KHERONN KHENNEDY MACHADO

COMPOSIÇÃO DINÂMICA DE SERVICOS WEB UTILIZANDO ONTOLOGIAS NA DESCRIÇÃO E PLANEJADORES HIERÁRQUICOS EM INTELIGÊNCIA ARTIFICIAL

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Marcos Alexandre Castilho

CURITIBA

2010

KHERONN KHENNEDY MACHADO

COMPOSIÇÃO DINÂMICA DE SERVICOS WEB UTILIZANDO ONTOLOGIAS NA DESCRIÇÃO E PLANEJADORES HIERÁRQUICOS EM INTELIGÊNCIA ARTIFICIAL

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Marcos Alexandre Castilho

CURITIBA

2010

KHERONN KHENNEDY MACHADO

COMPOSIÇÃO DINÂMICA DE SERVICOS WEB UTILIZANDO ONTOLOGIAS NA DESCRIÇÃO E PLANEJADORES HIERÁRQUICOS EM INTELIGÊNCIA ARTIFICIAL

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comisso formada pelos professores:

Orientador: Prof. Dr. Marcos Alexandre Castilho
Departamento de Informática, UFPR

Prof. Dr. Luiz Carlos Erpen de Bona
Departamento de Informática, UFPR

Prof. Dr. Edjard de Souza Mota
UFAM

Prof. Dr. Fabiano Silva
Departamento de Informática, UFPR

Curitiba, 30 de agosto de 2010

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus pela minha vida e das pessoas que me amam e torcem por mim. Pela minha família que sempre me incentivou e de alguma forma, tornou meu caminho nos estudos possível. Agradeço a minha esposa Cleice que sempre dedicou especial atenção, demonstrando toda paciência e carinho. Não posso deixar de agradecer meu filho Khaike que esperou um pouco para alegrar a minha vida Ao meu colega de curso, Franck, pelo incentivo e apoio. Aos meus professores, em especial ao professor Castilho que sempre dedicou seus conselhos, orientações e correções de forma maestra, com paciência e tolerância, principalmente nos últimos meses que morei longe da Universidade. Agradeço aos conselhos, sempre pertinentes do professor Bona que atuou como coorientador. Finalmente, um especial agradecimento ao meu pai, maior incentivador e ser humano que poderei conhecer.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A padronização de linguagens para exposição de serviços na Internet foi responsável pelo novo paradigma computacional: a Computação Orientada a Serviços. Nesse contexto, os serviços web surgem como tecnologia capaz de garantir interoperabilidade entre sistemas heterogêneos.

Várias técnicas têm sido propostas para a reutilização desses serviços afim de reutilizá-los através de sua composição, frequentemente chamado de *Workflow* de serviços. Neste trabalho investiga-se a composição de serviços através de planejadores hierárquicos utilizando ontologias para descrever as operações e serviços web.

Palavras chave: Serviços web, composição de serviços, ontologias, planejadores hierárquicos.

ABSTRACT

The standard of language for exposing services on the Internet was responsible the new computing paradigm: Service-Oriented Computing. In this context the main component, Web services, emerge as technology to ensure interoperability between heterogeneous systems.

Several techniques have been proposed for reuse these services in order to reuse them through its composition, often called the Workflow service. This work investigates the composition of services by planners using hierarchical ontologies to describe the operations and Web services

Key-words: Web services, service composition, ontology, hierarchical planners.

CAPÍTULO 1

INTRODUÇÃO

A padronização das linguagens para exposição de serviços na Internet foi responsável pelo surgimento de um novo paradigma: a computação orientada a serviços. Neste cenário destacam-se os serviços web (*Web Service*) como principal componente que possibilita a troca de mensagens, permitindo a interoperabilidade entre sistemas heterogêneos de forma transparente e segura. Um serviço web pode ser definido como um programa auto-descrito e auto-contido que é acessado e distribuído pela Internet [20]

Em razão da facilidade de desenvolvimento e da vantagem de publicá-los na Internet, surgiu a necessidade de agregá-los, permitindo que mais serviços sejam executados para satisfazer requisições do usuário. A técnica que permite a junção de serviços é chamada de Composição de serviços web.

Um cenário possível para aplicação de composição de serviços seria verificar os serviços necessários para uma viagem. Para exemplificar, suponha que um determinado cliente decide fazer uma viagem utilizando para isso uma prestadora de serviços. A agência de viagens, para realizar a ação "Marcar Viagem", precisa encontrar vários serviços na web que satisfaçam a requisição, sendo que cada serviço ("Marcar Hotel", "Marcar Voo"e "Alugar Carro"), provavelmente, será de uma empresa diferente. A figura 1.1 ilustra esta situação. A situação ideal seria que os serviços fossem encontrados e combinados automaticamente.

Os trabalhos relacionados na literatura descrevem duas abordagens em relação ao processo de agrupar serviços. Basicamente essa classificação relaciona-se com o nível de intervenção do usuário na composição de serviços, estáticos ou automáticos.

Para compor serviços de forma estática, uma técnica é mapeá-los como um conjunto coordenado de atividades, paralelas e/ou sequenciais, que são interligadas com o objetivo de alcançar uma meta comum. Essa técnica é definida como Workflow [51]. O esforço

nessa abordagem está no desenvolvimento de programas que gerenciem o fluxo de trabalho, automatizando parcialmente um processo de negócio através de linguagens como BPEL4WS.

BPEL4WS é uma linguagem baseada em XML, criada pelo consórcio OASIS, desenvolvida para especificar interações entre serviços web, importando e exportando informações entre os serviços e utiliza exclusivamente interfaces [20]. Ainda outro esforço baseado em workflow é uma plataforma para linguagem que define serviços compostos (CSDL) [27]. O elemento principal nessa abordagem é chamado de atividade de interação, que descreve uma troca de informações entre as partes, tendo foco no receptor.

Compor estaticamente serviços pode ser vantajoso quando em cenários de poucos serviços, no entanto, o ambiente web é caracterizado por uma quantidade enorme de serviços.

Para esse aspecto um recurso seria automatizar o problema da composição de serviços, através de mapeamento para um problema de planejamento em Inteligência Artificial, objeto de estudo do presente trabalho.

Umas das principais iniciativas nesse campo é a descrição dos serviços aplicando semântica, para que os planejadores possam fazer inferências com base nas descrições das condições e efeitos. Nesse contexto, a linguagem OWL (Ontology Web Language), mais especificamente a OWL-S [16], que é uma especificação para escrever serviços na web, é um dos principais esforços, além de ser recomendada pela W3C (World Wide Web Consortium). Essa abordagem permite a utilização de planejadores conforme descrito em [11] e [52] .

O presente trabalho propõe a discussão da abordagem que compõe serviços de forma automática, através da implementação de um protótipo que utiliza semântica na descrição dos serviços e planejador hierárquico para a construção automática de planos. Nesse sentido a escolha da linguagem *Ontology Web Language* (OWL) [5] foi eleita, pois ao contrário de outras linguagens que trabalham com fluxo, como BPEL4WS [15], não são capazes de atribuir condições e efeitos.

Os capítulos estão organizados da seguinte forma: o capítulo 2 é dedicado ao princi-

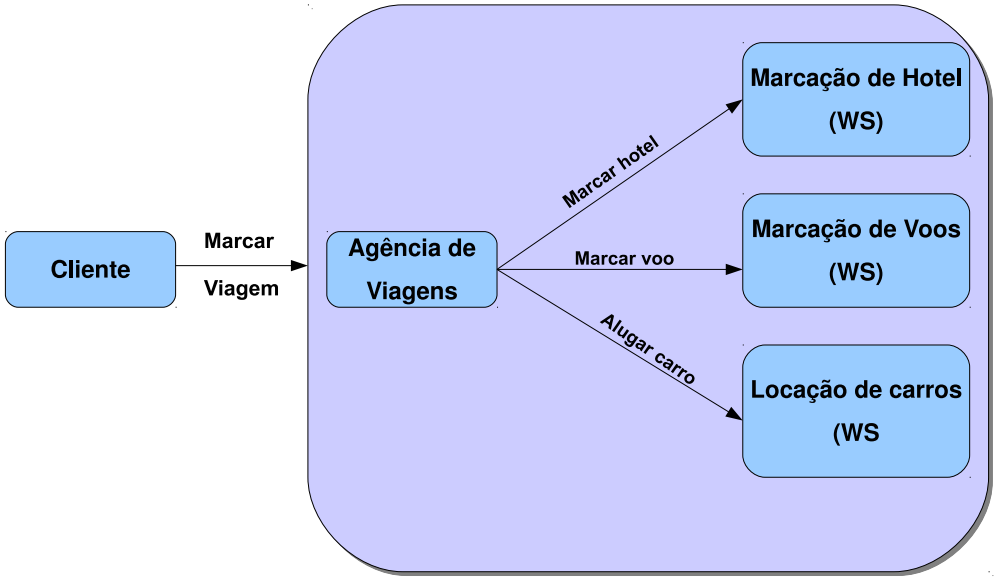

Figura 1.1: Exemplo do problema da composição de serviços - Domínio agência de viagens

pal componente do problema da composição de serviços, os serviços web. O capítulo 3 descreve a web semântica, as características e os esforços para inserção de representação de significados em documentos web.

O capítulo 4 trata de uma revisão literária sobre o planejamento, o planejamento clássico, suas restrições, as linguagens de representação, além do planejamento hierárquico, em especial o planejador JSHOP2.

O capítulo 5 contém uma proposta de aplicação para testar a composição serviços web, utilizando descrição semântica nos serviços e o planejador JSHOP2 para criar planos de serviços. Finalmente, no capítulo 6 apresenta a conclusão e propostas de trabalhos futuros.

CAPÍTULO 2

WEB SERVICE

Neste capítulo serão abordados os conceitos e fundamentos da tecnologia que permite a distribuição de programas na Internet ataravés de serviços web (*Web Services*).

Na seção 2.1 é apresentado o conceito de serviços web, suas características e funcionalidades, na seção 2.2 o protocolo para realizar invocações utilizando objetos; a seção 2.3 apresenta a notação básica de todos os componentes. Na seção 2.4, a linguagem para descrição de serviços e suas características, na seção 2.5 é descrito como publicar os serviços e finalmente na seção 2.6 outra abordagem para desenvolver serviços é mostrada.

2.1 Conceitos

A padronização de serviços na web permitiu uma infinidade de recursos e uma gama de aplicações que independem do dispositivo e plataforma no qual o programa será acessado. Esses programas são denominados de serviços web.

Conceitualmente, essa tecnologia permite que sistemas heterogêneos possam se integrar de forma transparente e segura, através de protocolos padrões da Internet.

A proliferação de tais serviços permitiu o surgimento de um novo paradigma computacional, onde o foco das aplicações é o conceito de serviços, definidos através de interfaces e não mais como uma "caixa-preta" como era até então.

Dessa forma serviços web são programas auto-descritos e auto-contidos que são acessados e distribuídos pela Internet [20]. O principal foco é a interoperabilidade entre diferentes sistemas, linguagens e plataformas.

Ainda conceituando, um serviço web é um componente de software que é invocado utilizando a estrutura de documentação XML (*Extend Markup Language*) sobre o protocolo SOAP (*Simple Object Access Protocol*). O comportamento dos serviços, além das entradas e saídas, é descrito utilizando uma linguagem chamada WSDL (*Web Service*

Description Language) [14] . Estes conceitos são detalhados a seguir.

Tecnicamente, um serviço web é composto de rotinas passíveis de serem ativadas por qualquer aplicativo disponibilizado pela Web, a partir de alguns parâmetros de autenticação e segurança.

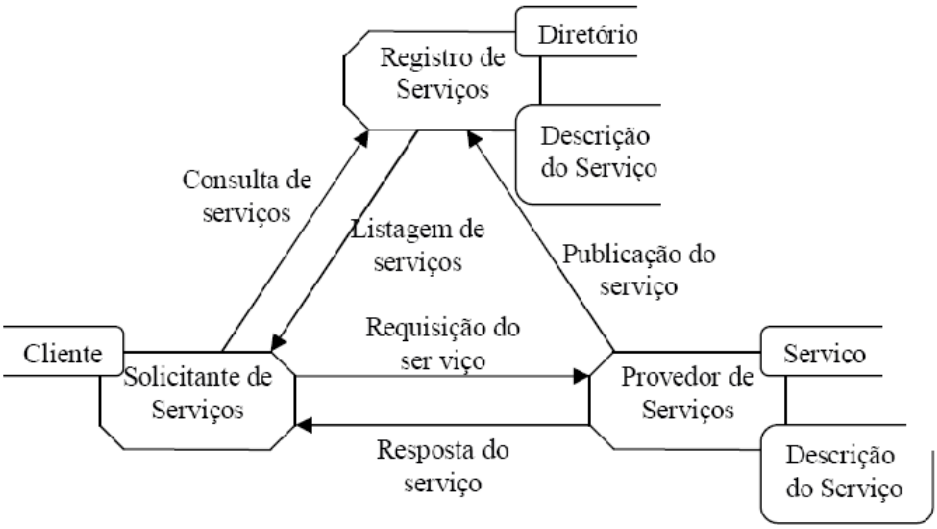

Figura 2.1: Arquitetura dos serviços

Basicamente a estrutura de um serviço depende de três componentes: o cliente; o serviço; e o diretório. A figura 2.1 ilustra essa arquitetura.

O processo inicia com a consulta de serviços ao Registro, a UDDI (Universal Description, Discovery and Integration), responsável por descrever, descobrir e integrar serviços, retornando uma listagem que corresponde à consulta. A requisição é feita pelo cliente ao Provedor de Serviços, que após a conferência da publicação, responde ao cliente qual é o serviço desejado/encontrado.

Nas próximas seções serão descritos cada componente e funcionalidades dessa interação:

- o protocolo para descrever requisições dos serviços;
- a linguagem específica dos serviços, descrição de parâmetros, entradas e saídas;
- a publicação em diretórios específicos.

2.2 O protocolo SOAP

O SOAP (*Simple object access protocol*) é um protocolo elaborado para facilitar a chamada remota de funções via Internet, permitindo que dois programas se comuniquem de uma maneira tecnicamente muito semelhante à invocação de páginas Web. O protocolo *SOAP* tem diversas vantagens sobre outras maneiras de se chamar funções remotamente (como *DCOM*, *CORBA* ou diretamente no TCP/IP):

- é simples de implementar, testar e usar;
- é um padrão da indústria, criado por um consórcio e adotado pela W3C[1] e por várias outras empresas;
- usa os mesmos padrões da Web para quase tudo: a comunicação é feita via *HTTP* com pacotes virtualmente idênticos; os protocolos de autenticação e encriptação são os mesmos; a manutenção de estado é feita da mesma forma; é normalmente implementado pelo próprio servidor Web;
- normalmente não é bloqueado por *firewalls* e roteadores, que consideram os pacotes como qualquer comunicação normal HTTP.
- tanto os dados como as funções são descritas em *XML*, o que torna o protocolo não apenas fácil de usar como também muito robusto;
- é independente do sistema operacional e CPU;
- pode ser usado tanto de forma anônima como com autenticação (nome/senha).

