métodos *getServicesSpecification*, *setServices* e *invokeProcess*. O primeiro tem como objetivo buscar as informações dos serviços que compõem o processo, o segundo guarda os serviços encontrados no processo de descoberta para que sejam invocados pelo último método.

4.4 Mecanismo de Invocação de Serviços

Como visto anteriormente, serviços podem ser implementados de diferentes maneiras, como SOAP (Booth et al. (2004)), UPnP (*UPNP Service Discovery for Heterogeneous Networks* (2006)), RESTful Web Services (Richardson & Ruby (2007)), entre outros. Assim, o Middleware oferece suporte à criação de diferentes mecanismos para invocação de serviços. Através da classe *Abstrac-*

tInvocationEngine, o desenvolvedor pode criar os mecanismos para invocar os diferentes tipos de serviços.

A classe *AbstractInvocationEngine* e exemplos de implementações de mecanismos para invocação de serviços são mostrados na Figura 4.6.

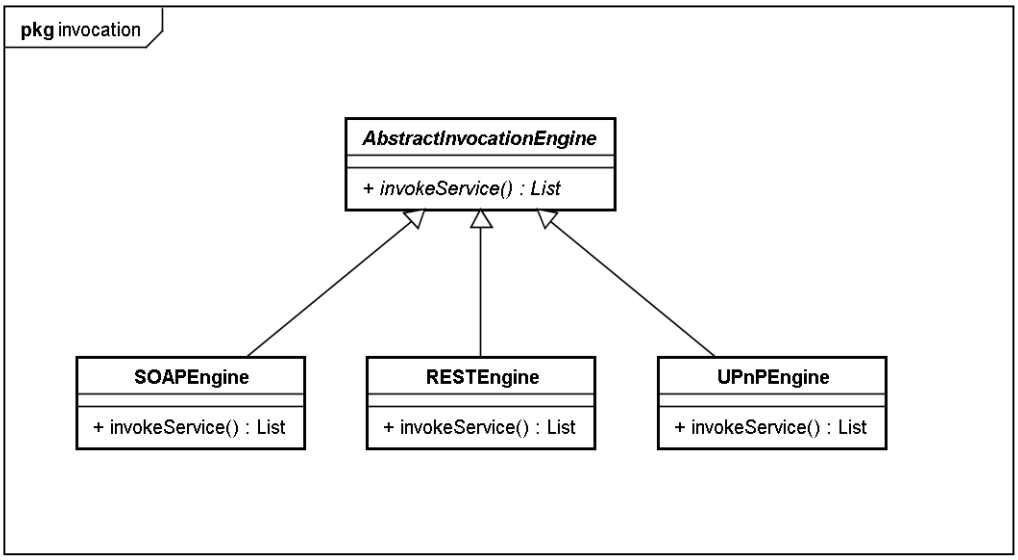


Figura 4.6: Diagrama de classes do Módulo de invocação de serviços

4.5 Repositório de Serviços

Como dito anteriormente, o Repositório de Serviços possui dois repositórios para armazenar os serviços que poderão ser utilizados pelo ambiente, sendo um deles o repositório padrão (ServicesRepository **SR**) e o outro um repositório para serviços que apresentaram algum problema, uma espécie de lista negra para serviços (Services under Observation (**SUO**)).

Existem diversas formas para armazenar estes serviços, como Banco de dados relacional, arquivo de dados, XML (Klein (2001)), propostas de extensão do Registro UDDI (Paolucci et al. (2002*a*)), entre outras abordagens. Apesar de possuir o mesmo objetivo estas abordagens diferem em relação à complexidade, escalabilidade e desempenho. Neste trabalho não foi realizada nenhuma pesquisa em relação a qual abordagem para armazenamento de serviços utilizar, mas criou mecanismos para possibilitar a adaptação destes repositórios ao Middleware.

Nesta perspectiva, a classe *RepositoryManager*, ver Figura 4.7, é responsável por buscar qual o repositório que está sendo utilizado no arquivo de configuração do sistema, ver seção 4.6, tanto para o repositório **SR** quanto para o **SUO**.

Como ambos os repositórios (**SR** e **SUO**) têm o mesmo objetivo, armazenar serviços e suas descrições semânticas, eles podem utilizar a mesma implementação para repositório, mudando apenas a localização destes repositórios. Da mesma maneira, as implementações podem ser diferentes, por exemplo, o **SR** pode utilizar um repositório baseado em UDDI e o **SUO** ser baseado em XML, de acordo com as preferências do usuário. O Mecanismo de repositório de serviços é mostrado na figura 4.7.

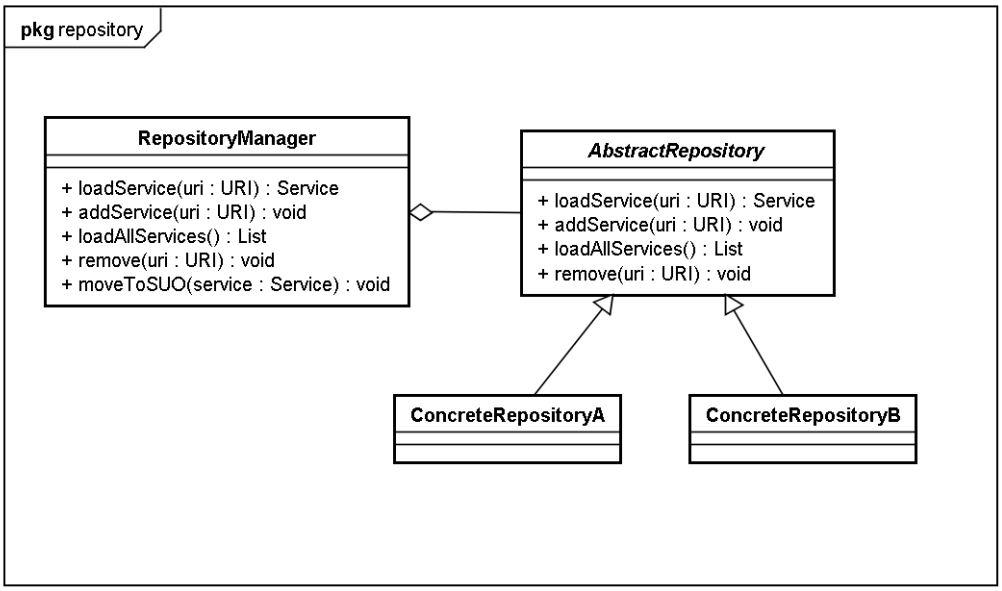


Figura 4.7: Diagrama de classes do Módulo de Repositório de serviços

4.6 Módulo de Configuração

O módulo de Configuração é responsável por verificar as preferências do desenvolvedor da aplicação, definidas através do arquivo de configuração do sistema, e fazer com que o sistema siga estas preferências.

O arquivo de configuração é um arquivo XML que contém as especificação dos pontos variáveis do sistema. Estes pontos foram discutidos no decorrer desta seção.

