

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL  
FACULDADE DE INFORMÁTICA  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO DE INTEGRAÇÃO DE TECNOLOGIAS E  
PADRÕES PARA EXECUÇÃO DE APLICAÇÕES BASEADAS  
EM AGENTES PARA A WEB SEMÂNTICA

**MAURICIO DA SILVA ESCOBAR**

Orientador: Prof. Dr. Marcelo Blois Ribeiro

Dissertação de Mestrado

Porto Alegre  
2008

**MAURICIO DA SILVA ESCOBAR**

**UM MODELO DE INTEGRAÇÃO DE TECNOLOGIAS E  
PADRÕES PARA EXECUÇÃO DE APLICAÇÕES BASEADAS  
EM AGENTES PARA A WEB SEMÂNTICA**

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre, pelo programa de Pós Graduação em Ciência da Computação da Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Marcelo Blois Ribeiro

Porto Alegre

2008

## **Dados Internacionais de Catalogação na Publicação (CIP)**

E74m Escobar, Mauricio da Silva

Um Modelo de integração de tecnologias e padrões para execução de aplicações baseadas em agentes para a web semântica / Mauricio da Silva Escobar.

Porto Alegre, 2009.

93 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.

Orientador: Prof. Dr. Marcelo Blois Ribeiro

**Ficha Catalográfica elaborada pelo  
Setor de Tratamento da Informação da BC-PUCRS**



Pontifícia Universidade Católica do Rio Grande do Sul  
FACULDADE DE INFORMÁTICA  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

## TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "**Um Modelo de Integração de Tecnologias e Padrões para Execução de Aplicações Baseadas em Agentes para a Web Semântica**", apresentada por Maurício da Silva Escobar, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas de Informação, aprovada em 12/12/08 pela Comissão Examinadora:

  
Prof. Dr. Marcelo Blois Ribeiro -  
Orientador

PPGCC/PUCRS


  
Prof. Dr. Ricardo Melo Bastos -

PPGCC/PUCRS

  
Prof. Dr. Ricardo Choren Nova -

IME-RJ

Homologada em...17.../03.../09..., conforme Ata No. 004/09 pela Comissão Coordenadora.

  
Prof. Dr. Fernando Gehm Moraes  
Coordenador.

PUCRS

### Campus Central

Av. Ipiranga, 6681 - P32 - sala 507 - CEP: 90619-900

Fone: (51) 3320-3611 - Fax (51) 3320-3621

E-mail: [ppgcc@pucrs.br](mailto:ppgcc@pucrs.br)

[www.pucrs.br/facin/pos](http://www.pucrs.br/facin/pos)

## **AGRADECIMENTOS**

*Ao meu orientador, Professor Marcelo Blois Ribeiro, pelo apoio e ensinamentos passados ao longo da realização deste trabalho, que contribuíram imensamente para o meu crescimento profissional.*

*Aos membros da banca, Professores Ricardo Choren e Ricardo Bastos, pela aceitação do convite de participação na avaliação deste trabalho.*

*A minha namorada, Bianca, por todo o apoio, compreensão e paciência em todos os momentos.*

*Aos meus pais, Ângela e Mauro, pela educação, pela incansável dedicação, e pelo suporte em todos os momentos.*

*A toda a minha família e amigos pelo apoio.*

*Aos meus colegas da PUCRS e do grupo de pesquisa ISEG.*

*A todos os professores e funcionários da Faculdade de Informática da PUCRS.*

*Ao convênio PUCRS/DELL, pelos auxílios concedidos.*

## RESUMO

A *Web Semântica* provê acesso a informações distribuídas e heterogêneas, permitindo aos softwares a mediação entre as necessidades dos usuários e as fontes de informações disponíveis. Os agentes de software são uma das tecnologias mais promissoras para o desenvolvimento da *Web Semântica*. Entretanto, as tecnologias baseadas em agentes não serão difundidas a menos que existam infra-estruturas adequadas para o desenvolvimento de sistemas multiagentes (SMAs). Tornar agentes de software uma abstração prática, capaz de lidar com ontologias e de ser integrado com as ferramentas *Web* existentes são desafios ainda existentes. Este trabalho propõe uma arquitetura para ser integrada a navegadores e a servidores *Web* para permitir a criação de domínios semânticos onde os agentes de software vivem, estendendo a *Web*, sem interferir na sua estrutura atual. Estes domínios alteram o atual paradigma *request-response* utilizado na *Web* para um paradigma híbrido *request-response / peer-to-peer*. Os agentes vivendo nas máquinas cliente em domínios semânticos associados a navegadores *Web* serão capazes de comunicar-se com outros agentes em outros domínios e processarem conteúdos anotados semanticamente em páginas *Web*.

**Palavras-chave:** *Web Semântica*, Sistemas Multiagentes, Infra-estrutura de agentes.

## **ABSTRACT**

The Semantic Web provides access to heterogeneous, distributed information, enabling software to attend user needs through understanding the information resources available. Agents are a promising technology for the development of Semantic Web software products. However, agent-based technologies will not become widespread until there are adequate infrastructures for the development of semantic multi-agent systems (MAS). Some challenges, such as turning software agents into a practical abstraction for dealing with ontologies and integrating distributed agents with the existing web tools still need to be addressed. This work proposes an architecture to be integrated to web servers and web browsers in order to create Semantic Domains where software agents live thus extending the Web without interfering with its current structure. These Domains turn the current request-response paradigm used on the Web into a hybrid request-response / peer-to-peer model. Agents living on the client's machine in a Semantic Domain associated to a web browser are capable of contacting other agents in other Semantic Domains, and of processing semantic content annotated in Web pages.

**Keywords:** Semantic Web, Multi-agent Systems, Agents Infrastructure.

## LISTA DE ILUSTRAÇÕES

Figura 1 - Exemplo de representação em HTML .....	24
Figura 2 - Exemplo de representação em XML.....	25
Figura 3 - Exemplo de elemento em XML .....	25
Figura 4 - Exemplo de atributo em XML .....	26
Figura 5 - XML <i>Schema</i> e instâncias (DACONTA, 2003).....	27
Figura 6 - Exemplo de estrutura de dados em XML.....	28
Figura 7 - Exemplo 2 de estrutura de dados em XML.....	28
Figura 8 - Grafo representando uma tripla RDF.....	30
Figura 9 - Exemplo de código RDF (RDF, 2004) .....	30
Figura 10 - Sintaxe da linguagem RDF para declaração de classe e subclasse .....	32
Figura 11 - Exemplo de hierarquia de classes.....	32
Figura 12 - origem da Linguagem OWL .....	34
Figura 13 - Exemplo de classe em OWL (OWL, 2004) .....	35
Figura 14 - Exemplo de subclasse em OWL (OWL, 2004).....	35
Figura 15 - Exemplo de propriedade em OWL .....	36
Figura 16 - Robôs controlados por agentes de software (VLASSIS, 2007).....	38
Figura 17 - Arquitetura de um agente no Semanticore (ESCOBAR <i>et al.</i> , 2006).....	44
Figura 18 - Representação do Modelo de Domínio (ESCOBAR <i>et al.</i> , 2006).....	46
Figura 19 - Interface do Semanticore: aplicação exemplo.....	51
Figura 20 - A arquitetura do Jena (MCBRIDE, 2002).....	53
Figura 21 - A arquitetura Magpie (GUIDI <i>et al.</i> , 2007) .....	56
Figura 22 - Comunicação entre ambientes.....	60
Figura 23 - Arquitetura do SWP .....	61
Figura 24 - Exemplo de anotações semânticas em uma página <i>web</i> .....	61
Figura 25 - Diagrama de classes da arquitetura do SWL cliente.....	63
Figura 26 - Diagrama de seqüência - processamento de anotações semânticas .....	65
Figura 27 - Diagrama de seqüência - controle das janelas de interação .....	68
Figura 28 - SWP: Interface com o usuário no navegador Mozilla Firefox.....	70
Figura 29 - Arquitetura para a integração com o navegador Firefox.....	71
Figura 30 - SWP - Domínio no navegador <i>Web</i> .....	75
Figura 31 - Código do agente do usuário.....	76
Figura 32 - Resultado apresentado pelo agente do usuário.....	78



## LISTA DE TABELAS

Tabela 1 - Conferência WWW: principais tópicos de pesquisa em <i>Web Semântica</i> .....	22
Tabela 2 - Conferência ISWC: principais tópicos de pesquisa em <i>Web Semântica</i> .....	23
Tabela 3 - Regra de inferência para a verificação do petróleo .....	76
Tabela 4 - Regra de inferência para comparação entre carros .....	77

## LISTA DE SIGLAS

ACL – *Agent Communication Language*  
API – *Application Programming Interface*  
HTML – *Hypertext Markup Language*  
HTTP – *Hypertext Transfer Protocol*  
KQML – *Knowledge Query Manipulation Language*  
OWL – *Ontology Web Language*  
OO – *Object-Oriented*  
RDF – *Resource Description Framework*  
RDFS – *RDF Schema*  
SDK – *Software Development Kit*  
SMA – *Sistema Multiagentes*  
SWP – *Semantic Web Plugin*  
UML - *Unified Modeling Language*  
URI – *Uniform Resource Identifier*  
URL – *Uniform Resource Locator*  
XML – *eXtensible Markup Language*  
XMLS – *XML Schema*  
XUL – *XML User Interface Language*  
WWW – *World Wide Web*

# SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>11</b>
1.1 QUESTÃO DE PESQUISA .....	13
1.2 OBJETIVOS .....	13
1.2.1 OBJETIVO GERAL.....	13
1.2.2 OBJETIVOS ESPECÍFICOS .....	14
1.3. ORGANIZAÇÃO DA DISSERTAÇÃO .....	14
<b>2 FUNDAMENTAÇÃO TEÓRICA.....</b>	<b>17</b>
2.1 WEB SEMÂNTICA.....	17
2.1.1 <i>Desafios na Web Semântica</i> .....	19
2.1.2 <i>Tópicos de Pesquisa em Web Semântica</i> .....	20
2.2 XML (EXTENSIBLE MARKUP LANGUAGE) .....	24
2.2.1 <i>Elementos XML</i> .....	25
2.2.2 <i>Atributos XML</i> .....	26
2.2.3 <i>XML Schema</i> .....	26
2.3 RESOURCE DESCRIPTION FRAMEWORK (RDF) .....	27
2.3.1 <i>RDF: Idéia Básica</i> .....	28
2.3.2 <i>Recursos</i> .....	29
2.3.3 <i>Propriedades</i> .....	29
2.3.4 <i>Sentenças</i> .....	29
2.3.5 <i>RDF Schema (RDFS)</i> .....	31
2.3.5.1 <i>Classes e Propriedades</i> .....	31
2.3.5.2 <i>Herança</i> .....	31
2.4 WEB ONTOLOGY LANGUAGE (OWL) .....	32
2.5 AGENTES E SISTEMAS MULTIAGENTES .....	36
2.5.1 <i>Propriedades e Características de um Sistema Multiagentes</i> .....	39
2.5.1.1 <i>Sociedade de agentes</i> .....	39
2.5.1.2 <i>Cooperação</i> .....	40
2.5.1.3 <i>Coordenação</i> .....	41
2.5.1.4 <i>Comunicação</i> .....	41
2.6 SEMANTICORE.....	42
2.6.1 <i>O Modelo de Agente</i> .....	43
2.6.2 <i>Modelo de Domínio</i> .....	45
2.6.3 <i>Exemplo de Utilização</i> .....	46
2.6.3.1 <i>O agente Gerenciador de Recursos</i> .....	46
2.6.3.2 <i>Demais agentes do sistema</i> .....	48
2.6.3.3 <i>Executando a aplicação</i> .....	51
2.7 JENA.....	52
<b>3 TRABALHOS RELACIONADOS .....</b>	<b>55</b>
3.1 POWER MAGPIE .....	55
<b>4 UMA ARQUITETURA PARA A EXECUÇÃO DE APLICAÇÕES BASEADAS EM AGENTES NA WEB SEMÂNTICA .....</b>	<b>59</b>
4.1 INTEGRAÇÃO .....	62
4.2 GERENCIAMENTO DE CONTEÚDOS .....	64
4.3 HISTÓRICO .....	66
4.4 INTERAÇÃO.....	66
<b>5 PROTÓTIPO E EXEMPLO DE USO.....</b>	<b>69</b>
5.1 EXEMPLO DE USO.....	73
5.2 CONSIDERAÇÕES SOBRE O EXEMPLO.....	78

<b>6 CONCLUSÃO E TRABALHOS FUTUROS .....</b>	<b>81</b>
<b>REFERÊNCIAS .....</b>	<b>85</b>
<b>APÊNDICE A .....</b>	<b>89</b>
<b>APÊNDICE B .....</b>	<b>91</b>

# 1 INTRODUÇÃO

A *World Wide Web* foi proposta como um projeto de desenvolvimento para o CERN (*European Organization for Nuclear Research*) em 1989, por Tim Berners-Lee, onde foi criado um sistema universal para a interconexão de informações. Um requisito básico para esse sistema era uma linguagem para a formatação da informação em hipertextos, o que motivou a criação da linguagem HTML (HTML, 2006) por Tim Berners-Lee. Em 1991 já havia um navegador portátil e disponível para distribuição capaz de trabalhar com a linguagem HTML. O Netscape<sup>1</sup> impulsionou a *Web* em 1994, através da criação de um navegador comercial. Na mesma época, o Yahoo<sup>2</sup> foi criado e fornecia um indexador de páginas *Web*. Existiam também mecanismos como o WebCrawler, que era um mecanismo de busca e possuía cerca de 2500 servidores no mundo todo. Já no final de 1995, haviam aproximadamente 73500 servidores *Web* em todo o mundo, e a Microsoft fornecia uma versão do Internet Explorer<sup>3</sup>.

Em contraste, a proposta da *Web Semântica* (BERNERS-LEE *et al.*, 2001) tornou-se popular em 2001 quando foi reconhecido que a *Web* foi construída para consumo humano, e embora tudo nela seja lido por máquina, os seus dados não estão em uma forma que permita o raciocínio automatizado. Portanto, a forma como a semântica contida em páginas *Web* é representada só faz sentido para os seres humanos, tornando difícil a automatização do processamento das fontes de informação na *Web*, e também por causa do grande volume de informações nela contida, torna-se impossível o seu gerenciamento manualmente.

Com o rápido crescimento da *Web*, torna-se cada vez mais difícil a manipulação das informações contidas nas suas páginas de maneira a gerar um resultado satisfatório. Em muitos casos, para que possamos melhorar a coleta de informação na *Web*, são necessários mecanismos de busca complexos. Mesmo com tais mecanismos, os resultados obtidos requerem ainda uma análise pelo usuário para que este possa extrair a informação desejada. A geração de conteúdos na *Web* de maneira automática a partir das bases de dados não inclui uma semântica formal para a sua representação.

---

<sup>1</sup> Página web: <http://browser.netscape.com/>

<sup>2</sup> Página web: <http://www.yahoo.com>

<sup>3</sup> Página web: <http://www.microsoft.com/brasil/windows/ie/default.msp>

As páginas *Web* são construídas utilizando-se a linguagem HTML, que é a linguagem predominante nos dias de hoje. Esta linguagem não possui recursos suficientes para permitir a atribuição de significado para as informações contidas nas páginas *Web* e não permite a criação de modelos semânticos, devido aos marcadores HTML representarem apenas a descrição de como os dados contidos na página devem ser exibidos.

A *Web Semântica* é uma iniciativa que busca a definição de formatos padrão para exprimir informações em uma forma processável pela máquina (SEMANTIC WEB, 1998). A idéia básica é criar formas de explicitar o relacionamento de conceitos de vários domínios de conhecimento, permitindo que máquinas possam trabalhar sobre estes conceitos, relacionando-os e inferindo novos conceitos.

Durante os últimos anos muitas tecnologias relacionadas a *Web Semântica* tem emergido ou tem sido elaboradas. A W3C<sup>4</sup>, que é o principal consórcio responsável pelos padrões *Web*, tem trabalhado intensivamente nos padrões semânticos, como o RDF (RDF, 2004) e o OWL (OWL, 2004). Estes padrões oferecem a criação de uma base sólida para aplicações semânticas e tem implicado em um significativo avanço da *Web Semântica* de um nível de pesquisa para sua aplicação na indústria para a construção de aplicações de uma nova geração.

Devido à grande quantidade de informações disponibilizadas na *Web*, torna-se evidente a necessidade de mecanismos – aplicações – que possam nos auxiliar na coleta e análise destas informações, bem como seu processamento automático. Na *Web semântica*, os agentes de software são usados como entidades autônomas capazes de consumir automaticamente os conteúdos publicados. Da mesma forma, os computadores necessitam ter acesso a coleções estruturadas de informações (dados e meta-dados) e a conjuntos de regras de inferência que os auxiliem no processo de dedução automática para que seja administrado o raciocínio automatizado, ou seja, a representação do conhecimento.

Para possibilitar a construção de sistemas abertos e flexíveis que utilizam a infra-estrutura da *Web*, uma plataforma de agentes deve prover algumas características como distribuição de processamento e integração com ferramentas existentes (incluindo navegadores *Web*). Estas qualidades são importantes para permitir uma fácil combinação entre a tecnologia de agentes e a *Web*, facilitando sua adoção.

Este trabalho apresenta uma arquitetura para a execução de aplicações baseadas em agentes na *Web Semântica*, por meio da integração de tecnologias de criação de

---

<sup>4</sup> Página web: <http://www.w3.org/>

agentes, formas de representação de conhecimento e ferramentas atuais utilizadas na *Web*. Esta arquitetura integra o *framework* SemantiCore (BLOIS E LUCENA, 2004) a navegadores e servidores *Web*, permitindo a interação direta entre os usuários e os agentes associados ao conteúdo em um determinado domínio.

## 1.1 Questão de Pesquisa

Nos últimos anos têm crescido o número de iniciativas acadêmicas e industriais para atender os requisitos da *Web* Semântica. Ferramentas de anotação de conteúdos, editores de ontologias e mecanismos de inferência tem sido criados para permitir o desenvolvimento de aplicações que podem beneficiar-se das características da *Web* Semântica. Nenhuma destas iniciativas provê uma combinação de todos os esforços já realizados que permita o desenvolvimento de aplicações na *Web* Semântica.

Com base nesta perspectiva, surge a seguinte questão de pesquisa: **“é possível criar uma arquitetura que integre as tecnologias existentes para prover transparência na execução de aplicações baseadas em agentes para a *Web* Semântica?”**.

## 1.2 Objetivos

Definida a questão de pesquisa, se faz necessária a definição dos objetivos do trabalho. A seguir são apresentados o objetivo geral e os objetivos específicos deste trabalho.

### 1.2.1 Objetivo Geral

O objetivo geral deste trabalho é a construção de uma arquitetura integrada a navegadores e a servidores *Web*. Através da integração com o *framework* SemantiCore, que possui como abstração a construção de sistemas mutliagentes abertos na *Web*, e das

tecnologias e padrões da *Web Semântica*, a arquitetura visa permitir a criação de ambientes onde os agentes de software vivem, estendendo a *Web*, sem interferir na sua estrutura atual.

### 1.2.2 Objetivos específicos

Os objetivos específicos são os seguintes:

- Aprofundar o estudo teórico sobre trabalhos relacionados ao problema abordado.
- Apontar a alteração necessária no protocolo HTTP (HTTP, 1996) para permitir a identificação automática de ambientes de agentes.
- Definir os elementos da arquitetura.
- Integrar a arquitetura proposta às ferramentas *Web*, como navegadores e servidores de aplicação.
- Mapear e implementar os elementos da arquitetura proposta.
- Projetar e implementar uma aplicação que possibilite a verificação das características da arquitetura proposta.

### 1.3. Organização da Dissertação

Este trabalho está dividido em três partes: a fundamentação teórica, a apresentação da proposta, e a demonstração da aplicabilidade desta proposta. O Capítulo 2 apresenta a fundamentação teórica necessária para um bom entendimento do trabalho, onde são esclarecidos aspectos referentes à *Web Semântica* e suas tecnologias associadas, Agentes de software e Sistemas Multiagentes e o *framework* Semanticore.

No Capítulo 3 são apresentados alguns trabalhos encontrados na literatura que visam apoiar o desenvolvimento da *Web Semântica*, e que se relacionam com a presente proposta. No Capítulo 4 é descrita a arquitetura proposta. Já no Capítulo 5 é apresentado o protótipo desenvolvido que mapeia as características da arquitetura, e um exemplo de uso da



proposta, através do desenvolvimento de um cenário de exemplo, que busca demonstrar todas as características da proposta. Para a apresentação deste exemplo, são mostrados trechos de código e explicações contextualizadas sobre os elementos da arquitetura. Por fim, no Capítulo 6 são apresentadas as conclusões e os trabalhos futuros.