Os pedidos SOAP podem ser feitos em três padrões: GET, POST e SOAP [40]. Os padrões GET e POST são idênticos aos pedidos feitos por navegadores Internet. O SOAP é um padrão semelhante ao POST, mas os pedidos são feitos em XML o que permite recursos mais sofisticados, como passar estruturas e vetores.

Independente de como é feito o pedido, as respostas são sempre em XML (ver 2.3). O XML descreve perfeitamente os dados em tempo de execução e evita problemas causados

[1] `http://www.w3.org/TR/SOAP/`

por mudanças inadvertidas nas funções, já que os objetos chamados têm a possibilidade de sempre validar os argumentos das funções, tornando o protocolo muito robusto. Além disto, O SOAP pode ser facilmente implementado em praticamente qualquer ambiente de programação [40].

O *SOAP* provê uma forma de comunicação entre aplicações rodando em diferentes sistemas operacionais, com diferentes tecnologias e linguagens de programação. Na subseção 2.2.1 é descrita a sintaxe e a estrutura do protocolo *SOAP*.

2.2.1 Sintaxe

Uma mensagem SOAP é um documento XML simples que deve conter os seguintes elementos:

- um elemento Envelope que identifica o documento XML como uma mensagem *SOAP*;
- um elemento de cabeçalho que contém informações gerais;
- um elemento de corpo que contém informações de chamada e resposta;
- um elemento *Fault* contém erros e informações de status.

Todos os elementos acima são declarados no *namespace* padrão para o envelope SOAP, conforme ilustrado na figura 2.2:

2.3 Linguagem de marcação sintática

O XML é uma versão simplificada do *Standart Generalized Markup Language* (SGML) e a principal característica da linguagem é descrever um documento com ajuda de elementos de marcação sintática. Moutis e Kirk [31] definem que um elemento em XML é algo que descreve um dado.

Ainda de acordo com esses autores, os documentos são formados de três componentes distintos:

```
<? Xml version = "1.0"?>
<soap:Envelope sabão <: Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope" xmlns: soap = http://www.w3.org/
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header> <soap:Header>
... ...
</soap:Header> </ Soap: Header>

<soap:Body> <soap:Body>
... ...
<soap:Fault> <soap:Fault>
... ...
</soap:Fault> </ Soap: Fault>
</soap:Body> </ Soap: Body>

</soap:Envelope> </ Soap: Envelope>
```

Figura 2.2: Esqueleto SOAP

- conteúdo de dados: as palavras propriamente ditas;
- estrutura: o tipo do documento e organização de seus elementos;
- apresentação: formato de como as informações são apresentadas.

A flexibilidade da linguagem XML permite aos usuários definirem seus próprios elementos e atributos, extrair dados de um documento, além de definir a estrutura de dados e dos documentos em qualquer nível de complexidade. Uma característica importante na flexibilidade são os DTDs (Document Type Definition). Esse item permite a qualquer desenvolvedor criar suas marcações específicas que atendam aos propósitos da descrição de dados. Os seguintes recursos de um DTD são listados:

- tipos de elementos que são permitidos dentro de um documento XML;
- diversas características de cada tipo de elemento, junto com os atributos e conteúdos que cada elemento é capaz de ler;
- qualquer notação encontrada dentro de um documento;
- as entidades usadas.

2.4 Descrevendo serviços com WSDL

Para que uma aplicação possa implementar uma interface que irá se comunicar com um serviço web, ela necessariamente precisa conhecer a sua estrutura para efetuar a comunicação [13]. Dessa maneira, um documento WSDL (*Web Service Description Language*) descreve sintaticamente um serviço, utilizando tags em liguagem de marcação.

Em outras tecnologias não existe nada análogo ou que tenha toda a eficiência de um documento WSDL. Como dito anteriormente, o documento WSDL é escrito utilizando XML como metalinguagem. Ele descreve a interface da implementação de um determinado código, incluindo o nome dos métodos, os parâmetros e seus tipos, o retorno (caso exista), o protocolo no qual o serviço vai se comunicar, as estruturas dos dados complexas, as URL's de onde se encontra o serviço, entre outros [14].

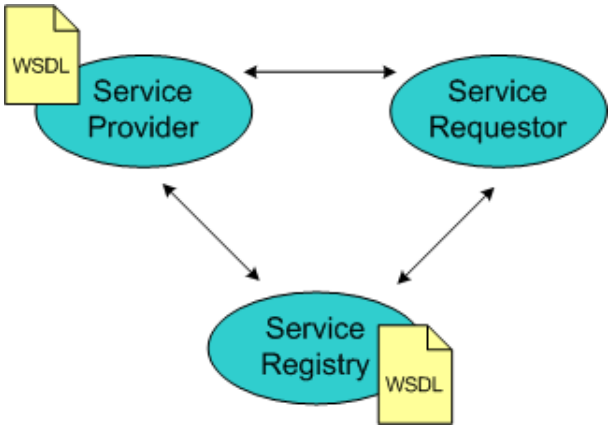

Figura 2.3: Componentes da especificação WSDL

A figura 2.3 ilustra os componentes de especificação de um documento WSDL. Cada serviço é representado por um círculo que executa uma determinada tarefa. O requisitante do serviço (*Service Requestor*) é que faz a requisição dos Registros para o provedor de serviços (*Service Provider*), que, por sua vez, verifica o registro no registro de serviços *Service Registry*. De posse desse documento, qualquer empresa é capaz de escrever o corpo inicial de uma aplicação, sem a necessidade de entrar em contato com o provedor do serviço. Isso é muito vantajoso no sentido de que uma empresa pode criar um serviço web público (como um que faça cálculos financeiros gratuitamente) e disseminá-lo pelo mundo. A implementação do serviço é feita em partes, conforme ilustrado na figura 2.4

permitindo a definição da implementação do serviço e a definição da interface de forma independente.

A próxima seção define como os serviços são registrados para serem acessados posteriormente.

Figura 2.4: Camadas de descrição

2.5 Registro de serviços web

Existem dois tipos de serviços web: os públicos e os privados [48]. Os serviços web privados são desenvolvidos para uma quantidade limitada e normalmente conhecida de clientes. Um exemplo seria uma empresa que fornece um serviço web para que seus clientes possam atualizar dados cadastrais. Já os serviços web públicos são de propósito aberto, ou seja, qualquer cliente no mundo poderia se conectar a ele.

Um exemplo disso seria um serviço web que efetua tradução de texto. O serviço poderia fazer propagandas, mala-direta, etc, porém isso não seria tão eficiente quanto divulgar seu produto em um registro universal.

O UDDI (*Universal Description, Discovery and Integration*) é definido como um banco de dados mundial de serviços web. Várias empresas já disponibilizaram um registro UDDI, incluindo Microsoft e IBM. Uma grande vantagem desses registros é que eles são replicados, garantindo que qualquer serviço web encontrado em um registro estará disponível em todos os outros.

Existem duas formas de busca em registros universais. A primeira é utilizar as páginas

web dos próprios registros. A outra é implementar uma interface de busca, já que os registros UDDI fornecem serviços web comuns de busca. Por fim, quando um serviço é encontrado em um registro universal, o cliente simplesmente informa o arquivo WSDL que contém todas as informações necessárias.

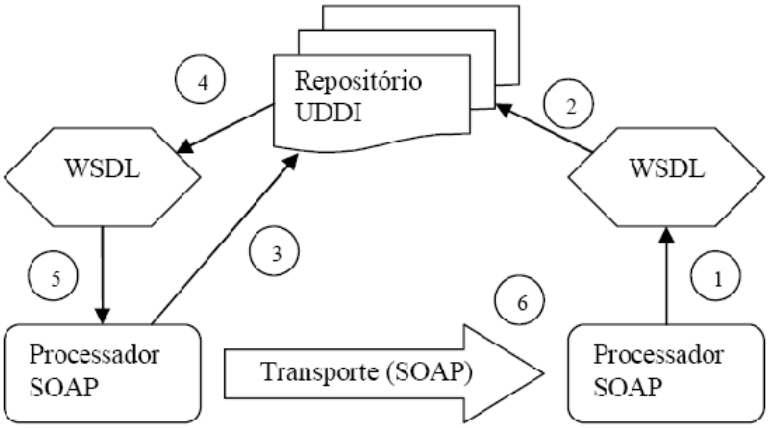

Figura 2.5: Fluxo do processo para descoberta do serviço

A figura 2.5 ilustra o fluxo para a descoberta e invocação dos serviços. Gera-se o arquivo WSDL para descrever o serviço web com suporte do processador SOAP (1) e utiliza-se a API(*Aplication Program Interface*) UDDI para registrar as informações no repositório (2). Os dados são transmitidos juntamente com as informações sobre contato e o registro possui uma entrada (URL que aponta para o Servidor SOAP) com a localização do WSDL, assim outro Processador SOAP pode requisitar o registro (3) para obter o WSDL (4). Em seguida, o cliente gera a mensagem apropriada (5) para enviar uma operação específica através de determinado protocolo (6). O cliente e o servidor devem estabelecer o mesmo protocolo e compartilhar a mesma semântica para a definição do serviço.

Na próxima seção, é definida uma nova abordagem para serviços web, serviços em grade (*grid services*), assim como plataformas e ferramentas dessa nova abordagem.

2.6 Serviços em grade computacional

Um serviço em grade é uma arquitetura de grade orientada a serviços que define uma semântica uniforme para exposição de serviços [21].

Essa arquitetura permite um ambiente com maior controle, onde os recursos computacionais devem ser compartilhados em grades de computadores. Além disso possibilita um ambiente colobarativo, utilizando-se de computadores de diferentes localidades, apropriado para sistemas que necessitam de alto desempenho.

Figura 2.6: Representação de uma Grade Computacional

A figura 2.6 exemplifica a estrutura de uma grade computacional, onde a requisição pode ser feita a partir de qualquer dispositivo com conexão para uma rede, a interface que, no exemplo é uma interface web, conjuga a informação na grade computacional. Além disso, todo o processo é transparente para o usuário final.

Entre as plataformas e projetos relacionados à grade computacionais, destaca-se o projeto *GLOBUS TOOLKIT* [47]. O *Globus Toolkit* provê um conjunto de serviços e bibliotecas para dar suporte a infra-estrutura em aplicações de grade.

A primeira e a segunda geração de grades desenvolveram esforços para interligarem estações de trabalhos e a promoção da interoperabilidade para a viabilização da computação em alta escala. No entanto, é na terceira geração que o desenvolvimento dos ambientes de grades são orientados a serviços, deslocando a abordagem que era então de compartilhamento de recursos para o compartilhamento de serviços conforme relatado em [47].

O OGSA (*Open Grid Services Architecture*) define uma arquitetura onde todos os recursos são baseados em serviços, ou seja, uma grade de serviço é um serviço web, que provê um conjunto de interfaces padrão bem definido [21].

Essas interfaces tem grande importância para a composição de serviços, pois elas

definem a descoberta, criação dinâmica de serviços, gerenciamento de tempo de vida e notificação dos objetos. Esse deslocamento ajudou a promover os serviços web, e consequentemente a composição de serviços dentro de um ambiente de grade computacional tem sido frequentemente relatado na comunidade científica [6] como o framework GSFL [33] e o eFlow [9].

2.7 Considerações

Esse capítulo apresentou a estrututa básica de um serviço web e as tecnologias e linguagens necessárias para o seu desenvolvimento. Também a convergência dos serviços para um ambiente com maior controle e colaboração, o qual é definido como serviços em grade.

A Internet possui uma grande quantidade de serviços web. A busca, seleção, composição e execução manual dos serviços necessários pode tornar-se uma atividade de extrema complexidade devido a essa quantidade de serviços. Além disso, os serviços são descritos sem valoração semântica, o que dificulta ainda mais o uso de agentes inteligentes para a satisfazer uma requisição do usuário, na tentativa de criar novas soluções a partir de serviços já implementados [44].

A fim de compor serviços existentes, sem intervenção humana, ou quase nenhuma, a composição dinâmica propõe a automação de tarefas como a busca, seleção, composição e execução dos serviços, com o uso de planejadores [52] e da descrição dos serviços com semântica.

A composição dinâmica envolve localizar automaticamente serviços que relacionam a propriedade solicitada, selecionar os serviços necessários e executá-los [39].

O uso da composição dinâmica também é pertinente quando um serviço muda sua implementação. Sem o uso de alguma notação semântica, apenas humanos podem decidir se o serviço ainda atinje o objetivo, ou seja, a descrição puramente sintática torna a tarefa da composição de serviços ainda mais complexa.

Entretanto, uma descrição semântica pode fornecer a um agente de software raciocínio sobre as operações de um serviço, requisitos e efeitos de sua ação [53].

O próximo capítulo mostra a evolução da Web Semântica, onde será mostrado como escrever serviços com anotações ricas.

CAPÍTULO 3

A WEB SEMÂNTICA

Este capítulo contém uma visão geral sobre a web semâtica e apresenta as definições e conceitos básicos sobre a evolução das linguagens para descrever significados em conteúdo Web. A seção 3.1 é dedicada aos conceitos e justificativa relacionada ao campo da Web semântica. O conceito de ontologia e sua aplicação é apresentado na seção 3.2. Na seção 3.3 é apresentada uma linguagem para descrever restrições. Na seção 3.4 é descrito o primeiro *framework* para modelar dados e finalmente na seção 3.5 a linguagem que efetiva a descrição semâtica para web e serviços.

3.1 Conceitos e a importância dos significados

Uma das grandes dificuldades encontradas na composição dinâmica de serviços web é o fato dos documentos na rede não serem descritos de forma que os computadores possam entender o significado das informações. A Internet foi estruturada de forma que os conteúdos possuem significado orientado ao contexto humano. Dessa forma, surgiu a necessidade de ordenar as informações, representando explicitamente significados.

A partir da valoração semântica dos conteúdos, agentes de softwares deverão ser capazes de processar informações de diversas fontes e compartilhadas com outros programas. A web semântica permite otimizar a organização das informações, utilizando linguagens que permitem exprimir representações de conhecimentos como XML, XML Schema, RDF [7] DAML+OIL [16]. Estes conceitos serão explicados a seguir.

Ontologias ([7] e [16]) fornecem vocabulários que definem relações entre conceitos. O conceito de ontologia, na Inteligencia Artificial, tomou outro significado. Segundo Gruber [25], ontologia é uma especificação formal, explícita e compartilhada de uma conceituação.