Baseando-se nestes pontos de variabilidade foi criado o Modelo de Configuração do Middleware que é mostrado na Figura 4.8. diagrama de classes que representa o Modelo de Configuração do Middleware. Os pontos variáveis e as informações necessárias para a correta configuração destes pontos são discutidos a seguir:

- **Discovery**: Representa a configuração das técnicas para Descoberta e

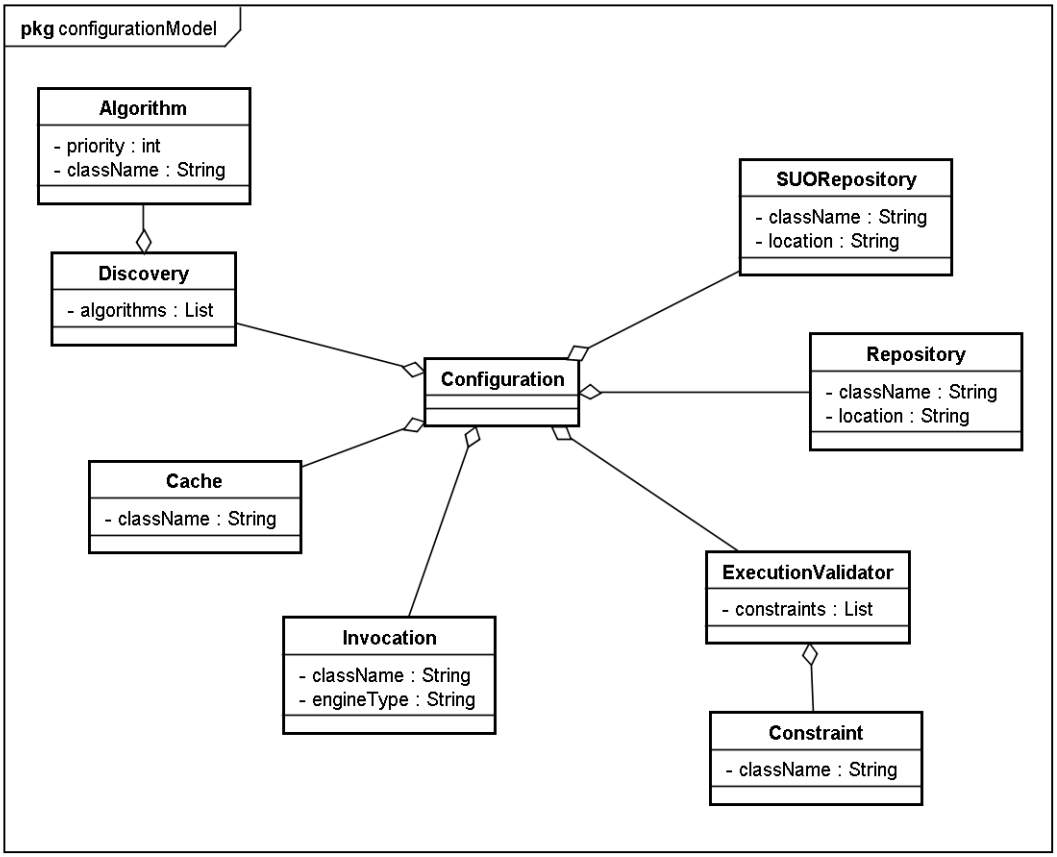


Figura 4.8: Classes que representam o arquivo de configuração do sistema

Composição é composto por um conjunto de algoritmos de descoberta (*Algorithm*), que possui as seguintes informações:

  - **priority**: Prioridade em que a técnica será utilizada.
  - **className**: Nome da classe Java que implementa esta técnica.
  - **parameter**: Parâmetros utilizados no processo de descoberta, por exemplo: Inputs, Outputs, Classificação, entre outros.

- **Cache**: possui apenas o nome da classe (*className*) que implementa a técnica para cache que será utilizada pelo sistema.

- **Invocation**: Referente aos mecanismos de invocação. Possuem as seguintes características:

  - **engineType**: Tipo de serviço que o mecanismo de invocação pode executar.
  - **className**: Nome da classe que implementa o mecanismo.

- **Repository** e **SUORepository**: Configuração dos repositórios de serviços, possuem as seguintes características:

  - **location**: localização do repositório de serviços, esta localização pode ser via Web (servidor remoto) ou local (arquivo).
  - **className**: Nome da classe que implementa o repositório.

- **ExecutionValidator**: É responsável pelas Restrições (*Constraints*) que serão utilizadas no processo de validação da invocação dos serviços, estas restrições possuem o nome das classes que as implementam (*className*).

Baseando-se nestas características, o Código 4.1 mostra um exemplo de arquivo de configuração.

Código 4.1: Exemplo de arquivo de configuração

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Configuration>
3
4   <!-- Repositorio SR -->
5   <Repository-SR>
6     <Class>br.ufal.ic.grow.grinv.repository.GrinvRep</Class>
7     <Location>#location_example</Location>
8   </Repository-SR>
9
10  <!-- Repositorio SUO -->
11  <Repository-SUO>
12    <Class>br.ufal.ic.grow.grinv.repository.GrinvRep</Class>
13    <Location>#location_example</Location>
14  </Repository-SUO>
15
16  <!-- Algoritmos de Descoberta -->
17  <Discovery>
18    <Algorithm>
19        <priority>1</priority>
20        <ClassName>grow.grinv.discovery.GBMatchmaker</ClassName>
21        <Parameter>inputs</Parameter>
22        <Parameter>outputs</Parameter>
23    </Algorithm>
24    <Algorithm>
25        <priority>0</priority>
26        <ClassName>grow.grinv.discovery.GrinvMatch</ClassName>
27        <Parameter>inputs</Parameter>
28        <Parameter>outputs</Parameter>
29    </Algorithm>
30  </Discovery>
```