## 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, será apresentado o embasamento teórico referente aos principais conceitos e características de *Web Semântica* e suas tecnologias, agentes de software e sistemas multiagentes, e o *framework SemantiCore*.

### 2.1 Web Semântica

A *Web Semântica* provê uma estrutura que permite o compartilhamento e reutilização dos dados. Ela é um esforço conduzido pela W3C com a participação de um grande número de pesquisadores e parceiros. Um dos objetivos da *Web Semântica* é desenvolver tecnologias e linguagens de representação de conhecimento legíveis para as máquinas, adicionando informações sobre o conteúdo contido nas páginas *Web* atuais e em futuras para que esta informação seja manipulável computacionalmente. Dentre os objetivos está o desenvolvimento de um modelo tecnológico que permita o compartilhamento global de conhecimento assistido por máquinas (SEMANTIC WEB, 2001).

A *Web Semântica* deve ser capaz de suportar serviços automatizados baseados em descrições formais da semântica das informações. A semântica é vista como uma alternativa na busca de informações relevantes, uma vez que os mecanismos de busca baseiam-se em técnicas predominantemente sintáticas, não atendendo muitas vezes as reais necessidades do usuário.

A *Web Semântica* não é uma *Web* alternativa, mas uma extensão da atual, onde a informação é recuperada através de uma semântica bem definida, habilitando uma cooperação entre o trabalho humano e computadores. Ela é baseada em uma estrutura descentralizada, onde agentes de software possam atuar em ambientes de diversas estruturas, compreendendo e realizando sofisticadas tarefas para os usuários através da valoração semântica destes conteúdos. Estas estruturas são independentes do tipo de mídia, isto é, podem ser textos, sons, imagens, vídeos ou gráficos.

De modo a organizar a informação na Internet, pesquisadores de inteligência artificial vêm propondo uma série de modelos. A idéia central é categorizar a informação de maneira padronizada, facilitando seu acesso (BREITMAN, 2007). No entanto, acredita-se que o maior fator de sucesso do crescimento da internet seja a liberdade que ela fornece a seus usuários. Esta liberdade vai desde sites básicos, construídos por leigos, e que contêm informações triviais; até os mais sofisticados sites, construídos por grandes empresas, e que utilizam diversas tecnologias *Web*.

Para a construção da *Web Semântica*, existem inúmeros esforços para padronização de tecnologias, bem como o XML (XML, 2006), o RDF, as arquiteturas de meta-dados, formas de representação de ontologias, agentes inteligentes, entre outras.

As ontologias, em ciência da computação, foram adotadas na inteligência artificial para facilitar o compartilhamento e reuso de conhecimento (FENSEL, 2001; DAVIES *et al.*, 2003). Atualmente, seu uso começa a ser expandido para áreas como integração de informações, sistemas cooperativos, engenharia de sistemas baseados em agentes e comércio eletrônico. As ontologias são modelos conceituais que capturam e tornam explícito o vocabulário utilizado em aplicações semânticas, garantindo dessa forma, a comunicação livre de ambigüidades.

O trabalho eficiente e efetivo com a *Web Semântica* deve ser suportado por ferramentas avançadas que permitam o uso total destas tecnologias. Em particular, isto requer os seguintes elementos (FENSEL *et al.*, 2003):

- Linguagens formais para expressarmos e representarmos ontologias.
- Editores e construção semi-automática de ontologias.
- Reuso e unificação de ontologias (ambientes que facilitem a criação de novas ontologias através do reuso de ontologias já existentes).
- Ferramentas de anotação que associem informações não estruturadas e semi-estruturadas através de metadados.
- Ferramentas para navegação e acesso a informação que permita o acesso inteligente a informação por usuários humanos.
- Serviços de tradução e integração de diferentes ontologias que realizam intercâmbio entre padrões (especialmente em comércio eletrônico).

Destacadas algumas das principais características da *Web Semântica*, na próxima seção serão apresentados alguns desafios de pesquisa em *Web Semântica*.

### 2.1.1 Desafios na Web Semântica

A *Web* continua a crescer e novas tecnologias, modos de interação e aplicações estão sendo desenvolvidas. As tecnologias da *Web Semântica* destinam-se a proporcionar um espaço compartilhado de informações semânticas, mudando qualitativamente as experiências na *Web*.

A *Web Semântica* possui ainda algumas tecnologias que não são maduras em diversas áreas. Portanto, existem alguns problemas que ainda são encontrados, ocasionando novas oportunidades e desafios na pesquisa. Tais desafios na *Web Semântica* incluem e não se limitam a: (i) criação e gerenciamento de conteúdo semântico, (ii) construção de aplicações robustas e escaláveis, e (iii) organização e integração de informações de diferentes fontes, tornando a semântica explícita a fim de melhorar nossa experiência geral com tecnologias de informação e assim, nos permitindo utilizar a grande quantidade de informação que está disponível em forma digital para enfrentarmos as nossas tarefas quotidianas.

O desenvolvimento da *Web Semântica* envolve muitas áreas da ciência da computação (EUZENAT, 2002), incluindo:

- **Inteligência Artificial:** mecanismos de raciocínio, linguagens de representação de conhecimento, aprendizado, descoberta de recursos, etc.
- **Web: profiling,** identificação, linguagens baseadas em XML e tecnologias.
- **Agentes de software:** computação distribuída, linguagens de comunicação e interação e protocolos de cooperação.
- **Teoria da computação e lógica (computacional):** linguagens formais, prova e teoremas.

- **Linguística computacional e reconhecimento de padrões:** aquisição de conhecimento de fontes primárias, o uso de recursos léxicos para o desenvolvimento de ontologias, etc.
- **Engenharia de documentos e bibliotecas digitais:** transformação, marcação e indexação.
- **Interfaces humano-computador:** trabalho cooperativo suportado por computador, avaliação de fatores de trabalho e estudos de comunicação.
- **Ciências sociais e humanas:** experimentos de validação de ontologias e processamento de informações sociais.

Acima de tudo, o desenvolvimento da *Web Semântica* deve considerar as atuais necessidades da *Web*, e que requerem soluções em diferentes áreas, como por exemplo, a personalização (e desse modo as questões de privacidade), a mobilidade (questões de confiabilidade) e a publicação (questões de segurança). Estas questões e tópicos são bastante tradicionais, mas a *Web Semântica* lida com eles ao extremo. Contudo, para a *Web Semântica* acontecer ela necessita mais do que somente tecnologias, questões econômicas e sociais também são importantes.

### 2.1.2 Tópicos de Pesquisa em Web Semântica

Esta seção apresenta o levantamento das pesquisas realizadas em *Web Semântica* nos últimos anos, apresentando uma visão da relevância e atualidade da área. Serão apresentados alguns dados coletados das principais conferências da área, bem como a categorização dos principais tópicos de pesquisa sugeridos.

A tabela 1 apresenta os tópicos de pesquisa sugeridos para a conferência internacional da WWW (*International World Wide Web Conference*). Estes tópicos foram coletados nas conferências dos anos de 2005 a 2008. A tabela apresenta cada tópico de pesquisa em uma linha da tabela. As demais colunas representam a marcação do ano em que este tópico foi abordado pela conferência.

A tabela 2 mostra os tópicos de pesquisa sugeridos para a conferência internacional de *Web Semântica (International Semantic Web Conference - ISWC)*. Estes tópicos foram coletados nas conferências dos anos de 2004 a 2007. A tabela apresenta cada tópico de pesquisa em uma linha da tabela. As demais colunas representam a marcação do ano em que este tópico foi abordado pela conferência.

Através destes dados podemos ter uma noção das tendências da pesquisa em *Web Semântica* onde, os temas tendem a se adaptarem às necessidades tecnológicas do momento. Neste caso, os congressos solicitam contribuições que demonstram como as tecnologias semânticas podem ser exploradas na *Web*. Em geral, são esperados os seguintes objetivos nas pesquisas:

- Mostrar como as tecnologias semânticas adicionam valor para a *Web*, conseguindo atender requisitos que as tecnologias atuais não atendem;
- Apresentar novas tecnologias para a *Web Semântica* ou novas aplicações de tecnologias semânticas existentes que provejam novos níveis de funcionalidades na *Web*;
- Verificar o efeito das comunidades na *Web* pode ser explorado para gerar semântica;
- Demonstrar como tendências emergentes na *Web* tais como *wikis*, softwares sociais, entre outros, podem ser enriquecidos com as tecnologias semânticas.

O enquadramento do presente trabalho, em relação aos tópicos de pesquisa dos congressos abordados, pode ser visto como: “Distributed Architectures for the Semantic Web” no congresso WWW, e “Semantic Web Middleware” no congresso ISWC.

**Tabela 1** - Conferência WWW: principais tópicos de pesquisa em *Web Semântica*

<b>Assunto / Ano</b>	<b>2005</b>	<b>2006</b>	<b>2007</b>	<b>2008</b>
Automated Reasoning				
Annotation of multimedia				
Agent systems on the web				
Convergence between semantic web and grid systems				
Distributed aspects of semantic representations				
Distributed architectures for the Semantic Web				
Distributed architectures for structured data on the Web				
Emergent Semantics				
Emergent semantics of structured data repositories				
Information Integration				
Ontologies and representation languages				
Provenance, Trust & Security				
Representations for the Semantic Web				
Search and retrieval based on document understanding				
Semantic Web User Interfaces				
Semantic Annotation				
Semantic Metadata				
Semantic Web Applications				
Semantic Querying (search) and Retrieval				
Semantic Web Repositories				
Semantic brokering, integration and interoperability				
Semantic web services				
Semantic web mining (ontology learnign)				
Semantics in peer-to-peer systems				
Social networks, web communities				
Semantic multimedia				
Semantic Web in e-Business, e-Learning, e-Science				
Web applications that exploit semantics				



**Tabela 2** - Conferência ISWC: principais tópicos de pesquisa em *Web Semântica*

<b>Assunto / Ano</b>	<b>2004</b>	<b>2005</b>	<b>2006</b>	<b>2007</b>
Applications with clear lessons learned				
Evaluations of Semantic Web technologies				
Semantic Web for e-business, e-science, e-government, and other application domains				
Semantic Web technologies for multimedia content				
Personal Information Management				
Languages, tools and methodologies for representing and managing Semantic Web data				
Database technologies for the Semantic Web				
Search, query, and visualization of the Semantic Web				
Robust and scalable knowledge management and reasoning on the Web				
Machine learning and human language technologies for the Semantic Web				
Semantic Web content creation, annotation, and extraction				
Ontology creation, extraction, and evolution				
Ontology mapping, merging, and alignment				
Evaluation and ranking of ontologies				
Ontology search				
Semantic Web middleware				
Semantic Web services				
Agents on the Semantic Web				
Semantics in peer-to-peer systems and grids				
Social networks and processes on the Semantic Web				
Semantic web technology for collaboration and cooperation				
Representing and reasoning about trust, privacy, and security				
Social Software				
Evaluation of Semantic Web techniques				
The Semantic Desktop				
User-centered Semantic Web applications and/or interaction design				
Security for the Semantic Web				
Large Scale Knowledge Management				
Data Semantics				
Knowledge Portals				
Semantic Brokering				
Semantic Integration and Interoperability				
Semantic Web Mining				
Semantic Web Inference Schemes				
Semantic Web for e-Business and e-Learning				
Searching, Querying and Viewing the Semantic Web				
User Interfaces				
Visualization and Modelling				

Todas as iniciativas contam com tecnologias básicas para o funcionamento da *Web Semântica*. As próximas seções apresentam estas tecnologias para o entendimento da infra-estrutura da *Web Semântica*.

## 2.2 XML (eXtensible Markup Language)

O XML é a linguagem universal de meta-dados. Ela provê uma estrutura uniforme e um conjunto de ferramentas que possibilitam a troca de dados e meta-dados entre aplicações. As linguagens de marcação utilizam etiquetas para que as informações possam ser adicionadas aos documentos, podendo assim indicar a maneira como estas informações serão exibidas, e até mesmo, atribuir um valor (agregar semântica) às palavras contidas no texto.

A HTML é uma linguagem padrão para a codificação das páginas *Web*. Ela fornece apenas a informação de como a página *Web* deve ser exibida, não fornecendo qualquer informação semântica sobre o texto que está nela contido. Considere o seguinte exemplo de página HTML, conforme mostrado na Figura 1.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head>
    <title>Mauricio Escobar - Semantic Web Plugin</title>
  </head>
  <body>
    <h1>Mauricio da Silva Escobar</h1>
    <p>Semantic Web Plugin</p>
  </body>
</html>
```

**Figura 1** - Exemplo de representação em HTML

A figura mostra a codificação de uma página *Web*, onde são descritos somente marcadores de formatação de texto, como o “<h1>” e o “<p>”. Eles indicam somente o modo como o texto contido neles será exibido. A mesma informação pode ser representada em XML, como mostrado na Figura 2.

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

<webPage>
  <title>Mauricio Escobar - Semantic Web Plugin</title>
  <author>Mauricio da Silva Escobar</author>
</webPage>
```

**Figura 2** - Exemplo de representação em XML

Conforme a figura, podemos perceber que a descrição em XML expressa as informações sobre uma determinada página *Web*, através da definição do marcador “webPage”, e também possui os elementos “title” e “author”, que respectivamente descrevem o título e o autor da página.

Ambas as linguagens HTML e XML utilizam linguagem de marcação, sendo descrita por *tags*. Estas podem ser aninhadas (*tags* dentro de *tags*). No XML, todas as *tags* necessitam ser fechadas (por exemplo, ao abrir a *tag* <title>, a mesma deve ser fechada utilizando </title>). Em HTML, *tags* como <br>, que indicam quebra de linha, não necessitam ser fechada.

A linguagem XML oferece uma distinção da informação, separando o conteúdo da estruturação do documento. Ela possibilita a criação ilimitada de marcadores, permitindo ao usuário definir inúmeros tipos. A seguir, serão apresentados os componentes da linguagem XML.

### 2.2.1 Elementos XML

Os elementos XML representam as “coisas” sobre as quais o documento fala, como uma página *Web*, um autor, e um título. Eles formam os principais conceitos de um documento XML. Um elemento consiste de uma abertura do marcador, o conteúdo, e o fechamento do marcador. A Figura 3 apresenta um exemplo de elemento em XML.

```
<title>Mauricio Escobar - Semantic Web Plugin</title>
```

**Figura 3** - Exemplo de elemento em XML

### 2.2.2 Atributos XML

Um elemento vazio não necessariamente é sem significado, porque este pode ter alguma propriedade em termos de atributo. Um atributo é o par nome-valor dentro de um marcador aberto de um elemento. Na Figura 4 é apresentado um exemplo, onde temos os atributos “email” e “country”, ambos com seus valores.

```
<author email="mauricio.escobar@pucrs.br" country="Brazil">  
    Mauricio da Silva Escobar  
</author>
```

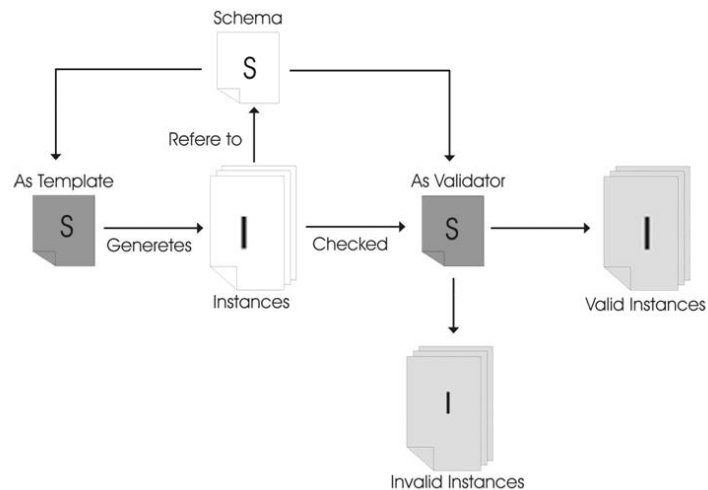
**Figura 4** - Exemplo de atributo em XML

O uso de elementos com ou sem atributos é frequentemente uma questão de gosto. No entanto, atributos não podem ser aninhados.

### 2.2.3 XML Schema

O objetivo desta seção é apresentar em linhas gerais as características básicas do XMLS, não sendo detalhados todos os componentes da linguagem. A *XML Schema* é uma linguagem que provê um meio para a definição de estruturas, conteúdo e semântica para documentos XML. Ela tornou-se recomendação da W3C em maio de 2001 (XMLS, 2001). A XMLS é análoga a um *schema* em banco de dados, onde são definidos os nomes das colunas e os tipos de dados na tabela do banco de dados.

Como mostrado na Figura 5, temos dois tipos de documento: o documento que representa o *schema* (ou documento de definição), e múltiplos documentos de instância conforme o *schema*. Ambos os documentos e instâncias do *schema* usam a sintaxe XML (marcadores, elementos e atributos).



**Figura 5** - XML *Schema* e instâncias (DACONTA, 2003)

A XMLS permite a validação das instâncias para garantir a precisão e restrição dos valores dos campos e estrutura dos documentos em tempo de criação. A precisão dos campos é verificada contra o tipo do campo; por exemplo, o campo “idade” de uma pessoa deve conter apenas números, conforme definição do tipo do campo como “integer”.

Considere o exemplo de declaração a seguir, onde é declarado um elemento e seu nome, e a restrição de valores para ele. No exemplo, é declarado o elemento chamado “author”, e seu tipo como sendo “string”, permitindo ao elemento ter atribuído qualquer valor textual.

```
<xsd:element name="author" type="xsd:string"/>
```

No documento XML de instância, o mesmo elemento poderia ser descrito como:

```
<author>Mauricio Escobar</author>
```

## 2.3 Resource Description Framework (RDF)

O XML provê uma estrutura uniforme e um conjunto de ferramentas que possibilitam a troca de dados e meta-dados entre aplicações. Entretanto, o XML não traduz

semântica, mas apenas a estrutura dos dados. Por exemplo, *tags* aninhadas não dizem nada em especial, como exemplificado na Figura 6.

Descrevendo “Paulo é professor de geografia”, em XML:

```
<curso nome="geografia">
  <professor>Paulo</professor>
</curso>
<professor nome="Paulo">
  <ensina>geografia</ensina>
</professor>
<ensina>
  <professo>Paulo</professo>
  <curso>geografia</curso>
</ensina>
```

**Figura 6** - Exemplo de estrutura de dados em XML

Imagine a seguinte situação (Figura 7):

```
<membro_academico>Maria</membro_academico>
<professor>Vitor</professor>
<curso nome="geografia">
  <ministradoPor>Paulo</ministradoPor>
</curso>
```

**Figura 7** - Exemplo 2 de estrutura de dados em XML

Qual seria a resposta para a consulta “recupere todos os membros acadêmicos”? Sintaticamente a resposta seria Maria. Semanticamente Maria, Vitor e Paulo. Conseguimos chegar a uma resposta semântica porque o ser humano é capaz de realizar a associação com base no modelo. Mas como formalizar essa semântica para o software?

### 2.3.1 RDF: Idéia Básica

O RDF é uma linguagem declarativa que fornece uma maneira padronizada de utilizar o XML para representar metadados no formato de sentenças sobre propriedades e relacionamentos entre itens na *Web* (BREITMAN, 2007). Esses itens, chamados recursos, podem ser virtualmente qualquer objeto (textos, figuras, vídeos e outros), desde que possuam uma URI.

O RDF é independente de domínio, ou seja, nenhuma informação específica de domínio é usada. A terminologia do domínio é escrita em RDFS (*RDF Schema*). Os

conceitos fundamentais do RDF são os recursos (*RDF resources*), as propriedades (*RDF properties*) e as sentenças (*RDF statements*).

### 2.3.2 Recursos

Podemos pensar em um recurso como um objeto, algo sobre a qual se deseja falar. Recursos podem ser autores, livros, lugares, pessoas, hotéis, salas, etc. Todo recurso possui uma URI (*Uniform Resource Identifier*), ou algum outro tipo de identificador que garanta a unicidade.

### 2.3.3 Propriedades

As propriedades são um tipo especial de recurso. Elas descrevem as relações entre os recursos, como por exemplo, “escrito por”, “possui idade”, “possui título”, e assim por diante. As propriedades em RDF também são identificadas por URIs. A idéia de usar URIs para identificar os recursos e as suas relações é muito importante quando se trabalha em um contexto global, como a *Web*, para que possamos garantir a unicidade de um recurso.

