Desenvolvimento de Sistemas Inteligentes sob a perspectiva da Web Semântica

Evandro Costa[1], Ig Bittencourt[2], Baldoíno Fonseca[1], Ivo Calado[1] e Guilherme Maia[1]

[1]Instituto de Computação – Universidade Federal de Alagoas (UFAL)
Campus A. C. Simões, BR 104 - Norte, Km 97, C. Universitária, Maceió, AL – Brasil

[2]Departamento de Sistemas e Computação – Universidade Federal de Campina Grande
Av. Aprígio Veloso, 882, Campina Grande - PB.

{evandro,bfsn,iaarc,jgmm}@tci.ufal.br, ibert@dsc.ufcg.edu.br

Resumo. *Atualmente, muito se discute sobre a realização autônoma e eficaz de atividades complexas por entidades inteligentes (Agentes de Software). Diante deste contexto, o presente tutorial objetiva apresentar um estudo sistemático do trabalho realizado no tema de construção de Ambientes Inteligentes através de uma abordagem de Agentes, Serviços e Web Semântica alcançando o atual estado da arte. O tutorial apresenta primeiramente cada uma das abordagens isoladamente, mostrando as principais motivações e características, por conseguinte é focada uma visão da interação das três abordagens. Por fim, são mostrados exemplos práticos através da utilização do Jade e outras ferramentas existentes para a construção de Ambientes baseados em Agentes, Serviços e Web Semântica. O tutorial é finalizado com uma reflexão acerca do conteúdo abordado, apontando para as atuais tendências no tema em questão.*

1. Introdução

Sistemas Inteligentes representam uma categoria de sistema de software com proposta de exibição de comportamento inteligente. Como exemplos de sistemas inteligentes citam-se os sistemas baseados em conhecimento e suas instâncias nos sistemas especialistas, chegando aos atuais sistemas baseados em agentes. Associados a muitos desses sistemas estão técnicas de representação de conhecimento e raciocínio, aprendizagem, etc.

O advento da Web tem proporcionado diversas oportunidades à área de Inteligência Artificial, com os denominados Sistemas Inteligentes. De fato, tão logo a Web se estabeleceu, surgiram investimentos em aplicações via Web em diversos domínios, tais como: comércio e governo eletrônico, gestão empresarial, educação, dentre outros. Recentemente, muitas dessas aplicações vêm se aprimorando e tendo os seus requisitos indo ao encontro das extensões da Web, como Web 2.0 e, mais atual ainda, a Web 3.0 ou Web Semântica. Normalmente, uma das características marcantes dessas novas aplicações diz respeito ao fato de lidarem com grande volume de dados e informações desestruturadas adequadas, geralmente, apenas ao entendimento por agentes humanos. Isso inclui, por exemplo, páginas Web sem caracterização de contexto e semânticas nas suas definições. Nesse sentido, agentes inteligentes têm sido concebidos para atuarem no âmbito da Web semântica, numa expectativa de oferecer um tratamento mais adequado da informação.

Este texto serve como um suporte para o tutorial sobre construção de Ambientes Inteligentes através de uma abordagem de Agentes, Serviços e Web Semântica, sem, portanto, ter a pretensão de ser exaustivo, cobre tanto os aspectos conceituais no tema, quanto os seus aspectos práticos.

2. Sistemas Inteligentes

Esta seção aborda conceitos de Sistemas Inteligentes, discutindo a diferença entre Sistemas Inteligentes (SI), Sistemas Baseados em Conhecimento (SBC) e Sistemas Especialistas (SE). Além disso, aspectos relacionados à representação do conhecimento e comportamento inteligente são apresentados.

2.1. Conceitos

Sistemas Inteligentes representam sistemas que exibem um comportamento inteligente, ou seja, sistemas que possuem a capacidade de adaptação de acordo com as interações, raciocínio de acordo com uma situação particular, descoberta de novos significados, aprendizado, entre outras. Como exemplos de sistemas inteligentes citam-se os sistemas baseados em conhecimento, sistemas especialistas, sistemas multi-agentes, entre outros.

É importante frisar a diferença entre sistemas inteligentes, sistemas baseados em conhecimento e sistemas especialistas. *Sistemas Baseados em Conhecimento*(SBC) são sistemas que possuem uma base de conhecimento explicitamente representada e um mecanismo de inferência separado da base. *Sistemas Especialistas* (SE), por sua vez, são tipos mais específicos de SBCs cujo objetivo é utilizar conhecimento específico do domínio para alcançar o desempenho de especialistas humanos na resolução de problemas. A Figura 1 ilustra a diferença entre SI, SBC e SE.

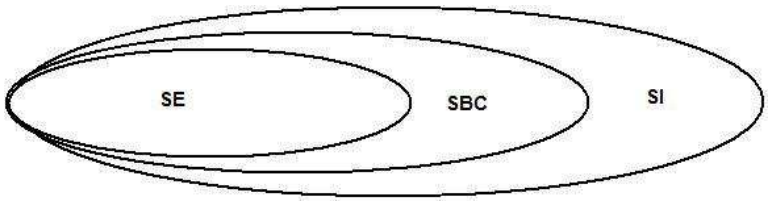

Figura 1. SI x SBC x SE

2.2. Conhecimento

Esta subseção objetiva descrever características do conhecimento, como construir sistemas baseados em conhecimento e como representar tal conhecimento.

Primeiramente deve-se esclarecer a diferença entre dado, informação e conhecimento, onde: *dado* equivale a uma representação simbólica de um objeto sem considerações de contexto, significado e aplicação, como por exemplos 40^0C, *20 anos*; *informação* equivale ao dado com o seu significado associado, como por exemplo *Febre alta* = 40^0C, *Idade de Andréa = 20 anos*; *conhecimento* é tudo que se usa para agir e criar novas informações. Conhecimento inclui a informação sobre o domínio e a forma como essa informação é utilizada para resolver problemas.

2.2.1. Engenharia do Conhecimento

O objetivo principal da **engenharia do conhecimento** é transformar o processo *ad hoc* de construir sistemas baseados em conhecimento em um processo de engenharia baseado em métodos, linguagens e ferramentas [Mastella 2005].

Segundo a engenharia do conhecimento, quatro atividades básicas são necessárias para desenvolver um SBC (Figura 2): *aquisição*, *formalização*, *implementação* e *refinamento*.

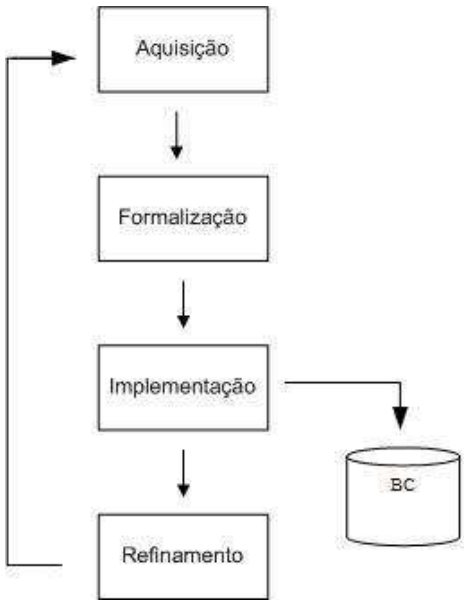

Figura 2. Atividades da Engenharia do Conhecimento.