```
31
32  <!-- Mecanismo de Cache -->
33  <Cache>
34    <Class>br.ufal.ic.grow.grinv.cache.CacheExample</Class>
35  </Cache>
36
37  <!-- Mecanismo de Invocacao -->
38  <Invocation>
39    <EngineType>OWL-S</EngineType>
40    <ClassName>grow.grinv.invocation.OwlsEngine</ClassName>
41  </Invocation>
42
43  <!-- Configuracao de Restricoes -->
44  <ExecutionValidator>
45    <Constraint>
46      <Class>constraints.Constraint_A</Class>
47    </Constraint>
48    <Constraint>
49      <Class>constraints.Constraint_B</Class>
50    </Constraint>
51  </ExecutionValidator>
52</Configuration>
```

Com o arquivo de configuração propriamente criado, o módulo de Configuração é responsável por criar instâncias de acordo com as requisições do sistema. A criação destas instâncias é feita através de reflexão computacional, ver Dowling et al. (2000), provida pela linguagem Java, isso permite que as instâncias sejam criadas em tempo de execução baseadas no nome da classe que está contida no arquivo de configuração.

Além disso, módulo de configuração possui construtores específicos para cada ponto de variabilidade, utilizando o padrão de projeto *Builder*, ver Gamma et al. (1994), com o objetivo de isolar a complexidade da criação de cada um destes pontos de variabilidade. Estas classes podem ser observadas na Figura 4.9.

Caso não seja possível instanciar um objeto com a classe informada pelo usuário, cada construtor define uma classe padrão que faz parte da implementação padrão do Middleware, estas classes são mostradas na Tabela 4.1.

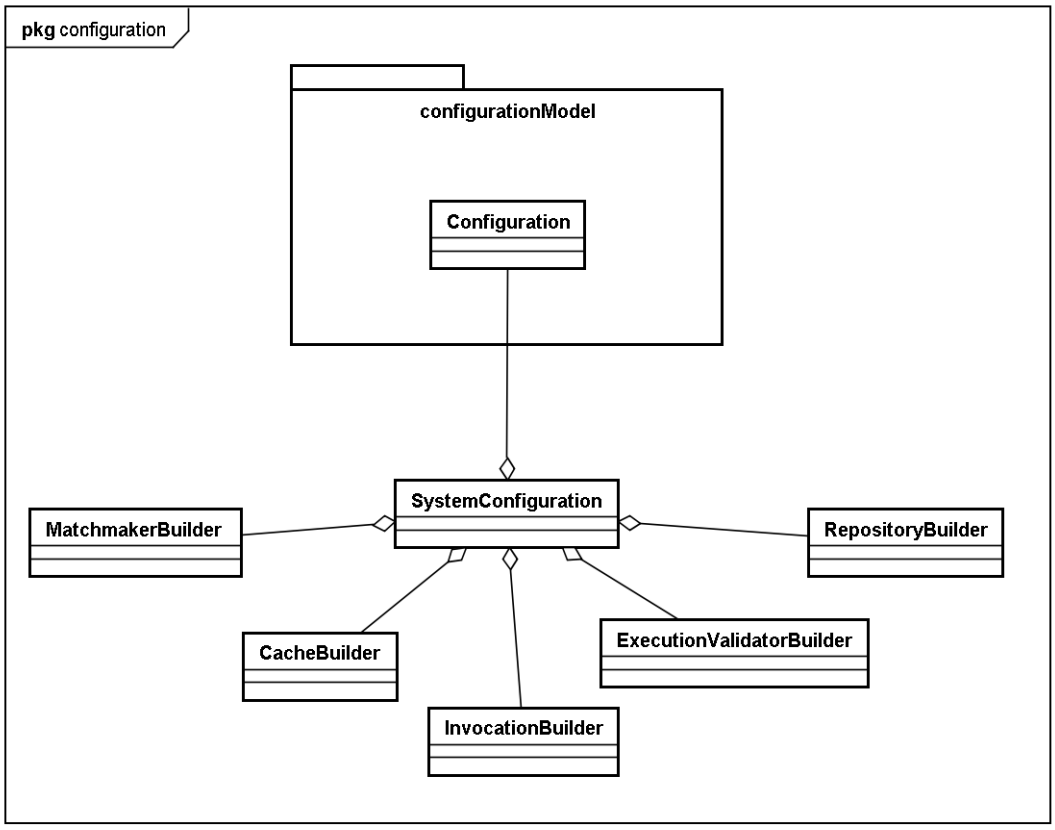


Figura 4.9: Classes que gerenciam a escolha das funcionalidades do sistema

Tabela 4.1: *Classes padrões para funcionamento*

Propriedade	**Classe**
Algoritmo de Descoberta	br.ufal.ic.grow.grinv.discovery.GBMatchmaker
Repositório SR	br.ufal.ic.grow.grinv.repository.GrinvRepository localização: ’./SR’
Repositório SUO	br.ufal.ic.grow.grinv.repository.GrinvRepository localização: ’./SUO’
Invocação(OWL-S)	br.ufal.ic.grow.grinv.invocation. OwlsInvocationEngine
Invocação outros tipos	X
Cache	X
Restrições	X

Capítulo 5

Estudo de Caso

Este capítulo apresenta um estudo de caso que tem como objetivo ilustrar o funcionamento do Middleware proposto neste trabalho. Neste cenário será mostrado o funcionamento do mecanismo para verificar a consistência dos resultados da invocação dos serviços, mostrando como especificar as restrições que farão parte do processo de validação.

Neste estudo de caso, será mostrado um cenário onde a aplicação precisa de um serviço para prover uma explicação para um determinado problema. Neste contexto serão discutidas as restrições que podem ser modeladas para avaliação destes serviços.

5.1 Descrição do Cenário

Neste cenário, considera-se uma situação na qual um estudante está interagindo com um Ambiente Interativo de Aprendizagem que utiliza Serviços Semânticos para realizar suas atividades, isto é, o sistema usa o Middleware proposto neste trabalho para encontrar um serviço que provê as funcionalidades requisitadas pelo estudante.

Além disso, assume-se que o estudante está realizando uma atividade de resolução de problemas. Entretanto, ele necessita de uma dica para resolver um determinado problema, isto é, o estudante necessita de um serviço (simples ou composto) que possa oferecer esta dica baseada no problema que ele está resolvendo. Então, o Middleware é responsável por encontrar um serviço que se encaixe na resolução da requisição do usuário (Parâmetro de Entrada: Problem, Parâmetro de Saída: Explanation).

5.2 Criando Restrições

No que diz respeito às Restrições para este cenário, podemos citar a seguinte situação:

*Um estudante que utiliza a **Língua Inglesa** está realizando a atividade de resolução de problemas. Em um momento, este estudante sente dificuldades na resolução de uma questão e necessita de uma explicação. A aplicação retorna um recurso educacional contendo uma explicação para este problema, entretanto esta explicação está escrita na **Língua Francesa**, desta forma o estudante não poderá entender o conteúdo desta explicação e, conseqüentemente, não resolverá o problema.*

Neste cenário, podemos ver que a linguagem em que a explicação está escrita influencia diretamente o aprendizado do estudante. Isto é, não faz sentido enviar ao estudante uma explicação que ele não poderá interpretar. Entretanto, de alguma forma, o sistema utilizou um serviço que forneceu uma explicação em uma outra linguagem, isto significa que o serviço gerou um resultado incorreto para a aplicação.

Neste contexto, para assegurar a corretude dos resultados da invocação dos serviços é necessário criar uma Restrição para verificar a linguagem do recurso recomendado pelo serviço. Esta verificação pode ser feita através de ferramentas como *Google Language API* [1] e *What language is This* website [2].

O Código 5.1 apresenta a implementação de uma Classe em Java que representa a Restrição de linguagem.

Código 5.1: Exemplo de código para Restrição de Linguagem

```
1
2 public class LanguageConstraint extends Constraint{
3    // tool that recognizes the language
4    LanguageTool tool;
5
6    public double validate(Service s,
7         Map descriptions, List Results){
8
9       // Verify if the service requires this
10      // constraint validation
11      if(descriptions.contains("language")){
12        //retrieving the required language(s)
13        String lang = descriptions.get("language");
14
15        //checking all results
```

[1] http://code.google.com/intl/pt-BR/apis/language/
[2] http://whatlanguageisthis.com/