### 2.3.4 Sentenças

As sentenças indicam a propriedade de um recurso. Um recurso é uma tripla “objeto-atributo-valor”, que consiste de um recurso, uma propriedade e um valor. Valores podem ser tanto recursos como *literais*. Os literais são valores atômicos, como por exemplo, os tipos primitivos de dados.

Um exemplo de uma sentença é:

“*Mauricio Escobar é dono da página web <http://www.domain.com/~me>*”

A mesma sentença poderia ser descrita utilizando uma tripla RDF, como mostrado abaixo:

“http://www.domain.com/~me, http://www.domain.com/site-owner, #MauricioEscobar”

Podemos pensar da tripla  $(x, P, y)$  como uma fórmula lógica  $P(x, y)$ , onde o predicado binário  $P$  relata o objeto  $x$  para o objeto  $y$ . De fato, RDF oferece somente predicados binários (propriedades). Note que a propriedade “site-owner” é identificada por uma URL. As triplas RDF podem ser representadas em grafos, facilitando a visualização e a compreensão por humanos, como mostrado na Figura 8, que representa o exemplo anterior através de um grafo.



**Figura 8** - Grafo representando uma tripla RDF

A visão da *Web Semântica* requer representações que possam ser acessíveis e processadas computacionalmente. A linguagem RDF associa cada propriedade que o esquema define através da linguagem XML e de *namespaces* (URIs). A Figura 9 mostra um exemplo de código utilizando RDF.

```

1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.     xmlns:dc="http://purl.org/dc/elements/1.1/"
4.     xmlns:exterm="http://www.example.org/terms/">
5.   <rdf:Description rdf:about="http://www.example.org/index.html">
6.     <exterm:creation-date>August 16, 1999</exterm:creation-date>
7.   </rdf:Description>
8.   <rdf:Description rdf:about="http://www.example.org/index.html">
9.     <dc:language>en</dc:language>
10.  </rdf:Description>
11. </rdf:RDF>
  
```

**Figura 9** - Exemplo de código RDF (RDF, 2004)

O código é iniciado com a declaração do prólogo, indicando que o RDF é baseado em XML. As linhas 2 a 11 contêm as expressões RDF e utilizam o vocabulário definido pelos *namespaces* especificados por “*xmlns*” (linhas 2, 3 e 4). O *namespace* na linha 2 refere-se à sintaxe padrão do RDF e os da linha 3 e 4 referem-se ao vocabulário



especificado pelas URLs descritas. A tag *<description>* (linha 5) é utilizada para referenciar o recurso da *Web* que terá seus meta-dados descritos em “<http://www.example.org/index.html>”.

### 2.3.5 RDF Schema (RDFS)

O RDF é uma linguagem universal que possibilita aos usuários a descrição de recursos usando um vocabulário próprio. Ela é genérica e não presume nenhum domínio, e para a especificação da semântica de um domínio, usa-se o *RDF Schema*. O RDFS (RDFS, 2004) também permite especificar os elementos relacionados a um domínio de discurso. Ele permite a declaração de classes de indivíduos, ou seja, conjuntos de indivíduos.

Neste trabalho não serão apresentadas todas as definições da linguagem, mas somente as principais para que possamos compreender como funciona a representação de conhecimento em RDF.

#### 2.3.5.1 Classes e Propriedades

Uma classe pode ser vista como um conjunto de elementos. Objetos individuais que pertencem a uma classe são referenciados como *instâncias* da classe, e descrevem objetos que compartilham as mesmas características. As propriedades descrevem as relações comuns a um conjunto de objetos.

#### 2.3.5.2 Herança

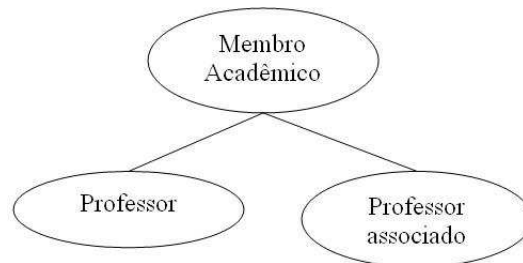
O conceito de herança em RDF é similar a OO (paradigma orientado a objetos). Por exemplo, todos os professores são membros acadêmicos. Podemos dizer que

“professor” é uma *subclasse* de “membro acadêmico”, ou a equivalência, onde “membro acadêmico” é *superclasse* de “professor”, conforme ilustrado na Figura 10.

```
<rdfs:Class rdf:about="Professor">
  <rdfs:subClassOf rdf:resource="FuncionarioAcademico">
</rdfs:Class>
```

**Figura 10** - Sintaxe da linguagem RDF para declaração de classe e subclasse

A relação de subclasse define uma hierarquia de classes como mostrado na Figura 11. Uma classe pode ter múltiplas superclasses (Herança múltipla). Sendo neste exemplo, a classe “Professor” e “Professor associado” como sendo subclasse de “Membro acadêmico”.



**Figura 11** – Exemplo de hierarquia de classes

## 2.4 Web Ontology Language (OWL)

As ontologias são desenvolvidas com o intuito de facilitar o compartilhamento e reuso de informações (FENSEL, 2001). Dentre as várias definições para o termo ontologia existentes na literatura, a mais citada é a oferecida por Thomas Gruber (GRUBER, 1993), onde: “Uma ontologia é uma especificação explícita de uma conceituação”. Na *Web Semântica*, é através do uso de ontologias que a semântica de um domínio é representada.

Em geral uma ontologia descreve formalmente um domínio de discurso (ANTONIOU e HARMELEN, 2004). Tipicamente, uma ontologia consiste de uma lista finita de termos e relacionamentos entre estes termos. Os termos denotam importantes conceitos (classes de objetos) do domínio. Por exemplo, no âmbito de uma Universidade, professores, alunos, administradores, cursos e disciplinas são conceitos importantes.

No contexto da *Web Semântica*, as ontologias provêm uma *compreensão compartilhada de um domínio*. Para tal compartilhamento, é necessária a superação de diferenças nas terminologias. Por exemplo, diferentes aplicações podem fazer uso de um mesmo termo de formas distintas. Este problema pode ser resolvido pela definição de um mapeamento direto entre ontologias. O mapeamento entre ontologias ainda é um dos grandes desafios de pesquisa em ontologias e *Web Semântica*.

As ontologias são úteis ainda para melhorar a precisão de buscas na *Web*. Os mecanismos de busca podem olhar para as páginas que se referenciam a um *conceito* preciso em uma ontologia ao invés de coletarem todas as páginas que contêm ocorrências deste conceito, geralmente de forma ambígua, por ocorrência de palavras-chave. Desta maneira, diferenças entre terminologias de páginas *Web* e buscas podem ser superadas.

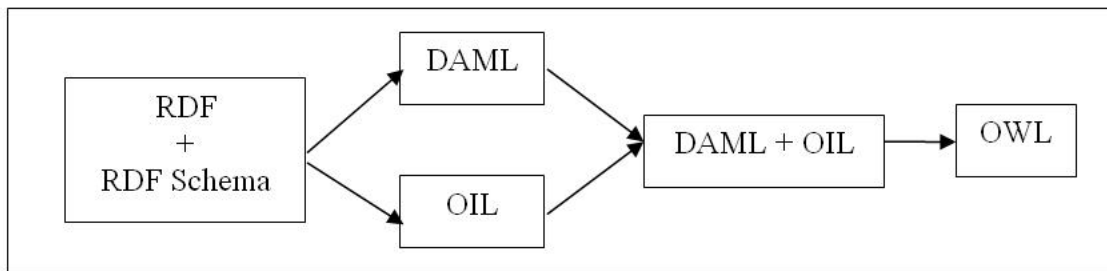
Para se definir e manipular ontologias sugere-se a utilização de linguagens que suportem estruturas para a representação do conhecimento. Esta representação é realizada através da descrição formal de um conjunto de termos sobre um domínio específico. A definição de uma linguagem é necessária para a representação e descrição formal da estrutura que especifica uma conceituação.

Existem várias formas de representar ontologias por meio de linguagens de marcação. Dessa forma, são necessários alguns dos seguintes requisitos para uma linguagem de definição de ontologias (ANTONIOU e HARMELEN, 2004):

- Uma sintaxe bem definida;
- Uma semântica formal;
- Conveniência de expressão;
- Suporte ao raciocínio;
- Poder de expressividade.

A linguagem OWL (OWL, 2004) é recomendada pela W3C para a descrição de ontologias. Ela é projetada para ser usada quando a informação contida em um documento precisa ser processada por aplicações. Ela apresenta maior expressividade semântica do que o XML, o RDF e o RDF *Schema*, fornecendo um vocabulário amplo para a descrição de propriedades e classes. Nela existem marcadores para a definição de: relações entre classes, cardinalidade, igualdade, tipagem de propriedades, características de propriedades, classes enumeradas, entre outros.

A sintaxe OWL é baseada em XML e RDF/RDF *Schema* e incorpora uma revisão sobre a linguagem DAML + OIL. A Figura 12 mostra a origem da OWL.



**Figura 12** - origem da Linguagem OWL

A OWL dispõe de três sublinguagens com níveis crescentes de expressividade. O *OWL Full* representa a linguagem completa do OWL, possibilitando o uso de todas as primitivas da linguagem. Possibilita a combinação de suas primitivas arbitrariamente com RDF e *RDF Schema*. Permite portanto, alterar o significado de primitivas pré-definidas, tanto em RDF quanto em OWL.

Já *OWL DL* é mais focada para usuários que buscam eficiência para processamento computacional. Esta sublinguagem restringe a forma como os construtores de OWL e RDF devem ser usados com o objetivo de aumentar a eficiência computacional.

A *OWL Lite* apóia usuários que necessitem, primeiramente, de uma classificação hierárquica e restrições simples. As restrições do subconjunto de construtores da linguagem OWL DL também são válidas para *OWL Lite*. Além deste subconjunto, também são excluídas as cardinalidades arbitrárias, enumeração das classes e declarações de disjunções. A vantagem na utilização de *OWL Lite* é pela simplicidade e facilidade de implementação, principalmente com ferramentas para construção de ontologias. A desvantagem, por outro lado, é a clara restrição na expressividade.

Um documento OWL pode ser estruturado através de:

- **Classes:** conjunto de instâncias com características comuns.
  - *Superclasse:* como as classes podem ser organizadas hierarquicamente, as instâncias diretas das subclasses são também instâncias das superclasses.
  - *Relacionamentos:* as classes podem ser sobrepostas arbitrariamente.
  - *Disjunções:* todas as classes podem potencialmente se sobrepor, porém em muitos casos é necessário fazer com que estas classes não compartilhem instâncias.

- **Propriedades:**
  - *Tipos*: identificam os valores primitivos das instâncias, como *integer, float, string, boolean, etc.*
  - *Objetos*: representam o vínculo de duas instâncias, isto é, seus relacionamentos.
  - *Inversa*: representam um relacionamento bidirecional. Adicionando valores a uma propriedade, conseqüentemente, adicionam-se valores em uma segunda.
  - *Transitivas*: Se a instância *x* está relacionada com a instância *y*, e a instância *y* está relacionada à instância *z*, então *x* está relacionado com *z*. Usado principalmente em relações “parte-de”.
- **Instâncias**: representam os objetos em um domínio, isto é, coisas específicas. Verifica-se aqui que dois nomes podem representar o mesmo objeto no mundo real.

Uma classe em OWL é definida usando o elemento *owl:Class*. Por exemplo, podemos definir classes como na Figura 13, onde são definidas as classes “Winery”, “Region e “ConsumableThing”.

```
<owl:Class rdf:ID="Winery"/>
<owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="ConsumableThing"/>
```

**Figura 13** - Exemplo de classe em OWL (OWL, 2004)

Já a Figura 14 define uma classe “PortableLiquid” como sendo uma subclasse de “ConsumableThing”, através do elemento *subClassOf*.

```
<owl:Class rdf:ID="PortableLiquid">
  <rdfs:subClassOf rdf:resource="#ConsumableThing" />
  ...
</owl:Class>
```

**Figura 14** - Exemplo de subclasse em OWL (OWL, 2004)

Uma propriedade é uma relação binária. Dois tipos de propriedades são distinguidos:

- *Datatype properties*: relações entre instâncias de classes e literais RDF e *XML Schema datatypes*.

- *Object properties*: relações entre instâncias de duas classes.

A Figura 15 ilustra a definição de propriedades. Neste exemplo, são declarados os conceitos “SistemaMultiAgentes” e “Agentes”, e eles são relacionados utilizando a propriedade “compostoPor”.

```
<owl:Class rdf:ID="Agentes"/>
<owl:Class rdf:ID="SistemaMultiAgente"/>
<owl:ObjectProperty rdf:ID="compostoPor">
  <rdfs:range rdf:resource="#Agentes"/>
  <rdfs:domain rdf:resource="#SistemaMultiAgente"/>
</owl:ObjectProperty>
</rdf:RDF>
```

**Figura 15** - Exemplo de propriedade em OWL

As ontologias estabelecem uma terminologia comum usada tanto por humanos quanto por agentes de software para o entendimento dos conceitos de um domínio (RIBEIRO, 2002). Na próxima seção, serão apresentados os conceitos referentes a Agentes e Sistemas Multiagentes.

## 2.5 Agentes e Sistemas Multiagentes

A tecnologia de agentes de software pode ser vista como uma abordagem complementar ao paradigma de objetos no desenvolvimento de sistemas de software complexos. Esta nova tecnologia propõe um nível de abstração mais alto que aquele proposto pela tecnologia de objetos, uma vez que o comportamento dos agentes situa-se mais próximo do comportamento de seres humanos, cujo trabalho em geral é substituído ou suportado por sistemas de software.

Um agente de software, segundo Weiss (WEISS, 1999), pode ser definido como um sistema de computador situado em algum ambiente e capaz de agir de forma autônoma para atingir um objetivo, onde a autonomia refere-se a capacidade de agir de acordo com sua própria linha de controle.

Na *Web Semântica*, agentes de software são usados como entidades capazes de consumir automaticamente conteúdos publicados. Assim, a *Web Semântica* pode ser vista como um sistema multiagentes global formado pela relação de um grande número de sociedades de agentes. Na literatura são encontrados diversos trabalhos que citam a

contribuição e uso de agentes de software na *Web Semântica*, como (HENDLER, 2001) e (LI, 2002), e trabalhos que visam apoiar o desenvolvimento de aplicações deste tipo, como (GUO *et al.*, 2005) e o SemantiCore, que será apresentado mais adiante.

A *Web Semântica* é baseada na idéia de dinamicidade, heterogeneidade, e fontes de conhecimento compartilhadas provendo conteúdo processável por máquina. Agentes de software podem usar este conhecimento para alcançar seus objetivos, produzindo novos conhecimentos que podem ser disseminados ou publicados dentro de uma *framework* comum. A *Web Semântica* pode beneficiar-se de agentes autônomos e distribuídos, responsáveis por coletar ou agregar conhecimento, raciocinando ou inferindo novos fatos, identificando e gerenciando inconsistências, e provendo mecanismos de confiança e segurança.

Algumas das características básicas presentes em agentes são: autonomia, pró-atividade e interatividade. A autonomia consiste na capacidade do agente agir sem intervenção externa; a pró-atividade indica que o agente pode atuar de forma que suas ações não necessariamente sejam de resposta ao ambiente e a interatividade permite que ele possa se comunicar com outros agentes e com o ambiente em que está presente.

Outras características podem ser encontradas em agentes:

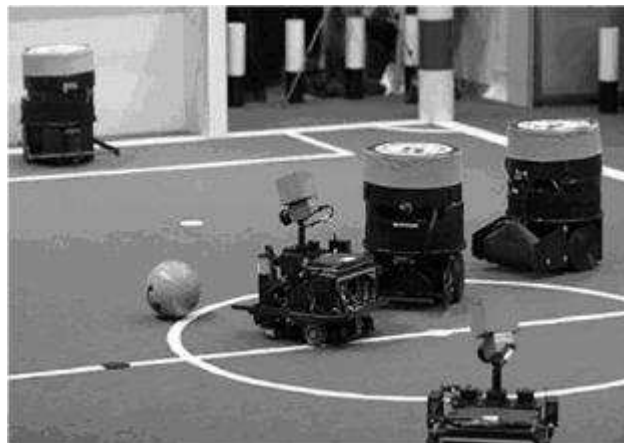
- Adaptação: são capazes de modificar, em algum grau, os seus comportamentos devido às mudanças de ambiente e de outros agentes.
- Aprendizado: são capazes de modificar o seu comportamento baseado na sua experiência.
- Racionalidade: são capazes de selecionar suas ações com base em seus objetivos.
- Mobilidade: são capazes de se mover de um ambiente para outro.
- Sociabilidade: um agente pode interagir com outro agente.

A definição precisa do que é realmente um agente de software é um dos temas mais debatidos e controversos na área de ciência da computação. Um agente de software pode ser definido de uma maneira simples como um objeto complexo com atitude (BRADSHAW, 1997). Mais especificamente, um agente pode ser encarado como uma entidade autônoma que possui uma série de capacidades e objetivos definidos, que interage com um ambiente e com outros agentes e é capaz, através destas interações, de adaptar seu estado e comportamento.

Assim como a definição de agentes de software, a definição de sistemas multiagentes também é diferenciada na literatura (FERBER, 1999; WEISS, 1999; WOOLDRIDGE e JENNINGS, 1999). Alguns autores consideram um sistema multiagentes como uma coleção de agentes que se coordenam através de suas interações. Segundo estes autores, um SMA só existe se existe coordenação entre os agentes que o formam. Para outros autores a definição de SMA deve incorporar um conjunto de agentes que interagem entre si e com objetos em um ambiente (WOOLDRIDGE e JENNINGS, 1999).

O desenvolvimento orientado a agentes tem como principais objetivos: a definição das entidades que serão representadas no sistema (domínio); as percepções e ações que cada agente pode realizar; a representação das crenças e objetivos e a definição dos relacionamentos de comunicação entre os agentes (protocolo de comunicação).

Os sistemas multiagentes estão ligados com o comportamento de uma coleção de agentes autônomos (possivelmente pré-existent) focados em resolver um determinado problema. Eles formam uma espécie de rede para resolução de problemas que estão além de suas capacidades individuais. Um sistema que consiste de um grupo de agentes que potencialmente interagem entre si é chamado de Sistema Multiagentes. A Figura 16 mostra um exemplo de uso de sistema multiagentes. No exemplo, temos robôs jogando futebol, onde cada robô é controlado por um agente de software.



**Figura 16** - Robôs controlados por agentes de software (VLASSIS, 2007)

Em um sistema multiagentes, os protocolos de comunicação permitem aos agentes trocarem e entenderem mensagens. Os protocolos de interação permitem aos agentes terem conversação, por meio de trocas estruturadas de mensagens. Na resolução distribuída de problemas, o conceito de agente é usado para simplificar a solução do problema. Desta



forma, os agentes são distribuídos em unidades colaborativas responsáveis pela resolução de tarefas específicas (VLASSIS, 2007).

## 2.5.1 Propriedades e Características de um Sistema Multiagentes

### 2.5.1.1 Sociedade de agentes

Em um SMA, os agentes podem ser organizados em sociedades. Um grupo de agentes pode formar uma pequena sociedade no qual desempenham diversos papéis. O grupo define os papéis, e os papéis definem os compromentimentos associados a ele (WEISS, 1999). Quando um novo agente junta-se a um grupo, a ele é associado um ou mais papéis, e ele adquire o comprometimento para aquele papel. Um agente junta-se a um grupo autonomamente, mas ele deve estar consciente dos compromentimentos associados ao papel adotado. Os grupos definem também o contexto social, no qual o agente interage.

O conceito de sociedades de agentes envolve abstrações da sociologia e teorias organizacionais. A sociabilidade é importante para a cooperação, que promove a mudança do paradigma cliente-servidor para um paradigma flexível e distribuído (*peer-to-peer*) que as modernas aplicações necessitam, e onde a tecnologia de agentes pode encontrar seu grande potencial.

Uma sociedade pode ser classificada sobre os seguintes aspectos:

- Homogênea: possui agentes do mesmo tipo (arquitetura).
- Heterogênea: agentes de tipos diferentes (arquitetura).
- Fechadas: agentes fixos no ambiente.
- Abertas: agentes podem migrar entre ambientes.
- Baseada em leis: regras explícitas de comportamento para todos os agentes.
- Sem leis: sem regras explícitas de comportamento para todos os agentes.

### 2.5.1.2 Cooperação

Os agentes recebem problemas com certo nível de abstração. Cada agente deve então decompor o problema em subproblemas que lhes digam respeito e com as quais possa lidar, de acordo com os recursos e os conhecimentos que possui ou possa buscar com outros agentes.