Uma definição importante sobre as ontologias está no uso dos metadados. Um metadado é definido como um dado associado com objetos que auxiliam usuários potenciais a ter

vantagem completa do conhecimento da sua existência e sua característica desses objetos.

Nesse sentido, na web semântica os documentos são descritos utilizando ontologias, estruturando e gerenciando os conteúdos na web. De acordo com Grau[23] a próxima geração da web visa combinar as tecnologias existentes na web com o formalismo para a representação do conhecimento.

Todos os esforços na área buscam elevar o patamar da web atual, isto é, de um emaranhado de links a um ambiente com organização e estruturação dos recursos disponíveis na Internet, conforme ilustrado na figura 3.1.

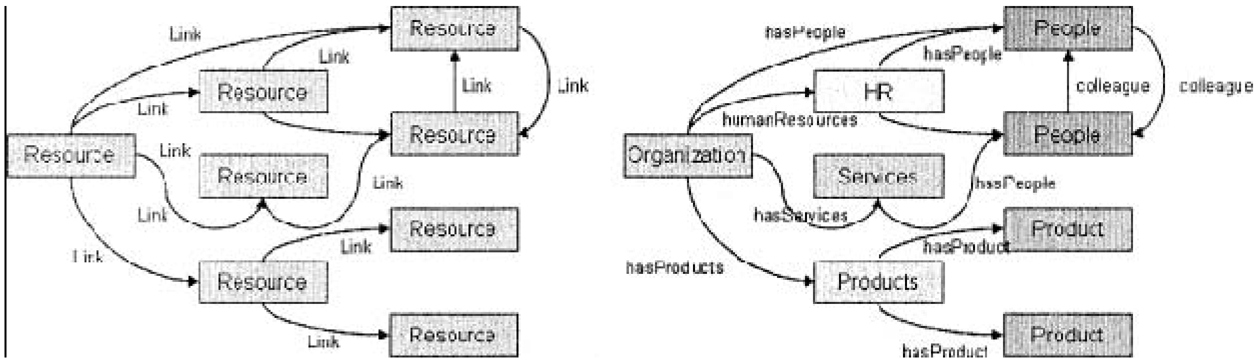

Figura 3.1: Evolução de Links para Significados

Para tornar viável a web semântica, várias pesquisas ocorrem no sentido de padronização de tecnologias, linguagens e frameworks. Entre os sujeitos dessa ação, o W3C, além de pesquisadores e empresas, estão desenvolvendo conceitos e implementando ferramentas para que tal proposta ocorra efetivamente.

As limitações dos padrões atuais para serviços web justifica a adoção de semântica[4]. Essas limitações são descritas utilizando as fases do ciclo de vida dos serviços. Em relação à descoberta, o problema maior está nos diferentes termos usado para anúncio e requisição do usuário. Na fase da invocação, diferentes especificações para mensagens e interfaces podem trazer resultados indesejados.

Outra grande deficiência dos padrões está no uso de diferentes terminologias e protocolos, resultado justamente da falta de padronização. As próximas seções descrevem os elementos que compõem a web semântica a) Ontologias; b) Representação do conhecimento desde a evolução das linguagens de marcação até as linguagens de mais alto nível de relacionamento semântico, RDF (*Resource Description Framework*) e, aquela que é

atualmente a mais utilizada e padronizada pela W3C, a linguagem de Ontologia para Web, a OWL *(Ontology Web Language)*.

3.2 Ontologias

Ontologia é definida como "uma especificação formal, explicita e compartilhada de uma conceituação"[25]. A importância para a web semântica é justamente o esclarecimento da estrutura de um conhecimento. Chandrasekaram [12] afirma que uma ontologia se refere a um conjunto de conhecimentos que descreve um domínio, usando um vocabulário representativo. Alguns benefícios apontados por Marietto[37], são apresentados a seguir:

- possibilitar uma melhor compreensão do domínio;
- possibilitar a troca de informações;
- auxiliar o compartilhamento do conhecimento, bem como o reuso;
- ontologias explicitam a especificação de sistemas, logo as ontologias ajudam no processo de verificação de um sistema computacional.

Estes pontos permitem concluir que o uso de ontologias na representação do conhecimento são de suma importância. No decorrer do trabalho serão analisadas as linguagens desenvolvidas para definição e editoração de ontologias. As próximas seções descrevem a evolução do XML (seção 2.3), desde o primeiro esforço com o *XML Schema* até a linguagem ontológica web (*OWL*).

3.3 Descrição de estruturas, conteúdos e restrições

A principal característica da linguagem XML Schema [1] é descrever a estrutura, o conteúdo, as restrições e os tipos de dados dos vários elementos e atributos de um documento XML.

[1]XML Schema é uma linguagem baseada no formato XML para definição de regras de validação ("esquemas") em documentos no formato XML.

Outra característica encontrada é o suporte à herança. Podem-se criar novos esquemas, derivados dos existentes. Isso proporciona efetivamente melhorias nos processos de desenvolvimento de aplicações XML, como exemplo, a manutenção do código e produtividade [46].

```
<?xml version="1.0"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">

  <xs:element name="note">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="to" type="xs:string"/>
         <xs:element name="from" type="xs:string"/>
         <xs:element name="heading" type="xs:string"/>
         <xs:element name="body" type="xs:string"/>
       </xs:sequence>
     </xs:complexType>
  </xs:element>

  </xs:schema>
```

Figura 3.2: Exemplo de código em linguagem XML Schema

Considere a figura 3.2, que apresenta um exemplo de código utilizando a linguagem XML Schema, na qual se percebe-se definições de objetos como “xs:complexType” e tipos como “xs:string’ para declaração de restrições de valores específicos para esses atributos.

A linguagem *XML Schema* permitiu um grande avanço ao aumentar o poder de representação sintática expressando restrições, no entanto a linguagem ainda não permite descrever relações entre dados e conceitos.

A próxima seção mostra a linguagem que permite tal definição sobre o domínio da informação.

3.4 Framework para modelagem de dados

Confome descrito na seção anterior, uma linguagem além de possuir recursos sobre a questão sintática, deve também representar o conhecimento. Nesse sentido, a linguagem RDF (*Resource Description Framework*), de acordo com Lassila e Swick[34], é uma tecnologia de representação de conhecimento que pode ser compartilhada por muitas linguagens de metadados.

O modelo de dados para documentos escritos em RDF, consiste basicamente de três tipos de objetos:

- recursos: Todo objeto que possa ser descrito por uma expressão RDF e que tenha uma URI (*Uniform Resource Identifier*) é considerado um recurso. Uma URI serve para nomear objetos de informação, podendo ser uma página web como um documento HTML, como por exemplo "http://www.kheronn.blogspot.com/contato.html", ou um elemento específico dentro de um documento XML;

- propriedades: uma propriedade é um atributo ou uma relação, usado para descrever um recurso. Cada propriedade ainda possui um significado que define quais são os valores permitidos, relações com outras propriedades e tipos de recursos que podem ser definidos;

- declaracões: uma declaração é a combinação de um recurso específico, uma propriedade nomeada, mais o valor dessa propriedade.

Além disso, esses itens também podem ser chamados de sujeito, predicado e objeto.

Existe ainda uma outra forma de representação de uma declaração RDF utilizano uma anotação em tripla, ([sujeito], [predicado], [Objeto]), pode-se representar através de grafos. Na figura 3.3 os recursos são representados como elipses, os arcos representam as propriedades e os valores como retângulos. Por exemplo, dada a declaração: *"Kheronn Khennedy Machado é estudante da disciplina Web Semântica, que adota o livro Semantic Web: Theory, Tools and Applications, cujo autor é Jorge Cardoso."* (3.1)

Recurso (Sujeito)	Kheronn Khennedy Machado - http://www.kheronn.com
Propriedade(Predicado)	estudante
Valor (Objeto)	Web semântica

Tabela 3.1: Modelo de Declaração RDF

A figura 3.3, representa o grafo da declaração que descreve o objeto Web Semântica, que no contexto é o recurso a ser descrito e que tem como tipo de propriedade livro e autor.

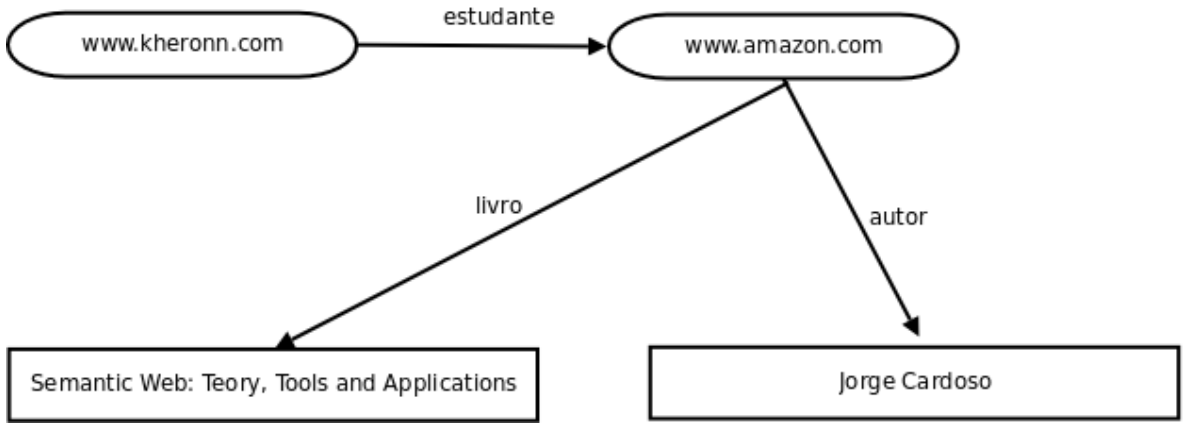

Figura 3.3: Grafo da Declaração

Logo se percebe que um recurso é um objeto do qual queremos fazer uma descrição, nem sempre é o sujeito de uma declaração. O relacionamento descreve que a disciplina Web Semântica tem um sítio http://www.amazon.com. A figura 3.3 ilustra as seguintes triplas associadas:

1. (estudante, [http://www.kheronn.com], http://www.amazon.com)

2. (livro, [http://www.amazon.com], “Semantic Web: Teory, Tools and Applications”)

3. (autor, [http://www.kheronn.com], “Jorge Cardoso”)

Representar objetos utilizando essas anotações foi o primeiro passo para descrever semânticamente documentos web.

A presente seção mostrou uma forma de representar graficamente recursos[36]. Na próxima seção veremos como representar formalmente ontologias, e as relações entre termos e conceitos.

3.4.1 Sintaxe RDF

Duas sintaxes são propostas para expressar modelos em RDF [10]:

- serializada: expressa toda potencialidade do modelo RDF;
- abreviada: que inclui construtores adicionais, expressando de forma mais compacta o modelo RDF.

Um exemplo de codificação serializada é ilustrado na figura 3.4:

```
<rdf:RDF>
  <rdf:Description about= "www.kheronn.com">
   <email>kheron@c3sl.ufpr.br </email>
  <estudante>
      <rdf:Description ID="www.amazon.com">
        <livro> Semantic Web: Theory, Tools and Applications </livro>
         <autor> Jorge Cardoso </autor>
     </rdf:Description>
    </estudante>
  </rdf:Description>
</rdf:RDF>
```

Figura 3.4: Codificação em RDF serializada do exemplo anterior

As marcas são sempre iniciadas com "rdf":, atribuída no URI do *namespace* RDF. O elemento "Description"contém a identificação do recurso a ser descrito e uma lista de propriedades que se aplicam a esse recurso. Os atributos "about"e "ID"identificam o atributo. Um recurso externo já existente é identificado pelo atributo "about". No caso da inexistência, utiliza-se ID. A propriedade do recurso que está sendo descrita é o email. O elemento estudante indica a relação entre os recursos. O elemento "Description"tem a função de descrever o recurso web Semântica.

3.4.2 Relacionamento entre propriedades em documentos RDF

Documentos escritos somente em RDF não permitem definir relacionamentos entre as propriedades e o recursos. Essa tarefa cabe ao RDF Schema.

Essa linguagem permite descrever ontologias, definindo formalmente relações entre termos e conceitos. Brickley e Guha[8] especificam que o sistema de tipos do RDF Schema

é similar ao tipo de dados de linguagens de programação orientada a objetos que, centrada na propriedade, facilita o desenvolvimento na descrição de recursos existentes na Web. O RDF Schema permite descrever estruturas de classes hierárquicas, tornando o sistema de classes extensíveis e reutilizáveis, podendo ser compartilhado e estendido.

A linguagem define uma estrutura de classes e subclasses, permitindo associações na forma de herança, incorporando extensibilidade ao modelo. Todos os recursos são definidos em um vocabulário, listados a seguir:

- rdfs:Resource: classe genérica do modelo RDF Schema. Todo objeto é descrito como um recurso;
- rdfs:Class: é subclasse do rdfs:Resource, similar a noção de classes na orientação a objetos;
- rdfs:Properties: propriedades, representando um aspecto do recurso sendo descrito.

Apesar de todas as propriedades e recursos, a linguagem RDF não fornece um vocabulário adicional com uma semântica formal capaz de facilitar uma forma comum de processamento de conteúdo semântico. Sobre essa questão é apresentada na próxima seção uma linguagem que permite tal funcionalidade.

3.5 Linguagem ontológica Web (OWL)

A linguagem ontológica Web (OWL) é a linguagem padronizada pela W3C e sucessora da linguagem DAMIL+OIL [5], desenvolvida especialmente para criação de ontologias para documentos web. Embora seja fundamentada na linguagem RDF, OWL permite um vocabulário maior e mais expressivo que seu antecessor.

A literatura inclui um "s"após a sigla OWL para diferenciar a linguagem especificamente para descrição de serviços web. A OWL-S, é um conjunto de ontologias para descrever as propriedades e capacidades dos serviços (OWL Services Coalition). O propósito da linguagem é descrever classes e as relações existentes entre elas, possibilitando que as classes sejam reutilizadas e herdadas.

Uma ontologia descrita utilizando OWL inclui a descrição de classes, propriedades e suas instâncias, especificando como derivar suas consequências lógicas.