```
16          for(Resource resource:Results.getOutputs()){
17             // checking if the 'resource' is written
18             // in language 'lang'
19             double value= tool.check(resource, lang);
20
21             // if the 'resource' has less than 70% of
22             // its content in the required language
23             // the result did not accomplish the
24             // constraint
25             if(value < 0.7){
26                  return 0.0
27             }
28          }
29       }
30
31       return 1.0;
32    }
33 }
```

5.3 Configuração

Inicialmente, será apresentado o processo de configuração do Middleware. Os serviços que estão disponíveis no repositório estão apresentados na Tabela 5.1 A Figura 5.1 mostra um exemplo de um serviço (Show Resource) descrito na linguagem OWL-S. Estes serviços foram descritos através da ontologia que descreve os Recursos educacionais, a representação gráfica de parte desta ontologia é ilustrada na Figura 5.2. Seguem algumas informações sobre estes serviços:

Tabela 5.1: *Repositórios de Serviços*

Serviços	**Parâmetros de Entrada**	**Parâmetros de Saída**
Give Concept	Problem	Concept
Get Activities Report	Student	Report
Give Explanation	Problem	Explanation
Show Resource	Student	Resource
Explain Concept	Concept	Explanation
Evaluate Problem	Problem	Evaluation

- **Give Concept**: Este serviço ajuda o usuário na atividade de resolução de problemas. Ele relaciona um certo problema ao Conceito necessário para responder este problema.

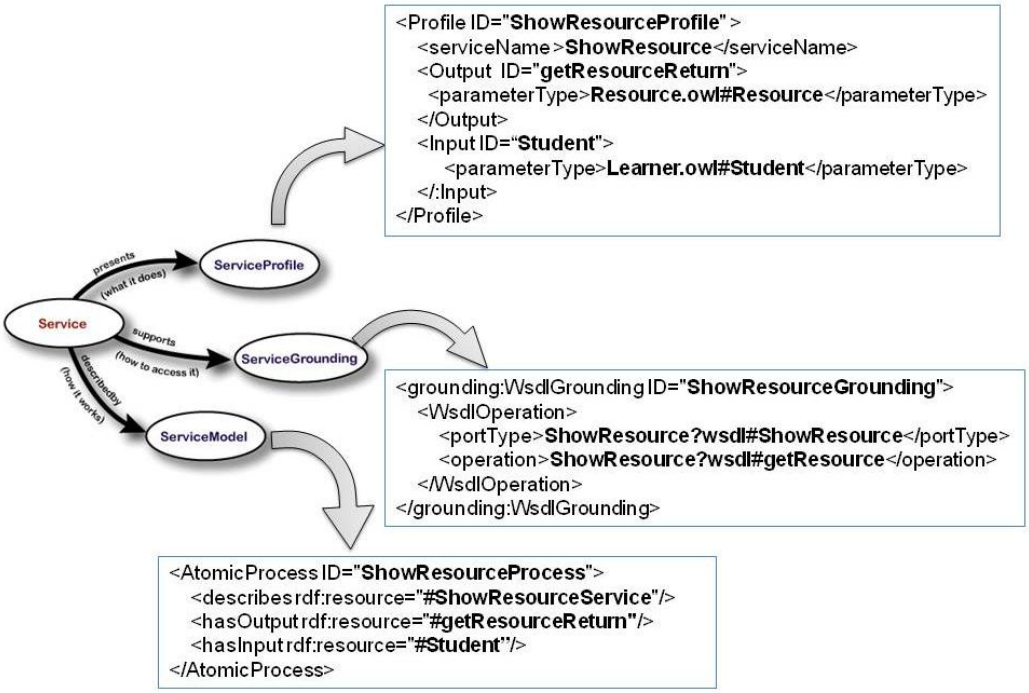


Figura 5.1: Serviço Show Resource descrito com OWL-S

- **Get Activities**: Este serviço retorna um relatório de atividades realizadas pelo estudante da plataforma.
- **Give Explanation**: Este serviço ajuda o usuário na atividade de resolução de problemas. Ele envia uma explicação baseado no problema que o estudante está respondendo. Entretanto, neste cenário, este serviço foi desenvolvido por um outro grupo com o objetivo de ser executado em um sistema que possui estudantes que usam a Língua Francesa, então este serviço recomenda explicações na **Língua Francesa**.
- **Show Resource**: Este serviço recebe como entrada um estudante, e baseado no curso do estudante, este serviço retorna ao usuário um recurso que será mostrado ao estudante.
- **Explain Concept**: Este serviço visa prover uma Explicação para auxiliar o estudante no entendimento de um determinado Conceito.
- **Evaluate Problem**: Este serviço avalia a resolução de um problema feito por um estudante.

Em relação ao processo de descoberta, duas abordagens serão utilizadas para descoberta e composição. A primeira abordagem é um simples algoritmo de busca Não-Informada e não oferece suporte a composição de serviços, em compensação ele possui bom desempenho. A segunda abordagem foi proposta

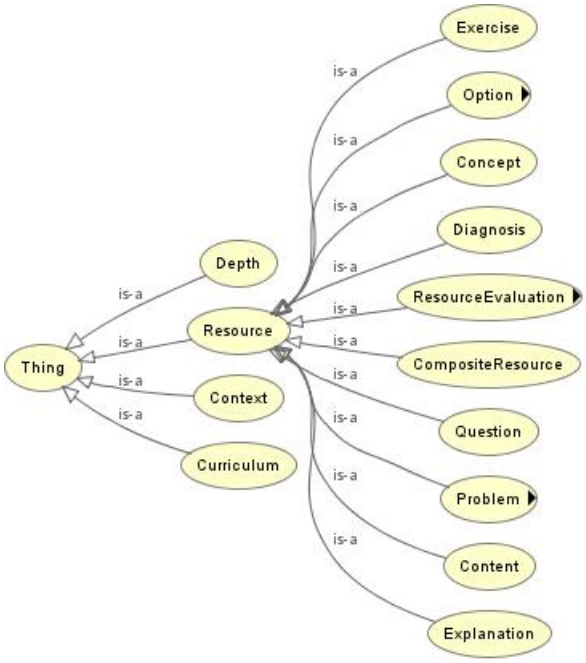


Figura 5.2: Ontologia que descreve recursos educacionais

por Calado et al. (2009), e usado em um ambiente de E-Learning em Barros et al. (2009), e utiliza um algoritmo baseado em Grafos para realizar a descoberta e composição de serviços, mostrando melhores resultados que a primeira técnica mas com menor desempenho. Mais detalhes sobre esta abordagem são mostrados a seguir:

O segundo algoritmo utiliza um Grafo dirigido, onde os vértices representam os serviços e os pesos das arestas representam as relações entre estes serviços. O peso de uma aresta $\overrightarrow{AB}$ representa a similaridade entre os parâmetros de saída do serviço A e os parâmetros de entrada do serviços B, esta similaridade é calculada através da Métrica de similaridade do Cosseno. A Métrica do Coseno trata as entidades como um vetor n-dimensional, determinando o angulo formado por estes vetores. Quanto maior a proximidade destes vetores, maior é a similaridade entre as entidades (mais detalhes acerca das métricas de similaridades podem ser encontrados em Klusch et al. (2006)).

Assim, o grafo é construído de tal forma que as relações entre os serviços representam a possibilidade de utilizar os parâmetros de um serviço como parâmetros de entrada para o próximo serviço, para possibilitar a execução da composição destes serviços em sequência.

Finalmente, o Código 5.2 mostra o arquivo de configuração utilizado neste cenário.

Código 5.2: Configuração do Middleware

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```
<Configuration>
  <Repository-SR>
    <Class>repository.GrinvRepository</Class>
    <Location>./SR</Location>
  </Repository-SR>
  <Repository-SUO>
    <Class>repository.GrinvRepository</Class>
    <Location>./SUO</Location>
  </Repository-SUO>
  <Discovery>
    <Algorithm>
       <priority>0</priority>
       <ClassName>grinv.UninformedSearch</ClassName>
       <Parameter>inputs</Parameter>
       <Parameter>outputs</Parameter>
    </Algorithm>
    <Algorithm>
       <priority>1</priority>
       <ClassName>grinv.GBMatchmaker</ClassName>
       <Parameter>inputs</Parameter>
       <Parameter>outputs</Parameter>
    </Algorithm>
  </Discovery>
  <Invocation>
    <EngineType>OWL-S</EngineType>
    <ClassName>invocation.OwlsEngine</ClassName>
  </Invocation>
  <ExecutionValidator>
    <Constraint>
       <Class>constraints.LanguageConstraint</Class>
    </Constraint>
  </ExecutionValidator>

</Configuration>
```

5.4 Atendimento de Requisições

Com o Middleware propriamente configurado, a requisição do usuário deve ser especificada. Esta requisição é enviada para o *Request Manager* para iniciar a execução do Middleware. Neste cenário, os parâmetros de entrada e saída devem ser especificados para que seja realizada a execução, a Tabela 5.2 apresenta esta requisição. Além disso, a aplicação deve descrever as Restrições que serão utilizadas para a requisição, neste cenário existe o parâmetro

Linguagem.

Além disso, o problema que o usuário está resolvendo é adicionado a esta requisição para ser utilizado na invocação do serviço. Outros parâmetros podem ser adicionados para as requisições, como Pré-condições e Efeitos, mas os algoritmos apresentados na seção anterior utilizam apenas parâmetros de entrada e saída.