Em um ambiente com recursos limitados, os agentes devem coordenar suas atividades para cumprirem seus próprios interesses ou para satisfazerem os objetivos do grupo (WEISS, 1999). As ações de múltiplos agentes necessitam ser coordenadas porque existem dependência entre estas ações, e existe também a necessidade de manter as restrições globais, e nenhum agente tem a competência, recursos ou informações suficientes para atingir sozinho, um objetivo de sistema.

Para produzir sistemas de coordenação, muitas pesquisas em inteligência artificial têm se concentrado em técnicas para distribuição de controle e dados. O controle distribuído significa que os agentes têm um grau de autonomia na geração de novas ações, e podem decidir qual objetivo perseguir. A desvantagem da distribuição de controle e dados é que o conhecimento geral do sistema fica disperso, e cada agente possui uma imprecisa e parcial perspectiva do sistema. Existe então, um aumento de incerteza sobre as ações dos agentes, e, dessa forma, torna-se difícil a obtenção de um comportamento global coerente.

Podemos dizer que agentes estão cooperando, se eles assumem ações em comum após identificarem e adotarem um objetivo comum (FERBER, 1999). Em um ambiente cooperativo, a troca de informações entre agentes é fundamental, existindo assim, um compartilhamento da informação. É possível distinguir de duas formas a cooperação:

- Partilha de resultados: ocorre após a conclusão de um objetivo. O agente verifica se existem outros agentes interessados nas informações provenientes do alcance de seu objetivo.
- Partilha de tarefas: ocorre quando um agente detecta que não tem capacidade ou informação suficiente para a execução de determinada tarefa. Mas para isso, o agente deve verificar se existem outros agentes com “interesse” em ajudar. Esta partilha pode ser vista como o balanceamento de carga computacional do sistema.

### 2.5.1.3 Coordenação

A cooperação acontece quando vários agentes planejam e executam suas ações de uma forma coordenada. O problema da coordenação é o gerenciamento das *interdependências entre as atividades desempenhadas pelos agentes* (WOOLDRIDGE, 2002): mecanismos de coordenação são essenciais se as atividades que uma agente possui ocasionam a interação de alguma maneira.

A coordenação de agentes visa garantir que:

- Todas as partes componentes de um problema estejam incluídas nas atividades de pelo menos um agente.
- Os agentes interajam de forma a permitir que suas atividades sejam desenvolvidas e integradas no sentido de uma solução global.
- Os membros do grupo de trabalho atuem de forma determinada e consistente.
- O grupo de agentes respeite restrições globais a solução do problema.
- Existam procedimentos que garantam a harmonia na execução de uma única ação de forma conjunta por mais de um agente.

Alguns requisitos são fundamentais para permitir a coordenação:

- A comunicação entre os agentes.
- O reconhecimento das potenciais interações entre os planos de ação dos agentes.
- E a capacidade de negociação entre os agentes.

### 2.5.1.4 Comunicação

Em sistemas multiagentes, assim como para os humanos, a comunicação é a base para interações sociais e organizacionais (FERBER, 1999). Sem comunicação, o agente é meramente um indivíduo isolado, surdo e mudo para outros agentes, fechado em seu ciclo

de percepção-deliberação-ação. Isso porque, basicamente os agentes que se comunicam podem possivelmente cooperar e coordenar suas ações, entre outros aspectos.

Segundo Vlassis (VLASSIS, 2007), interações são frequentemente associadas com alguma forma de comunicação. A comunicação em um SMA envolve o envio e recebimento de mensagens entre os agentes. A comunicação pode ser usada em diversos casos, como por exemplo, para a coordenação entre agentes cooperativos ou para negociação entre agentes. Entretanto, a comunicação adicionalmente gera o problema de saber qual o protocolo de rede deve ser utilizado, para permitir a troca de informações, e em qual *linguagem* o agente deve falar para entender os outros agentes (especialmente em ambientes heterogêneos).

Existem muitas formas de comunicação encontradas em sistemas multiagentes, dentre as mais utilizadas, podemos citar os (i) atos de fala, (ii) conversações, e (iii) linguagens de comunicação entre agentes, como o KQML (KQML, 2008) e o FIPA ACL (FIPA ACL, 2002).

Dentre as formas de comunicação, podemos citar duas estratégias comuns para a comunicação entre agentes, que são:

- Comunicação através de troca de mensagens
  - Comunicação direta, onde os agentes lidam com sua própria coordenação.
  - Coordenação assistida, na qual os agentes dependem de agentes especiais para a coordenação.
- *Blackboard*
  - É um repositório no qual os agentes escrevem mensagens, armazenam resultados parciais e obtêm informações. Normalmente é particionado em vários níveis de abstração, de acordo com o problema em questão.

## 2.6 SemantiCore

Com o objetivo de permitir o desenvolvimento de SMAs para a *Web*, incorporando os requisitos necessários à criação da *Web Semântica*, foi desenvolvido em

meados de 2002 a arquitetura *Web Life* (RIBEIRO, 2002). Nessa época, falava-se muito sobre a *Web Semântica*, mas existiam poucas ferramentas que permitiam (ou ao menos objetivavam) que os seus requisitos fossem implementados.

O SemantiCore, apresentado inicialmente em 2004 (BLOIS e LUCENA, 2004), surgiu a partir de uma extensão na arquitetura *Web Life*. O SemantiCore é estruturado como um *framework* que abstrai a plataforma de implementação e provê primitivas para a criação de aplicações organizadas em um conjunto de agentes que realizam suas tarefas no ambiente *Web*.

Com o surgimento de novas tecnologias, bem como com o aprimoramento das já existentes, foi apresentado o SemantiCore 2006 (ESCOBAR *et al.*, 2006). Uma das principais inovações presentes na nova versão diz respeito ao modelo de arquitetura do agente. No SemantiCore 2006, agentes de software podem ser descritos através de uma estrutura ontológica. Ontologias são usadas para definir os conceitos de forma que os membros de uma comunidade ou domínio tenham o mesmo entendimento.

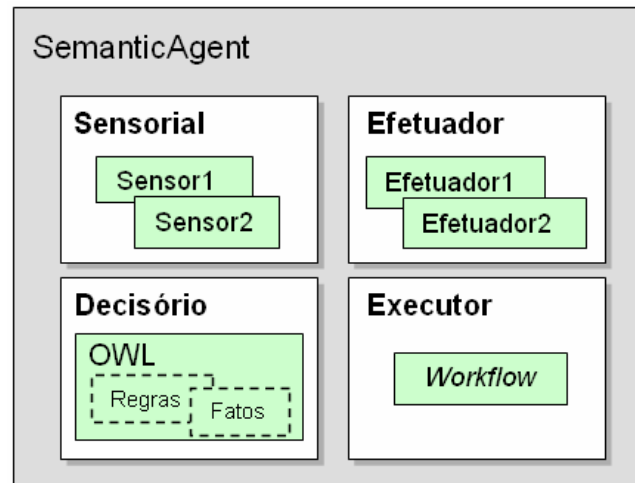
O SemantiCore é dividido em dois modelos: o modelo do agente e o modelo do domínio semântico. Os dois modelos dispõem de pontos de flexibilidade (*hotspots*) permitindo aos desenvolvedores associarem diferentes padrões, protocolos e tecnologias.

### **2.6.1 O Modelo de Agente**

O modelo do agente possui uma estrutura orientada a componentes, onde cada componente contribui para uma parte essencial do funcionamento do agente, agregando todos os aspectos necessários a sua implementação. Com a retirada de um ou mais componentes não relacionados ao desempenho das tarefas do agente é possível simplificar a sua arquitetura.

Os agentes são compostos basicamente por quatro componentes (Figura 17). Cada componente é responsável por tarefas especializadas, onde cada componente é um processo de software independente, e que colaboram entre si como um grupo. A estrutura de componentes do agente ajuda o desenvolvedor a focalizar o agente em partes. Esta modularização oferece benefícios em termos de manutenção e organização do código. O

padrão de projeto *Factory* (GAMMA *et al.*, 1994) é usado para encapsular as instanciações que o agente necessita ao ser iniciado, isto é, a implementação dos pontos de flexibilidade.



**Figura 17** - Arquitetura de um agente no Semanticore (ESCOBAR *et al.*, 2006)

Para perceber e capturar recursos que trafegam pelo ambiente, o agente possui o Componente Sensorial (*Sensorial Component*). Este componente agrupa os elementos que são habilitados à recepção de objetos através do ambiente. O componente sensorial armazena os diversos sensores definidos pelo desenvolvedor e também verifica se algum destes sensores deve ser ativado pelo recebimento de um objeto semântico do ambiente. Se um ou mais sensores forem ativados, os objetos são enviados para outros componentes para processamento.

O desenvolvedor pode definir diferentes tipos de sensores para capturar diferentes tipos de objetos no ambiente. Um sensor RDF (*RDF Sensor*) é um tipo especial de sensor que captura objetos RDF no ambiente.

O Componente Decisório (*Decision Component*) encapsula o mecanismo de tomada de decisão do agente. O mecanismo decisório presente no componente é um dos pontos de flexibilidade do SemantiCore. O componente decisório opera sobre ontologias que permitem a definição de conceitos, também como um conjunto de relações, restrições, axiomas, instâncias e vocabulários. Torna-se necessário o uso de uma linguagem apropriada para a definição semântica dos dados, como OWL.

Para que a saída gerada pelo componente decisório possa ser entendida, ela deve ser uma instância de uma Ação (*Action*). As ações mapeiam todos os possíveis comandos que um agente deve entender para trabalhar de forma apropriada. As ações podem

ser aplicadas internamente aos elementos do agente ou aos elementos do ambiente. O desenvolvedor pode definir suas próprias ações através da extensão da classe *Action (hotspot)* presente no *framework*.

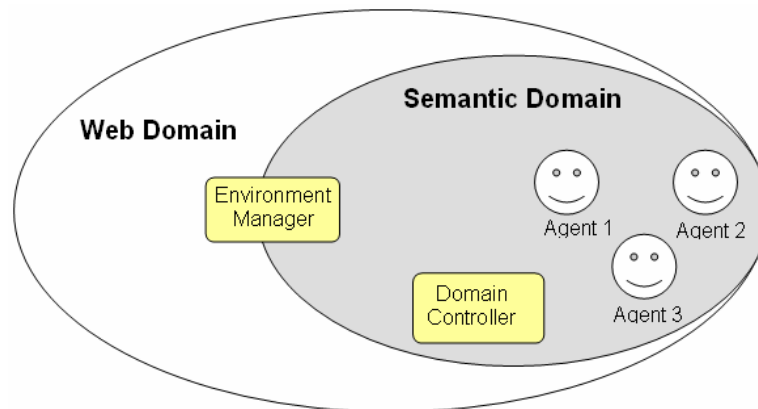
O Componente Executor (*Execution Component*) contém os planos de ação que serão executados pelo agente. Este componente trabalha com o mecanismo de *workflow*, e ele é necessário para o controle das transições de atividades dentro de um processo do *workflow*.

Toda publicação de um objeto semântico no ambiente requer um efetuator apropriado no agente. Os efetutores são controlados pelo Componente Efetuator (*Effector Component*) do agente. Este componente recebe dados dos outros componentes e encapsula estes em objetos para serem transmitidos no ambiente. Assim como o mecanismo sensorial, o efetuator abstrai a representação do recurso, permitindo assim, diferentes representações para este conforme a necessidade. Os objetos são encapsulados em mensagens para que possam trafegar pelo ambiente.

## 2.6.2 Modelo de Domínio

Para que um agente possa atuar, é necessário ele estar dentro de um ambiente. A camada de abstração do *SemantiCore* define o lugar onde um agente executa como um **domínio semântico**. Um domínio semântico requer um domínio *Web* para operar. Ele pode ser pensado como uma região no domínio *Web* no qual os agentes vivem. Um domínio contém sociedades de agentes que são organizadas para alcançar um objetivo.

A Figura 18 ilustra uma representação de domínio semântico. Cada domínio semântico é composto por algumas entidades administrativas, tais como o *Domain Controller* e *Environment Manager*. O *Domain Controller* é o agente responsável pelo registro dos agentes no ambiente. Este agente também é responsável pelas características de segurança e pela recepção de agentes móveis vindos de outros domínios. O *Environment Manager* é uma ponte entre o domínio do *SemantiCore* e os domínios *Web* convencionais.



**Figura 18** - Representação do Modelo de Domínio (ESCOBAR *et al.*, 2006)

### 2.6.3 Exemplo de Utilização

Nesta seção será apresentado um exemplo de aplicação construída utilizando-se o *framework* Semanticore 2006. Este exemplo busca ilustrar os aspectos básicos envolvidos na criação de um agente no SemantiCore, cobrindo todos os elementos presentes no modelo de agente, conforme apresentado anteriormente na seção 2.6.1, e a troca de mensagens entre os agentes.

O exemplo consiste na troca de mensagens entre três agentes, sendo eles o (i) *AgResourceManager*, (ii) *AgMachine1* e (iii) *AgMachine2*. O primeiro agente é responsável por coletar informações de *hardware* sobre os computadores de todos os agentes presentes no domínio. As informações a serem coletadas são: a capacidade de memória virtual e o número de processadores da máquina onde o agente está situado. Após a coleta das informações, o agente (i) é responsável por exibir os resultados no console da aplicação. Nas próximas seções serão detalhados os aspectos de cada agente, e a execução do exemplo.

#### 2.6.3.1 O agente Gerenciador de Recursos

O agente Gerenciador de Recursos (*AgResourceManager*) possui um plano de ação que é responsável por realizar a busca das informações de *hardware*. A execução deste



plano envolve o envio de uma mensagem para os agentes do domínio, o recebimento das respostas, e a apresentação de um resultado para o usuário do sistema, neste caso, o resultado compreende a listagem das máquinas e suas informações de *hardware*, conforme descrito anteriormente.

O agente gerenciador de recursos possui um mecanismo decisório (*AgResourceManagerDecision*), um sensor (*AgResourceManagerSensor*), um mecanismo executor (*AgResourceManagerExecution*), um plano de ação (*AgResourceManagerPlan*) e uma ação (*AgResourceManagerAction*).

O sensor é responsável por receber mensagens do ambiente e repassar para o decisório. O decisório por sua vez, é responsável pela tomada de decisão do agente, e neste caso, verifica se uma mensagem recebida contém o texto “machine\_info” no campo assunto. Essa mensagem contém a resposta de um agente perante a uma solicitação de informações de *hardware*. A seguir é apresentado o código fonte da classe que implementa o *AgResourceManager*.

```

1  package agent.resourcemanager;
2
3  import semanticore.agent.execution.model.ActionPlan;
4  import semanticore.domain.Environment;
5  import semanticore.domain.model.SemanticAgent;
6
7  public class AgResourceManager extends SemanticAgent {
8  public AgResourceManager(Environment env, String agentName, String arg) {
9  super(env, agentName, arg);
10 }
11
12 protected void setup() {
13 addSensor(new AgResourceManagerSensor("AgResourceManagerSensor"));
14 setDecisionEngine(new AgResourceManagerDecision());
15 setExecutionEngine(new AgResourceManagerExecution());
16 ActionPlan actionPlanAgResourceManager =
17 addActionPlan("AgResourceManagerActionPlan");
18 actionPlanAgResourceManager.addAction(new AgResourceManagerAction());
19
20 getExecutionComponent().getExecutionEngine().startActionPlan(
21 actionPlanAgResourceManager, null);
22 }
23 }

```

O método *setup* contém o código necessário para a inicialização do agente. Após sua inicialização, o agente gerenciador de recursos realiza a execução do seu plano de ação. Este plano contém uma única ação, responsável por realizar o envio da solicitação de informações de *hardware* para os demais agentes do domínio. A seguir é apresentada a classe que implementa o mecanismo decisório.

```

1  package agent.resourcemanager;
2
3  import java.util.Vector;
4
5  import semanticore.agent.decision.DecisionEngine;
6  import semanticore.domain.model.SemanticMessage;

```

```

7  import agent.common.MachineInfo;
8
9  public class AgResourceManagerDecision extends DecisionEngine {
10
11     public Vector decide(Object msg) {
12         SemanticMessage message = (SemanticMessage) msg;
13         if (message.getSubject().equals("machine_info"))
14             if (message.getContent() instanceof MachineInfo) {
15                 System.out.println("Agent: " + message.getFrom());
16                 System.out.println(message.getContent().toString());
17             }
18         }
19         return null;
20     }
21 }

```

No método *decide* é implementada a lógica decisória do agente. O decisório verifica se o assunto da mensagem contém o texto “machine\_info”. Caso o texto esteja contido, isto indica o recebimento de uma resposta à solicitação de informações por um dos agentes do domínio. Após a verificação, as informações são exibidas no console da aplicação.

Toda mensagem de resposta que é recebida possui em seu campo “conteúdo” um objeto do tipo *MachineInfo*. Este objeto encapsula as informações de *hardware*, e contém os atributos *quantidade de memória* e *número de processadores*.

### 2.6.3.2 Demais agentes do sistema

Os agentes *AgMachine1* e *AgMachine2* são instâncias da classe *AgMachine*, que implementa um agente no *SemantiCore*. A esta classe de agente, existem associados um mecanismo decisório (*AgMachineDecision*), um sensor (*AgMachineSensor*), um mecanismo executor (*AgMachineExecution*), um plano de ação (*AgMachineActionPlan*) e uma ação (*AgMachineAction*).

O sensor é responsável por receber mensagens do ambiente e repassar para o decisório. O decisório é responsável pela tomada de decisão do agente, e neste caso, verifica se uma mensagem recebida contém o texto “get\_machine\_info” no campo assunto. A seguir serão apresentados os códigos-fonte das classes *AgMachine*, *AgMachineSensor* e *AgMachineDecision*.

```

1  package agent.machine;
2
3  import semanticore.domain.model.SemanticAgent;

```

```

4 import semanticore.domain.Environment;
5 import semanticore.agent.execution.model.ActionPlan;
6
7 public class AgMachine extends SemanticAgent {
8     public AgMachine(Environment env, String agentName, String arg) {
9         super(env, agentName, arg);
10    }
11    protected void setup() {
12        addSensor(new AgMachineSensor("AgMachineSensor"));
13        setDecisionEngine(new AgMachineDecision());
14        setExecutionEngine(new AgMachineExecution());
15        ActionPlan actionPlanAgMachine = addActionPlan("AgMachineActionPlan");
16        actionPlanAgMachine.addAction(new AgMachineAction());
17    }
18 }

```

O método *setup* contém o código necessário para a inicialização do agente.

```

1 package agent.machine;
2
3 import semanticore.agent.sensorial.Sensor;
4 import semanticore.domain.model.SemanticMessage;
5
6 public class AgMachineSensor extends Sensor {
7     public AgMachineSensor(String arg0) {
8         super(arg0);
9     }
10
11    public Object evaluate(Object dadosDoAmbiente) {
12        if (dadosDoAmbiente instanceof SemanticMessage)
13            return dadosDoAmbiente;
14        return null;
15    }
16 }

```

Através do método *evaluate* uma informação é recebida do ambiente. Após o recebimento de uma informação, o sensor verifica se ela é uma mensagem (classe *SemanticMessage*). Caso seja uma mensagem, esta é enviada para processamento pelo componente decisório. A seguir é apresentada a classe que implementa o mecanismo decisório.

```

1 package agent.machine;
2
3 import java.util.Vector;
4
5 import semanticore.agent.decision.DecisionEngine;
6 import semanticore.domain.actions.lib.ExecuteProcessAction;
7 import semanticore.domain.model.SemanticMessage;
8
9 public class AgMachineDecision extends DecisionEngine {
10
11    public Vector decide(Object information) {
12        Vector result = new Vector();
13        String subject = ((SemanticMessage) information).getSubject();
14        if (subject.equalsIgnoreCase("get_machine_info"))
15            result.add(new ExecuteProcessAction("ActionPlanGetInfo"));
16        return result;
17    }
18 }

```

O método *decide*, herdado da classe *DecisionEngine*, contém o código responsável pela tomada de decisão do agente. Neste exemplo, a tomada de decisão do *AgMachine* é verificar se o assunto da mensagem contém o texto “get\_machine\_info”. Caso contenha o texto, ele envia um sinal (objeto *ExecuteProcessAction*) para o componente executor, solicitando a execução do plano de ação “ActionPlanGetInfo”.