- Aquisição: é o processo de extrair, estruturar e organizar o conhecimento de uma ou mais fontes, que podem ser documentadas ou não. As fontes documentadas são, em geral, mapas, livros, filmes, bases de dados, entre outras. O conhecimento não documentado reside na mente das pessoas que atuam no domínio e a aquisição ocorre por meio da interação direta. Existem várias técnicas para aquisição do conhecimento, como por exemplo:
  - Imersão na literatura;
  - Entrevistas não-estruturadas, entrevistas estruturadas e questionários;
  - Rastreamento de processos que podem ser na forma de relatórios verbais, relatórios não-verbais e análise de protocolo;
  - Técnicas aplicadas em engenharia de software: brainstorming e prototipação.
- Formalização: também chamada de representação do conhecimento. Nessa fase o conhecimento adquirido é estruturado em um formalismo que pode ser compreendido pelo computador e é codificado na base de conhecimento.
- Implementação: essa fase compreende a utilização de ferramentas para a construção, propriamente dita, da base de conhecimento. Na fase de implementação destacam-se duas atividades:
  - Validação do conhecimento: a base de conhecimento é verificada utilizando casos de testes;

  - Inferência: isto é, inferências de novos conhecimentos a partir da base.
- Refinamento: consiste em melhorar a capacidade e eficiência do processo de inferência sobre a base de conhecimento.

2.2.2. Formalismos de Representação

Os anos 80 do século passado foram marcados por discussões sobre formalismos de representação do conhecimento. Isto fez com que dois tipos de grupos de formalismos fossem definidos [Baader and Nutt 2003]:

1. Não Baseados em Lógica: tais formalismos refletem uma noção cognitiva e pretendem estar mais próximos de uma noção intuitiva. Exemplos que podem ser citados como formalismos que não são baseados em lógica são Redes Semânticas, Frames e representação baseada em regras. Um dos principais problemas com estes formalismos equivale ao *gap* semântico;
2. Baseados em Lógica: os formalismos baseados em lógica são variantes da lógica de primeira ordem e são suficientes para expressar fatos referentes ao mundo real. Os formalismos baseados em lógica ”herdam” a sintaxe, semântica e a prova da lógica de primeira ordem.

Formalismos não Baseados em lógica Esta seção aborda os formalismos que não são baseados em lógica. Ao longo da evolução dos formalismos de representação de conhecimento, dois formalismos que influenciaram bastante os atuais formalismos são frames e redes semânticas[1]. Alguns formalismos são citados a seguir:

- Redes Semânticas: As redes semânticas são grafos direcionados, ligados por nós, para representar objetos, conexões e a relação entre objetos. A rede semântica é usada para representar elementos tal como uma classe, suas instâncias e suas características. Seus arcos são direcionados e representam as relações entre os atributos. Quando um atributo não deve ser herdado, as redes semânticas necessitam de tratamento de exceção [Durkin 1994]. Os elos *é-um* e *parte-de* são bastante comuns em sistemas de redes semânticas. Este tipo de elo é utilizado para determinar a herança de propriedades. Os demais elos são específicos do domínio e representam propriedades de conceitos;
- Frames: É uma variação das redes semânticas que permite a representação de estruturas internas dos objetos. Frames são estruturas de dados com lacunas que podem ser preenchidas com informações declarativas ou processuais [Júnior and YONEYAMA 2000]. Frames foram descritos por Marvin Minsky [Minsky 1975], onde um frame possui seu formalismo similar aos roteiros, sendo utilizado para capturar as conexões implícitas da informação num domínio de problema em estruturas de dados explicitamente organizadas. Um frame representa uma entidade através de suas características e potenciais habilidades. Suas características estão representadas por pares atributo-valor e as potencialidades são representadas por métodos. Um frame abstrato (ou

[1]Atualmente, existem abordagens de redes semânticas baseadas em lógica, porém, descreve-se nesta seção as redes semânticas originalmente projetadas.

frame de classe) não tem instâncias, por esta razão seus atributos não são valorados, suas subclasses são ligadas à instâncias da entidade representada por essa classe [Minsky 1975].

- Roteiros: equivale a uma representação estruturada que descreve uma seqüência esteriotipada de eventos num contexto particular, Sendo bastante utilizadas na compreensão de linguagem natural. Temos que os conhecimentos sobre o que acontece tipicamente num esquema ou frame, podem ser representados por um roteiro ou script. Eles são utilizados para o conhecimento das ações de terceiros, contanto que executem aquilo que se espera dentro de uma determinada seqüência. Pode-se dizer que são estruturas que auxiliam o conhecimento de situações padronizadas. Ajudam a interpretar um novo evento a partir da utilização de scripts em casos passados [RIESBECK and SCHANK 1989];

Formalismos Baseados em lógica Esta seção aborda formalismos baseados em lógica. O conceito de sistemas formais será utilizado objetivando descrever o formalismo. Um sistema formal pode ser entendido como um sistema de manipulação de símbolos contendo uma linguagem formalmente descrita e um conjunto de regras para manipulá-los [do Carmo Costa 1992]. Logo, um sistema formal equivale a uma quádrupla $<\mathcal{L},\mathcal{A},\mathcal{R}, \vdash>$, onde $\mathcal{L}$ é uma linguagem da lógica, $\mathcal{A}$ é um subconjunto de expressões de $\mathcal{L}$, chamados axiomas, $\mathcal{R}$ é um conjunto de regras de manipulação e o símbolo $\vdash$ representa uma relação entre conjunto de expressões de $\mathcal{L}$.

Além disso, quando a representação do conhecimento de um determinado domínio (sintático e semântico) ocorre através de uma notação através da descrição de atributos, este formalismo é chamado de Linguagem Atributiva[2].

Abaixo são abordados dois formalismos de representação do conhecimento baseados em lógica.

- Lógica de Primeira Ordem: Lógica de primeira ordem pode ser considerado um formalismo adequado para representação de conhecimento no contexto da web semântica, considerando seu poder de expressão, o qual é maior do que o da lógica proposicional. Entretanto, a lógica de primeira ordem é apenas semi-decidível, tendo assim uma operacionalidade pouco adequada aos propósitos em questão. Daí, buscou-se como uma das alternativas para suprir essa inadequação, investir em lógica de descrição.;
- Lógica de Descrição: Lógica de descrição[3] equivale a um formalismo de representação de conhecimento baseado em lógica. Em meados dos anos 80, o termo Lógica de Descrição surgiu objetivando dar ênfase aos subjacentes sistemas baseados em lógicas, como lógica de primeira ordem. DL equivale a mais recente família de formalismo que representa o conhecimento i) através da definição dos conceitos do domínio (terminologia); e, ii) usar os conceitos para especificar propriedades de objetos e seus indivíduos (descrição do universo de discurso) [Baader and Nutt 2003]. Diferentemente de lógica de

[2]Do Inglês Attributive Language ou simplesmente $\mathcal{AL}$.
[3]Do Inglês Description Logic ou simplesmente DL.

primeira ordem, a lógica de descrição está interessada em um procedimento que garanta que as respostas (positivas ou negativas) sejam dadas em tempo finito. Alguns dos recursos presentes em lógica de primeira ordem não foram requeridos para garantir um bom nível de expressividade/representação.

2.3. Inferência

O mecanismo de inferência é responsável por raciocinar baseado em um determinado conhecimento representado. Aqui são discutidos os mecanismos de inferência baseados em regras e casos.