Tabela 5.2: *Requisição da Aplicação*

Descrições	
Input	Problem
Output	Explanation
Linguagem	English
Parâmetros	
Problem	Problem_120a

Após a especificação da requisição, o processo de descoberta é iniciado utilizado o Algoritmo de busca Não-informada citado anteriormente, já que ele possui a menor prioridade no arquivo de configuração. Sua execução é mostrada no Código 5.3.

Código 5.3: Log de execução do primeiro algoritmo

```
1  Discovery Request:
2  > Descriptions:
3  >> Inputs: {Problem}
4  >> Outputs: {Explanation}
5  - - - - - -
6  Starting Uninformed Search:
7  ->Service: Give Concept
8  ---> Inputs: {Problem} - 100%
9  ---> Outputs: {Concept} - 50%
10 ->Service: Get Activities
11 ---> Inputs: {Student} - 0%
12 ---> Outputs: {Report} - 0%
13 ->Service: Give Explanation
14 ---> Inputs: {Problem} - 100%
15 ---> Outputs: {Explanation} - 100%
16 ->Service: Show Resource
17 ---> Inputs: {Student} - 0%
18 ---> Outputs: {Resource} - 71%
19 ->Service: Explain Concept
20 ---> Inputs: {Concept} - 50%
21 ---> Outputs: {Explanation} - 100%
22 ->Service: Evaluate Problem
```

```
23 —> Inputs: {Problem} – 100%
24 —> Outputs: {Evaluation} – 50%
25
26 Calculating Similarity:
27 > Service Give Concept: 75%
28 > Service Get Activities: 0%
29 > Service Give Explanation: 100%
30 > Service Show Resource: 35,5%
31 > Service Explain Concept: 75%
32 > Service Evaluate Problem: 75%
33 Matching Result: 100% (Give Explanation)
```

Este algoritmo encontrou o serviço *Give Explanation* como resultado de sua execução, e o envia para o Módulo de Invocação para ser executado. De fato, o serviço encontrado se encaixa com as descrições informadas pelo usuário, entretanto, como informado anteriormente, este serviço provê apenas Explicações escritas em Francês. Então, o Módulo de Invocação invoca este serviço e recebe como resultado uma Explicação para o problema (*Problem_120a*).

Apesar do serviço foi desenvolvido para Linguagem Francesa e o Problema foi escrito em Inglês, o serviço pode interpretar este problema através das descrições semânticas apresentadas na ontologia e identificar os conceitos que são relacionados ao problema, e assim conceber uma Explicação.

Como a etapa de invocação do serviço obteve sucesso, a etapa de *Validação dos Resultados* é iniciada, como mostrado no diagrama de atividades (Figura 3.2). Nesta etapa, o *ExecutionValidator* verifica se os resultados estão válidos de acordo com as Restrições especificadas. Neste cenário, a Restrição de Linguagem não é satisfeita pelo fato da aplicação ter indicado que a Explicação precisava estar escrita em Inglês e o serviço retornou uma explicação em Francês. Como as restrições não foram atendidas, a execução foi considerada inválida.

Pelo fato do serviço não produzir um resultado correto, este serviço foi removido do repositório de Serviços (SR) e inserido no repositório de serviço em observação (SUO). Feito isso, um novo processo de descoberta é iniciado sem o serviço defeituoso.

Neste novo processo de descoberta, o uso do algoritmo de Busca Informada retornou um serviço com 75% de similaridade (*Give Concept*). Como este não é um resultado perfeito, o algoritmo com prioridade 1 (*Graphs Matchmaker*) é utilizado com o objetivo de encontrar um melhor resultado que o encontrado pela abordagem anterior.

Como podemos ver no relatório de invocação, Código 5.4, a execução do algoritmo encontrou o serviço *Give Concept* como resultado de *matching di-*

reto, este é o serviço simples que melhor se encaixa da requisição do usuário, de fato, seu parâmetro de saída é igual ao requerido pelo usuário, entretanto seus parâmetros de entrada são diferentes. Como *matching indireto*, o algoritmo indicou os serviços *Give Concept* e *Explain Concept* como composição em seqüência, de fato o primeiro serviço tem os parâmetros de entrada iguais aos indicados pela aplicação (Problem) e o segundo serviço possui parâmetros de saída iguais aos definidos pela aplicação (Explicação). Além disso, o primeiro parâmetro de saída do primeiro serviço é igual à entrada do segundo, fazendo assim uma boa opção de composição. Comparando duas soluções, a composição foi escolhida pelo módulo de descoberta com 100% de similaridade, mais detalhes sobre o funcionamento do algoritmo podem ser encontrados em Calado et al. (2009); Barros et al. (2009), a Figura 5.3 ilustra a composição escolhida.

Código 5.4: Relatório de descoberta

```
Discovery Request:
> Descriptions:
>> Inputs: {Problem}
>> Outputs: {Explanation}
- - - - - -
Direct Matching Search:
->Service: Give Concept - 75%
->Service: Get Activities - 0%
->Service: Show Resource - 35,5%
->Service: Explain Concept - 75%
->Service: Evaluate Problem - 75%

- - - - - -
Indirect Matching Search:
>Path found, Services:
>> Give Concept -> Explain Concept - 100%
- - - - - -
Direct Matching Result: 75%
Indirect Matching Result: 100%

Final Result: Indirect Matching
Composition: {Give Concept, Explain Concept}
```

Com o serviço encontrado, este é passado para o módulo de invocação que utiliza a OWL-S API para mapear os parâmetros enviados pelo usuário com os parâmetros do serviço encontrado. Em seguida, o serviço é invocado pela OWL-S API que provê os mecanismos para executar serviços compostos. Como resultado da invocação do serviço encontrado, foi obtida a Explicação

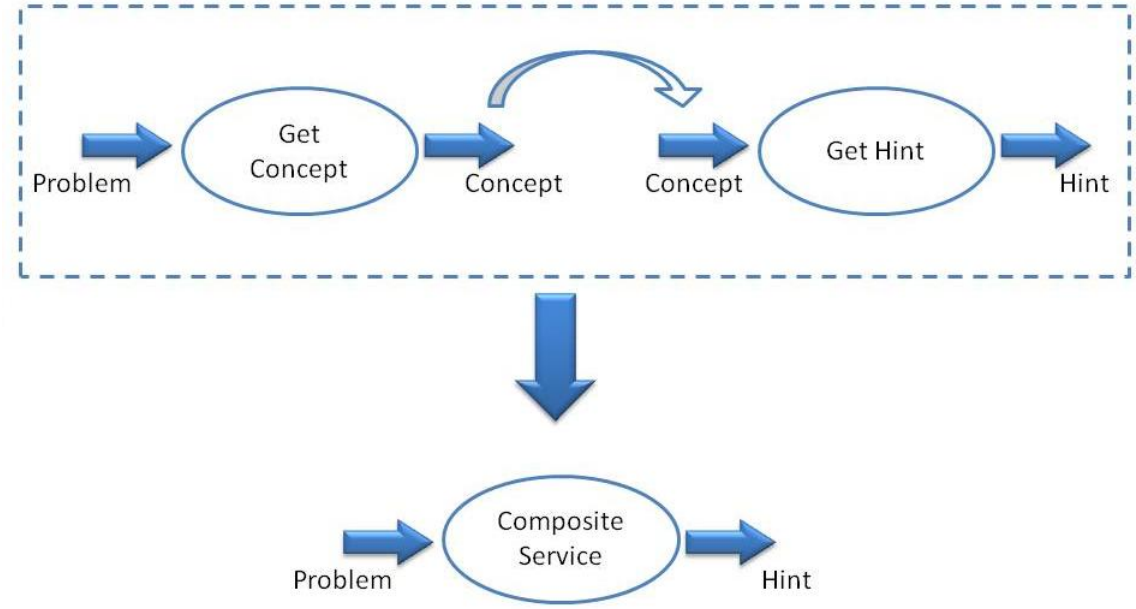


Figura 5.3: Composição de Serviços

requisitada pelo estudante.