O plano de ação “ActionPlanGetInfo” contém uma única ação, que é responsável por recuperar as informações de hardware da máquina local e enviá-las para o agente solicitante. A seguir é apresentado o código-fonte da ação.

```

1  package agent.machine;
2
3  import semanticore.agent.execution.model.Action;
4  import semanticore.domain.model.SemanticMessage;
5  import agent.common.MachineInfo;
6
7  public class AgMachineAction extends Action {
8
9      public AgMachineAction() {
10         super("AgMachineAction", null, null);
11     }
12
13     public void exec() {
14         long totamMemory = Runtime.getRuntime().totalMemory();
15         int processors = Runtime.getRuntime().availableProcessors();
16         MachineInfo info = new MachineInfo(totamMemory, processors);
17         SemanticMessage reply =
18             new SemanticMessage("machine_info", getOwner()
19                 .getName(), "AgResourceManager", info);
20         transmit(reply);
21     }
22 }
23

```

O método *exec* contém o código necessário para a execução da ação. A ação mostrada realiza a coleta de informações relativas à memória virtual disponível e o número de processadores da máquina, ambos estes comandos estão disponíveis através da API do Java. As informações recuperadas são encapsuladas em um objeto, definido através de classe *MachineInfo*, e este objeto é enviado via mensagem para o agente *AgResourceManager*, através do método *transmit* presente na classe *Action*. O campo assunto da mensagem contém o texto “machine\_info”, que permitirá ao agente Gerenciador de recursos identificar que esta é uma mensagem de resposta a sua solicitação.

O método *transmit* enviará as informações para o componente efetuator do agente, e este por sua vez, invoca o efetuator padrão presente no agente, responsável por enviar uma mensagem no ambiente.

### 2.6.3.3 Executando a aplicação

Após a instanciação das classes necessárias, e a construção da lógica de troca de mensagens e processamento de informações, são definidos os agentes que participam do sistema. Para este exemplo, todos os agentes estão situados em um mesmo domínio e presentes em uma mesma máquina, sendo eles o *AgMachine1*, *AgMachine2* e *AgResourceManager*.

A Figura 19 mostra a interface gráfica do semanticore e os agentes<sup>5</sup> associados ao domínio semântico.



**Figura 19** - Interface do Semanticore: aplicação exemplo

Após a execução do exemplo, é apresentado o seguinte resultado no console da aplicação:

```
Agent: AgMachine1
Machine info:
  Virtual Memory: 7753728
  Processors: 2
Agent: AgMachine2
Machine info:
  Virtual Memory: 7753728
  Processors: 2
```

Como os agentes estão em execução na mesma máquina, as informações recuperadas são as mesmas.

<sup>5</sup> Os agentes DomainController e CommunicationManager são agentes de controle do SemantiCore

## 2.7 JENA

O Jena é um kit que foi desenvolvido pela HP Labs<sup>6</sup>, com o objetivo de facilitar o desenvolvimento de aplicações que utilizam os modelos e linguagens da *Web Semântica* (MCBRIDE, 2002). O Jena é uma interface de programação de aplicações (API<sup>7</sup>) e é disponibilizado como software livre.

O coração de sua arquitetura é a API RDF, como mostrado na Figura 20, que suporta a criação, manipulação e consultas em grafos RDF. A API também suporta diversas tecnologias de armazenamento de dados, permitindo a associação de diferentes tecnologias.

Além da API RDF, o Jena dispõe de uma API para a manipulação de ontologias. Esta API é independente de linguagem, isto é, permite a utilização de várias linguagens para a representação de ontologias, como o OWL. Esta API é responsável pela transformação de uma ontologia em um modelo abstrato de dados orientado a objetos, e possibilita que suas primitivas (conceitos e relacionamentos) também possam ser tratadas como objetos.

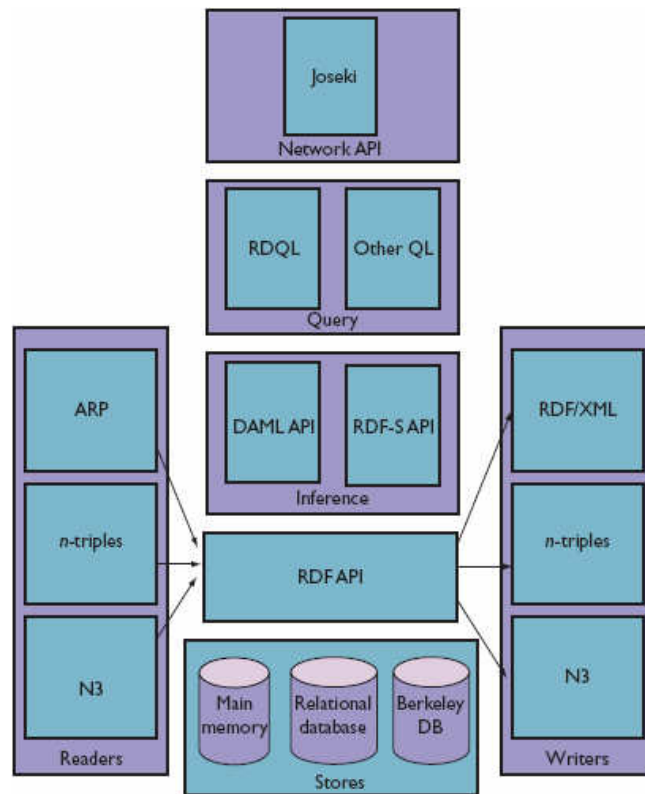
O Jena também possui mecanismos de inferência. Os principais usos destes mecanismos são para suportar o uso de linguagens como RDFS e OWL permitindo que fatos sejam inferidos das instâncias de dados e de suas descrições (JENA, 2006). Entretanto, estes mecanismos são projetados para serem genéricos, e, em particular, incluem mecanismos genéricos de regras de inferência.

Além das capacidades de processamento de linguagens e modelo semânticos, o Jena dispõe de mecanismos capazes de serem associados a fontes de dados na *Web*. A *Network API* permite a uma aplicação o acesso e atualização a fontes de dados na *Web*, similarmente como um navegador acessa um *web server*. A *network API* possui operações similares aos comandos do protocolo HTTP, como o GET e o PUT, utilizados para a recuperação de metadados. Este mecanismo pode ser utilizado em aplicações cliente (no navegador ou outras aplicações que consumam conteúdos na *Web*) ou em servidores, para a publicação de conteúdos.

---

<sup>6</sup> Página web: <http://www.hpl.hp.com/>

<sup>7</sup> Do inglês Application Programming Interface



**Figura 20** - A arquitetura do Jena (MCBRIDE, 2002)

No contexto da presente proposta, o Jena é utilizado para permitir a manipulação das ontologias que são capturadas em páginas *Web*, permitindo que estas sejam encapsuladas e utilizadas como objetivos na linguagem Java, e para permitir aos agentes a capacidade de inferência e raciocínio baseada em regras sobre as ontologias capturadas.





## 3 TRABALHOS RELACIONADOS

Nos últimos anos têm crescido o número de iniciativas acadêmicas e industriais para implementar os requisitos da *Web Semântica*. Ferramentas de anotação de conteúdos, editores de ontologias e mecanismos de inferência tem sido criados para permitir o desenvolvimento de aplicações que podem beneficiar-se das características da *Web Semântica*. Nenhuma destas iniciativas provê uma solução que combine os esforços realizados anteriormente para permitir o desenvolvimento de aplicações para a *Web Semântica*.

Na literatura não foram encontrados trabalhos que se propõem aos mesmos objetivos do presente trabalho. Neste sentido, entre as propostas encontradas, existem algumas que se assemelham a esta proposta e contemplam parcialmente os requisitos para a execução de aplicações da *Web Semântica*.

Atualmente, existem diversas ferramentas disponíveis para darem suporte a *Web Semântica*. Tais ferramentas podem ser classificadas em três grandes grupos (BREITMAN, 2007): ontologias e editores de metadados, *plugins* e APIs, e mecanismos de inferência. Entre as ferramentas existentes, podemos citar o Disco (DISCO, 2008), que é um navegador que opera sobre conjuntos de fontes de dados na *Web Semântica*, exibindo informações que são encontradas em páginas *Web* que contenham um documento RDF associado; o OntoBroker (ONTOBROKER, 2008) é uma máquina de inferência que processa ontologias e possibilita a construção de aplicações semânticas; o Pellet (PELLET, 2003) é um mecanismo de inferência para o OWL DL, construído em Java; e o SEAL (*SEmantic portAL*) (MAEDCHE, 2001), que é constituído essencialmente por ontologias, sendo utilizado para busca semântica e compartilhamento de conhecimento na *Web*.

### 3.1 Power Magpie

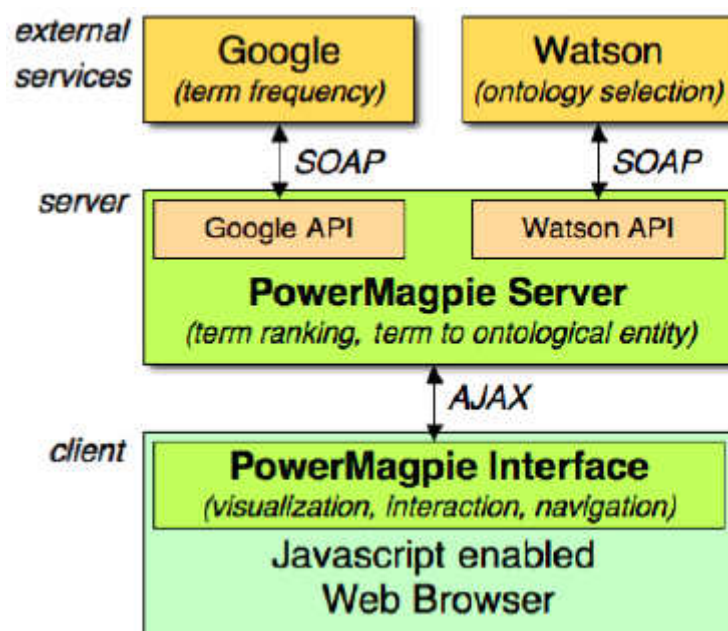
Esta seção irá abragerá em detalhes uma proposta que mais se assemelha com o presente trabalho. Em (GUIDI *et al.*, 2007) é mostrado um navegador para a *Web semântica* que representa uma extensão de um navegador *Web* convencional, aumentando

suas funcionalidades com a habilidade de atuar sobre recursos descritos em páginas *Web* e encontrar recursos semanticamente relacionados com uma página *Web*.

Na visão dos autores, o protótipo construído provê três principais características:

1. A habilidade de encontrar e recuperar automaticamente ontologias que são relacionadas a uma página *Web* que o usuário está navegando.
2. Uma interface simples com o usuário para permitir a navegação entre os dados encontrados.
3. A habilidade para prover serviços capazes de aproveitar os dados descobertos.

A arquitetura consiste de dois componentes principais: uma interface com o usuário e um servidor (chamado de *Power Magpie Server*). A *interface com o usuário* é uma extensão de um navegador *Web* convencional. O *servidor* é o componente que atua como suporte a aplicação cliente, provendo as funções necessárias. O cliente e servidor podem executar em diferentes máquinas. A arquitetura do *Power Magpie*, em alto nível, pode ser vista conforme a Figura 21.



**Figura 21** - A arquitetura Magpie (GUIDI *et al.*, 2007)

Para permitir a descoberta de ontologias, o Power Magpie pode contar com arquiteturas externas como o Swoogle, SWSE, Watson ou redes *peer-to-peer* (GUIDI *et al.*, 2007). Entretanto, no protótipo atual, a principal fonte de informações é provida pela

arquitetura Watson (GUIDI *et al.*, 2007). A Watson é uma arquitetura que coleta conteúdo semânticos disponíveis na *Web Semântica*, analisa e extrai metadados úteis e os indexa para permitir um eficiente acesso a estes metadados. Dessa forma, ela pode ser vista como um repositório de ontologias.

Segundo os autores, o resultado do trabalho é um *framework* que recupera em tempo real ontologias relacionadas a uma página *Web*. O protótipo atual permite aos usuários explorarem as ontologias relacionadas e destacarem conceitos em uma página *Web*. Também são apresentados dois trabalhos futuros. O primeiro é atacar o problema da ambigüidade que pode ser gerada pelo uso de uma mesma palavra em diferentes domínios. O segundo é prover serviços que aproveitem os conteúdos encontrados nas páginas *Web*.

Em relação a presente proposta, este trabalho apresenta alguns pontos em comum, que são: (i) permitir a captura de conteúdos anotados semanticamente em páginas *Web*, (ii) a divisão de sua arquitetura em dois modelos, cliente e servidor, e (iii) a extensão de um navegador *Web* convencional.

Este trabalho difere-se da presente proposta em alguns aspectos. O SWP utiliza a abstração de agentes de software, que serão os responsáveis pelo processamento dos conteúdos anotados semanticamente que são capturados dinamicamente nas páginas *Web*, e não somente na visualização e armazenamento destas anotações, como acontece no Power Magpie, que cita o processamento dos conteúdos recuperados como trabalho futuro. O SWP divide a arquitetura em Cliente e Servidor para permitir a integração do Semanticore a navegadores e servidores *Web*, possibilitando a formação de ambientes onde os agentes de software atuam. E por fim, o protótipo do SWP também estende um navegador *Web* convencional, o *Mozilla Firefox*.



## 4 UMA ARQUITETURA PARA A EXECUÇÃO DE APLICAÇÕES BASEADAS EM AGENTES NA WEB SEMÂNTICA

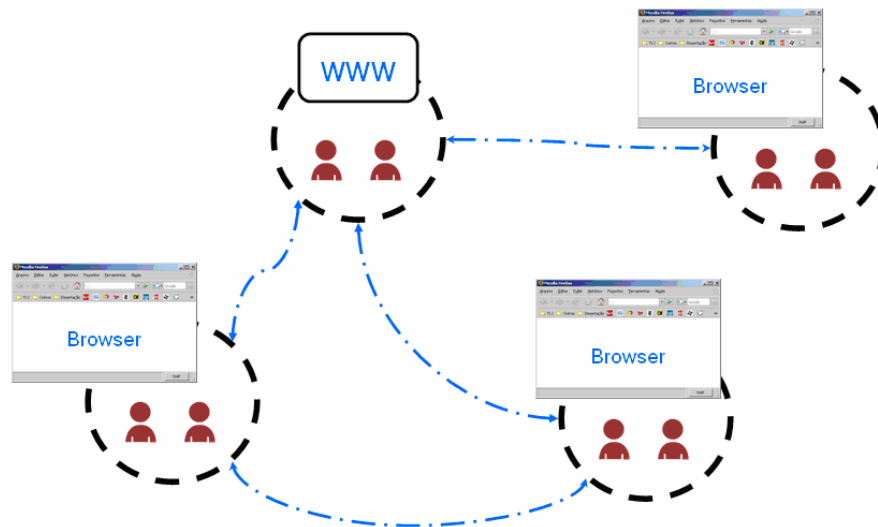
Neste trabalho, está sendo proposta uma arquitetura que permite a integração das tecnologias existentes para viabilizar o desenvolvimento de aplicações baseadas em agentes para a *Web* semântica, chamada Semantic Web Plugin (SWP).

A arquitetura possui o SWP *cliente* e *servidor*, que integram o SemantiCore a navegadores e servidores *Web*, de forma a permitir a criação de ambientes onde agentes de software atuam, estendendo a *Web*, sem interferir na sua estrutura atual. A extensão do Semanticore agrega ao SWP o modelo de agente e o modelo domínio. No Semanticore, o modelo de agente é orientado a componentes, agregando todos os aspectos necessários a sua implementação; e o modelo de domínio define o ambiente onde os agentes atuam.

O modelo *cliente* permite o processamento de páginas que possuem anotação semântica pelos agentes. Após a leitura de páginas *Web* anotadas semanticamente, são extraídas as estruturas semânticas das demais informações da página e encapsuladas para envio posterior. O formato de anotação utilizado é abstraído, permitindo o processamento de diferentes tecnologias de representação semântica.

Além da captura e disponibilização de anotações semânticas em páginas *Web*, existe a possibilidade de que ambientes presentes em outros navegadores ou em servidores *Web* (que possuam o SWP servidor) sejam automaticamente reconhecidos. Este reconhecimento torna-se possível através da utilização de marcadores especiais anexados ao protocolo de comunicação HTTP. A arquitetura estabelece novas informações que são adicionadas ao cabeçalho da resposta a requisição feita ao servidor, indicando a identificação do domínio semântico.

Esta arquitetura altera o atual paradigma *request-response* usado na *Web* para um modelo híbrido *request-response / peer-to-peer*, onde, agentes atuando em ambientes associados a navegadores *Web* nas máquinas cliente podem comunicar-se diretamente com outros agentes distribuídos em outros ambientes que estejam acessíveis através da Internet. A Figura 22 mostra um esquema de comunicação entre os diferentes ambientes providos pela integração com o SWP.

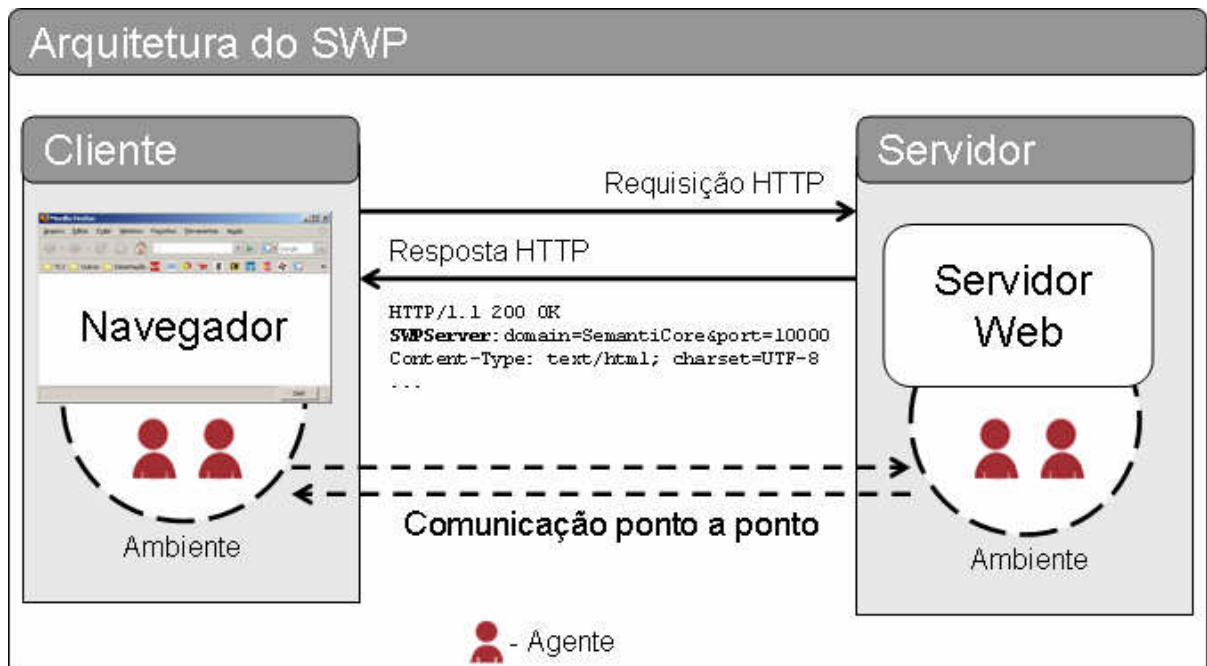


**Figura 22** - Comunicação entre ambientes

A figura ilustra a comunicação entre navegadores e servidores *Web*. A comunicação entre os navegadores é possível após o acesso por eles a páginas providas por um servidor que possui o SWP. Dessa forma, este servidor atua como o mediador da descoberta de outros ambientes, informando a existência de outros ambientes para cada navegador que o acessa. Após essa divulgação, os navegadores podem iniciar a comunicação de forma ponto a ponto, e, neste momento, já não é necessária a comunicação direta com o servidor.

A Figura 23 apresenta a arquitetura proposta. Nesta figura é ilustrada a comunicação entre um navegador e um servidor *Web* habilitado como servidor *web* semântico, ambos com o SWP. No primeiro momento, o navegador realiza uma requisição ao servidor, e na resposta desta requisição, são enviadas as informações que indicam a existência de um domínio *web* semântico. Após a identificação do servidor pelo navegador, é realizada a comunicação ponto a ponto entre os ambientes.

As páginas que contêm anotação semântica, e os servidores *web* semânticos identificados são mantidos em históricos. O histórico de ambientes possibilita que o presente ambiente no navegador do usuário conecte-se a outros ambientes, formando assim uma rede, e, possibilitando a troca de mensagens entre os agentes, tornando possível a cooperação entre eles. Dessa forma, os agentes poderiam executar suas tarefas (visando o alcance de seus objetivos) sem necessariamente o usuário estar navegando na *Web*.