2.3.1. Raciocínio Baseado em Regras

Os sistemas baseados em regras, como o próprio nome sugere, possuem o conhecimento descrito na forma de regras. Estas regras são utilizadas pelos sistemas especialistas, onde, através deste conhecimento, objetiva-se a resolução de problemas e a tomada de decisão, com a qualidade comparável a um especialista humano. Sendo assim, o sistema precisa ter uma base de regras consistente armazenada em uma base de conhecimento, a qual inclui também um conjunto de fatos adquiridos para se chegar à conclusão do problema e tomada de decisão.

Na Figura 3 apresenta-se uma arquitetura detalhada de um sistema baseado em regras.

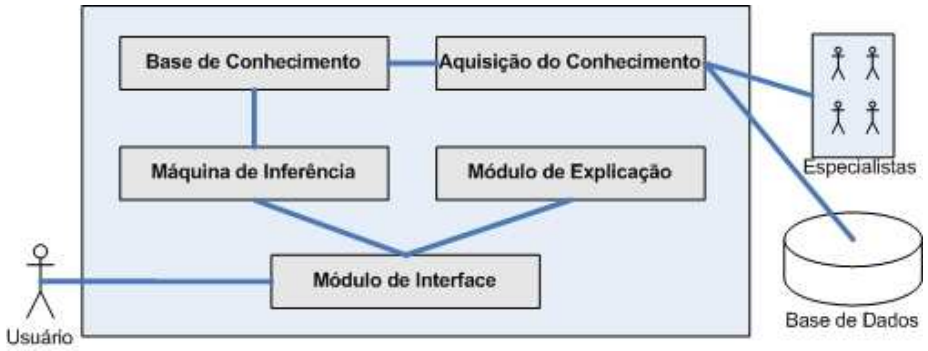

Figura 3. Arquitetura simplificada de RBR

Segue-se uma explicação sobre os aspectos presentes na Figura 3

- Engenheiro do Conhecimento: profissional que desempenha o papel de capturar o conhecimento especializado em regras e inserir na base de conhecimento;
- Base de Regras: conhecimento armazenado sobre um domínio particular, representado por uma coleção de regras. Essas regras são formadas pela relação “se-então” (if-then), comparável ao que se verifica em uma típica linguagem de programação tradicional;
- Máquina de Inferência: parte do raciocínio baseado em regras que seleciona e executa regras, resultando em uma resposta ao usuário, de acordo com os resultados que foram inferidos ao longo da execução das regras;

- Explanação: depois da conclusão da inferência ser mostrada ao usuário, pode-se haver necessidade de uma explanação acerca de “por que” (uma questão posta pelo sistema) e “como” (how) se chegou àquele resultado, e isso é responsabilidade do subsistema de explanação;
- Memória de Trabalho: local onde se encontram os fatos utilizados pelo mecanismo de inferência do sistema;
- Aquisição de Conhecimento: este subsistema permite o fluxo do novo conhecimento de um especialista humano para a base de conhecimento;

2.3.2. Raciocínio Baseado em Casos

O Raciocínio Baseado em Casos (RBC) teve início com os estudos de Schank [Schank 1982] sobre a compreensão da linguagem. Segundo sua teoria, a linguagem é um processo baseado em memória. Seus pressupostos teóricos afirmam que em cada experiência do ser humano, sua memória passa a se ajustar em resposta a essas experiências, implicando dessa forma, que o aprendizado depende dessas alterações na memória. Através desse estudo, conclui-se que: o processo de compreensão de uma linguagem depende de informações armazenadas previamente na memória, ou seja, pessoas não compreendem eventos sem deixar de fazer referências a fatos que já vivenciaram e que já conhecem [Silva 2005].

O RBC é justamente uma abordagem capaz de possibilitar que algum novo problema enfrentado por certo usuário seja confrontado com uma base de conhecimentos, formada por um conjunto de situações prévias - especialmente organizadas denominada base de casos. Qualquer sistema cuja modelagem esteja fundamentado nessa abordagem pode ser imaginado como sendo um sofisticado sistema de casamento de padrões [Bittencourt et al. 2006].

A representação esquemática das operações que a comunidade de RBC tem utilizado denomina-se de algoritmo dos 4-R. O algoritmo possui este nome devido às fases presentes no ciclo, sendo elas: (1^oR) recuperar caso(s); (2^oR) reusar o(s) caso(s); (3^oR) revisar a(s) solução(ões); (4^oR) reter solução(ões). Este ciclo utilizando RBC é ilustrado na Figura 4.

Com isso, existem quatro fases no processo de inferência em RBC, sendo elas:

1R - Resgate: objetiva achar um caso ou um conjunto de casos similares, provendo solução do problema atual. Algumas formas de recuperação de casos podem ser seqüenciais, em dois níveis, entre outras. Outro aspecto importante nesta fase é a similaridade, da qual existem dois tipos. Primeiro, similaridade local, sendo feita para descobrir a similaridade entre atributos; segundo, similaridade global, para a similaridade entre casos;

2R - Reuso: objetiva fazer o reuso do caso da base de casos para ser utilizado para a solução de um problema. Nesta fase, ocorre a adaptação do caso, quando há a necessidade. Algumas formas de adaptação que podemos citar são: composicional, hierárquica, geracional, entre outras. Esta fase é bastante complicada de prever e generalizar, pois é muito específica de cada domínio;

3R - Revisão: objetiva confirmar uma solução reusada. O processo de revisão pode ser feito de forma automática ou automatizada. Esta fase é bastante

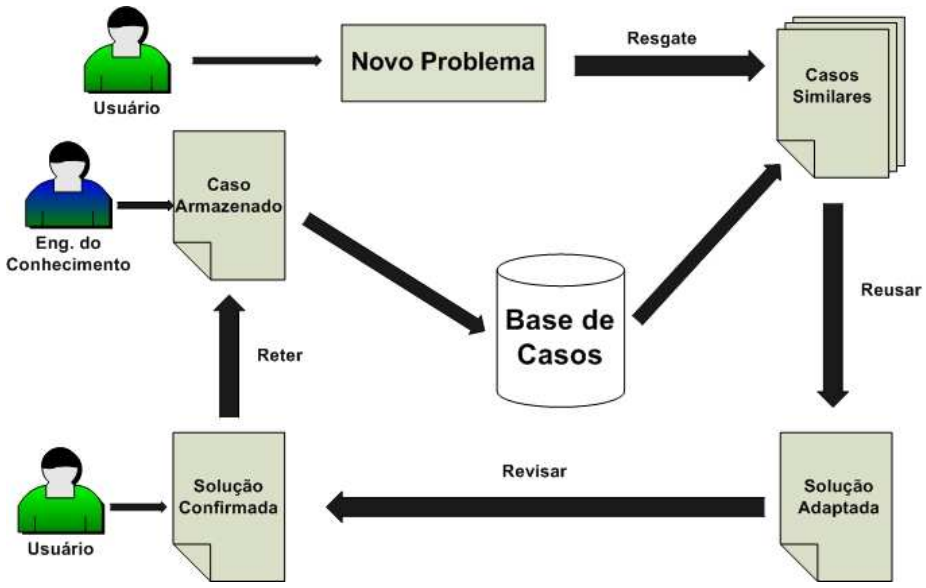

Figura 4. Representação esquemática do algoritmo dos 4-R

complicada e, em alguns casos, pode até haver outra forma de inferência dentro desta fase, apenas para validar;

4R - Retenção: é a fase onde ocorre a aprendizagem do RBC, pois depois dos 3Rs, o novo caso é armazenado na base de casos e depois é retornada a solução para o usuário.