Como na primeira tentativa, os resultados desta invocação também são avaliados. No caso da Explicação atender todas as restrições (ser escrita em Inglês), esta explicação será enviada ao estudante. Caso contrário o processo se repetirá e o sistema irá buscar por um novo serviço que possa prover a explicação. Neste caso, se nenhum novo serviço puder ser encontrado, o Middleware enviará uma mensagem informando que nenhum resultado pôde ser obtido.

Finalmente, este cenário buscou ilustrar a execução do Middleware proposto neste trabalho, apresentando descoberta, composição e invocação automática de serviço de maneira integrada e transparente ao usuário da aplicação (estudante). Além disso, os mecanismos para prover uso confiável de Serviços Semânticos foram mostrados em um cenário que o seu uso se mostrou necessário.

Capítulo 6

Trabalhos Relacionados

Este Capítulo tem como objetivo fornecer uma comparação entre algumas abordagens similares ao Middleware proposto neste trabalho. Recentemente, a comunidade tem proposto algumas ferramentas para a composição, descoberta e invocação de Semantic Web Services. Esta seção irá apresentar algumas dessas obras, comparando-os com o Middleware proposto neste trabalho.

Para esta comparação, buscou-se analisar ferramentas que, de alguma forma, realizam os processos de descoberta, composição e invocação de Serviços Semânticos. Nesta perspectiva, foram selecionadas ferramentas que abrangem os mais diferentes modelos conceituais para descrição semântica de serviços, como o WSMX (proposto por Galway & Innsbruck (2008)), baseado em WSMO, o METEOR-S (proposto por Aggarwal et al. (2004)), baseado em WSDL-S/SAWSDL, e alguns baseados em OWL-S como o SAREK (de Claro & de Araújo Macêdo (2008)), o OSIRIS NEXT (de Moller & Schuldt (2010)) e o Transplan (proposto por Silva & Claro (2009)). Além do sistema de composição ADDO (de Weise et al. (2008)) que utiliza uma solução própria com WSDL+MECE (Bleul et al. (2009)).

O trabalho de Aggarwal et al. (2004) apresenta a ferramenta METEOR-S que fornece mecanismos para a descoberta e invocação dos serviços semânticos e apóia a descoberta de serviços através do planejamento manual baseando-se em uma extensão da linguagem BPEL4WS (Andrews et al. (2003)), em sua implementação padrão não há suporte para planejamento automático. Além disso, este trabalho não oferece nenhum mecanismo de tolerância a falhas ou suporte a adição de novas abordagens para a descoberta.

O arcabouço proposto por Majithia et al. (2004) fornece composição de descoberta e de serviços através de uma abordagem encadeamento para frente e tem um mecanismo de tolerância a falhas, mas é restrito a tolerância a falhas no processo de descoberta (não disponibilidade de um serviço). Este trabalho

também não permite a adição de mecanismos para uma nova descoberta.

No trabalho de Weise et al. (2008) é proposta uma arquitetura que suporta três diferentes abordagens para a composição de serviços, e os resultados obtidos apresentam a eficácia desta abordagem. Ou seja, este trabalho demonstra que o uso de diferentes abordagens para descoberta de serviços em um mesmo sistema aumenta a eficácia do processo de descoberta, dentro das ferramentas analisadas esta foi a única que também oferece esta funcionalidade. No entanto, o sistema ADDO não permite a adição de novas técnicas para descoberta de serviços, e não prevê qualquer mecanismo para invocar serviços ou tolerância a falhas.

Em Galway & Innsbruck (2008) é proposto o WSMX (Web Service Modelling eXecution environment) que é a implementação de referência do WSMO. É um ambiente de execução para a integração de aplicativos de negócios, onde os serviços web descritos semanticamente são integrados para aplicações de negócio. WSMX incide sobre as empresas apoio à criação de processos baseados em serviços e também fornece suporte para a descoberta e composição, Embora essa abordagem também permitir que o desenvolvedor personalizar o ambiente de execução (por exemplo, algoritmos de descoberta e composição), este trabalho só suporta um algoritmo de descoberta em cada execução, ao contrário do Middleware proposto nesta dissertação que suporta vários algoritmos. O documento de referência (Galway & Innsbruck (2008)) não menciona o uso de qualquer técnica para fornecer tolerância a falhas, como previsto no Grinv. Suporte para planejamento (orquestração e coreografia) é fornecida através de ferramentas gráficas que ajudam o desenvolvedor. Grinv não oferece essas ferramentas, mas elas podem no futuro ser incorporadas ao projeto.

A ferramenta OSIRIS NEXT (Moller & Schuldt (2010)) apresentou o CFI (Control Flow Intervention), uma abordagem de recuperação automática e flexível que dinamicamente substitui serviços que falharam por serviços com semântica equivalente. Entretanto, esta ferramenta lida apenas com tolerância a falhas na invocação de serviços, sem prover mecanismos para verificar se os resultados das invocações dos serviços estão corretos.

No framework SAREK, proposte em Claro & de Araújo Macêdo (2008), um mecanismo de tolerância a falhas é proposto, ele utiliza um algoritmo de planejamento automático para composição de serviços baseando-se em precondições e efeitos. Este mecanismo constrói Esquemas de Replicação para prover Confiabilidade, isto é, ele encontra um conjunto de possíveis soluções, se existir alguma falha na execução de um serviço, uma solução alternativa é utilizada. Entretanto, esta abordagem não prevê a mudança de técnicas para a descoberta de serviços, além de não oferecer mecanismos para verificar se

os resultados das invocações dos serviços estão corretos.

No contexto de variabilidade no processo de descoberta de serviços, o Transplan de Silva & Claro (2009) também se destaca, possibilitando alterações nas técnicas de descoberta e composição de serviços. Entretanto esta ferramenta foca apenas na etapa planejamento automático no ciclo do uso serviços semânticos.

Dentre os trabalhos analisados, apenas o Grinv apresenta a possibilidade de mudança de sua configuração em tempo de execução através da alteração no arquivo de configuração do Middleware.

Por fim, vale se ressaltar que não se pôde realizar uma análise mais profunda das ferramentas pelo fato delas utilizarem diferentes modelos conceituais e, até onde se sabe, não existir uma coleção de testes que possa ser utilizada por todas ou grande parte destas ferramentas para os processos de descoberta, composição e invocação de serviços. Além disso algumas ferramentas não possuem a implementação distribuída como o ADDO (Weise et al. (2008)).

A Tabela 6.1 mostra uma comparação entre as abordagens analisadas neste capítulo e suas funcionalidades.

Tabela 6.1: *Comparação entre trabalhos relacionados*

Funcionalidade	Grinv	Sarek	Transplan	Osiris Next	WSMX	METEOR-S	ADDO
Linguagem	OWL-S 1.2	OWL-S 1.1	OWL-S 1.1	OWL-S 1.2	WSML	WSDL-S /SAWSDL	WSDL +MECE
Composição Automática	✓	✓	✓	✓			✓
Composição Manual	✓				✓	✓	
Invocação	✓	✓		✓	✓	✓	
Descoberta Variável	✓		✓		✓		
Tolerância a falhas	✓	✓		✓			
Mecanismo de Cache	✓						
Isolamento de Serviços Defeituosos	✓						
Reconfiguração em tempo de execução	✓						

Capítulo 7

Conclusão e Trabalhos Futuros

Este trabalho teve como objetivo propor um Middleware para prover o uso de Serviços Web Semânticos para as aplicações, através dos processos de descoberta, composição e invocação automática de serviços. De um modo particular, esta proposta se destaca por oferecer fácil extensibilidade e adaptação para possibilitar que os usuários da ferramenta, desenvolvedores de aplicações que utilizem Serviços Web Semânticos, possam adaptar a configuração do Middleware e as funcionalidades que estão sendo providas de acordo com a necessidade da aplicação em desenvolvimento. O desenvolvimento de aplicações utilizando o Middleware é detalhado em Barros et al. (2011*a*).