**Figura 23** - Arquitetura do SWP

O SWP reconhece as anotações semânticas de uma página *Web* através de URLs inseridas no seu cabeçalho, como mostra a Figura 24.

```
<head>
  <link rel="owl" type="text/owl" href="http://www.inf.pucrs.br/~mescobar/swp/economia.owl" />
</head>
```

**Figura 24** - Exemplo de anotações semânticas em uma página *web*

A URL mostrada na figura é capturada e interpretada pelo SWP. Este então realiza a captura do conteúdo apontado pela URL, encapsula e disponibiliza este conteúdo para os agentes. O elemento “link”, da linguagem HTML, possui como atributos o “rel”, “type” e “href”. O atributo “rel” descreve o relacionamento do documento HTML com o endereço especificado em “href”. Neste caso, estamos informando apenas que o conteúdo apontado é uma ontologia em *Owl*. O atributo “type” indica ao mecanismo que processa o HTML o tipo do conteúdo que está associado ao endereço presente em *href*. A indicação de um conteúdo em *owl* é dada através do valor “text/owl”, informado no atributo *type*. Esta convenção é utilizada no contexto do SWP, não sendo oficializada pela W3C. Por fim, o atributo *href* especifica a localização de um recurso na *Web*.

Como pré-requisitos para o funcionamento de aplicações que utilizam a arquitetura do SWP, podemos citar: (i) o desenvolvimento dos agentes que atuam no ambiente, processando as ontologias capturadas, devem ser realizados conforme os formatos

de desenvolvimento do *framework* Semanticore; e (ii) as páginas *Web* que serão acessadas devem ser anotadas semanticamente para permitir a identificação e captura de ontologias pelo SWP durante a navegação.

Devido à limitação de tempo deste trabalho, foi realizada apenas a construção da arquitetura cliente, para permitir a integração a navegadores *Web*. A arquitetura do SWP cliente, como mostrado na Figura 25 e representada através de um diagrama de classes UML (JACOBSON *et al.*, 1999), pode ser dividida em quatro partes principais, descritas nas próximas seções.

## 4.1 Integração

A *Integração* contém os elementos responsáveis por realizar a extensão do Semanticore, permitindo o controle sobre o ambiente onde os agentes atuam, e, a integração com o navegador. A seguir serão detalhados os elementos que compõem a *Integração*, e a relação entre eles.

A classe **SWP** representa o núcleo da arquitetura. Ela é a classe principal, sendo responsável por centralizar os principais elementos da arquitetura e também por estender as funções do *framework* Semanticore, através da relação de herança com a classe *Environment*. Esta relação é necessária para permitir ao SWP o acesso aos agentes de software em execução, o controle sobre o ambiente e o envio das anotações semânticas capturadas das páginas *Web* para os agentes.

A classe **Mediator** é responsável por mediar a integração entre o navegador e o restante da arquitetura. É nesta classe que estão às operações públicas que podem ser realizadas no SWP, tais como consultar os agentes presentes no domínio, o histórico semântico, os domínios semânticos conhecidos, entre outras.



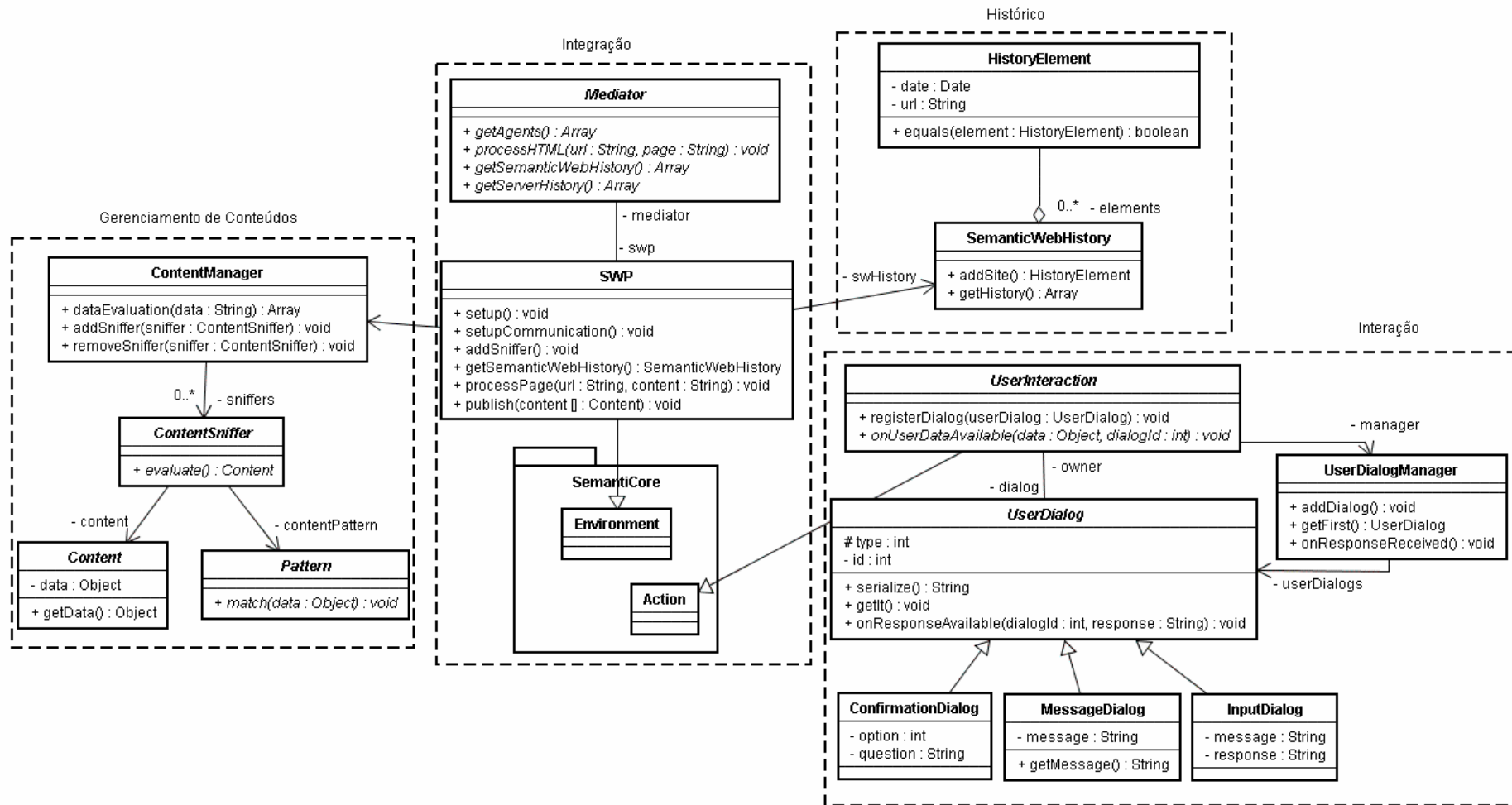


Figura 25 - Diagrama de classes da arquitetura do SWL cliente

## 4.2 Gerenciamento de Conteúdos

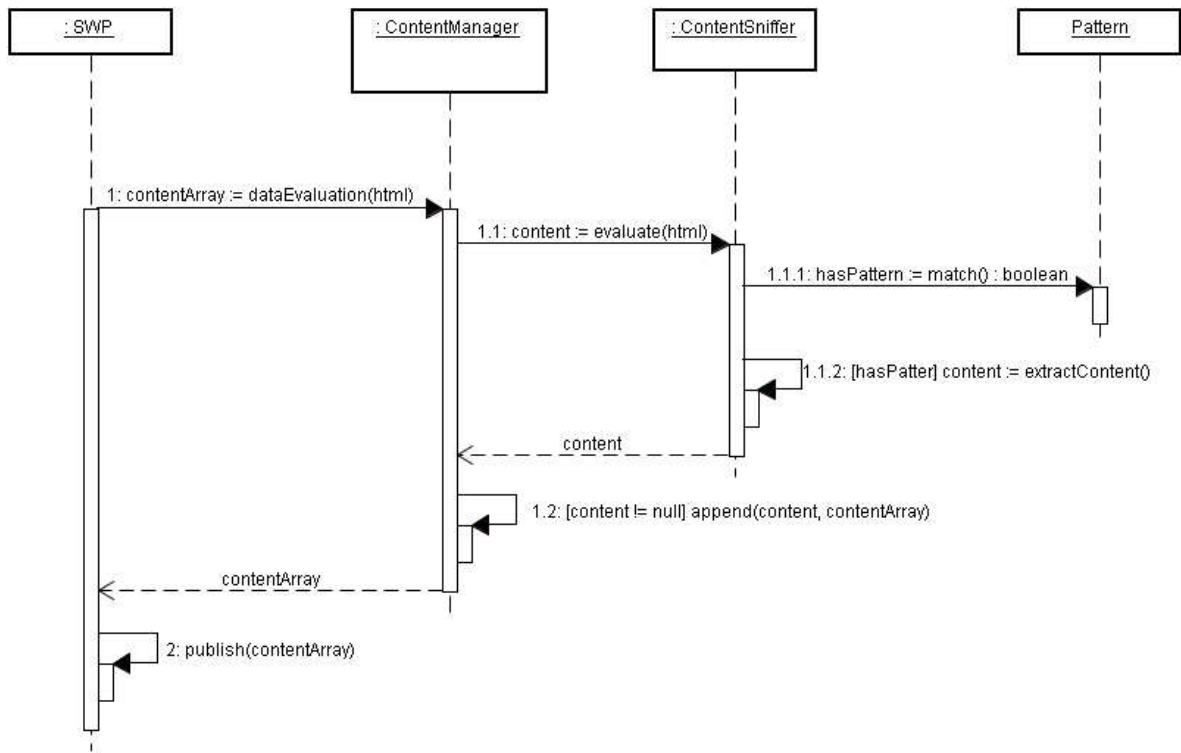
O *Gerenciamento de Conteúdos* contém os elementos responsável pelo processamento do conteúdo das páginas *Web* visitadas pelo usuário, que foram capturados na Integração, e verificar a existência de anotações semânticas. Caso uma anotação seja encontrada, esta é extraída das demais informações da página e encapsuladas para envio aos agentes presentes no ambiente. O envio destas informações no ambiente é realizado na Integração. A seguir serão detalhados os elementos que compõem o Gerenciamento de Conteúdos, e a relação entre eles.

Quando uma página *Web* é acessada, seu conteúdo é capturado e enviado pelo SWP à classe *ContentManager*. Esta por sua vez, é responsável por centralizar e organizar os elementos que realizam o processamento dos conteúdos capturados. Estes elementos são chamados de *Sniffers*, e são representados pela classe *ContentSniffer*.

Um *ContentSniffer* possui a função de processar o conteúdo de uma página *Web* na tentativa de encontrar um determinado padrão. Este padrão é definido através da classe *Pattern*. Quando um padrão é reconhecido, a classe *sniffer* é responsável por extrair o conteúdo das demais informações da página e por encapsular esse conteúdo em um determinado formato definido na classe *Content*. É através destes elementos que são definidos os tipos de anotações que são reconhecidos pelo SWP, como por exemplo, anotações em RDF e OWL.

O *ContentManager* também é responsável por enviar os conteúdos detectados de volta para a classe *SWP*. Neste momento, o SWP fica responsável por encapsular os conteúdos no formato de mensagem do Semanticore e por disponibilizá-los para processamento pelos agentes.

A etapa de reconhecimento e processamento de anotações semânticas das páginas *Web* pelo SWP pode ser vista conforme o diagrama de seqüência UML, mostrado na Figura 26.



**Figura 26** - Diagrama de seqüência - processamento de anotações semânticas

O diagrama oferece uma melhor compreensão do fluxo realizado pelo SWP, que compreende o processamento e disponibilização de anotações semânticas das páginas *Web*. Em (1), o SWP invoca a operação *dataEvaluation* da classe *ContentManager*. Esta operação é responsável por realizar o envio do conteúdo HTML para processamento pelos *Sniffers*. Para cada *sniffer*, é invocada a operação *evaluate* (1.1). Ao receber uma informação no *evaluate*, o *sniffer* verifica se a informação recebida combina com o seu padrão de reconhecimento, através da operação *match* (1.1.1) presente na classe *Pattern*. Caso um padrão seja reconhecido, a classe *Sniffer* realiza a extração do conteúdo (1.1.2), e o retorna para a *ContentManager*. Após o processamento de todos os *sniffers*, a classe *ContentManager* gera um vetor contendo todos os conteúdos identificados (1.2), e este vetor é retornado para a classe SWP, que por sua vez possui a responsabilidade de publicar estes conteúdos no ambiente (2) para uso dos agentes, através da operação *publish*.

### 4.3 Histórico

O *Histórico* é responsável por armazenar a localização das páginas *Web* que foram identificadas como contendo anotações semânticas, os dados de acesso de outros ambientes com o SWP que foram identificados durante a navegação, e por prover os mecanismos necessários para permitir o acesso aos dados dos históricos pelos agentes no ambiente.

Além das funções de integração, a classe SWP é responsável por manter o histórico das páginas *Web* anotadas semanticamente. Isto é realizado através da associação com a classe *SemanticWebHistory*. Ela é a classe responsável por manter as informações necessárias de histórico, através da agregação de registros de histórico, representados pela classe *HistoryElement*.

Cada elemento do histórico possui os atributos *date* e *url*, que respectivamente representam a data de acesso e a localização da página com anotação semântica. Este histórico, além de armazenar as páginas, serve como base para consulta dos agentes, podendo estes, por exemplo, visitarem as páginas do histórico na busca de informações atualizadas.

### 4.4 Interação

Como o SWP estende um navegador *Web*, é importante permitir que os agentes presentes no ambiente associado ao navegador possam solicitar interações com o usuário. Estas interações são gerenciadas na *Interação*. A *Interação* possui os elementos necessários para permitir que os agentes realizem suas ações e possam indicar ao SWP a existência de uma interação. As interações ocorrem através de janelas que serão exibidas no navegador. A exibição da janela, a captura dos dados informados pelo usuário, e o envio destes dados até o agente solicitante são gerenciados na parte de *Interação*.

Além do processamento de conteúdos e da possibilidade de ser mantido um histórico das páginas anotadas semanticamente, o modelo dispõe de elementos capazes de permitir que os agentes presentes no domínio especifiquem interações com o usuário. Uma

interação é possível através da utilização da classe *UserInteraction*. Esta classe estende a classe *Action* do Semanticore, que representa a ação de um agente.

Para que um agente possa interagir com o usuário, seja para informar ou coletar dados, são necessários mecanismos que gerenciem tais operações de maneira transparente. No SWP, essa interação ocorre através de janelas gráficas que são exibidas diretamente no navegador *Web*.

Estas janelas são representadas através da classe *UserDialog*, e possuem três especializações, *ConfirmationDialog*, *MessageDialog* e *InputDialog*. A classe *ConfirmationDialog* possui o objetivo de obter uma confirmação do usuário, como por exemplo, uma questão com as alternativas “Sim” ou “Não”. A classe *MessageDialog* tem como função mostrar uma mensagem textual ao usuário. A classe *InputDialog* possui a função de coletar dados textuais a serem informados pelo usuário.

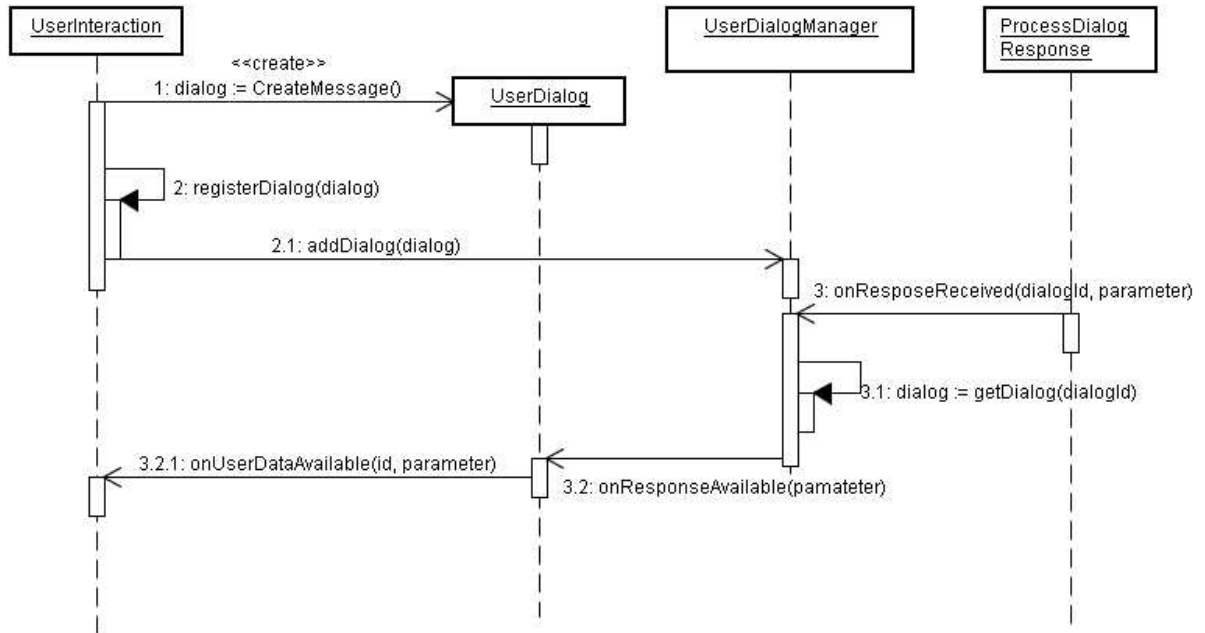
A classe *UserInteraction* possui duas operações principais. A primeira operação permite que uma janela de interação seja registrada pelo agente para envio ao navegador. A segunda operação é responsável por receber os dados de retorno que foram informados pelo usuário e o identificador da janela, caso a janela seja do tipo que requer entrada de dados.

O gerenciamento destas janelas é realizado na classe *UserDialogManager*. Esta classe mantém uma fila de janelas registradas. Ela é responsável por disponibilizar estas janelas para o navegador, e quando necessário, receber e enviar as respostas dos usuários para as Ações (*UserInteraction*) associadas com a janela.

A etapa de cadastramento e tratamento das janelas de interação com o usuário pode ser vista conforme o diagrama de seqüência UML, mostrado na Figura 27.

O diagrama representa o fluxo que compreende a criação, disponibilização e recebimento de uma resposta para janelas de interação com o usuário. Em (1) é criada uma janela de interação (*UserDialog*). Após a criação em (1), em (2) é invocada a operação *registerDialog* indicando que a janela criada deve ser registrada para ser disponibilizada para o usuário. Este registro é feito na classe *UserDialogManager*, através do método *addDialog* (2.1). Após o registro, a classe *UserDialogManager* é encarregada de ficar aguardando uma resposta da janela de interação. A resposta é recebida de forma assíncrona, através da operação *ProcessDialogResponse*, e esta então invoca a operação (3), e passa o parâmetro capturado da janela de interação. Após receber a resposta em (3), em (3.1) é recuperada a janela referente a resposta recebida, em (3.2) é invocada a operação que indica a

disponibilidade da resposta, e em (3.2.1) é invocada a operação que finaliza o fluxo, e fornece para a ação do agente a resposta informada pelo usuário para a janela de interação.



**Figura 27** - Diagrama de seqüência - controle das janelas de interação

No próximo capítulo será apresentada a arquitetura do protótipo que permite a integração entre o SWP e o navegador Mozilla Firefox (MOZILLA FIREFOX, 2008).