3. Web Semântica

Esta seção descreve as características relacionadas à Web Semântica. Segundo [Berners-Lee et al. 2001], a Web Semântica propiciará uma revolução de novas oportunidades.

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

Esta seção está organizada em duas partes. Primeiramente, as camadas propostas por Tim Berners-Lee são abordadas, além das linguagens para representação de conhecimento e serviços. Na segunda parte, algumas ferramentas são apresentadas, de forma que os conceitos apresentados na primeira parte possam ser aplicados.

3.1. Camadas

As camadas da Web Semântica foram inicialmente propostas por Tim Berners-Lee. A figuda 5 ilustra as camadas da Web Semântica.

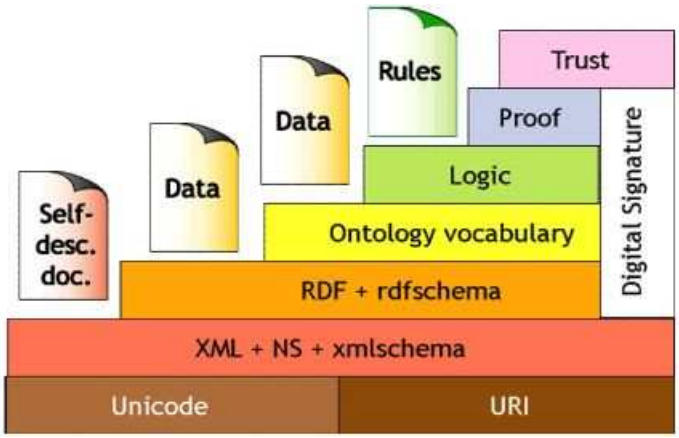

Figura 5. Camadas da Web Semântica.

Como mostrado pela figura 5, as camadas da Web Semântica são:

- URI e Unicode: uma das principais características da Web é a sua flexibilidade para atribuir *coisas* ou *recursos*. Essa flexibilidade é provida pelo URI (Unified Resource Identifiers) e Unicode. URI é um identificador que consiste em uma sequência de caracteres que obedecem a uma regra de sintaxe <URI>. Além disso, tal identificador possibilita a identificação única de um recurso através de um conjunto estensível de esquemas de nome [Berners-Lee et al. 2005];
- XML e XML Schema: estes estão acima da camada URI e Unicode. XML (eXtensible Markup Language) é utilizado para prover estruturas que possam ser processadas por aplicações computacionais. XML oferece aos usuários uma descrição de dados estruturados. Portanto, XML Schema é utilizado para descrever a estrutura de um documento XML. Apesar de aplicações computacionais processarem documentos XML, esta linguagem descreve a estrutura sintática apenas. Por essa razão, uma nova camada é indispensável para definir a estrutura semântica;
- RDF e RDF Schema: o objetivo do RDF (Resource Description Framework) é permitir que recursos da Web Semântica possam ser acessados por máquinas [Breitman 2005]. Em outras palavras, o objetivo do RDF é prover uma representação minimalista do conhecimento da Web [Shadbolt et al. 2006]. Além disso, RDF descreve qualquer recurso presente na Web, como imagens, páginas Web, videos, pessoas e assim por diante. RDF Schema é uma estensão semântica ao RDF e é responsável por prover um conjunto de recursos interrelacionados. Dessa forma, RDF Schema é útil para descrever a estrutura de um documento RDF. No entanto, RDF Schema não suporta inferência nem expressividade. Portanto, uma nova camada é necessária para garantir expressividade;
- Ontologia: de acordo com [Gruber 1993], ontologia é especificação de um vocabulário representacional para um domínio de discurso compartilhado. Tal termo foi emprestado pela filosofia, onde ontologia é uma contagem sistemática de existência. No entanto, no contexto da Web Semântica, o que "existe" é exatamente aquilo que pode ser representado. Não obstante, uma

nova camada é indispensável para prover inferência explícita;

- Lógica: esta camada permite a criação de regras. Dessa forma, inferência explícita é possibilitada por linguagens de regras;
- Prova e Confiança: A camada de prova executa as regras e verifica junto com os mecanismos da camada de confiança se uma dada prova é confiável ou não [Koivunen and Muller 2002]. No entanto, essas camadas estão atualmente sendo pesquisadas e pequenas aplicações demosntrativas estão sendo construídas;
- Assinatura Digital: essa camada provê integridade, garantia e não repudiabilidade sobre os dados da Web. Em outras palavras, assinatura digital pode ser utilizada para verificar a autoridade de um documento.

3.2. Linguagens

Existem diferentes linguagens relacionadas a Web Semânticas e com propósitos diferentes. Esta subseção descreve as linguagens utilizadas para representação, inferência e busca no contexto da Web Semântica.

3.2.1. OWL

OWL (Ontology Web Language) [OWL 2006] é a linguagem ontológica da Web. Ela permite expressividade de alto nível e inferência implícita. Além disso, há três dialetos diferentes de OWL:

- *OWL Lite*: representa uma linguagem muito simples. Portanto, OWL Lite é recomendada para a definição de hierarquias simples e/ou restrições;
- *OWL-DL*[4]: é mais expressiva que a OWL Lite e é baseada na lógica de descrição [Baader and Nutt 2003]. É importante destacar que *OWL-DL* permite a verificação de satisfatibilidade de conceitos e classificação de hierarquias;
- *OWL FULL*: representa a linguagem OWL mais expressiva. Essa linguagem deve ser utilizada quando expressividade do conhecimento for mais importante que garantias de computabilidade.

Outro ponto importante é que expressividade e inferência da OWL são possibilitados pela lógica de descrição. A versão atual da OWL-DL assegura $\mathcal{SHOIN}^{(D)}$ expressividade. Além disso, a versão OWL 1.1 é baseada em $\mathcal{SROIQ}^{(D)}$. Dessa forma, os componentes presentes na OWL são: *i) Classe*: representam conceitos atômicos; *ii) Propriedades*: representam relações binárias entre conceitos; e *iii) Indivíduos*: representam valores de um dado domínio. Além disso, construtores e axiomas são definidos na OWL-DL, como apresentado nas tabelas 1 e 2, respectivamente.

3.2.2. SWRL

SWRL (Semantic Web Rule Language) é uma extensão dos axiomas OWL para incluir regras do tipo *Horn* [lex Sakharov 2003], submetida pelos membros da W3C

[4]DL significa *Description Language*

Tabela 1. Construtores OWL.

Construtor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \cdots \sqcap C_n$	Man $\sqcap$ Woman
unionOf	$C_1 \sqcup \cdots \sqcup C_n$	Doctor $\sqcup$ Lawyer
complementOf	$\neg C$	$\neg$ Woman
oneOf	$x_1 \cdots x_n$	{John, Marie}
allValuesFrom	$\forall P.C$	$\forall$hasSon.Doctor
someValuesFrom	$\exists P.C$	$\exists$hasSon.Lawyer
maxCardinality	$\leq nP$	$\leq$1hasSon
minCardinality	$\geq nP$	$\geq$2hasSon

Tabela 2. Axiomas OWL.