Com o objetivo de prover confiabilidade para a aplicação, um mecanismo de tolerância a falhas foi proposto e implementado com o objetivo de garantir que caso haja um erro na invocação de um serviço, o Middleware busque um novo serviço que possa substituir o serviço defeituoso. Neste contexto, este mecanismo de tolerância a falhas foi publicado em Barros et al. (2010).

Além disso foi proposto um modelo baseado em Restrições com o objetivo de verificar a corretude dos resultados providos pelos serviços com o objetivo de não permitir que um serviço forneça resultados inconsistentes para a aplicação. Este modelo é detalhado no artigo Barros et al. (2011*b*).

Para prover adaptabilidade, o Middleware provê pontos de extensibilidade que os desenvolvedores podem utilizar para extender as funcionalidades que necessitam de mudanças. Estes pontos de extensibilidade são:

- O usuário do Middleware pode adicionar técnicas para realizar os processos de descoberta e composição de serviços de acordo com as características e necessidades da aplicação.
- O uso de um conjunto de técnicas para descoberta e composição de serviços no atendimento cada requisição, aumentando assim a possibilidade de se obter sucesso na busca por um serviço.

- Criação de novas estruturas para descrição de controladores para execução de serviços compostos.
- Desenvolvimento de mecanismos para realizar a invocação dos serviços que forneçam informações ou transformações necessárias para a aplicação.
- Adição de Restrições com o objetivo de verificar os resultados fornecidos pelos serviços que a aplicação faz uso.
- Uso de diferentes abordagens para descrição de Planejamento estático de composições de serviços.

Ainda foram apresentados detalhes acerca da concepção, desenvolvimento e uso do Middleware, mostrando o uso do arquivo de configuração do sistema que mostra o procedimento necessário para que o usuário da ferramenta possa personalizá-la de acordo com suas necessidades.

Com o objetivo de auxiliar aos desenvolvedores no processo de configuração do Middleware, estão sendo desenvolvidas ferramentas gráficas que sejam responsáveis por conduzir o desenvolvedor no processo de customização do Middleware, desde a escolha para as técnicas de descoberta, Restrições que serão utilizadas e manipulação do repositório de serviços.

Como trabalhos futuros, pode-se citar a necessidade de criação/adaptação de outras técnicas mais sofisticadas para descoberta e composição de Serviços Web Semânticos, que utilizem outras informações presentes nas descrições semânticas dos serviços, ou seja, que levem em consideração parâmetros como pré-condições, efeitos, qualidade de serviço (QoS), entre outros.

Ainda neste contexto, este trabalho não procurou estudar profundamente quais técnicas para descoberta e composição de serviços devem ser utilizadas para cada tipo de aplicação. Indícios deste questionamento podem ser encontrados no trabalho proposto em Weise et al. (2008), mas um estudo mais detalhado deverá realizado dentro utilizando o Middleware proposto no presente trabalho.

Em outra perspectiva, o desenvolvimento de mecanismos para verificar periodicamente o estado dos serviços que estão nos repositórios, por um lado detectando serviços que estejam apresentando problemas em sua execução antes que a aplicação faça uso destes serviços, por outro lado verificando se os serviços que apresentaram problemas anteriormente e estão localizados no repositório (SUO) já retornaram ao seu funcionamento padrão. O uso de Agentes de Software aparenta ser adequado para este propósito.

Bibliografia

Aggarwal, R., Verma, K., Miller, J. & Milnor, W. (2004), Constraint driven web service composition in meteor-s, *in* 'Proceedings of IEEE International Conference on Services Computing'.

Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A. & Verma, K. (2005), 'Web service semantics - wsdl-s. w3c member submission.'. URL http://www.w3.org/Submission/WSDL-S/.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S. et al. (2003), 'Business process execution language for web services version 1.1'. URL http://msdn.microsoft.com/en-us/library/ee251594(v=bts.10).aspx, Last access in October 2011.

Ankolekar, Burstein, M., Hobbs, J., Lassila, O., Martin, D., Mcilraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K. & Zeng, H. (2001), 'Daml-s: Semantic markup for web services'.

Antoniou, G. & van Harmelen, F. (2004), *A Semantic Web Primer*, MIT.

Baader, F. & Nutt, W. (2003), *The Description Logic Handbook: Theory, implementation and applications.*, Cambridge University Press, chapter Basic Description Logics, pp. 48–100.

Barros, H., Bittencourt, I. I., Costa, E., Silva, M. & Véras, D. (2010), A fault tolerance approach for semantic web services based systems: A case study in e-learning, *in* 'Anais do XXI Simpósio Brasileiro de Informática na Educação (SBIE 2010)', III Brazilian Workshop on Semantic Web and Education.

Barros, H., Calado, I., Silva, M., Bittencourt, I. I. & Costa, E. (2009), Using semantic web services to automatically attend to educational requests, *in* 'Anais do Simpósio Brasileiro de Informática na Educação. SBIE'.

Barros, H., Silva, A., Bittencourt, I. I., Costa, E., Holanda, O. & Sales, L. (2011*a*), 'Steps, techniques, and technologies for the development of intelligent applications based on semantic web services: A case study in e-learning systems', *International Scientific Journal Engineering Applications of Artificial Intelligence*.

Barros, H., Silva, M., Costa, E., Bittencourt, I. I., Véras, D. & Machado, A. (2011*b*), 'Providing reliable use of semantic web services in educational systems.', *IEEE Multidisciplinary Engineering Education Magazine*.

Berners-Lee, T., Fielding, R. T. & Masinter, L. (2005), 'Unified resource identifiers', http://gbiv.com/protocols/uri/rfc/rfc3986.html. Acesso: Agosto de 2007.

Berners-Lee, T., Hendler, J. & Lassila, O. (2001), 'The semantic web', *Scientific American* **284**(5), 34–43. URL http://www.scientificamerican.com/article.cfm?id=the-semantic-web.

Bleul, S., Comes, D., Geihs, K. & Kirchhoff, M. (2009), Automated integration of web services in bpel4ws processes, *in* K. David & K. Geihs, eds, 'Kommunikation in Verteilten Systemen (KiVS)', Informatik aktuell, Springer Berlin Heidelberg, pp. 105–116. URL http://dx.doi.org/10.1007/978-3-540-92666-5_9, 10.1007/978-3-540-92666-5_9.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. & Orchard, D. (2004), Web services architecture, Technical report, W3C.

Breitman, K. (2005), *Web Semântica: A Internet do Futuro*, LTC.

Calado, I., Barros, H. & Bittencourt, I. I. (2009), 'An approach for semantic web services automatic discovery and composition with similarity metrics.', *SAC* pp. 694–695.

C.Daconta, M., Obrst, L. J. & Smith, K. T. (2003), *The Semantic Web: A Guide to the Future of XML, Web Services and Knowledge Manangement*, Joe Wilkert.

Claro, D. B. & de Araújo Macêdo, R. J. (2008), Dependable web service compositions using a semantic replication scheme, *in* '26 Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos'.