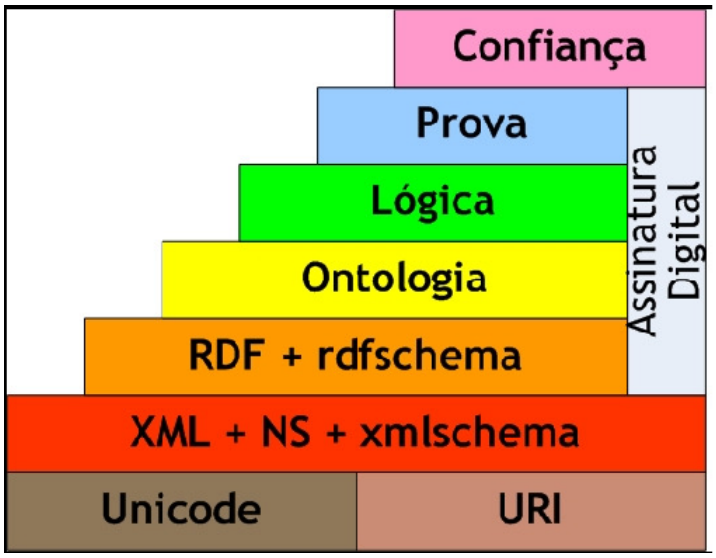

Figura 3.5: Camadas na Web Semântica

A figura 3.5 ilustra a posição da OWL na hierarquia das tecnologias semânticas, entre as camadas RDF e Lógica. De acordo com [3], a OWL-S foi desenvolvida para suportar uma efetiva automação dos serviços web, envolvendo atividades como:

- descoberta;
- execução;
- monitoramento;
- composição.

Nas seções seguintes são apresentadas características de cada versão OWL.

3.5.1 Linguagem ontológica Web simplificada (OWL Lite)

A versão mais simplificada da OWL suporta somente os requisitos básicos de classificação hierárquica e restrições simples, tais como restrições de cardinalidade e na construção de classes.

3.5.2 Linguagem ontológica Web com descrição lógica (OWL DL)

Essa versão é destinada aos usuários que desejam maior expressividade, com completude computacional e capacidade de decisão dos sistemas de raciocínio, ou seja, a computação será finalizada em tempo finito.

Esta versão possui ainda todos os construtores da linguagem OWL, exigindo separação dos tipos: classes, propriedade e indivíduo.

3.5.3 Linguagem ontológica Web completa (OWL FULL)

A linguagem OWL completa possui o máximo poder de expressividade, mas sem nenhuma garantia computacional, isto é, não garante que todas as conclusões sejam computáveis e que terminarão em tempo finito.

De acordo com a W3C, cada uma das sub-linguagens é uma extensão de sua predecessora mais simples, conforme apresentado a seguir:

- toda ontologia OWL Lite válida é uma ontologia OWL DL válida;
- toda ontologia OWL DL válida é uma ontologia OWL Full válida;
- toda conclusão OWL Lite válida é uma conclusão OWL DL válida;
- toda conclusão OWL DL válida é uma conclusão OWL Full válida.

Nas próximas subseções são apresentados os elementos principais e exemplos da linguagem OWL.

3.5.4 Sintaxe

Nesta seção são descritos os principais atributos que descrevem os componentes em OWL. A seguir são descritos alguns atributos em relação a propriedade da classe:

- oneOf : usado para descrever classes através da enumeração de indivíduos. Os membros da classe são exatamente o conjunto de indivíduos enumerados. Por exemplo,

uma classe "Pessoa"pode ser denida simplesmente pela enumeração dos indivíduos "Física"e "Jurídica";

- hasValue: usado em uma propriedade para descrever uma classe restrita a todos os indivíduos para os quais a propriedade tem o valor definido através de *owl:hasValue*;
- disjointWith: declara quais classes são disjuntas entre si. Nesse caso, pode-se inferir estados inconsistentes se indivíduos forem declarados instâncias de um conjunto de classes disjuntas, ou ainda que um indivíduo de uma certa classe não seja instância de classes declaradas disjuntas da sua classe.
- unionOf, complementOf, intersectionOf : usadas para descrever combinações booleanas de classes e restrições.

A figura ilustra 3.6 um exemplo de uso da *tag unionOf*:

```
<rdf:ID="Fruta">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#FrutaDoce" />
    <owl:Class rdf:about="#NaoFrutaDoce" />
  </owl:unionOf>
</owl:Class>
```

Figura 3.6: Exemplo de uso *unionOf*

A classe "Fruta"inclui ambas as extensões "FrutaDoce"e "NaoFrutaDoce".

Horridge [28] conceitua dois tipos de classes, sobre as quais podem ser aplicadas restrições:

a) Classes Primitivas - Uma classe é Primitiva se as restrições sobre seus indivíduos são apenas condições do tipo necessária. Condições necessárias podem ser entendidas como: se o membro é alguma coisa desta classe então é necessário que satisfaça estas condições. Diz-se que uma classe primitiva é apenas descrita. Logo, descrever uma classe (Primitiva) é declarar restrições usando condições necessárias. Por exemplo: se algo é instância da classe Cidade, então é necessário para ele ter, pelo menos, uma base do tipo CidadeBase.

xsd:string	xsd:normalizedString	xsd:boolean
xsd:decimal	xsd:float	xsd:double
xsd:integer	xsd:nonNegativeInteger	xsd:positiveInteger
xsd:nonPositiveInteger	xsd:negativeInteger	xsd:long
xsd:int	xsd:short	xsd:byte
xsd:unsignedInt	xsd:unsignedShort	xsd:unsignedByte

Tabela 3.2: Tipos de dados recomendados

b)Classes Derivadas - Uma classe é Derivada se as restrições sobre seus indivíduos são condições do tipo necessária e suficiente. No exemplo do parágrafo anterior, se a restrição é descrita como necessária e suficiente, isso implica que se um indivíduo qualquer é relacionado com a classe CidadeBase através da propriedade hasBase, então ele é uma instância da classe Cidade. Ou seja, essa condição é suficiente para inferir o seu tipo.

3.5.4.1 Propriedades e relacionamentos

Outra característica importante da linguagem é o relacionamento entre os indivíduos. Os valores de dados são descritos pelas propriedades, sendo estas divididas em duas categorias:

- datatype property : define indivíduos a valores;
- object property: associa indivíduos a individuos.

A tabela 3.2 ilustra alguns tipos recomendados pela W3C para os Datatypes.

3.6 Considerações

No presente capítulo foi apresentada a evolução das linguagens puramente sintáticas como XML, até a inclusão de vocabulários ontológicos para descrição de serviços como a OWL-S. A descrição sintática de serviços não permite a composição automática, já que não existe a descrição dos conceitos e seus relacionamentos.

A importância da web semâtica é justamente dar significados a conteúdos, isto é, descrever explicitamente pré-condições e efeitos. No entanto, vale ressaltar que sem agentes capazes de inferir ações sobre as descrições não se justifica o seu uso.

Nesse contexto, surge a necessidade de planejadores que façam uso da estrutura semântica descrita nos serviços para a criação de planos.

O próximo capítulo apresenta conceitos sobre planjedores em Inteligência Artificial, o formalismo para escrever problemas e domínio, além de uma análise específica sobre os planejadores hierárquicos.

CAPÍTULO 4

PLANEJAMENTO HIERÁRQUICO

Este capítulo descreve um referencial teórico sobre o planejamento que utiliza técnicas em inteligência artificial. A seção 4.1 apresenta definições e conceitos sobre o problema de planejamento. A seção 4.2 descreve sobre o planejamento clássico e suas restrições. A seção 4.3 é dedicado à primeira representação formal de problemas, e a seção 4.4 diz respeito à linguagem de definição de domínios (PDDL), por fim, a seção 4.6 apresenta outra abordagem de representar e implementar domínios e uma justicativa de seu uso neste trabalho.

4.1 Conceitos

O problema de planejamento define-se por, partindo-se de uma situação inicial conhecida, obter uma determinada situação objetivo, a partir da aplicação de ações de um conjunto de ações previamente definido. O problema está em descobrir quais ações devem ser aplicadas e em que ordem de execução.

Um exemplo que pode ilustrar esse problema é "assistir um filme no cinema" (4.1). A partir de uma situação inicial, onde uma pessoa está em casa e a situação final onde a pessoa está no cinema assistindo filme. Para alcançar o objetivo, pode-se agir como indicado:

- ir ao cinema;
- escolher o filme;
- comprar ingresso;
- assistir o filme.

A sequência de ações que encontra o objetivo é chamado de plano. Nota-se em um

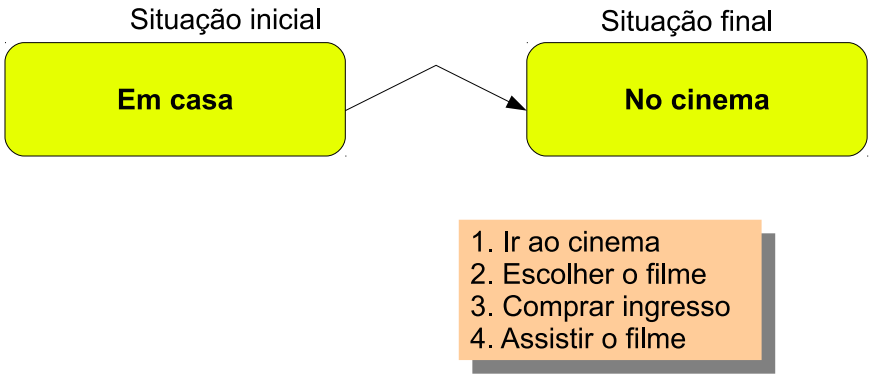

Figura 4.1: Camada da arquitetura

plano uma relação de ordem das ações que o compõem, logo considerando a ordenação das ações, pode-se classificar um plano como parcialmente ou completamente ordenado.

Um plano parcialmente ordenado é definido com base em que apenas parte das ações estão ordenadas, enquanto que o completamente ordenado é aquele em que todas as ações estão ordenadas, impondo uma sequência cronológica, conforme ilustrado na figura 4.1.

O planejamento é dividido em duas categorias, de acordo com a descrição do domínio dos problemas: planejamento clássico e não clássico. A próxima seção descreve as particularidades do planejamento clássico no qual, entre outras características os ambientes são totalmente conhecidos.

4.2 O Problema do Planejamento

Planejamento consiste em construir sequências de ações que formem um caminho entre o estado inicial e o objetivo. A sequência com que as ações serão processadas é chamada de plano.

Formalmente, o problema do planejamento é definido como $\leq$ S, S_0, G, A, T $\geq$, onde:

- S é o conjunto de todos os estados possíveis do mundo;
- S_0 denota o estado inicial do mundo;
- G denota o objetivo;
- A é o conjunto de ações que o planejador pode executar na tentativa de mudar de um estado para outro estado no mundo;
- T é a relação S $\times$ A , que define a condição e os efeitos para a execução de cada ação.

Um plano consistente é uma lista de ações que leva o mundo do estado inicial até o estado desejado. O estado inicial de um problema consiste em um conjunto de átomos que devem ser verdadeiros no mundo no começo do processo de planejamento. O objetivo é um conjunto de átomos que devem ser verdadeiros no mundo após a execução do plano.

As próximas seções descrevem linguagens formais para representar ações e estados, e a seção 4.6 apresenta planejadores que fornecem uma forma particular de especificar regras, aproveitando-se da natureza hierárquica inerente a muitos problemas.

4.3 O Formalismo STRIPS

O formalismo STRIPS foi inicialmente modelado por Fikes e Nilson [41], que definiram uma representação formal simples e um processo de busca, conhecido como sistema *STRIPS* (STanford Research Institute Problem Solver).

A simplicidade na representação *STRIPS* tornou-se a base no planejamento em Inteligência Artificial, permitindo que problemas que eram tratados através de provas de

teoremas e uso da lógica clássica[24], passaram a ser tratados como um problema de busca em um espaço de estados, ou outras técnicas tais como SAT a programação por restrições, dentre outras.

STRIPS utiliza uma representação de literais de primeira ordem, que devem ser básicos e livres de funções. Ainda, define o problema de planejamento em duas partes: a descrição do domínio e a descrição do problema.

O conjunto de literais e a descrição define a descrição do domínio. Considere o exemplo:

- uma pessoa está em casa;
- uma pessoa está sem ingresso.

O exemplo pode ser representado em *STRIPS* da seguinte forma:

casa(pessoa) $\wedge$ semingresso(pessoa)

Nota-se, que na definição de um estado, é utilizado o operador $\wedge$ para representar um conjunto de literais, definindo que tal estado só existe, se e somente se, todos os literais indicados também existirem. Ainda em *STRIPS*, três partes formam uma ação: no nome da ação uma lista de parâmetros; a pré-condição, que é uma conjunção de literais positivos que devem ser verdadeiros para que a ação possa ser executada; e o efeito, que é uma conjunção de literais positivos ou negativos que descrevem as alterações nos estados após a execução da ação.

Ainda na representação, os literais positivos são divididos em uma lista de adição (add) e os literais negativos em uma lista de eliminação (del), sendo uma ação definida então como:

nome da ação(parâmetros)

pre: lista das pré-condições

add: lista de efeitos adicionados (literais positivos)

del: lista de efeitos excluídos (literais negativos)

Por exemplo, a representação em *STRIPS* para a ação comprar ingresso pode ser descrita como:

- compraringresso(pessoa,dinheiro)
- pre: ter(dinheiro) $\wedge$ cinema(pessoa)
- add: comingresso(pessoa)
- del: semingresso(pessoa)

Destaca-se no exemplo na verdade uma descrição genérica da ação, denominada esquema da ação, que será instanciada pelas constantes do problema, gerando várias outras ações.

Literais não mencionados são assumidos como falsos (hipótese do mundo fechado [45]), logo para representar o estado inicial, todos os literais positivos devem ser listados, e literais negativos podem ser retirados da descrição inicial. Logo, nota-se que a proposta do STRIPS é a representação simples de ações e estados.

A grande vantagem nessa abordagem está na independência entre a linguagem que descreve o problema e o algorítimo que resolve o problema. No entanto, o benefício da independência gera custos como: não suporte a predicados de igualdade, não ser lógica [2] nem tipada.

Para contornar esses empecilhos, na próxima seção é descrita a linguagem PDDL com intuito de oferecer mais suporte e recurso para descrição de problemas de planejamento.

4.4 Linguagem PDDL

Na tentativa de fornecer um padrão para declaração de problemas e domínios, fornecendo uma comparação precisa entre diferentes planejadores, surgiu em 1998 a linguagem PDDL [38] .

A linguagem PDDL é uma combinação de outras duas linguagens, STRIPS e ADL[17], desenvolvida especialmente para o primeira competição de planejadores[1]. Entre suas características destacam-se efeitos condicionais, quantificação universal em universos dinâmicos, axiomas de domínios, ações hierárquicas e múltiplos problemas em múltiplos domínios.

[1]Congresso internacional AIPS (Artificial Intelligence Planning and Scheduling Systems)

Algumas das características são descritas abaixo:

- a quantificação universal permite aplicar uma ação a um conjunto de objetos presentes em um determinado ambiente;
- axiomas de domínio permitem derivar relações entre literais de acordo com a ocorrências de determinadas ações, ou seja, os axiomas permitem deduzir a partir da descrição do estado vigente deduzir novas informações;
- ações hierárquicas é uma funcionalidade onde o objetivo é conseguir compor ações a partir de ações primitivas disponíveis. Os planejadores SHOP [42] e JSHOP [35] utilizam esse recurso.