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq$ $\neg$ Female
sameIndividualAs	$\{x_1\} \equiv \{x_n\}$	Luís Inácio Lula da Silva $\equiv$ Lula
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_n\}$	{John} $\equiv$ {Joseph}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasDaughters
equivalentProperty	$P_1 \equiv P_2$	Cost $\equiv$ Price
inverseOf	$P_1 \equiv P_2^-$	hasDaughters $\equiv$ hasSons$^-$
transitiveProperty	$P_1^+ \sqsubseteq P_2$	Ancestral$^+$ $\sqsubseteq$ Ancestral
functionalProperty	$\top \sqsubseteq \leq 1P$	$\top \sqsubseteq \leq$ 1hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leq 1P^-$	$\top \sqsubseteq \leq$ 1hasSSN$^-$

em 2004. SWRL combina sublinguagens de OWL DL e Lite com algumas sublinguagens de RWL, adicionando uma forma simples de construir e manipular regras do tipo Horn de uma maneira sintaticamente e semanticamente coerente, tendo a finalidade de realçar a expressividade.

O principal axioma adicionado à OWL-DL são as cláusulas Horn, que são uma forma de uma implicação entre um antecedente (corpo) e o conseqüente (cabeça). O significado informal de uma regra pode ser lido como: sempre que as circunstâncias especificadas na expressão antecedente forem verdadeiras, então as circunstâncias especificadas no conseqüente devem também ser.

Uma regra pode ser representada da forma abaixo, escrita em uma sintaxe informal compreensível pelo ser humano:

$$antecedente \rightarrow consequente.$$

O antecedente e o conseqüente de uma regra consistem sempre de zero ou mais cláusulas atômicas. As cláusulas atômicas podem ser da forma C(x), P(x, y), Q(x, z), sameAs(x, y) ou differentFrom (x, y). Onde C é uma descrição no OWL-DL, podendo ser um nome de uma classe ou então uma combinação complexa de cláusulas booleanas, etc. P pode ser uma propriedade individual na OWL-DL, e Q pode ser uma propriedade OWL DL avaliada sobre x, e z sendo qualquer uma a variável ou um valor dos dados de OWL. Ou seja, uma cláusula atômica C (x) será verdadeira se x for uma instância da uma classe descrita por C, e uma cláusula atômica P (x, y) (ou Q (x, z)) será verdadeira se x estiver relacionado a y (ou z) pela

propriedade P (ou Q). A cláusula atômica sameAs (x, y) será verdadeira se x for considerado o mesmo objeto que y, e a cláusula atômica differentFrom (x, y) será verdadeira se x e y forem considerados instâncias diferentes [Li et al. 2005].

Considerando as regras citadas acima, torna fácil criar relacionamentos entre as propriedades, um exemplo pode ser visto abaixo:

$$pai(?a, ?b) \wedge irmao(?b, ?c) \rightarrow tio(?a, ?c).$$

Para representar variáveis em SWRL é usado o ponto de interrogação antes do identificador da variável Ainda, para a expressão acima, considere que pai e irmão são propriedades no OWL-DL. Sendo assim, a regra acima afirma que a composição das propriedades pai e irmão implicam na propriedade tio. Em outras palavras, se João é pai de José, e Carlos é irmão de João, então isso implica em dizer que Carlos é tio de José.

Usando SWRL, pode-se definir algumas regras similares como a apresentada anteriormente para expressar relacionamentos entre Classes e propriedades em alguma ontologia OWL. Podendo deixar assim o significado de uma propriedade mais claro. SWRL permite ainda que se possa inserir políticas de controle de acesso, na forma de regra, usando entidades definidas na própria ontologia.

Um definição importante proposta pela W3C é a XML Concrete Syntax, ou swrlx. Essa proposta é uma combinação da OWL Web Ontology Language XML Presentation Syntax com a RuleML XML Syntax para expressar como as regras SWRL devem ser representadas em arquivos XML. XML Concrete Syntax fornece uma série de vantagens: qualquer classe OWL pode ser usada como predicado em regras; axiomas de regras e ontologias podem ser livremente misturados; a interoperabilidade entre OWL e RuleML é simplificada [Horrocks et al. 2004].

A W3C também propôs um conjunto Build-Ins (swrlb) para aumentar a expressividade e o poder de SWRL. Um Build-In é um clausula atômica, da forma como foi descrita anteriormente, e cada um possui uma função bem específica. Os Build-Ins são divididos em módulos, que tratam de problemas específicos, esses módulos são: Built-Ins para Comparação, Built-Ins Matemático, Built-Ins para Valores Booleanos, Built-Ins para Strings, Built-Ins para Datas, Tempo e Duração, Built-Ins URIs e Built-Ins para Listas; para maiores informações sobre esses módulos consulte [Horrocks et al. 2004]. Cada implementação do SWRL seleciona quais módulos deve cobrir.

SWRL é implementada por uma série de frameworks, porém ela é raramente implementada completamente porque ela não é decidível.

3.2.3. SPARQL

SPARQL [SPARQL 2007] é uma linguagem de consulta para repositório RDF (*Resource Description Framework*). RDF é um grafo rotulado e direcionado para representação de informações na Web. Essa especificação define a sintaxe e semântica da linguagem de consulta SPARQL.

Um grafo RDF é um conjunto de triplas. Cada tripla consiste de *sujeito*, *predicado* e *objeto*. *Sujeitos* são recursos ou ”coisas”das quais deseja-se tratar. Por exemplo, Projeto de Pesquisa, Instituição e Pesquisador. Cada sujeito possui um identificador, uma URI (*Universal Resource Identificator*). Esta URI pode ser uma URL (*Unified Resource Locator*) ou outro tipo de identificador. *Predicados* descrevem relacionamentos entre os sujeitos. Por exemplo, *”pertence a”* e *”possui”*. Predicados também são identificados através de URIs.

Para exemplificar essas definições, considere a seguinte sentença: *Ig Bittencourt participa do Projeto de Pesquisa GRoW*. Essa sentença possui as seguintes partes:

- **Sujeito**: Projeto de Pesquisa GRoW;
- **Predicado**: participa;
- **Objeto**: Ig Bittencourt.

Em RDF, essa sentença seria descrita da seguinte forma:

```
<rdf: RDF>//
    <rdf:Description about: "Projeto de Pesquisa GRoW">//
        <f: participa>//
           Ig Bittencourt//
        </f: participa>//
    </rdf:Description>//
</rdf:RDF>//
```

Através da linguagem de consulta SPARQL é possível extrair informações sobre valores de atributos, extrair subgrafos RDF e construir novos grafos RDF baseados nos resultados de consultas. Os resultados das consultas podem ser ordenados e restringidos de diferentes formas [SPARQL 2007] e [David Nadler Prata 2006].

Para responder a pergunta: *”Qual(is) projeto(s) Ig Bittencourt participa?”*, a consulta em SPARQL deve ser construída como descrita abaixo:

```
SELECT ?ProjetoPesquisa WHERE
{?ProjetoPesquisa:participa:IgBittencourt}
```

3.3. Anotação de Páginas Web

Outro ponto importante em aplicações na Web Semântica é a anotação de páginas Web. Um exemplo é o Dublin Core Metadata Initiative o qual descreve recursos na Web. Os elementos presentes no Dublin Core são assunto, título, criador, descrição, editora, data, tipo, formato, identificador, relação, fonte, linguagem e direitos.