## 5 PROTÓTIPO E EXEMPLO DE USO

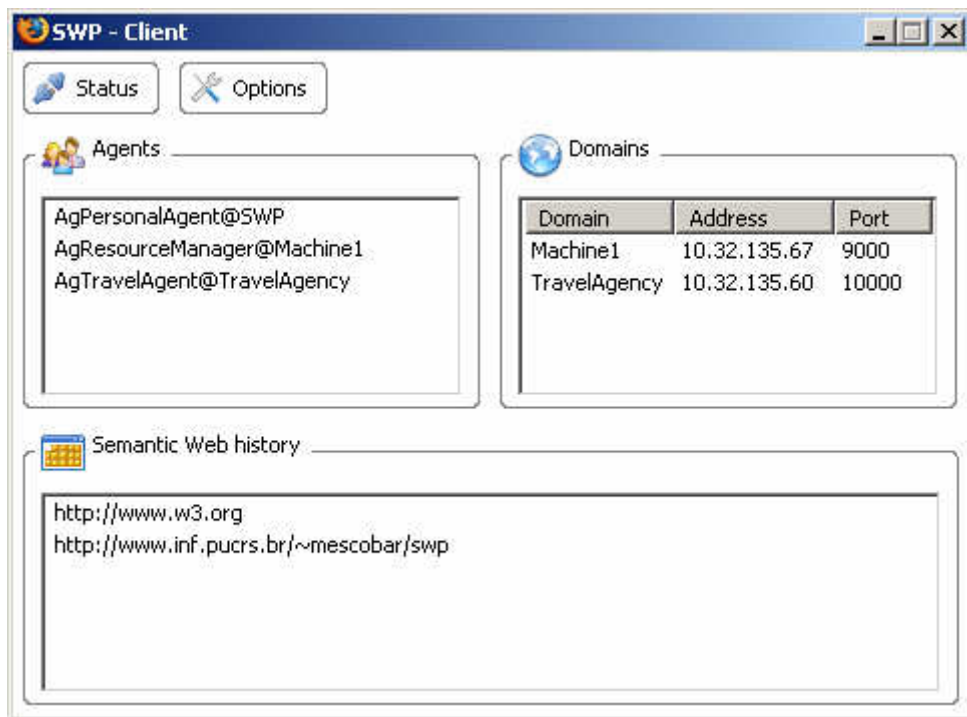
Devido a falta de navegadores para a *Web Semântica* capazes de trabalhar com o processamento de linguagens de representação de conhecimento, é apresentada neste trabalho uma iniciativa que visa dar um passo na direção desta concretização com o uso de navegadores amplamente aceitos no mercado. Portanto, este trabalho visa utilizar o navegador Mozilla Firefox, que além de possuir uma grande aceitação no mercado, possui suporte a extensão de sua arquitetura, possibilitando novas funcionalidades.

No navegador *Firefox*, as extensões são pequenas adições que oferecem novas funcionalidades. As extensões visam adicionar ou modificar componentes, que vão desde uma barra de ferramentas a uma característica completamente nova. Dessa forma, elas permitem que a aplicação seja personalizada para satisfazer as necessidades pessoais de cada usuário.

As interfaces com o usuário no *Firefox* são construídas utilizando as linguagens XUL (XUL, 2008) e JavaScript (FLANAGAN, 2002). A XUL é uma gramática XML que proporciona a definição de componentes gráficos como botões, menus, barras de ferramentas, árvores, etc. O *JavaScript* é uma linguagem dinâmica de *scripts* que suporta a construção de elementos procedurais e orientados a objetos. As ações do usuário nos componentes gráficos são gerenciadas através do *JavaScript*, que é utilizado para o controle de eventos dos componentes, e eventos internos do navegador, como por exemplo o evento gerado quando uma página HTML é carregada.

Para este trabalho foi criada uma extensão que gera uma nova janela no navegador, permitindo ao usuário a visualização das informações do domínio semântico que é associado ao navegador, como mostrado na Figura 28.

A janela mostrada na figura possui três listas, que informam respectivamente: os agentes presentes no domínio semântico (*Agents*), os ambientes já identificados pelo SWP (*Domains*), e o histórico das páginas *Web* detectadas contendo anotações semânticas (*Semantic Web History*). O botão “Status” indica se o SWP está ativo ou não. O botão “Options” exibe uma janela onde o usuário pode informar se deseja que o SWP processe o cabeçalho, o corpo da página, ou ambos, devido a possibilidade de as anotações semânticas estarem dispostas em qualquer parte da página *Web*. A seguir, será apresentada a arquitetura de integração com o navegador *Firefox*.



**Figura 28** - SWP: Interface com o usuário no navegador Mozilla Firefox

O protótipo da arquitetura SWP cliente foi desenvolvido utilizando a linguagem de programação Java. O navegador *Firefox* é construído utilizando a linguagem C++. Dessa forma, não foi realizada uma integração nativa entre as diferentes arquiteturas. Mesmo sendo possível a integração entre diferentes linguagens, isto acarretaria em uma redução das funcionalidades do SWP, onde muitos dos recursos que são providos pela máquina virtual Java não estariam disponíveis. Para cobrir essa necessidade de integração, foi realizada a construção de componentes de software que permitam a comunicação entre a extensão criada para o *Firefox* e o SWP. A Figura 29 mostra o diagrama da arquitetura para a integração com o navegador.

A classe *MozillaMediator* estende as funcionalidades da classe *Mediator* do SWP. Esta classe é responsável por cadastrar todas as possíveis operações que são realizadas com o navegador, e também por manter a classe que realiza a comunicação com o navegador, denominada *Mozilla Connector*.

A classe *MozillaConnector* opera como servidora, na espera de conexões (requisições) que são originadas na extensão do navegador. Esta classe utiliza comunicação por *sockets*, e implementando o protocolo HTTP para a comunicação. No navegador, as requisições para o SWP são implementadas utilizando a linguagem Javascript, que é responsável por acessar a API de *socket* que o Firefox disponibiliza para suas extensões.



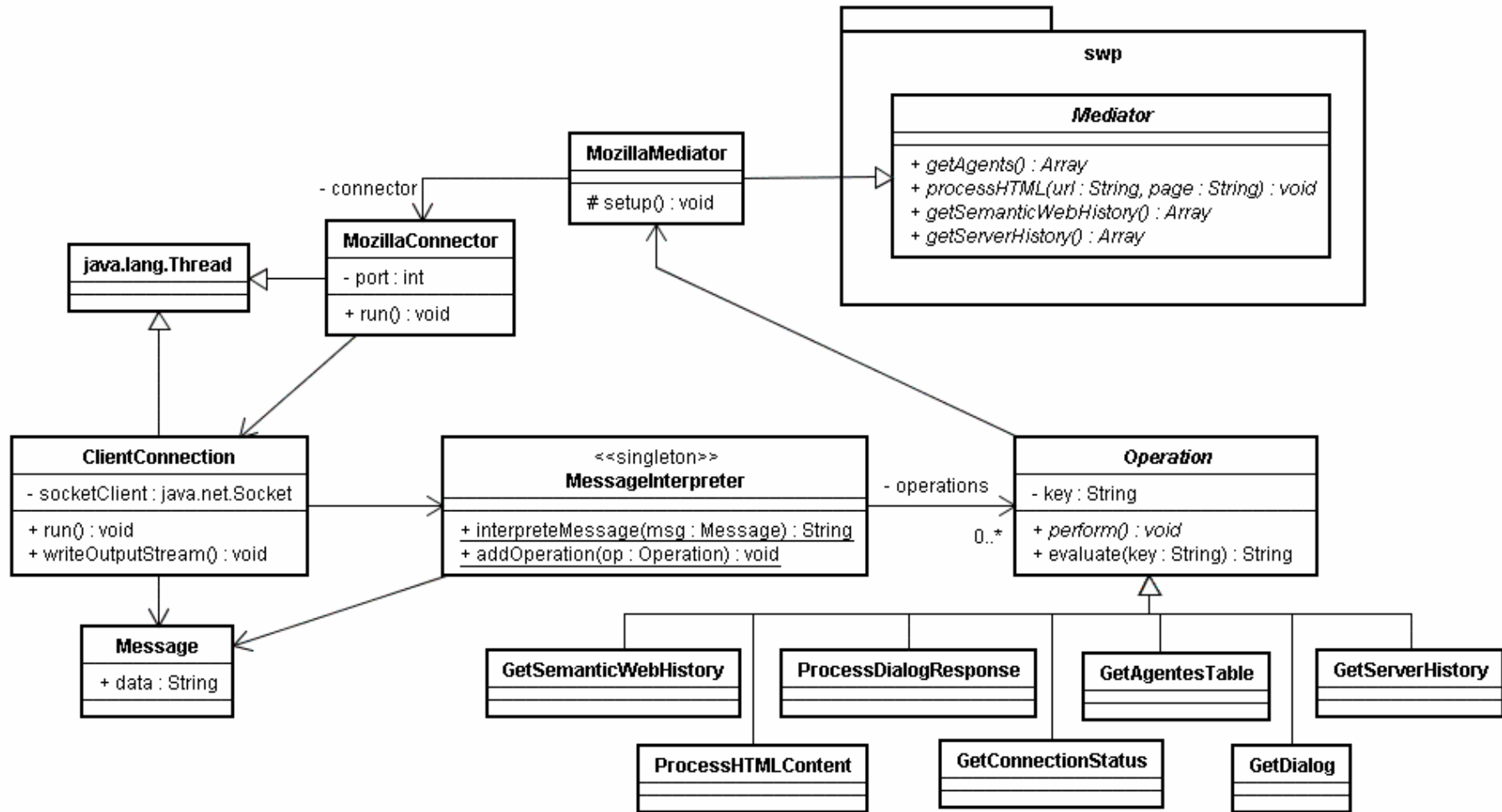


Figura 29 - Arquitetura para a integração com o navegador Firefox

Quando uma requisição é aceita, a classe *ClientConnection* é associada a requisição e esta é responsável pelo gerenciamento dos dados trocados (leitura e escrita). Os dados que são recebidos do navegador são encapsulados na classe *Message* e estes são passados para a classe *MessageInterpreter*.

A classe *MessageInterpreter* é responsável por analisar o conteúdo das mensagens e decidir qual *operação* o SWP deve realizar, gerando ou não uma resposta a ser enviada de volta ao navegador.

As operações são definidas através da classe *Operation*. Esta classe é responsável por abstrair as operações que devem ser realizadas no SWP ou no navegador. Como especializações dessa classe, encontramos as seguintes operações:

- *GetDialog*, responsável por recuperar as janelas de interação que estão registradas pelos agentes, e repassá-las para envio ao navegador;
- *GetSemanticWebHistory*, responsável por recuperar os dados do histórico *Web* semântico, contendo a referência das páginas anotadas semanticamente que foram detectadas, e disponibilizá-los para envio ao navegador;
- *GetAgentsTable*, responsável por recuperar os dados referentes aos agentes presentes no domínio semântico e disponibilizá-los para envio ao navegador;
- *ProcessDialogResponse*, responsável por gerenciar o envio dos dados de uma janela de interação com o usuário e enviá-las ao agente responsável;
- *ProcessHTMLContent*, responsável por receber e repassar o conteúdo HTML de uma página *Web* acessada pelo usuário da extensão para processamento no SWP;
- *GetServerHistory*, responsável por recuperar os dados do histórico de domínios *Web* semânticos, e disponibilizá-los para envio ao navegador.

No próxima seção será descrito um exemplo de cenário que ilustrará a utilização do SWP na captura de anotações semânticas em páginas *Web*, e a comunicação entre ambientes presentes em navegadores e servidores *Web*, através da internet.

## 5.1 Exemplo de uso

O modelo proposto oferece uma série de facilitadores para a construção de aplicações que possam consumir conteúdos anotados semanticamente em páginas *Web*, e permitir que agentes de software presentes em diferentes domínios na *Web* possam se comunicar. Para ilustrar algumas das funcionalidades providas pelo SWP, neste capítulo será apresentado o desenvolvimento de um exemplo de aplicação na *Web Semântica*.

Abaixo seguem as características do ambiente de desenvolvimento utilizado para a construção do exemplo:

- Linguagem de programação Java (SDK 5.0).
- Ambiente de desenvolvimento Eclipse.
- Construção das ontologias através da ferramenta Protégé (PROTÉGÉ, 2008).
- Manipulação das ontologias em Java através do *framework* Jena (versão 2.2).
- Linguagem HTML para a codificação das páginas *Web*, e CSS para codificação de estilos.
- Microsoft Visual Web Developer Express para a edição dos arquivos HTML e CSS.

O exemplo proposto ilustra a captura de anotações semânticas em páginas *Web*, o processamento destas anotações por um agente de software presente no navegador do usuário, e a apresentação para o usuário, do resultado obtido pelo agente a partir do processamento das anotações semânticas, demonstrando o surgimento de links semânticos entre duas páginas não associadas a priori.

O exemplo é composto por duas páginas *Web*, onde a primeira página é uma notícia retirada da seção de economia do site O Globo<sup>8</sup>, e a segunda é uma notícia retirada do site Quatro Rodas<sup>9</sup>. As páginas originais não possuem quaisquer tipos de anotações semânticas em suas codificações. Dessa forma, foi necessária a realização da captura do

---

<sup>8</sup> <http://oglobo.globo.com>

<sup>9</sup> <http://quatrorodas.abril.com.br>

código HTML das páginas e a adaptação destes para permitir a adição da anotação semântica.

Cada página foi armazenada em um arquivo HTML, e em seus cabeçalhos foram adicionadas informações que indicam a existência de ontologias que descrevem semanticamente o domínio de cada página, para permitir a detecção. As ontologias, descritas na linguagem OWL, encontram-se nos apêndices A e B deste trabalho.

Abaixo seguem os marcadores que foram adicionados nas páginas.

Na primeira notícia:

```
<link rel="owl" type="text/owl"
href="http://www.inf.pucrs.br/~mescobar/swp/economia.owl" />
```

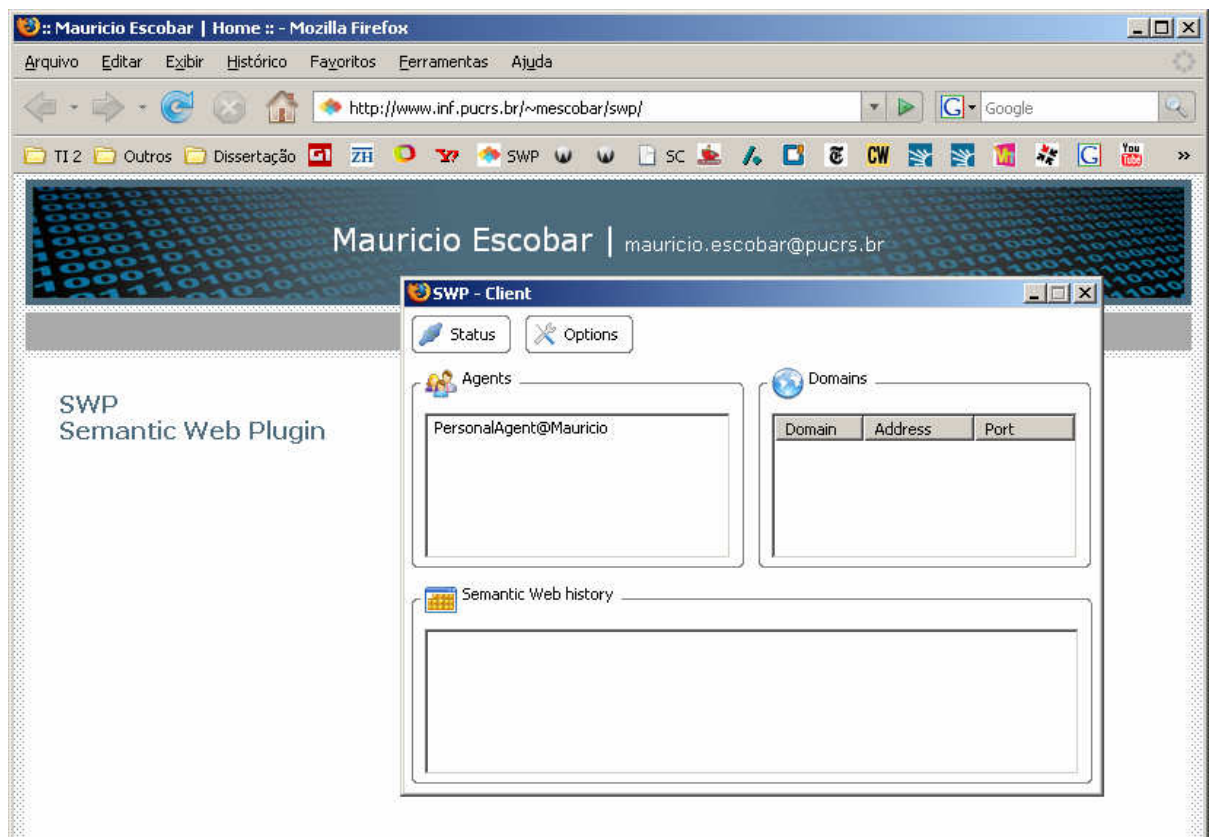
Na segunda notícia:

```
<link rel="owl" type="text/owl"
href="http://www.inf.pucrs.br/~mescobar/swp/veiculos.owl" />
```

A primeira página é uma notícia que indica a retirada de aviões da frota operante de uma empresa e o corte do quadro de funcionários devido a alta do petróleo. A segunda notícia é um comparativo entre dois carros populares. Esta página reúne informações referentes a consumo de combustível, o rendimento dos motores, entre outras informações.

Ambas as páginas são desassociadas, isto é, elas não possuem em sua codificação a declaração de *links* que indicam a existência uma da outra. Dessa forma, só existe uma relação semântica entre elas, referente ao conceito relativo ao petróleo, pois a primeira notícia fala sobre a alta do preço do petróleo, e a segunda notícia contém informações sobre o consumo de combustíveis derivados do petróleo.

Este exemplo também é composto por um agente de software que representa o usuário. Este agente está em execução no domínio semântico presente no navegador *Web* do usuário. A Figura 30 mostra o navegador *Web* e a extensão do SWP, onde na lista de agentes, aparece no nome do agente do usuário, chamado de *PersonalAgent*. Neste caso, é mostrado “PersonalAgent@Mauricio”, onde “Mauricio” é o nome do domínio semântico.



**Figura 30** - SWP - Domínio no navegador Web

Como o SWP é integrado ao *framework* SemantiCore, a programação do agente é feita de acordo com o modelo de agente definido por ele. Na Figura 31 é mostrado o código-fonte da classe que implementa o agente do usuário para este exemplo.

A classe *PersonalUserAgent* estende a classe *SemanticAgent*, do *framework* Semanticore. O método *addMessage*, é o método do agente por onde chega uma mensagem vinda do ambiente.

Este agente só aceita conteúdos em OWL, e por isso, na linha 17 é feita uma conversão do conteúdo textual da mensagem para o objeto *OntModel*, que é o objeto responsável por encapsular uma ontologia, disponível através do uso do *framework* Jena.

O objeto gerado com o conteúdo recebido é enviado para o componente decisório do agente, conforme o método na linha 18. Já nas linhas 23 a 25, é definido o método *setup* do agente. Este método deve conter todo o código de inicialização do agente, e no caso deste exemplo, contém apenas a definição do mecanismo de tomada de decisão do agente (linha 24).

```

1 package instancias.demoPetroleo.agent;
2 import java.io.StringReader;
3 import semanticore.domain.model.SemanticAgent;
4 import semanticore.domain.model.SemanticMessage;
5 import com.hp.hpl.jena.ontology.OntModel;
6 import com.hp.hpl.jena.ontology.OntModelSpec;
7 import com.hp.hpl.jena.rdf.model.ModelFactory;
8
9 public class PersonalUserAgent extends SemanticAgent {
10
11 public PersonalUserAgent(String name) {
12     super(name);
13 }
14
15 public void addMessage(SemanticMessage message) {
16     try {
17         OntModel model = buildOntModel(message.getContent().toString());
18         getComponent("decision").put(model);
19     } catch (Exception e) {
20     }
21 }
22
23 public void setup() {
24     createDecisionComponent(new Decisorio());
25 }
26
27 protected OntModel buildOntModel(String data) {
28     StringReader content = new StringReader(data);
29     OntModel model = ModelFactory.createOntologyModel(OntModelSpec.CWL_MEM,
30     null);
31     model.read(content, null);
32     return model;
33 }
34 }

```

**Figura 31** - Código do agente do usuário

O mecanismo de tomada de decisão deste agente utiliza uma máquina de inferência provida pelo Jena. A cada mensagem recebida, o agente realiza o processo de inferência, e baseado em suas regras que também são definidas no formato do Jena, verifica se algum resultado é gerado.

Na tabela 3 é apresentado o trecho de uma regra de inferência utilizada pelo agente.

**Tabela 3** - Regra de inferência para a verificação do petróleo

Regra 1
[verificaTendencia: (?petroleo rdf:type n:Petroleo) (?petroleo n:tendenciaPreco 'alta') -> (?petroleo n:alertaDePreco 'Alta dos combustiveis')]

Esta primeira regra é responsável por verificar na ontologia se existe uma informação sobre Petróleo, e caso exista, verifica também se a tendência do preço está em alta. Caso a tendência seja de alta, é gerado um novo fato, indicando um alerta para o preço do petróleo.

Já na segunda regra, conforme mostrado na tabela 4, existe um comparativo que recupera dois carros distintos na ontologia, e verifica a medida de consumo destes carros, comparando-as e gerando um novo fato que indica qual destes carros é o que possui melhor desempenho, isto é, menor consumo de combustível.