A estrutura da linguagem PDDL é ilustrada na figura 4.2. São necessário dois arquivos, no arquivo domínio são especificados o nome do domínio, os requistos da linguagem, os predicados e as ações disponíveis.

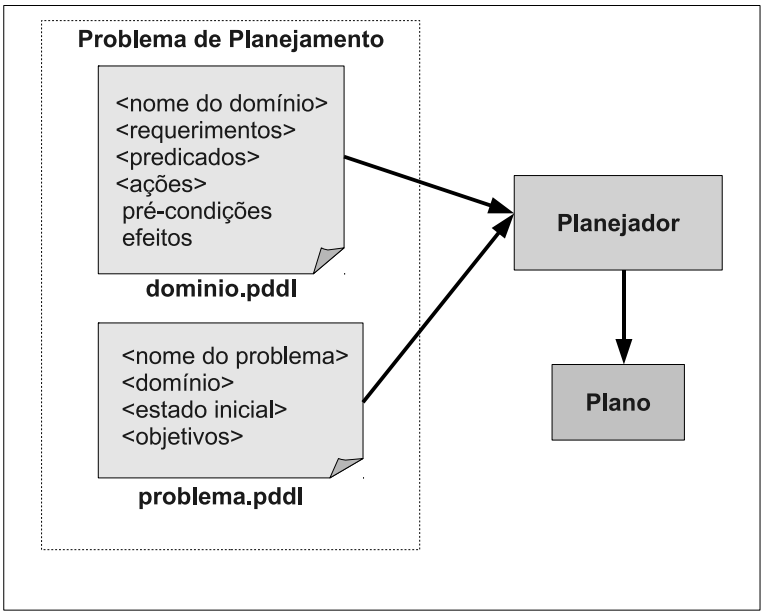

Figura 4.2: Estrutura de um problema de planejamento em PDDL

O arquivo para descrever um problema é apresentado na figura 4.3, contém o nome do problema, o domínio. Além disso contém o estado inicial e os objetivos.

```
(define (problem wb)
    (:domain logistics-typed)
    (:objects
        b1 - BLOCK
        b2 - BLOCK
    )
    (:init
        (empty)
        (ontable b2)
        (on b1 b2)
        (clear b1)
    )
    (:goal (and
        (empty)
        (ontable b1)
        (on b2 b1)
        (clear b2)
    ))
)
```

Figura 4.3: Exemplo de descrição de problema em PDDL

O arquivo para descrever um domínio referente ao exemplo do problema é apresentado na figura 4.4 , contém os predicados e ações. Para cada ação são descritos os parâmetros, pré-condições e efeitos.

A próxima seção descreve as restrições para o problema da composição utilizando o planejamento clássico.

4.5 Restrições em planejamento clássico

Esta seção apresenta algumas restrições em relação ao domínio, problema e planejador para a abordagem clássica [22]. Essas restrições justificam o uso de outra abordagem na forma de tratar o probela para compor serviços na web.

Alguma dessas restrições:

- finito: o ambiente deve possuir um conjunto finito de estados;
- determinístico: se uma opção é aplicável a um estado, ela resulta em um único outro estado;
- objetivos restritos: os objetivos são especificados como um estado objetivo explícito

```
(define (domain logistics-typed)
  (:requirements :strips :typing)
  (:types BLOCK)

  (:predicates
(holding ?b)
(empty)
(on ?b1 ?b2)
                (ontable ?b)
                (clear ?b)
  )
  (:action PICKUP
:parameters
 (?b1 - BLOCK
  ?b2 - BLOCK)
:precondition
(and  (empty)
(clear ?b1)
                        (on ?b1 ?b2)
                 )
:effect
(and  (holding ?b1)
                        (clear ?b2)
(not (empty))
                        (not (on ?b1 ?b2))
                        (not (clear ?b1))
                 )) ...
)
```

Figura 4.4: Exemplo de descrição de domínio em PDDL

ou como um conjunto de estados objetivo;

- planos sequenciais: a solução para o problema deve ser uma sequência de ações finita e linearmente ordenada;

- ações discretas: as ações não têm duração, ou seja, são transações de estados instantâneos.

O problema de compor serviços web se depara com situações imprevisíveis decorrente de fatores externos, tais como: tem serviços não disponíveis; planos não sequenciais de execução; um serviço pode não ser executado imediatamente devido a algum problema de rede; o conjunto de serviços pode produzir objetivos múltiplos.

Esses fatores inviabilizam a utilização de planejadores clássicos, sendo necessário um planejador que permita mapaear problemas inerente ao contexto da Internet.

A próxima seção apresenta outra técnica para representar problemas de planejamento,

baseados em redes de tarefas.

4.6 Redes de tarefas hierárquicas (HTN)

A abordagem do planejamento baseado em redes de tarefas se dá através da abstração na qual a dependência das ações podem ser dada na forma de rede.

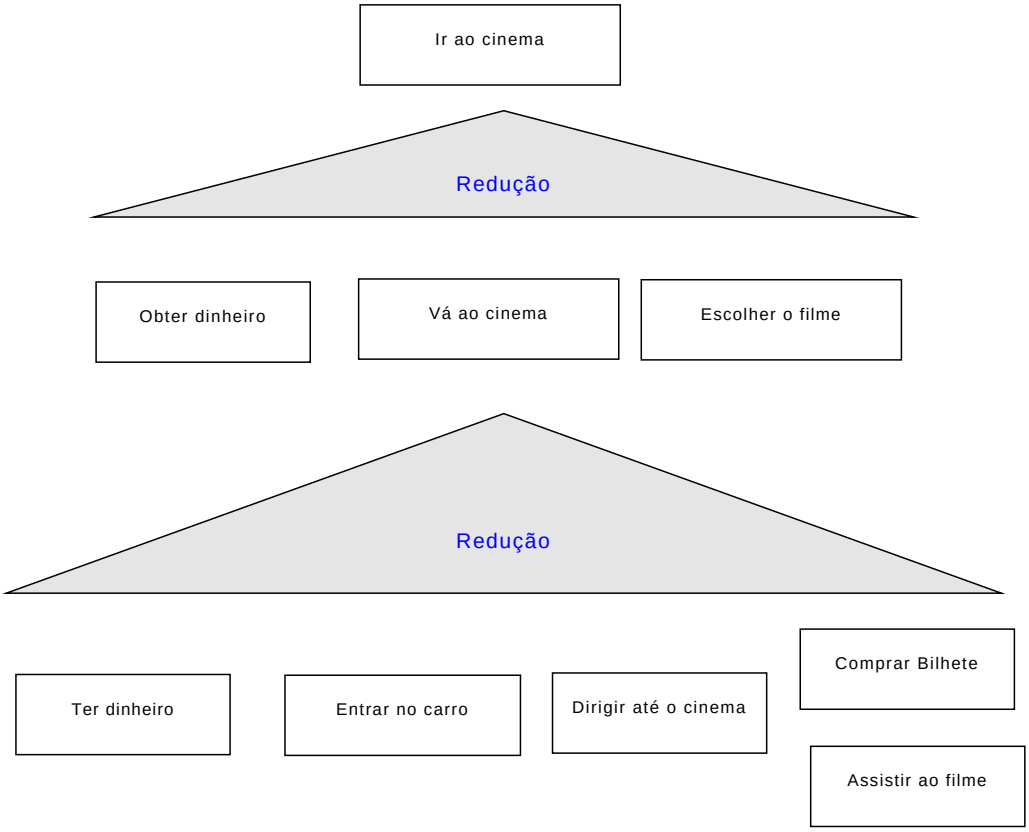

Figura 4.5: Exemplo de decomposição de tarefa composta

O objetivo do planejamento que utiliza essa técnica é produzir uma sequência de ações que executem algumas atividades ou tarefas [18]. A figura 4.5 ilutra a tarefa “Ir ao cinema”, é reduzida para uma lista de tarefas “Obter dinheiro”, “va ao cinema”e “Escolher o filme”, e logo depois é mais uma vez reduzida a “Ter dinheiro”, “Entrar no carro”, “Dirigir até o cinema”, “Comprar bilhete”e “Assistir ao filme”A partir de um objetivo abstrato, decompor em ações primitivas.

A diferença entre um planejador clássico e hierárquico está justamente relacionada ao objetivo, enquanto o primeiro busca alcançar um estado meta, o segundo realiza um

conjunto de tarefas, decompondo em sub-tarefas, através de métodos, até que a rede seja composta somente de ações primitivas, que são as ações executadas de fato.

Os planejadores hierárquicos utilizam métodos para reduzir as tarefas que estão no maior nível de abstração. São especificadas as possiveis reduções das tarefas para conjuntos de zero ou mais tarefas abstratas ou primitivas. A realização das tarefas através dos operadores tem o mesmo efeito da instanciação das ações de um plano na abordagem clássica, além disso, outra caracterítica semelhante é quando o processo termina, ou seja, quando existir no plano apenas ações ou tarefas primitivas que possam ser executadas. O planejador HTN [19] procura realizar tarefas, decompondo e resolvendo conflitos entre elas. Ainda, no HTN as ações são chamadas de tarefas primitivas, representando uma passagem de um conjunto de estados para o outro.

4.6.1 Métodos

Os métodos são de extrema importância no planejamento em redes hierárquicas, pois eles são os meios pelos quais se dá a redução das tarefas. Formalmente um método tem a forma (h,T) onde:

- h é uma tarefa não primitiva;
- T é uma lista de tarefas que realiza h.

A definição formal para uma rede de tarefas é descrita como $[(n_1{:}h_1),...,(n_N{:}H_1n)$,F], onde:

- cada h_i é uma tarefa não primitiva;
- n_i é um rótulo que distingue determinada instância de h_n de qualquer outra ocorrência dessa tarefa na rede;
- F é uma fórmula booleana constituída da ligação de restrições sobre variáveis.

De acordo com Erol, Hendler e Nau [18], as tarefas podem ser divididas em três tipos:

- tarefas primitivas: são diretamente executadas pela execução de uma ação (Ex: comprar bilhete);
- tarefas de objetivo: são tarefas que deve ser verdadeiras no mundo (Ex: ter dinheiro);
- tarefas compostas: significam o "desejo de mudança", formada por vários objetivos e tarefas primitivas (Ex: ir ao cinema).

Nota-se a diferença entre as tarefas de objetivo e composta, justamente pela complexidade na realização da tarefa. A primeira basta uma ação atômica "trabalhar esse mês", a segunda ação seria necessária várias ações tais como "ter dinheiro", "entrar no carro", "dirigir até o cinema", etc...

4.6.2 O planejador Shop2

A família *Simple Hierarchical Ordered Planner* (SHOP)[42] de planejadores permite planejamento hierárquico utilizando o STN[2] em ordem total (SHOP) ou em ordem parcial (SHOP2) [43]. , em que a abordagem está em planejar as ações na ordem em que serão executadas (da esquerda para direira), relaxando a restrição da ordenação de tarefas [43]. Permite a decomposição de uma tarefa em um conjunto parcialmente ordenado de subtarefas.

Uma grande vantagem desse algorítimo está na capacidade de manipular domínios de planejamento que envolvam tempo. Como os operadores do SHOP2 permitem atribuir valores a variáveis, é possível fazer operações aritméticas, sendo possível manter uma informação temporal.

O estado em SHOP2 é um conjunto de átomos lógicos instanciados e um conjunto de axiomas. A definição da sintaxe da estrutura é definido assim:

- variáveis: pode ser qualquer símbolo cujo nome começe com ponto de interrogação;
- tarefa primitiva: qualquer símbolo cujo nome começe com ponto de exclamação;

[2]Rede Simples de Tarefas (*Simple Task Network* (STN) [42] [22] é versão simplificada de planejamento hierárquico.

- constante, predicado, função e tarefa composta: qualquer símbolo cujo nome comece com uma letra ou sublinhado.

As váriaves em SHOP2 podem guardar valores ou objetos. A atribuição tem a forma (assign VT), onde:

- V é uma variável;
- T é termo.

O objetivo da atribuíção é ligar o valor de T à variável V. Uma tarefa possui a descrição no formato ([: immediate] s_1 τ_2 τ_3...τ_n) onde:

- S representa o nome da tarefa;
- τ_n são seus argumentos definidos como termos.

JSHOP2 [29] é uma re-implementação em Java de SHOP2 e a principal diferença é uma fase adicional que, a partir da descrição do domínio, gera código Java de um planejador especializado, dependente de domínio, tirando proveitos de otimizações feitas pelo próprio compilador Java.

4.7 Considerações

Esta capítulo apresentou definições básicas sobre planejamento, as linguagens que representam domínios e problemas, além das abordagens classicas e hierárquicas. Apresentou também as restrições recorrentes da abordagem clássica e a difiuldade em representar problemas de composição de serviços utilizando essa abordagem.

Logo, para que os planejadores possam surtir efeito satisfatório utilizando cenários reais, a abordagem do planejamento hierárquico foi escolhida por possuir maior porder de expressividade, aliado aos problemas inerentes ao contexto Web. Outra justificativa da escolha do planjedor SHOP2 está na descrição dos serviços utilizando semântica, através da OWL [3] [5], é possível em consonância com o planejador hierarquico SHOP2 planos satisfatórios, conforme demonstrado em [52].

O próximo capítulo mostra a implementação do ambiente para testes de composição fim a fim, utilizando semântica nas descrição dos serviços e a implementação do planejador JSHOP2.

CAPÍTULO 5

DESENVOLVIMENTO DE UM SISTEMA PARA COMPOSIÇÃO DE SERVIÇOS

Neste capítulo será apresentado o SCS (Sistema de Composição de Serviços), o qual propõe otimizar e diminuir esforços para testes em composição de serviços web, utilizando planejamento em inteligência artificial, mais especificamente o planejador JSHOP2 e semântica na descrição dos serviços.

A seção 5.1 situa o problema da composição do serviço, identificando as etapas para sua realização, a seção 5.2 descreve a arquitetura da implementação, as seções 5.2.1 5.2.3 apresentam os serviços semânticos bem como o desenvolvimento do banco de dados. Na seção 5.3 é apresentado camada de controle e interface do protótipo. Na seção 5.4 descreve a integração com o planejador e finalmente na seção 5.5 os experimentos realizados a partir de domínios de serviços compostos.

5.1 O problema

Assim como o ciclo de vida de um serviço web, a composição de serviços na Internet pode ser automatizada sob diversos aspectos, busca, seleção e composição.

Nosso experimento foi projetado para automatizar a fase da composição, fazendo uso de um planejador para criar o plano com as sequências dos serviços a serem executados.

Nas pesquisas foram encontradas alguns sistemas que utilizaram abordagem semelhante, no entanto, as implementações desses mostraram não ser completas, em relação a organização dos serviços, leitura dos documentos OWL e integração com o planejador.