Além disso, o termo metadata refere-se a qualquer informação utilizada na identificação, descrição e localização de um recurso. Como ilustração, a figura 6 mostra uma página Web anotada.

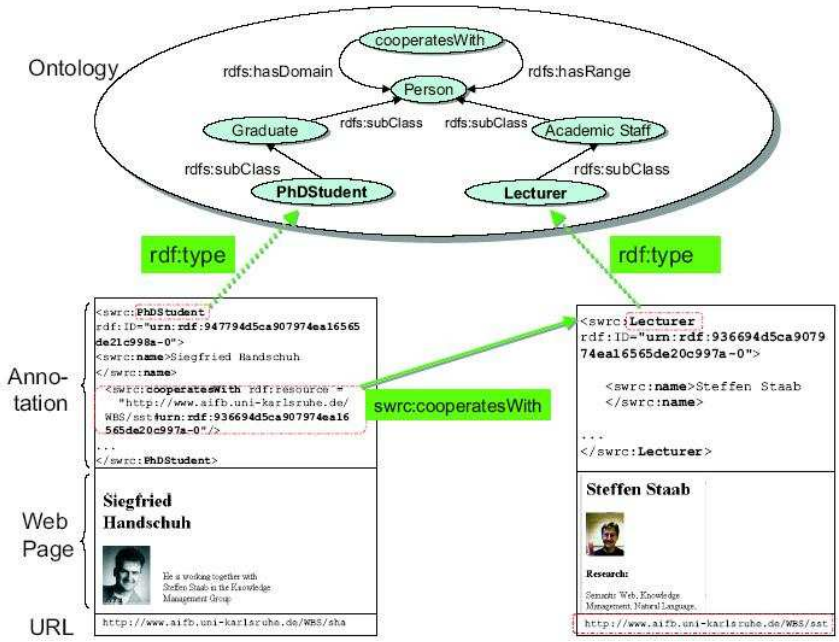

Figura 6. Exemplo de uma página Web anotada.

3.4. Ferramentas

Esta subseção descreve algumas ferramentas que servem de apoio para a construção de aplicações baseadas na Web Semântica.

3.4.1. Jena

Jena é um framework Java utilizado na construção de aplicações para Web Semântica. Ele provê um ambiente programático para RDF, RDFS, OWL, SPARQL e inclui um motor de inferência baseado em regras [HP 2006].

Este framework é utilizado por desenvolvedores na construção de sistemas baseado em conhecimento devido as seguintes características:

- Open-Source;
- É escrito em Java;
- Existe um grande número de aplicações escritas utilizando Jena;
- Possui vasta documentação explicando sua utilização;
- Possui listas de discussões e desenvolvedores trabalhando em novas versões;

Além disso, Jena disponibiliza uma API para RDF e OWL. A API RDF possui muitas classes para criação a manipulação de grafos RDF. Da mesma forma, a API OWL fornece uma interface para criar e manipular ontologias.

3.4.2. Protégé

Protégé [Protégé 2006] é um ambiente integrado para a modelagem e a aquisição de conhecimento voltado para o desenvolvimento de ontologias, desenvolvido pelos

pesquisadores do *Stanford Medical Informatics* [SMI 2006]. O desenvolvimento de ontologias no protégé pode ser feito de duas maneiras, utilizando o i) *Protégé-Frames*: o usuário pode construir e povoar ontologias que são baseadas em *frames*, em conformidade com *Open Knowledge Base Connectivity* [OKBC 1995] ou ii) *Protégé-OWL*: o usuário pode criar ontologias para a Web Semântica, em particular em W3C's *Ontology Web Language*.

Esta ferramenta é usada por desenvolvedores de sistemas e especialistas do domínio para construir sistemas baseado em conhecimento, pois oferece vantagens como:

- Livre;
- Código aberto;
- Pode ser exportado em uma variedade de formatos, como RDF(S), OWL, XML Schema, outros;
- Baseado em Java;
- Geração automática de código java;
- É extensível e disponibiliza um ambiente baseado em plug-ins, tornando-o flexível para o desenvolvimento de aplicação e rápida prototipação;
- Disponibiliza diversos plug-ins para uso (aproximadamente 90);
- Possui documentação abordando o uso da ferramenta;
- Possui listas de discussões e uma equipe de desenvolvimento dando continuidade na geração de novas versões;
- É suportado por uma comunidade de mais de 76.000 usuários.
- Uma conferência dedicada com discussões sobre a ferramenta Protégé;

Atualmente o protégé está na versão 3.3, onde o principal foco desta versão é o desenvolvimento de ontologias em *OWL*. Além de criar ontologias em OWL, o desenvolvedor poderá fazer o gerenciamento de mudanças, depurarem ontologias em OWL-DL, dentre outras características. O grupo de desenvolvedores responsável pelo protégé já está desenvolvendo a versão 4.x.

Protégé-OWL O Protégé-OWL é uma extensão do Protégé para dar suporte a Web Ontology Language (OWL). Dentre diversas funcionalidades, o editor para o Protégé-OWL permite ao desenvolvedor:

- Carregar e salvar ontologias nos formatos OWL e RDF;
- Editar e visualizar classes, propriedades e regras (SWRL[5]);
- Definir características lógicas das classes através de expressões OWL-DL;
- Editar indivíduos OWL.

3.4.3. RacerPro

RacerPro significa Renamed ABox and Concept Expression Reasoner Professional. É a nova geração de sistemas de Lógica de Descrição [RacerPro 2000]. RacerPro é um sistema capaz de manipular ontologias baseadas em OWL. Suas caracteríticas são:

[5]Do Inglês: Semantic Web Rule Language.

- Raciocínio e Repositório: RacerPro processa documentos OWL Lite e OWL-DL tentando verificar a consistência e a satisfatibilidade da ontologia. Além disso, RacerPro implementa SWRL;
- Sistema de Lógica de Descrição: RacerPro implementa lógica de descrição $\mathcal{SHIQ}$;
- Combinação de Lógica de Descrição e Álgebras Relacionais Específicas: RacerPro combina raciocínio em lógica de descrição com, por exemplo, raciocínio sobre relações espaciais (ou temporais) no contexto da linguagem de busca nRQL [GmbH and KG 2005].

Outro ponto importante é que o Protégé utiliza o RacerPro para verificar a consistência, classificar taxonomias e computar tipos inferidos, como mostrado na figura 7.

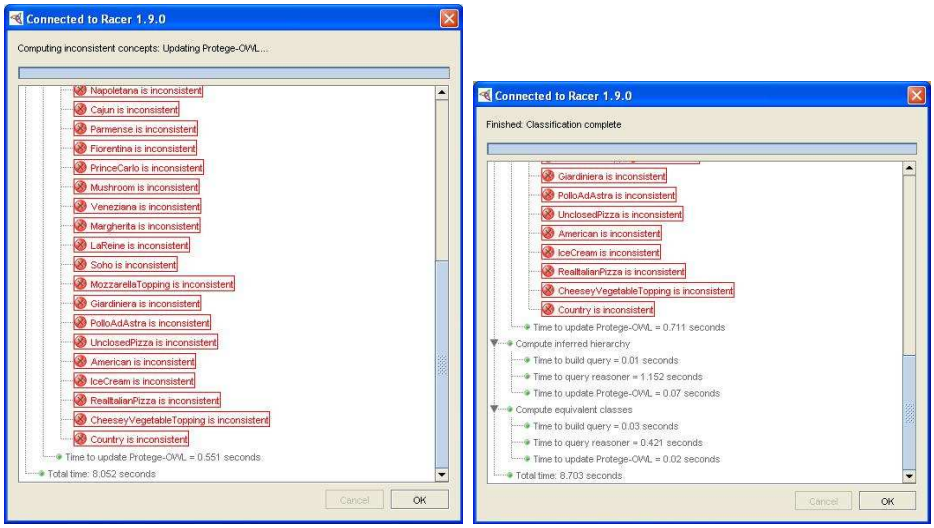

Figura 7. Tela exibindo consistência.