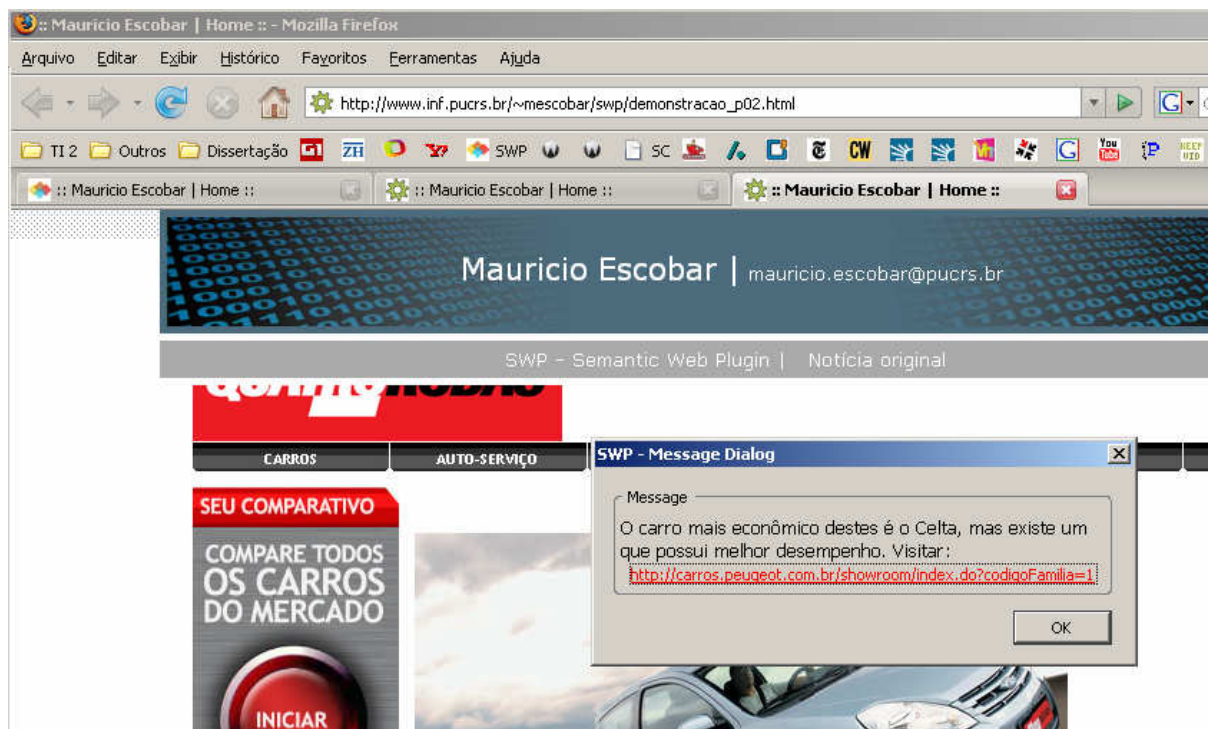
**Tabela 4 - Regra de inferência para comparação entre carros**

Regra 2
<pre>[verificaDesempenho:   (?carro1 rdf:type n:Carro)   (?carro2 rdf:type n:Carro)   notEqual(?carro1, ?carro2)   (?carro1 n:desempenho_km_litro ?desempenho1)   (?carro2 n:desempenho_km_litro ?desempenho2)   greaterThan(?desempenho1, ?desempenho2)   -&gt;   (?carro1 n:maiorDesempenho ?desempenho1)]</pre>

Após o usuário visitar a primeira página (com informações sobre o petróleo), o SWP identifica que existe uma ontologia associada a página e realiza a captura, extraindo o conteúdo OWL e o disponibilizando para o agente. O agente do usuário captura o conteúdo e realiza a inferência, satisfazendo a primeira regra, mas um resultado não é gerado para o usuário, pois o agente inferiu somente a alta no preço do petróleo.

Após o usuário visitar a segunda página (contendo o comparativo entre os carros), que também possui uma ontologia associada, o agente realiza novamente a inferência. Nesse momento, a segunda regra é satisfeita.

Após satisfazer as duas regras, o agente infere que o usuário possivelmente estava procurando por carros com baixo consumo de combustível. Neste momento, o agente realiza uma consulta ao histórico de páginas *web* semânticas, e recupera uma informação previamente processada, onde um carro de baixo consumo já havia sido identificado. O agente compara estes dados do histórico, com os dados capturados e infere que existe outro carro, que possui menor consumo do que os mostrados na notícia, e gera uma mensagem para o usuário, enviada através dos mecanismos do SWP, como mostrado na Figura 32.



**Figura 32** - Resultado apresentado pelo agente do usuário

A mensagem mostrada na figura é uma janela de interação com o usuário, provida pelo SWP e mostrada diretamente no navegador. Neste exemplo, o agente informa o endereço do site da fabricante de carros, que possui um carro com consumo de combustíveis menor do que os que ele inferiu baseado na notícia capturada no site em que o usuário estava navegando.

## 5.2 Considerações sobre o exemplo

O exemplo de aplicação descrito neste capítulo procurou mostrar, em linhas gerais, as primitivas do modelo proposto e a forma de usá-las na definição das anotações necessárias nas páginas *Web*, e o modo como o agente processa um conteúdo capturado. Como pôde ser visto ao longo do capítulo, o uso do modelo não traz um aumento significativo de trabalho aos desenvolvedores, salvo as questões referentes à implementação dos agentes de software e a definição das ontologias necessárias para representa o



conhecimento associado a um domínio, aspectos esses que poderiam ser considerados complexos não do ponto de vista de desenvolvimento, mas de entendimento do problema.

Durante a descrição do exemplo, não foram salientadas as questões referentes ao mapeamento entre as ontologias utilizadas nas notícias. Para este exemplo, as ontologias possuíam a mesma base de conceitos, e dessa maneira, o agente consegue realizar a inferência e a associação direta entre os conceitos presentes nas duas ontologias utilizadas no exemplo.

O mapeamento e composição de ontologias de diferentes domínios ainda é um problema em aberto na área de Ontologias e *Web Semântica*, não sendo tratado neste trabalho, que preocupa-se apenas no reconhecimento de padrões de anotação semântica em páginas *Web* e a disponibilização destas anotações para o agentes em um ambiente.



## 6 CONCLUSÃO E TRABALHOS FUTUROS

O principal requisito para a *Web Semântica* é a interoperabilidade. As máquinas devem ser capazes de explorar os recursos na *Web*, podendo acessá-los e usá-los. Conseqüentemente, os recursos devem estar abertos e compreensíveis, e não escondidos em um sistema proprietário que publica informações somente em formatos – ou ferramentas – orientados a humanos. Eles devem ser invocados e publicados em um formato aberto e estruturado que permite as máquinas fazerem o melhor uso destes.

O atendimento aos requisitos da *Web Semântica* necessitará de diversas camadas de desenvolvimento. A infra-estrutura permitirá a identificação, localização e transformação de recursos de forma robusta e segura. As linguagens são necessárias para expressar o conteúdo da *Web Semântica*. Recursos como as ontologias, transformações, metadados e banco de dados devem sustentar as camadas base da arquitetura da *Web Semântica*. Nas camadas mais acima teríamos os recursos e as aplicações explorando estes recursos.

A falta de navegadores para a *Web semântica* capazes de trabalhar com o processamento de linguagens de representação, e que permitam a associação direta entre os usuários e os agentes associados a conteúdos em domínios *Web* motivaram a realização deste trabalho. De conta disto, foi realizada a construção de uma arquitetura que estende as funcionalidade do *framework* SemantiCore, para ser integrada a navegadores e a servidores *Web*, permitindo a estes trabalharem com a *Web Semântica*.

A extensão do Semanticore agregou ao SWP o modelo de agentes de software e o modelo domínio semântico. No Semanticore, o modelo de agente é orientado a componentes, agregando todos os aspectos necessários a sua implementação; e o modelo de domínio semântico define o ambiente onde os agentes atuam, e contém entidades administrativas, responsáveis pelo controle do domínio. O Semanticore também abstrai as tecnologias de comunicação de dados - permitindo a associação de diferentes protocolos e tecnologias - e o controle de distribuição, reduzindo a complexidade no desenvolvimento de aplicações orientadas a agentes.

O modelo cliente do SWP permite que os agentes interpretem conteúdos anotados semanticamente na *Web*. O SWP realizada a captura de anotações semânticas em

páginas *Web*, onde estas anotações são extraídas das demais informações da página, e são disponibilizadas para os agentes associados ao ambiente no navegador. Além da captura e disponibilização de conteúdos, o SWP permite a identificação automática de outros ambientes que utilizam o SWP através da internet.

O modelo servidor é responsável por adicionar informações que indicam a existência de um ambiente criado pelo SWP no cabeçalho da resposta a uma requisição HTTP feita para o servidor *Web* no qual o SWP está integrado. Estas informações permitem a identificação automática deste ambiente pelo navegador *Web*. Quando o navegador identifica o servidor, estes automaticamente conectam-se, formando uma rede. O modelo servidor também é responsável pelas políticas de divulgação de ambientes conhecidos, isto é, o servidor mantém uma lista de todos os ambientes que o estão acessando, e quando um novo ambiente o acessa, o servidor divulga a sua lista de ambientes, possibilitando assim, a formação de novas conexões e a expansão da rede virtual de ambientes.

Devido à limitação de tempo, a integração com o servidor *Web* não foi realizada dentro deste trabalho. Apenas foi realizada a construção da arquitetura *cliente*, e a arquitetura para viabilizar a integração com o navegador Mozilla Firefox. O modelo servidor, portanto, foi simulado na construção dos cenários, e as informações necessárias para a identificação do ambiente foram adicionadas direto nas páginas *Web* do cenário exemplificado.

Os seguintes pontos são objetos de trabalhos futuros:

1. Embora seja proposta pelo Tim Berners-Lee uma idéia, não existe atualmente a concretização destas idéias em softwares comerciais. Esta é uma iniciativa que visa dar um passo na direção desta concretização com o uso de navegadores *Web* amplamente aceitos no mercado.

2. A questão de segurança da informação é um aspecto muito importante, especialmente por se tratar de um sistema presente na Internet. Mecanismos de segurança devem ser incorporados, para garantir a integridade e segurança na troca de dados, sejam nas informações capturadas na *Web*, e na troca de mensagens entre os agentes de diferentes ambientes.

3. O estudo mais aprofundado das recomendações da W3C, para a utilização das tecnologias da *Web* semântica, e o avanço desta iniciativa podem direcionar a novas melhorias e a novas funcionalidades para esta proposta.

4. A possibilidade de serem agregados a atual proposta, mecanismos de identificação dinâmica dos objetivos dos agentes presentes em diferentes domínios, e mecanismos que permitam a composição e coordenação destes objetivos para permitir que os agentes alcancem a resolução de objetivos que vão além de suas capacidades individuais.



## REFERÊNCIAS

- ANTONIOU, G.; HARMELEN, F. (2004). "A Semantic Web Primer". Massachusetts: The MIT Press, 2004, 238p.
- BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. (2001). "The Semantic Web". Scientific American Magazine, vol. 1, no. 5, Maio 2001, pp. 34-43.
- BLOIS, M.; LUCENA, C. (2004). "Multi-Agent Systems and the Semantic Web – The SemanticCore Agent-based Abstraction Layer". In: Proceedings of Sixth International Conference on Enterprise Information Systems (ICEIS), Porto, 2004, pp. 263-270.
- BRADSHAW, J. "An Introduction to Software Agents". In: J. Bradshaw (Ed.), Software Agents, American Association for Artificial Intelligence/MIT, Cambridge, 1997, pp. 3-46.
- BREITMAN, K. (2007). "Semantic web: concepts, technologies and applications". London : Springer, 2007, 327p. (The Nasa Monographs in Systems and Software Engineering).
- DACONTA, M., OBRST, L. SMITH, K. (2003). "The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management". New York: John Wiley & Sons, 2003, 281p.
- DAVIES, J.; DUKE, A.; SURE, Y. (2003). "OntoShare: a knowledge management environment for virtual communities of practice". In: Proceedings of the 2nd international Conference on Knowledge Capture, Sanibel Island, FL, USA, Outubro 2003, pp. 20-27.
- GUIDI, D., GRIDINOC, L., DZBOR, M., MOTTA, E. (2007). "Building the bases for a Semantic Web Browser". In: Proceedings of 6th International and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007), Busan, Korea, 2007, pp. 25-26.
- DISCO (2008). "Disco - Hyperdata Browser". Capturado em: <http://www4.wiwiss.fu-berlin.de/bizer/ng4j/disco/>, Novembro 2008.

- EUZENAT, J. (2002). "Research challenges and perspectives of the Semantic Web". Intelligent Systems, IEEE [see also IEEE Intelligent Systems and Their Applications] Volume 17, Issue 5, Sep/Oct 2002, pp. 86-88.
- FENSEL, D. (2001). "Ontologies: a silver bullet for knowledge management and electronic commerce". New York: Springer, 2001, 138p.
- FENSEL, D.; HENDLER, J.; LIEBERMAN, H.; WAHLSTER, J. (2003). "Spinning the semantic web: bringing the world wide web to its full potential". Cambridge (UK): The Mit Press, 2003, 479p.
- FERBER, J. (1999). "Multi-agent systems: an introduction to distributed artificial Intelligence". Boston: Addison-Wesley, 1999, 528p.
- FIPA ACL (2002). "Message Structure Specification". Desenvolvido por: The Foundation for Intelligent Physical Agents, 2000-2002. Capturado em: <http://www.fipa.org/specs/fipa00061/>. Dezembro 2008.
- FLANAGAN, D. (2002), "Javascript: the definitive guide." 4<sup>a</sup> ed. Sebastopol (USA): O'Reilly, 2002, 916p.
- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. (1994). "Design Patterns: elements of reusable object-oriented software". Massachusetts: Addison-Wesley, 1994, 395p.
- GRUBER, T. R. (1993). "Towards Principles for the Design of Ontologies Used for Knowledge Sharing". International Journal of Human and Computer Studies, vol. 43, no. 5, pp. 907-928.
- GUO, L.; ROBERTSON, D.; CHEN-BURGER, Y. (2005). "A Generic Multi-agent System Platform For Business Workflows Using Web Services Composition". In: Proceedings of 2005 IEEE Intelligent Agent Technology, France, 2005, pp. 301-307.
- HENDLER, J. (2001). "Agents and the Semantic Web". IEEE Intelligent Systems, vol. 16, no. 2, Mar./Abr. 2001, pp. 30-37.
- HTML (2006). "HTML - HyperText Markup Language". Capturado em: <http://www.w3.org/MarkUp/>, Março 2006.



- HTTP (1996). “HTTP – Hypertext Transfer Protocol”. Capturado em <http://www.w3.org/Protocols/>, Novembro 2008.
- JACOBSON, I., BOOCH, G., RUMBAUGH, J. (1999). “The Unified Software Development Process”. Boston, MA: Addison-Wesley, 1999.
- JENA (2006). “Jena - A Semantic Web Framework for Java”. Capturado em: <http://jena.sourceforge.net/>, Agosto 2008.
- KQML (2008). “KQML - Knowledge Query Manipulation Language”. Capturado em: <http://www.cs.umbc.edu/kqml/>, Novembro 2008.
- LI, W. (2002). “Intelligent Information Agent with Ontology on the Semantic Web”. In: Proceedings of the 4th World Congress on Intelligent Control and Automation, Shanghai, 2002, pp. 1501-1504.
- MCBRIDE, B. (2002). “Jena: A semantic Web Toolkit”. IEEE Internet Computing. Volume 6, Issue 6, Novembro 2002, pp. 55-59.
- ONTOBROKER. (2008). “Ontobroker”. Desenvolvido por Ontoprise GmbH. Capturado em: [http://www.ontoprise.de/content/e1171/e1231/index\\_eng.html](http://www.ontoprise.de/content/e1171/e1231/index_eng.html), Dezembro 2008.
- OWL (2004). “OWL - Web Ontology Language”. Capturado em: <http://www.w3.org/2004/OWL/>, Junho 2006.
- PELLET (2003). “Pellet: The Open Source OWL DL Reasoner”. Capturado em: <http://www.mindswap.org/2003/pellet/>, Novembro 2008.
- PROTÉGÉ - Ontology Editor and Knowledge Acquisition System (2005). Capturado em: <http://protege.stanford.edu/>, Novembro 2008.
- RDF (2004). “RDF - Resource Description Framework”. Capturado em: <http://www.w3.org/RDF/>, Março 2006.
- RDFS (2004). “RDFS – RDF Schema”. Capturado em: <http://www.w3.org/TR/rdf-schema/>, Novembro 2008.

- RIBEIRO, M. B. (2002). “Web Life: Uma arquitetura para a implementação de sistemas multi-agentes para a Web”. Tese de Doutorado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, 2002, 204p.
- SEMANTIC WEB (2001). “W3C Semantic Web Activity”. Capturado em: <http://www.w3c.org/2001/sw/>, Março 2005.
- MAEDCHE, A. et al. (2001). “SEAL – A Framework for Developing Semantic Web PortALs”. In: 18th British National Conference on Databases: Advances in Databases. London: 2001, pp. 1-22.
- MOZILLA FIREFOX (2008). “Mozilla Firefox”. Capturado em: <http://pt-br.www.mozilla.com/pt-BR/firefox/>, Julho 2008.
- WEISS, G. (1999). “Multiagent systems: a modern approach to distributed artificial intelligence”. Cambridge: The MIT Press, 1999, 619p.
- WOOLDRIDGE, M. (2002). “An Introduction to Multiagent Systems”. Chichester: John Wiley and Sons Ltda, 2002, 366p.
- WOOLDRIDGE, M.; JENNINGS, N.; KINNY, D. (1999). “A methodology for agent oriented analysis and design”. In: Proceedings of International Conference on Autonomous Agents, Seattle, EUA, 1999, pp. 69-76.
- XML (2006). “XML – eXtensible Markup Language”. Capturado em: <http://www.w3.org/XML/>, Março 2006.
- XMLS (2001). “XML Schema”. Capturado em: <http://www.w3.org/XML/Schema>, Novembro 2008.
- XUL (2008). “XUL – XML User Interface Language”. Capturado em: <http://www.mozilla.org/projects/xul/>, Julho 2008.
- VLASSIS, N. (2007). “A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence”. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 1ª edição, 2007, 84p.

## **APÊNDICE A**

– Ontologia representando a notícia da alta do petróleo –

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns="http://semanticore.pucrs.br#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://semanticore.pucrs.br">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Combustivel"/>
  <owl:Class rdf:ID="Petroleo"/>
  <owl:Class rdf:ID="Carro"/>
  <owl:ObjectProperty rdf:ID="consome">
    <rdfs:domain rdf:resource="#Carro"/>
    <rdfs:range rdf:resource="#Combustivel"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="derivaDe">
    <rdfs:domain rdf:resource="#Combustivel"/>
    <rdfs:range rdf:resource="#Petroleo"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="desempenho_km_litro">
    <rdfs:domain rdf:resource="#Carro"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="precoLitro">
    <rdfs:domain rdf:resource="#Combustivel"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="fabricante">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Carro"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="tendenciaPreco">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Petroleo"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="marca">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Carro"/>
  </owl:DatatypeProperty>

  <Petroleo rdf:ID="Petroleo_16">
    <tendenciaPreco rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >alta</tendenciaPreco>
  </Petroleo>

</rdf:RDF>

```

## **APÊNDICE B**

– Ontologia representando a notícia com o comparativo entre veículos –

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns="http://semanticore.pucrs.br#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://semanticore.pucrs.br">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Combustivel"/>
  <owl:Class rdf:ID="Petroleo"/>
  <owl:Class rdf:ID="Carro"/>
  <owl:ObjectProperty rdf:ID="consome">
    <rdfs:domain rdf:resource="#Carro"/>
    <rdfs:range rdf:resource="#Combustivel"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="derivaDe">
    <rdfs:domain rdf:resource="#Combustivel"/>
    <rdfs:range rdf:resource="#Petroleo"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="desempenho_km_litro">
    <rdfs:domain rdf:resource="#Carro"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="precoLitro">
    <rdfs:domain rdf:resource="#Combustivel"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="fabricante">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Carro"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="tendenciaPreco">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Petroleo"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="marca">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Carro"/>
  </owl:DatatypeProperty>

  <Carro rdf:ID="FordKa">
    <desempenho_km_litro rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >10</desempenho_km_litro>
    <marca rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Ka</marca>
    <fabricante rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Ford</fabricante>
  </Carro>

```

```
<Carro rdf:ID="Celta">  
  <desempenho_km_litro rdf:datatype="http://www.w3.org/2001/XMLSchema#int"  
  >11</desempenho_km_litro>  
  <marca rdf:datatype="http://www.w3.org/2001/XMLSchema#string"  
  >Celta</marca>  
  <fabricante rdf:datatype="http://www.w3.org/2001/XMLSchema#string"  
  >Chevrolet</fabricante>  
</Carro>  
</rdf:RDF>
```