O problema em compor serviços já descobertos implica em buscar a URI de cada OWL, ler as pré-condições e efeitos de cada serviço descrito no documento. Além disso deve ser criado a partir da referida leitura, arquivos com a descrição do domínio e do problema, esse último, com base na escolhas do usuário.

Nossa implementação foi projetada baseada em camadas para efetivar cada etapa da composição apresentada.

A próxima seção descreve a arquitetura do protótipo.

5.2 Arquitetura proposta

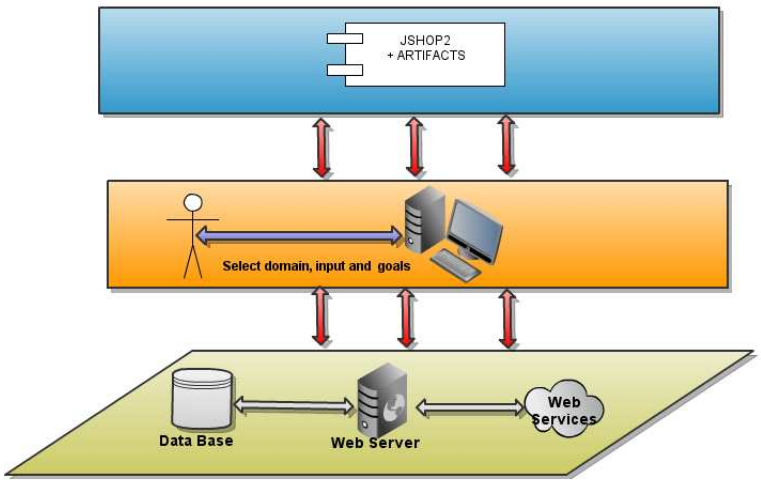

Figura 5.1: Camada da arquitetura

A plataforma proposta para a composição de serviços web foi projetada para facilitar e simplificar o planejamento de workflows, permitindo a reutilização e flexibilização na adição de novos domínios e serviços escritos em WSDL e OWL-S. A figura 5.1 ilustra as camadas na arquitetura.

A camada interior foi proposta no sentido de permitir a inclusão de outros domínios e serviços OWL-S, através de um banco de dados que relaciona serviços a um determinado domínio. Outra possibilidade ainda relacionada a essa camada é que os serviços podem estar hospedados na Internet ou qualquer servidor web local.

A próxima camada apresenta uma interface para seleção dos domínios inseridos, permitindo ao usuário a escolha das entradas e saídas (internamente as entradas são tratadas como pré-condições e as saídas como efeitos) para que o sistema realize a integração com o planejador JSHOP2, representado na camada do topo.

Ainda descrevendo a estrutura como um todo do sistema, a figura 5.2 ilustra os principais requisitos funcionais da implementação. Como o objetivo é a funcionalidade de um sistema de testes para compor serviços utilizando a semântica, o protótipo implementado deve permitir a inclusão de domínios e ontologias.

Ainda na figura é possível descrever as especificidades que precedem a integração com o planejador. O sistema tem como requisito funcional a geração de documentos escritos em PDDL que descrevam o domínio e o problema, arquivos esses que servirão como entrada para o planejador.

Finalmente, o sistema implementado tem como funcionalidade final a invocação e a exibição do plano, como base nas escolhas do usuário.

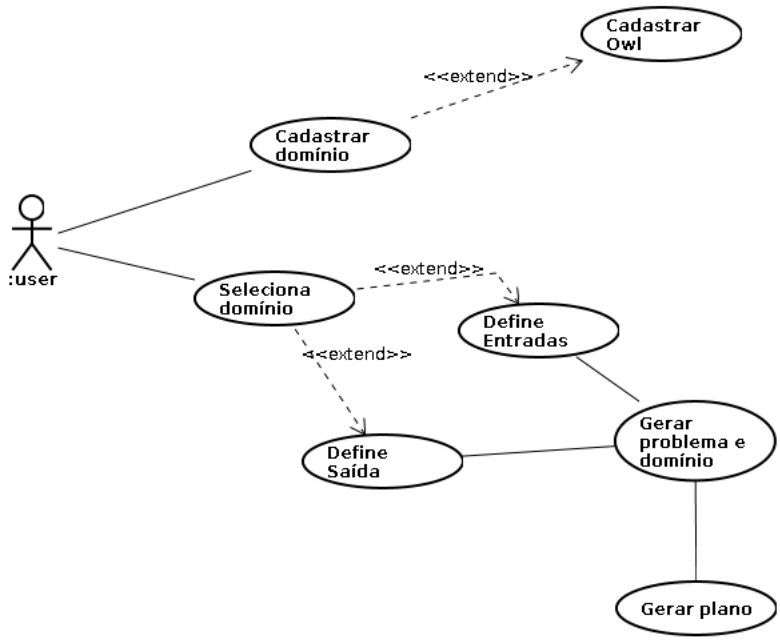

Figura 5.2: Use case do protótipo

As próximas seções descrevem cada camada da implementação utilizada na arquitetura.

5.2.1 Serviços semânticos

A primeira implementação foi no sentido de encontrar serviços escritos em WSDL que tenham seu documento correspondente descrito semanticamente em OWL-S. Na pesquisa

optou-se pela escolha do projeto OWLS-TC[1] mostrou ser uma boa alternativa por possuir especificações de domínios e ontologias já implementadas, além de utilizar a versão 1.1 da OWL que permite mais recursos semânticos na descrição dos serviços.

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<rdf:RDF  xmlns:owl       = "http://www.w3.org/2002/07/owl#"
...
<profile:textDescription xml:lang="en">
This service returns author of the given book.
</profile:textDescription>
<profile:hasInput  rdf:resource="#_BOOK"/>
<profile:hasOutput rdf:resource="#_AUTHOR"/>
<profile:has_process rdf:resource="BOOK_AUTHOR_PROCESS" />
</profile:Profile>
```

Figura 5.3: Trecho de código OWL-S

A figura 5.3 mostra um trecho de código implementado em linguagem ontológica: pode-se perceber no código informações como condições e efeitos descritos no documento como “hasInput” e “hasOutPut” além dos recursos usados “service” e a descrição do serviço “profile:textDescription”.

5.2.2 Funcionalidades

A partir dessa etapa, o trabalho saiu do campo das configurações e ajustes para configurar o planejador para a programação.

A figura 5.4 ilustra do diagrama de pacotes implementado para contemplar cada funcionalidade do sistema proposto. Faz-se necessário descrever que a construção dos pacotes utilizada seguiu como abordagem o padrão MVC (Model-View-Control).

Na implementação a camada modelo é responsável pelas classes que mapeiam as tabelas no banco de dados, mais especificamente para representar os domínios e os Owls. A camada de controle integra o planejador JSHOP2 ao sistema, sendo responsável pelas interação com o usuário e a criação de artefatos para a chamado do planejador.

[1]O projeto OWLS-TC é mantido pelo pesquisador Matthias Klusch disponível em `http://fusion.cs.uni-jena.de/opossum/index.php` que e atualmente encontra-se na versão 3.0.

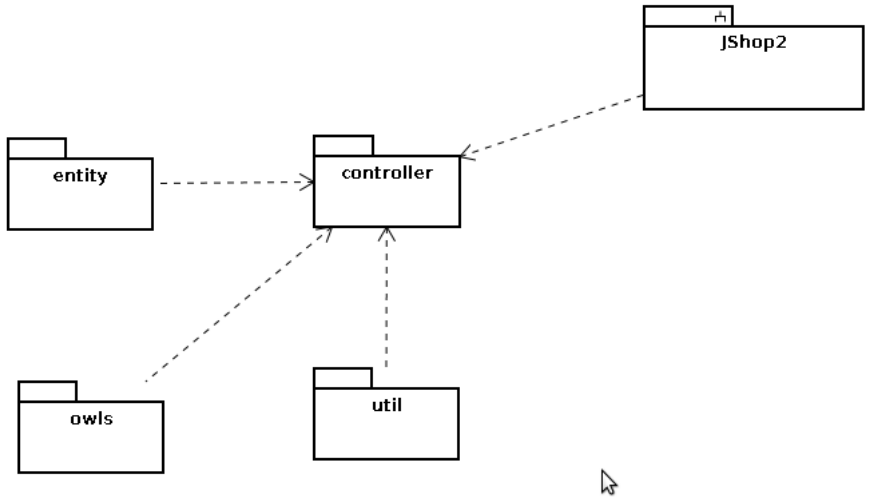

Figura 5.4: Diagrama de pacotes da implementação

Finalmente, a camada de visão é responsável pela interface do sistema e a atualização das ações correspondentes a essas interações.

A seguir a descrição dos pacotes e suas respectivas classes e funcionalidades:

- entity: esse pacote contém as classes que mapeiam as tabelas, mais detalhada na seção 5.2.3. Representa basicamente na forma relacional os domínios e os Owls. Essa estratégia permite que o usuário já estabeleça os domínios e uma coleção de Owls, facilitando a inclusão e leitura dos documento na integração do planejador.

- controller: contem as classes que geram artefatos para o planejador, problema e domínio respectivamente. Possui ainda uma classe para conectar com o banco, mantendo um pool de conexão afim de otimizar a performance. Ainda contém uma classe responsável pela interação com o usuário. Essa classe armazena as ações executadas pelo usuário, atualizando o sistema (ver seção 5.3 .

- owls : possui classes responsáveis pela leitura de pré-condições e efeitos em documentos Owls. Essas classes utilizam como dependência a api OWL-S [2]

- util: contem uma classe com funcionalidades genéricas. Desde a recuperação do caminho absoluto do servidor até gerenciamento de mensagens de erros.

[2]A biblioteca OWL-S é um framework que facilita a leitura e escrita para descrever serviços semânticos disponível em `http://www.mindswap.org/2004/owl-s/api`

5.2.3 Implementação do banco de dados

Esta seção apresenta nossa implementação das classes para o banco de dados. A abordagem utilizada foi o uso de um framework de mapeamento objeto-relacional, o JPA [30] (Java Persistence API). A implementação utilizada na Classe "Dominio" é ilustrada na figura 5.5, utilizando o framework JPA.

```
@Entity
@Table(name = "Dominio")
public class Dominio implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "idDom")
    private Integer idDom;
    @Basic(optional = false)
    @Column(name = "nome")
    private String nome;
    @Column(name = "descricao")
    private String descricao;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "idDom")
    private List<Ontology> ontologyList;

    public Dominio() {
    }
```

Figura 5.5: Código da classe Domínio

As primeiras linhas com "@Entity" e "@Table" informam ao framework que a classe em questão é uma entidade e que está mapeada para a tabela "Dominio". Em seguida à assinatura da classe, temos a declaração dos atributos que possuem, logo acima da sua assinatura a descrição "@Column" indicando um mapeamento direto entre o atributo da classe e a coluna da tabela.

É importante destacar a técnica utilizada na classe "Dominio" em relação ao atributo ontologyList. Ao utilizar o framework JPA, conseguimos encapsular a lógica de seleção para as classes referenciadas, assim, imaginando o cenário para o domínio "e-book", e três serviços com esse domínio uma simples invocação ao método "getOntologyList()" recupera todos os serviços do domínio selecionado.

Do mesmo modo a implementação utilizada na classe “Ontology” 5.6 , foi mapear para classe pai “Dominio”.

```
@Entity
@Table(name = "ontology")
public class Ontology implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "idOnt")
    private Integer idOnt;
    @Basic(optional = false)
    @Column(name = "owl")
    private String owl;
    @Column(name = "wsdl")
    private String wsdl;
    @JoinColumn(name = "idDom", referencedColumnName = "idDom")
    @ManyToOne(optional = false)
    private Dominio idDom;
```

Figura 5.6: Código da classe Ontology

As diferenças da implementação aqui estão nas linhas com os códigos “@JoinColumn” e “@ManyToOne” indicando respectivamente que essa classe possui uma relação de um para muitos, e que foi mapeada na classe Dominio. Essa implementação utilizada permite reutilização de funções conforme descrito posteriormente na camada de controle.

As classes entidades do sistema bem como a lógica de acesso estão representadas no diagrama de classes 5.7.

5.3 Controle e interface

Essa etapa se inicia com o desenvolvimento da interface entre o sistema de integração e o usuário. Para isso, observou-se em alguns trabalhos ([35] e [32]) o papel da interface na dinâmica da composição. Após esse processo foram indicados alguns pré-requisitos de interface listados a seguir:

- clareza: A interface deverá exibir somente informações necessárias na realização do planejamento;

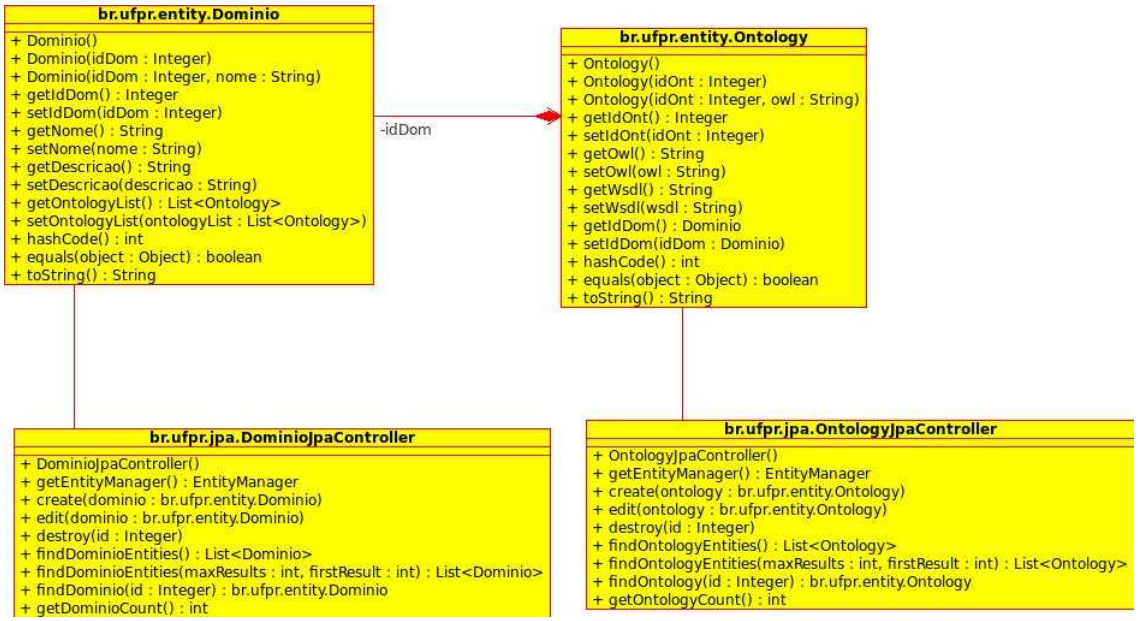

Figura 5.7: Diagrama de classes do protótipo

- usabilidade: O usuário deverá influenciar o fluxo de processamento selecionando somente as informações pertinentes à execução quando estas forem carregadas;
- tempo de resposta: A interface deverá exigir o mínimo de recurso na tentativa de diminuir o tempo de espera do processamento.

Foi utilizado na implementação da interfaces a linguagem Java Server Faces (JSF) [30], que fornece separação de funções, quesito importante na construção de aplicações web. Um trecho da implementação utilizada na página principal do sistema é ilustrada em 5.8.

Nas quatro primeiras linhas temos a definição das bibliotecas necessárias para a utilização dos componentes em JSF. Logo após, podemos perceber que a construção dos componentes que serão renderizados em qualquer navegador web segue um padrão, descrito a seguir:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
<%@ taglib uri="http://richfaces.org/a4j" prefix="a4j"%>
<%@ taglib uri="http://richfaces.org/rich" prefix="rich"%>
<f:view>
    <a4j:form>
        <rich:panel header="Composição de serviços web com JSHOP2"
  style="width: 100%; heigth:100%">
            <h:panelGrid columns="4">
            <h:outputText value="Selecione o dominio: " />
            <h:selectOneMenu  id="dominio" value="#{MBPlanner.dominio}"
    valueChangeListener="#{MBPlanner.selecaoDominio}" >
                <f:selectItems value="#{MBPlanner.selectDominio}" />
            </h:selectOneMenu>
            <h:commandButton actionListener="#{MBPlanner.lerOwl}" value="Ler" />
            <h:commandButton actionListener="#{MBPlanner.reset}"
value="Limpar estados" />
            </h:panelGrid>
```

Figura 5.8: Código em Java Servers Faces da página inicial

- Inicia sempre com uma letra que representa qual biblioteca utilizada (h, f, *rich* e *a4j*);
- A descrição do tipo do componente. Exemplo: outputTex = texto de saída / commadButton = Botão de ação;
- Um conjunto de opções que seriam parâmetros necessários para o funcionamento dos campos como valor de carga (value) e qual ação instanciada em caso de interação com o usuário (actionListener).