4. Serviços

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

A seguir é apresentada algumas das principais características da Arquitetura Orientada a Serviços[Singh and Huhns 2004]:

1. Neutralidade de Implementação: serviços implementados em diferentes tecnologias podem se comunicar, visto que em SOA, o que importa é a interface do serviço. Por exemplo, uma aplicação desenvolvida na plataforma JEE pode invocar um serviço que foi desenvolvido na plataforma .NET;
2. Fraco acoplamento: Em SOA cada serviço, deve ser entendido como um elemento independente, da mesma forma que um componente é independente na Arquitetura Orientada a Componentes;

3. Granularidade: os serviços em SOA devem ser tomados como elementos de baixa granularidade, pois deve-se ter uma visão mais abstrata quanto possível, ocutando detalhes como modelagens de ações e interações;
4. Configuração flexível: os sistemas devem suportar configurações de maneira flexível e dinâmica.

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

Pelo fato de SOA ser uma arquitetura, não existe um implementação padrão. Ela pode ser implementada utilizando diversas tecnologias, como Corba, RPC, Web Services, DCOM, etc. Nesse minicurso será apresentado uma forma de implementação de SOA, que é a partir de *Web Services*. A tecnologia *Web Services* foi escolhida pelo fato de ser uma das mais utilizadas ao se trabalhar com SOA e também por ser uma das tecnologias em que mais se tem trabalhado para tentar agregar semântica as descrições de serviços.

4.1. *Web Services*

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

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

Segundo a visão da IBM(ver [Kreger 2001]) a arquitetura dos *Web Services* está baseada na interação dos seguintes papéis: provedor do serviço, registrador de serviços e consumidor de serviços. As interações envolvem a publicação, a busca e a invocação do serviço. Neste esquema, o provedor do serviço desenvolve o serviço e publica sua descrição em um registrador de serviços, o consumidor de serviço efetua busca nos registros de serviços para descobrir o serviço que atende as suas necessidades, através das descrições adicionadas pelos provedores de serviços. Após descobrir o serviço ideal o consumidor utiliza a descrição para interagir diretamente com o consumidor e invocar o serviço desejado. Esta interação pode ser vista na figura 8.

Diante do exposto algumas tecnologias foram desenvolvidas como forma a permitir tanto a publicação quanto a descoberta e invocação de serviços, como: registros UDDI, linguagem WSDL e o protocolo de troca de mensagens SOAP.

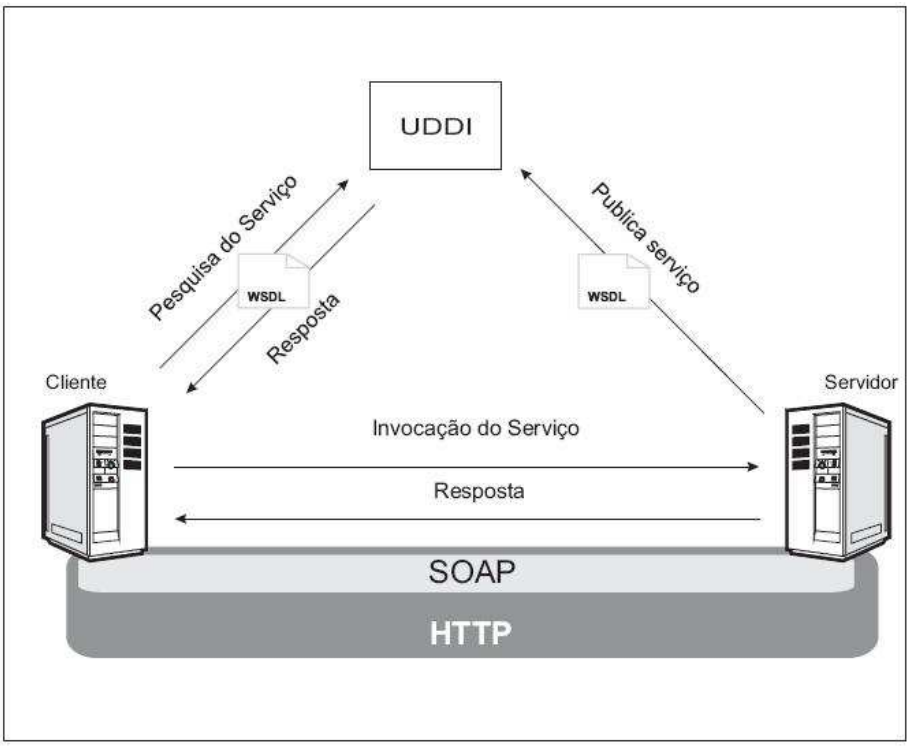

Figura 8. Modelo da interação entre cliente, servidor e registro UDDI

4.1.1. Servidores de registro UDDI

Os registros UDDI[6] são repositórios de descrições de serviços que permitem a busca. Os consumidores de serviços podem efetuar busca pelos serviços nesses registros tanto em tempo desenvolvimento, quanto em tempo de execução (*on-the-fly*), para recuperar as informações necessárias à invocação. A utilização dos registros UDDI para recuperação em tempo de desenvolvimento muitas vezes é desnecessária, já que o desenvolvedor pode obter essas informações de outras fontes, como um site ou mesmo diretamente do provedor do serviço. A real serventia dos registros UDDI aparece quando o consumidor precisa descobrir um serviço em tempo de execução para que possa invocá-lo.

A especificação UDDI consiste de 4 entidades que efetuam a descrição do serviço:

- *BusinessEntity*: define a entidade que está provendo o serviço;
- *BusinessService*: define qual a finalidade do serviço;
- *BindindTemplate*: define qual a localização do serviço e como invocá-lo;
- *tModel*: possibilita a interoperabilidade do serviço. Em geral, essa entidade é representada pelo arquivo WSDL que descreve o serviço.

Apesar dos registros UDDI possibilitarem o armazenamento e a descoberta de novos serviços, sua especificação limita a forma como pode ser automatizado o processo de descoberta, já que sua estrutura está descrita meramente em termos

[6] *Universal Description, Discovery and Integration*

sintáticos, não permitindo nenhuma interoperação semântica, o que possibilitaria que todo o processo de descoberta, seleção e execução fosse feito de maneira automática.

4.1.2. Linguagem de descrição de *Web Services* – WSDL

WSDL é uma linguagem baseada em XML para descrição de interfaces, protocolos de invocação e detalhes de distribuição de *Web Services*. Um documento WSDL define serviços como coleções de pontos de acesso via rede chamados *ports*. A definicição de cada *Web Services* define sua localização, os parâmetros necessários e seus tipos, seu retorno, ou seja, todas as informações necessárias à invocação.