Devedzic, V. (2004), 'Education and the semantic web', *International Journal of Artificial Intelligence in Education* **14**, 39–65.

Dowling, J., Schäfer, T., Cahill, V., Haraszti, P. & Redmond, B. (2000), Using Reflection to Support Dynamic Adaptation of System Software: A Case Study Driven Evaluation, *in* W. Cazzola, R. J. Stroud & F. Tisato, eds, 'Reflection and Software Engineering', Vol. 1826 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, Berlin, Heidelberg, chapter 10, pp. 169–188. URL http://dx.doi.org/10.1007/3-540-45046-7_10.

Farrell, J. & Lausen, H. (2007), 'Semantic annotations for wsdl and xml schema', http://www.w3.org/TR/sawsdl/.

Fensel, D. & Bussler, C. (2002), 'The web service modeling framework wsmf', *Electronic Commerce Research and Applications* **1**, 113–137.

Galway, D. & Innsbruck, S. (2008), 'Wsmx (web service modelling execution environment)', http://www.wsmx.org/. Last access in September 2010.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994), *Design Patterns: Elements of Reusable Object-Oriented Software*, AddisonWesley Professional.

Giv, R. D., Kalali, B., Zhang, S., Zhong, N. & Lopez-Ortiz, A. (2004), Algorithms for direct and indirect semantic matching of web services, Technical report, University of Waterloo.

Gruber, T. R. (1993), 'A translation approach to portable ontology specifications", *Knowledge Acquisition* **5**(2), 199–220.

Haas, H. & Brown, A. (n.d.), 'Web services glossary'.

Klein, M. (2001), 'XML, RDF, and relatives', *Intelligent Systems, IEEE [see also IEEE Intelligent Systems and Their Applications]* **16**(2), 26–28. URL http://dx.doi.org/10.1109/5254.920596.

Klusch, M. (2008), Semantic service coordination [semantic service discovery and composition: A survey], *in* 'CASCOM - Intelligent Service Coordination in the Semantic Web', Springer, p. Chapter 4.

Klusch, M. & Gerber, A. (2005), Semantic web service composition planning with owls-xplan, *in* 'In Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web', pp. 55–62.

Klusch, M., Fries, B. & Sycara, K. (2006), Automated semantic web service discovery with owls-mx, *in* 'AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems', ACM, New York, NY, USA, pp. 915–922.

Koivunen, M.-R. & Muller, E. (2002), W3c semantic web activity, *in* 'Semantic Web Kick-Off in Finland - Vision, Technologies, Research, and Applications'.

Kreger, H. (2001), 'Web service conceptual architecture (wsca 1.0)', Disponivel em http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

Kuster, U., KunigRies, B., Stern, M. & Klein, M. (2007), Diane an integrated approach to automated service discovery, matchmaking and composition., *in* 'WWW 2007'.

Majithia, S., Walker, D. W. & Gray, W. (2004), 'A framework for automated service composition in service-oriented architectures', *The Semantic Web: Research and Applications* **Lecture Notes in Computer Science**, 269–283.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N. & Sycara, K. (2004), 'Owl-s: Semantic markup for web services', http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/. W3C Member Submission.

Martin, D., Domingue, J., Brodie, M. L. & Leymann, F. (2007*a*), 'Semantic web services, part 1', *IEEE Intelligent Systems* **22**(5), 12–17.

Martin, D., Paolucci, M. & Wagner, M. (2007*b*), Bringing semantic annotations to web services: Owl-s from the sawsdl perspective, *in* 'Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference', pp. 340–352.

McIlraith, S. A., Son, T. C. & Zeng, H. (2001), 'Semantic web services', *IEEE Intelligent Systems* **16**(2), 46–53.

Microsoft (1996), 'Dcom technical overview', http://msdn.microsoft.com/en-us/library/Aa286561. Último acesso em Janeiro de 2008.

Moller, T. & Schuldt, H. (2010), 'Osiris next: Flexible semantic failure handling for composite web service execution', *International Conference on Semantic Computing* **0**, 212–217.

Musa, J. D., Iannino, A. & Okumoto, K. (1987), *Software reliability: measurement, prediction, application*, McGraw-Hill, Inc., New York, NY, USA.

Nandigam, J., Gudivada, V. N. & Kalavala, M. (2005), 'Semantic web services', *J. Comput. Small Coll.* **21**(1), 50–63.

OASIS (2006), 'Uddi - universal description, discovery and integration', http://uddi.xml.org/. URL http://www.uddi.org/.

OWL (2007), 'Owl web ontology language overview', Disponível em: http://www.w3.org/TR/owl-features/. Last access in June 2010.

Paolucci, M., Kawamura, T., Payne, T. R. & Sycara, K. (2002*a*), 'Importing the semantic web to uddi'.

Paolucci, M., Kawamura, T., Payne, T. R. & Sycara, K. (2002*b*), 'Semantic matching of web services capabilities', *The Semantic Web - ISWC 2002: First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002. Proceedings* pp. 333+.

Paolucci, M., Srinivasan, N. & Sycara, K. (2004), Expressing wsmo mediators in owl-s, *in* 'In Proceedings of the workshop on Semantic Web Services: Preparing to Meet the World of Business Applications held at the 3rd International Semantic Web Conference (ISWC 2004)'.

Payne, T. & Lassila, O. (2004), 'Semantic web services', *IEEE Intelligent Systems*.

Pedrinaci, C., Domingue, J. & Sheth, A. (2010), Semantic web services, *in* 'Handbook of Semantic Web Technologies', Vol. volume 2 of *Semantic Web Applications*, Springer.

Preist, C. (2004), A conceptual architecture for semantic web services, *in* 'International Semantic Web Conference'.

Richardson, L. & Ruby, S. (2007), *RESTful web services*, O'Reilly. URL http://portal.acm.org/citation.cfm?id=1406352.

Roman, D., Lausen, H. & Keller, U. (2005), 'Web service modeling ontology (wsmo)', http://www.wsmo.org/TR/d2/v1.3/. Last access in June 2010.

Shadbolt, N., Berners-Lee, T. & Hall, W. (2006), 'The semantic web revisited', *IEEE Intelligent Systems* **21**(3), 96–101.

Silva, M. V. & Claro, D. B. (2009), Transplan: Uma ferramenta para mapear e planejar serviços web, *in* '8th International Information and Telecommunication Technologies Symposium'.

Singh, M. P. & Huhns, M. N. (2004), *Service-Oriented Computing: Semantics, Processes, Agents*, Wiley.

Sivashanmugam, K., Verma, K. & Miller, J. (2003), Adding semantics to web services standards, *in* 'Proceedings of the 2003 International Conference on Web Services (ICWS'03)'.

Srinivasan, N., Paolucci, M. & Sycara, K. (2006), Semantic web service discovery in the owl-s ide, *in* 'HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences', IEEE Computer Society, Washington, DC, USA, p. 109.2.
UPNP Service Discovery for Heterogeneous Networks

UPNP Service Discovery for Heterogeneous Networks *(2006). URL* `http://dx.doi.org/10.1109/PIMRC.2006.254286`.

Verma, K. & Sheth, A. (2007), 'Semantically annotating a web service', IEEE Internet Computing **11**, *83–85.*

Weise, T., Bleul, S., Comes, D. & Geihs, K. (2008), Different approaches to semantic web service composition, in *'Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services'.*

Zhang, J. & Zhang, L.-J. (2005), Criteria analysis and validation of the reliability of web services-oriented systems, in *'Proceedings of the IEEE International Conference on Web Services', ICWS '05, IEEE Computer Society, Washington, DC, USA, pp. 621–628. URL* `http://dx.doi.org/10.1109/ICWS.2005.44`.