A técnica utilizada permite o desenvolvimento por camadas, separando funções do sistema. No exemplo da listagem, a codificação da tela só será responsável pela renderização dos componentes do sistema, cabendo a uma outra camada a interação com o usuário.

O fluxo de informações revela quais são os passos entre a aplicação e a ação do usuário.

Ao iniciar a aplicação, conforme lustrado na figura 5.9, são mostrados todos os domínios persistidos no banco. A ação de clique no botão “carregar” realiza a leitura dos atributos em todos os documentos owls relacionado ao domínio selecionado.

Em 2 e 3 na figura 5.9 as entradas e saídas são exibidas. Os formatos dos campos satisfazem os requisitos de clareza e usabilidade, pois o sistema permite a múltipla seleção

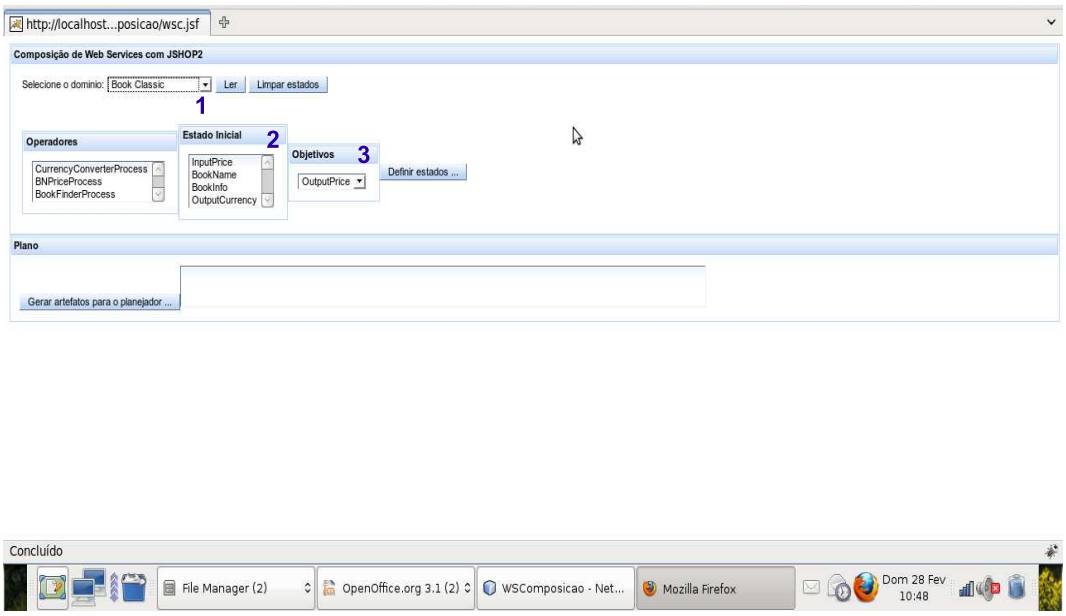

Figura 5.9: Interação do usuário com a tela

em ambos os casos, de entradas e saídas.

Na sequência, o usuário deve selecionar o botão "gerar artetafos", para que o planejador possa ser executado.

A figura 5.10 ilustra um diagrama de sequências dos objetos envolvidos na escolha das entradas e definições do objetivo. A abordagem utilizada foi coerente com a utilização do banco dos dados, no entanto, o que muda agora é que a escolha de um domínio em específico acarreta no carregamento de uma coleção de objetos Owls.

Esses objetos mapeiam os atributos e informa ainda os operadores e as opções de parâmetros para definir as entradas dos sistemas para criação dos artefatos, problema e domínio.

Na próxima seção descreve-se como foi realizada a integração da aplicação com o planejador hierárquico JSHOP2.

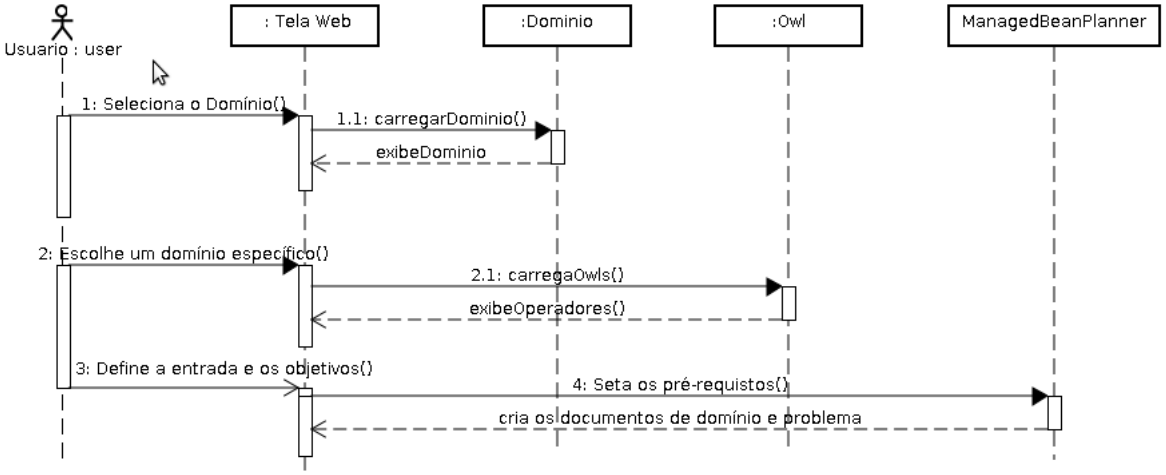

Figura 5.10: Sequência de eventos dos objetos relacionados com a intervenção do usuário

5.4 Integração com o planejador

Como o autor[3] disponibiliza o código fonte do planejador JSHOP2 [35], a abordagem utilizada foi utilizá-lo como dependência do projeto. O planejador é instanciado passando os arquivos necessários para a geração do plano. O algoritmo 1 ilustra a implementação para criar os arquivos de entrada do planejador, tendo como referência a escolha do usuário.

input : D dominio list de owl
output: List<Input> and List<Goals>

if *D is not empty* **then**
  foreach *Cada elemento owl in D* **do**)
  List<Input> ← owl.hasInput;
  List<Goals> ← owl.hasOutput;
end

Algorithm 1: Ler Domínio

Tem-se como entrada um domínio selecionado pelo usuário contendo uma lista de endereços OWL-S. Após verificar se o domínio não é uma lista vazia, percorre-se cada documento OWL lendo os atributos de entrada e objetivos e salvando-os em uma lista.

Após esse processo, um arquivo deve ser escrito dois arquivos em PDDL, respeitando

[3]O autor Okhtay lghami disponibiliza o projeto em `http://sourceforge.net/projects/shop`.

o formalismo do planejador JSHOP2. Um arquivo é a descrição do problema, conforme ilustrado na figura 5.11. São descritos neste arquivo os operadores e objetivos através do operador "achieve-goals".

```
(defproblem problem domain
  (
    (InputPrice)
    (BookInfo)
  )
 (
    (achieve-goals ((BookPrice) ((OutputCurrency))))
  )
)
```

Figura 5.11: Descrição de um problema

O próximo arquivo a gerado descreve o domínio, conforme ilustrado no figura 5.12.

Os dois exemplos de arquivos mostrados (5.12 e 5.11) são descritos em PDDL, respeitando algumas convenções específicas do planejador JSHOP2[4] como, a descrição dos axiomas e a palavra reservada "defdomain" para definir o domínio.

A próxima implementação foi integrar o planejador JSHOP2 ao projeto. A técnica utilizada foi a compilação dinâmica de classes com a utilização do compilador versão 6 (seis) da máquina virtual do Java [50]. A figura 5.13 ilustra a integração escrita em linguagem Java.

Através do método getSystemJavaCompiler() da classe ToolProvider, consegue-se invocar o compilador passando como último argumento os arquivos necessários para o planejador. O planejador é invocado em "JSHOP2.InternalDomain.main(value);".

Nesse momento ele cria arquivos .java necessários, com base na descrição do domínio e do problema, para a compilação e exibição do plano. A figura 5.14 ilustra o código gerado a partir dessa compilação.

Para cada operador e pré-condição é criada uma sub-classe contendo os métodos e atributos necessários para a geração do plano, selecionado a partir das preferências do usuário.

[4]A documentação do JSHOP2 é disponibilizada em [35]

```
(defdomain domain_generated
(
    (:operator (!BNPriceProcess)
      ( (BookInfo)      )
      ( (BookInfo)      )
      ( (BookPrice)     )
    )
    (:operator (!CurrencyConverterProcess)
      ( (InputPrice)  (OutputCurrency)      )
      ( (InputPrice)  (OutputCurrency)      )
      ( (OutputPrice)                       )
    )

(:operator (!BookFinderProcess)
      ( (BookName)  )
      ( (BookName)  )
      ( (Book )))
;Axiomas
(:- (InputPrice)
((BookPrice)))
```

Figura 5.12: Domínio gerado a partir das escolhas do usuário.

```
(i)   JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
(ii)  String value = new String[] {CAMINHO_SERVIDOR};
(iii) options = new String[] {"domain_generated"};
(iv)  JSHOP2.InternalDomain.main(value);
(v)   value = new String[] {"-ra","problem"};
(vi)  JSHOP2.InternalDomain.main(value);

(vii)  int res = compiler.run(null, null, null, value);
```

Figura 5.13: Compilação dinâmica utilizando Java na versão 6.0

5.4.1 Geração do plano

Após a implementação do ambiente para serviços já implementados em OWL-S com seu correspondente em WSDL, a criação de uma estrutura de banco de dados para mapear esses serviços agregando em forma de domínios, implementar a camada de interface e controle, o próximo passo foi a chamada para efetivamente ocorrer a composição.

Como o planejador JSHOP2 necessita como entrada os arquivos de problema e domínio, os quais devem ser escritos em PDDL com base nas seleções do usuário, [5] basta invocar através do próprio sistema a classe principal do planejador que crie os planos.

[5]Essa operação foi realizada utilizando a abordagem em 5.13

```
import JSHOP2.*;
class Operator0 extends Operator
{
public Operator0()
{
super(new Predicate(0, 0, TermList.NIL), -1, -1, new TermNumber(1.0));
DelAddElement[] delIn = new DelAddElement[1];
delIn[0] = new DelAddAtomic(new Predicate(0, 0, TermList.NIL));
setDel(delIn);
DelAddElement[] addIn = new DelAddElement[1];
addIn[0] = new DelAddAtomic(new Predicate(1, 0, TermList.NIL));
setAdd(addIn);
}

public Precondition getIterator(Term[] unifier, int which)
{
Precondition p;
p = (new PreconditionAtomic(new Predicate(0, 0, TermList.NIL), unifier)).setComparator(null);
p.reset();
return p;

} ...
```

Figura 5.14: Trecho de código java referente ao domínio

O algorimo 2 implementado realiza essa operação. Basicamente o algorítimo consiste

input : Gui classe principal do planejador JSHOP2, List<Efeitos>, List<Condições>
output: List<Planos>

if List<Efeitos> *is not empty and* List<Condições> *not empty* **then**
foreach *Cada elemento p in Gui* **do**)
List<Planos> $\leftarrow$ $p.servicePlan()$;
end

Algorithm 2: Gerar Plano

em verificar se existe as pré-condições e efeitos para só então recuperar dentro da classe principal uma lista de planos criados.

A classe principal, assim como a construção das classes problema e domínio, é compilada e invocada dinamicamente.

A próxima seção descreve os experimentos realizados com o protótipo implementado neste trabalho.

5.5 Experimentos

O protótipo implementado neste trabalho, o SCS (Sistema de Composição de Serviços) desenvolvido para validar a proposta de testar composição de serviços web a partir da descrição semântica em documentos OWL, utilizando o planejador hierárquico JSHOP2. A presente seção descreve cada domínio, suas características e os resultados em comparativo com outras aplicações, JSHOP2GUI [35], Transplan [49] e OWL-S Plan2 [32]. .

Os domínios utilizados foram os do projeto OWLS-TC [1] mantido pelo pesquisador Matthias Klusch. O projeto mantém uma quantidade de serviços já implementados com WSDL e descritos semanticamente com OWL.

A seguir a descrição dos domínios:

- e-book: é um domínio clássico na composição de serviços composto por três serviços que realizam operações de converter entre diferente formatos de moedas, encontrar uma publicação com base no número ISBN e encontrar o preço ou informação tendo como parâmetros o nome ou uma informação sobre obra. Para esse domínio os testes foram executados apontandos para urls externas;

- E-commerce: é uma versão mais complexa do domínio e-book. Além dos serviços já citados, possui serviços de verificação do cartão de crédito, caso o objetivo selecionado seja a efetivação da venda, serviço de autenticação do usuário e síntese da obra selecionada;

- Science Fiction Book: é um domínio que possui 9 (nove) operadores, entre eles operadores compostos por entradas múltiplas (autor e nome da obra), seis estados iniciais e oito objetivos como recomendação de preço e editora.

A figura 5.15 ilustra um arquivo em PDDL descrevendo um problema do domínio "Science Fiction Book" gerado pela implementação a partir do escolhas do usuário.

Logo após a assinatura do problema, estão as entradas definidas pelo usuário, "SCIENCEFICTIONBOOK" e "USER". Em seguida é descrito os objetivos que o usuário

```
(defproblem problem sciencebook
   (
    (_SCIENCEFICTIONBOOK)
    (_USER)
   )

 (
    (achieve-goals ((_AUTHOR)
    (_PRICE)
    (_REVIEW)
    (_PUBLIHSER)
    (_RECOMMENDEDPRICE))
    )
)

)
```

Figura 5.15: Código de um arquivo problema do domínio Science Fiction Book

definiu na sistema. Para esse domínio em específico, o usuário deseja uma lista de argumentos como autor, preço, revisão, editora e recomendação de preço. Esses objetivos estão representados pelas variáveis após a palavra "achieve-goals".

O plano gerado para esse problema está representado em 5.16.

São mostradas informações como o custo na execução do plano, os objetivos e quais os processos necessários para que a partir das escolhas do usuário chegue a um objetivo, além do tempo gasto para composição do plano.

O protótipo SCS demonstrou ser funcional ao executar todos os domínios levando em consideração o local de hospedagem. Enquanto os domínios "E-commerce" e "Science Fiction Book" estavam hospedados em um servidor web local, o domínio"e-book " estava hospedado em um servidor externo.

Na tabela 5.1 estão os principais aspectos de implementação em comparação com os trabalhos JSHOP2GUI [35], Transplan [49] e OWL-S XPlan2 [32]. Todos os trabalhos implementaram protótipos para compor serviços utilizando semântica em conjunto com planejador hierárquico, exceto o OWL-S XPlan2 que apresenta um planejador híbrido, hierárquico combinado com o planejador FastFoward[26].

A primeira consideração sobre outros sistemas de composição é o quesito fim-a-fim.

```
1 plan(s) were found:

Plan #1:
Plan cost: 1.0

(!!assert (goal (_author)))
(!!assert (goal (_price)))
(!!assert (goal (_review)))
(!!assert (goal (_publihser)))
(!!assert (goal (_recommendedprice)))
(!sciencefictionbook_authorprice_process)
--------------------

Time Used = 0.0060
```

Figura 5.16: Plano gerado para um problema do domínio Science Fiction Book

Nosso protótipo é o único dentre os comparados a efetivar todas as etapas para compor os serviços, desde o armazenamento das URIS (Identificador Uniforme de Recursos) para a leitura dos documentos OWL com as respetivas descrições de pré-condições e efeitos de cada serviço a integração com o planejador JSHOP2.

Outro aspecto é referente ao carregamento de URIS. Dos sistemas analisados, a leitura é realizada de forma manual e uma-a-uma. Nossa abordagem permite que seja realizada a inclusão de domínios com várias URIS, para posteriormente carregar múltiplos endereços dos serviços semânticos.

Finalmente, nossa implementação permite que usuário defina mais de um objetivo, flexibilizando a usabilidade do sistema implementado.

5.6 Considerações

Na presente seção foi descrita nossa implementação, o protótipo SCS, para validar a composição de serviços web utilizando o planejador hierárquico JSHOP2 fazendo uso de descrições semânticas nos serviços escritas em documento OWL.

Foi descrito o problema em compor serviços através das etapas para a realização e arquitetura desenvolvida em forma de camadas para atender todos os requisitos de uma

Atributos	SCS	JSHOP2GUI	Transplan	OWLS-XPlan2
Interface web	X			
Sistema fim-a-fim	X			
Carregar múltiplas uris específicas de um domínio	X			
Compatível com JSHOP2	X	X	X	
Especificação de critérios para exibição do plano	X			X
Definição de múltiplas entradas e saídas na descrição do problema	X			X

Tabela 5.1: Trabalhos correlatos

aplicação fim-a-fim.

A primeira implementação foi definir a estratégia para armazenar e ler os documentos com descrições semânticas. Utilizamos como técnica de armazenamento das URIS, classes java que mapeiam os endereços dos documentos OWL para um banco dados, relacionando também com seus respectivos domínios.

A contribuição nesse aspecto permitiu que o protótipo realize a leitura e o carregamento de múltiplas URIS, com base no domínio selecionado.

Preferimos a utilização de interface web, por se tratar de um problema do contexto da Internet, além disso, nossa implementação contribuiu permitindo que o usuário selecione mais de um objetivo. Para tal tarefa, o protótipo permite a definição de objetivos múltiplos, aumentando as possibilidades do planejador na criação dos planos.

Nossos testes com domínios de aplicações reais demonstraram que o sistema é funcional ao integrar o planejador JSHOP e efetivar a composição de serviços.

Por todos essas características, pode-se destacar como a principal contribuição do trabalho a realização da composição de serviços de ponta a ponta, isto é, desde a personalização dos domínios e serviços até a integração final com planejador, atributo ausente

em todos os sistemas comparados.

CAPÍTULO 6

CONCLUSÃO

Associar serviços de diferentes organizações expostos na web tem demonstrado grande importância na computação distribuída, especificamente para um novo paradigma de sistemas orientado a serviços.

Este trabalho apresentou uma abordagem para testar serviços web com composição automática. A técnica utilizada foi a descrição semântica de serviços associada a planejador hierárquico.

Para alcançar os objetivos do trabalho foi realizado uma revisão dos principais conceitos sobre os serviços web, seus componentes e funcionalidades.

Foi descrito a utilização de semântica em documentos web através da inclusão de ontologias e a importância em construir significados. Também foi descrita a evolução das linguagens puramente sintáticas para linguagens com atributos semânticas.

Ainda na revisão da literatura foi feito uma revisão sobre planejamento, incluindo as representações STRIPS e PDDL. Também foi apresentado o planejamento hierárquico, suas características e sua justificativa para utilizá-lo como ferramenta para compor serviços, mais especificamente o planejador JSHOP2.

A implementação do protótipo através de camadas desenvolvendo um sistema web fim-a-fim. Cada camada especifica um requisito do sistemas, iniciando da camada de banco que mapeia os serviços e os uris, a camada de interface que controla as ações do usuário referente ao domínio selecionado e os estados estados iniciais e objetivos, até a camada de controle que integra a aplicação com o planejador JSHOP2.

Nossa implementação propôs um sistema que integra todos as etapas para compor serviços web, facilitando a pesquisa na área de sistemas distribuídos e inteligência artificial.

Os resultados experimentais mostram que a abordagem é promissora, ao utilizar o planejador JSHOP2 para criar planos através das descrições semânticos em serviços web

utilizando domínios com diferentes níveis de complexidade e locais de hospedagem diferentes.

Como trabalho futuro, destacamos que a implementação deve contemplar a verificação de requisitos não-funcionais, entre eles quesitos em relação a autenticação de serviços seguros.

Outro caminho pode ser o replanejamento. Um cenário para tal situação poderia ser ilustrado caso não existisse uma combinação possível de serviços para entradas e objetivos em domínio, selecionados pelo usuário.

BIBLIOGRAFIA

[1] *OWL-S Service Retrieval Test Collection Version 3.0 revision 1.*

[2] *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2003.

[3] Support Ontologies AI e Service Profile. Owl-s applications and issues.

[4] Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Sheila A. Mcilraith, Srini Narayanan, Massimo Paolucci, e Terry Payne. DAML-S: Semantic Markup For Web Services, 2001.

[5] Grigoris Antoniou e Frank Van Harmelen. Web ontology language: Owl. *Handbook on Ontologies in Information Systems*, páginas 55–62. Springer Berlin Heidelberg, 2003.

[6] Ioannis G. Baltopoulos. Web service composition in the grid environment, 2005.

[7] T Berners-Lee, J Hendler, e O Lassila. The semantic web. http://kill.devc.at/system/files/scientific-american_0.pdf, 2001.

[8] Dan Brickley e R.V. Guha. Resource description framework(rdf) schema specification 1.0. http://www.w3.org/TR/rdf-schema/, 2004. Acessado em 03 de Abril de 2009.

[9] Fabio Casati, Ski Ilnicki, LiJie Jin, e Vasudev Krishnamoorthy. Adaptive and dynamic service composition in eflow, mar de 2000.

[10] Pierre-Antoine Champin. Rdf tutorial, jun de 2001.

[11] Seog chan Oh, Dongwon Lee, e Soundar R. T. Kumara. A comparative illustration of ai planning-based web services composition. *ACM SIGecom Exchanges*, 5:2005, 2005.

[12] B. Chandrasekaran e V. Richard Benjamins Jorn R. Josephson. What are ontologies and why do we need them? *IEEE Inteligent Systems*, páginas 20–25, 1999.

[13] Roberto Chinnici. Web services description language (wsdl) version 2.0 part 1: Core language. http://www.w3.org/TR/wsdl20/, 2001.

[14] E Christensen. W3c web services description language (wsdl). http://www.w3.org/TR/wsdl, 2001.

[15] Francisco Curbera e Rania Khalaf. Implementing BPEL4WS: The architecture of a BPEL4WS implementation. *In Proceedings of the Grid Workflow Workshop at GGF-10*, páginas 2006, 2004.

[16] M Dean, D Connolly, F van Harmelen, J Hendler, I Horrocks, D L McGuinness, P F Patel-Schneider, e L A Stein. OWL Web Ontology Language 1.0 reference, 2002.

[17] E.P.D.Pednault. Adl: Exploring the middle ground between strips and the situation calculus, 1989.

[18] Kutluhan Erol, James Hendler, e Dana S. Nau. HTN Planning: Complexity and expressivity. *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, páginas 1123–1128. AAAI Press, 1994.

[19] Kutluhan Erol, James Hendler, e Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, páginas 249–254. AAAI Press, 1994.

[20] Curbera F., Duftler M., Khalaf R, Nagy W, Mukhi N., e Weerawarana S. Unraveling the web services : an introduction to SOAP, WSDL, and UDD. volume 6, páginas 86–93. IEEE, apr de 2002.

[21] Ian Foster. The physiology of the grid: An open grid services architecture for distributed systems integration. The Forth Global Grid Forum, 2002.

[22] Malik Ghallab, Dana Nau, e Paolo Traverso. Automated planning, 2004.

[23] Bernardo Cuenca Grau. A possible simplification of the semantic web architecture, may de 2004.

[24] C. C. Green. Application of theorem proving to problem solving. *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, 1969.

[25] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[26] J Hoffmann e B Nebel. The ff planning system: Fast plan generation through heuristic search. Relatório técnico, 2001.

[27] Yang Hongli, Zhao Xiangpeng, Qiu Zongyan, Pu Geguang, e Wang Shuling. A formal model of web service choreography description language(WS-CDL). Relatório técnico, 2006.

[28] Matthew Horridge, Holger Knublauch, Alan Rector, Robert, Stevens, e Chris Wroe. A practical guide to building OWL ontologies using the protege-OWL Plugin and CO-ODE Tools Edition 1.0. http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf, 2004. Acessado em 10 de Abril de 2009.

[29] Okhtay Ilghami e Dana S. Nau. A general approach to synthesize problem-specific planners. Relatório técnico, 2003.

[30] Eric Jendrock, Jennifer Ball, Debbie Carson, Ian Evans, Scott Fordin, e Kim Haase. *The Java EE 5 Tutorial*, oct de 2008.

[31] Cheryl Kirk e Natanya Pitts-Moultis. *XML Black Book*. Makron, 1st edition, 2000.

[32] Matthias Klusch e Andreas Gerber. Semantic web service composition planning with OWLS-XPlan. *In Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, páginas 55–62, 2005.

[33] Sriram Krishnan, Patrick Wagstrom, e Gregor Von Laszewski. GSF: A Workflow Framework for Grid Services. Relatório técnico, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, 2002.

[34] Ora Lassila e Ralph R. Swick. Resource description framework (RDF) model and syntax specification, 1998.

[35] Okhtay lghami. Documentation for JSHOP2. Relatório técnico, 2006.

[36] Frank Manola e Eric Miller. RDF Primer. http://www.w3.org/TR/rdf-primer/, 2004. Acessado em 30 de Março de 2009.

[37] Maria Bruno Marietto, Nuno David, Jaime Simo Sichman, e Helder Coelho. Requirements analysis of multi-agent-based simulation. 1998.

[38] D. McDermottl. Pddl — the planning domain definition language, 1998.

[39] S McIlraith, T C Son, e H Zeng. Semantic web services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, (16):46–53, 2001.

[40] Nilo Mitra. SOAP Version 1.2 Part 0: Primer. http://www.w3.org/TR/soap12-part0/, 2009.

[41] Fikes R.;e Nilsson N. Strips: A new approach to the application of theorem proving to problem solving. *Journal of Artificial Intelligence*, 1971.

[42] D Nau, Y Cao, A Lotem, e H Munos-Avila. The SHOP planning system. *AI Magazine*, (22):91–94, 2001.

[43] Dana Nau, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, e Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[44] J Peer. Web Service Composition as AI Planning - a Survey. Relatório técnico, 2005.

[45] R. Reiter. On closed world data bases. *In Logic and Data Bases (H. Gallaire and J. Minker, editors)*, 1978.

[46] A. Roy, J. Ramanujan. XML schema language: taking XML to the next level, apr de 2001.

[47] Thomas Sandholm e Jarek Gawor. Globus Toolkit 3 Core - A Grid Service Container Framework. Relatório técnico, 2003.

[48] Mauro Santana. Web services assincronos. http://www.msdnbrasil.com.br/tecnologias/web_servic 2001.

[49] Marcus Vinicius Almeida Silva. TRANSPLAN: Uma solução para mapear e planejar serviços semânticos, 2008.

[50] H. Mssenbck T. Rodriguez K. Russell T. Kotzmann, C. Wimmer e D. Cox. Design of the java hotspottm client compiler for java 6. Relatório técnico, 2007.

[51] Crown Walk. The workflow management coalition specification. http://www.wfmc.org/standards/docs/TC-1012_Nov_99.pdf, 1994.

[52] Dan Wu, Bijan Parsia, Evren Sirin, e James Hendler. Automating DAML-S Web Services Composition Using SHOP2. *In Proceedings of 2nd International Semantic Web Conference ISWC2003*, páginas 195–210. Springer Berlin / Heidelberg, 2003.

[53] R Zhang, I B Arpinar, e Aleman-Meza. Automatic composition of semantic web services. *Intl. Conf. on Web Services, Las Vegas NV*, páginas 38–41, 2003.