WSDL complementa o modelo de representação de serviços utilizado pelos registros UDDI, ao prover uma descrição abstrata da interface do serviço e as formas de invocá-los [Zhou et al. 2006]. Para tal, utiliza uma estrutura para descrição de serviços dividida em 6 elementos principais[Christensen et al. 2001]: *Types*, *Message*, *PortType*, *Operation*, *Binding*, *Port*, *Service*.

Assim como a descrição provida pelos registros UDDI, a definição de serviços efetuada pelo WSDL, é puramente sintática, já que sua estrutura é baseada em estrutura de XML schema, não permitindo interoperação semântica a nível de ontologias.

4.2. *Semantic Web Services*

Os *Semantic Web Services* são a próxima etapa dos *Web Services*, pois no mundo da Web, em que a cada momento mais e mais tenta-se agregar conhecimento entendível por máquinas aos recursos, os *Semantic Web Services* podem ser vistos como um ponto de intersecção entre a tecnologia dos *Web Services* e a Web semântica por prover a interoperabilidade semântica encontrada na Web semântica juntamente com a dinamicidade de recursos presentes nos *Web Services*[C.Daconta et al. 2003] (figura 9).

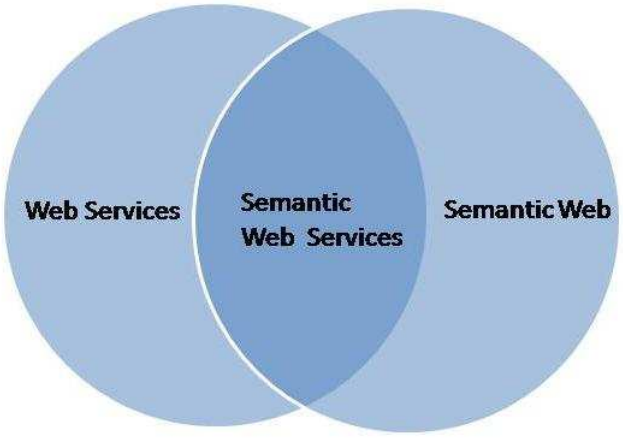

Figura 9. SW + WS = Semantic Web Services

Segundo [Payne and Lassila 2004], os *Semantic Web Services* podem ser definidos como um aprimoramento das descrições dos *Web Services* através da

utilização de anotações semânticas, de forma que possa ser conseguido um alto grau de automação, tanto na descoberta de serviços, quanto na composição, monitoramento e invocação de *Web Services*.

Para um melhor entendimento, imagine um cenário fictício em que um estudante de computação resolve tirar suas férias viajando para Europa. Para conseguir realizar tal tarefa ele terá de seguir um conjunto de passos como: fazer uma busca por preços de passagens aéreas, efetuar a reserva, confirmar o pagamento, fazer reservas em hotéis, etc. Suponha que cada uma dessas tarefas possam ser satisfeitas à partir da invocação de *Web Services* independentes. Utilizando *Web Services* tradicionais, em cada uma das etapas deveria haver algum tipo de intervenção do usuário. Se, ao contrário os *Web Services* estivessem anotados semanticamente, seria possível o estudante configurar um agente de software capaz de, sem intervenção humana, buscar e selecionar serviços que atendam aos requisitos do usuário, montar um plano de execução efetuando uma composição de todos os serviços selecionados e invocá-los de maneira automática.

Diante dos desafios de cenários como o colocado, impõem diversas iniciativas de pesquisa estão sendo desenvolvidas com o intuito de agregar semântica à atual tecnologia de *Web Services*. Alguns modelos propõem um melhoramento das atuais linguagens de descrição de serviços, como o WSDL-S, que neste caso teria como principal benefício a descoberta de serviços. Contudo, as principais iniciativas tem um objetivo mais ousado, que é prover uma maior semântica as descrições dos serviços, o que permitiria não apenas a descoberta automática, mas também a composição e a invocação automáticas de serviços.

De todas as iniciativas, as que mais se destacam por apresentarem modelos mais consistentes e completos são uma iniciativa européia, a WSML[7](*Web Service Markup Language*) e uma iniciativa norte-americana, a OWL-S (Ontology Web Languagen for Services). Nessa disputa, a OWL-S leva, atualmente, uma vantagem em relação a sua concorrente, já que é recomendada pela W3C, o que atrai um maior número de pesquisadores a trabalhar com ela. Diante deste fato, neste trabalho será utilizada a OWL-S como ontologia padrão para descrição de serviços.

4.3. OWL-S: *Ontology Web Language for Services*

Embora algumas vezes a OWL-S seja referida como uma linguagem, porque provê um vocabulário padrão para representação de serviços, trata-se de uma ontologia de topo (*Upper Ontology*) lançada em meados de 2004 em continuação ao projeto DAML-S[Ankolekar et al. 2004]. Por ser uma recomedação da W3C, a OWL-S tem atraído o interesse de grande parte da comunidade científica relacionada a *Semantic Web Services* e grande parte das ferramentas desenvolvidas tem como base a utilização dela.

Motivação A OWL-S foi desenvolvida tendo como objetivo permitir a criação de *Semantic Web Services* capazes de atender a grande parte dos requisitos esperados de um serviço semântico[McIlraith et al. 2001]. Entre eles, os principais que podem ser são:

[7]Anteriormente conhecida como WSMO (Web Service Modelling Ontology)

- Descoberta automática de serviços: a OWL-S possibilita a descrição de serviços com um alto grau de semântica, o que permite que um agente de software descubra e selecione de maneira automática um certo serviço de acordo com seu objetivo;
- Invocação automática de serviços: de forma semelhante ao item anterior, a OWL-S permite que os serviços possam ser invocados sem intervenção de nenhum agente humano;
- Composição automática de serviços: dado a elevada semântica presente nas descrições de serviços, agentes de software estão aptos a realizar a combinação de serviços, sem qualquer intervenção humana, de maneira que uma tarefa complexa possa ser executada;

Estruturação Diferentemente da linguagem de descrição de serviços WSDL, a OWL-S efetiva a descrição de serviços semanticamente, o que permite uma maior interopeção. Segundo [Martin et al. 2004] a estruturação de uma ontologia de serviços é motivada pela necessidade de três essenciais tipos de conhecimento básicos sobre um serviço, o que poderia ser referenciado pela resposta às três perguntas abaixo:

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

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

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

1. Um perfil(*Profile*) que apresenta as "intenções" do serviço, isto é, qual a funcionalidade do serviço, o que ele provê;
2. Um modelo do Serviço(*ServiceModel*) que apresenta um maior detalhamento do funcionamento do serviço;
3. Um *Grounding*[8] que apresenta detalhes relacionado a maneira como o serviço pode ser invocado.

O modelo abstrato de um *Semantic Web Services* em OWL-S com todos os relacionamentos entre as entidades presentes em um *Semantic Web Services* pode ser vito na figura 10.

A seguir será apresentado uma descrição mais detalhada da estrutura da OWL-S (perfil, modelo e Grounding), sendo dado maior enfoque ao perfil, por ter uma maior correlação com este trabalho.

Perfis de serviços O objetivo do Service Profile é fornecer uma descrição sobre qual o objetivo da utilização do Web service, de forma que o consumidor seja capaz de selecionar o serviço que mais atende as suas expectativas. Em geral as principais

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

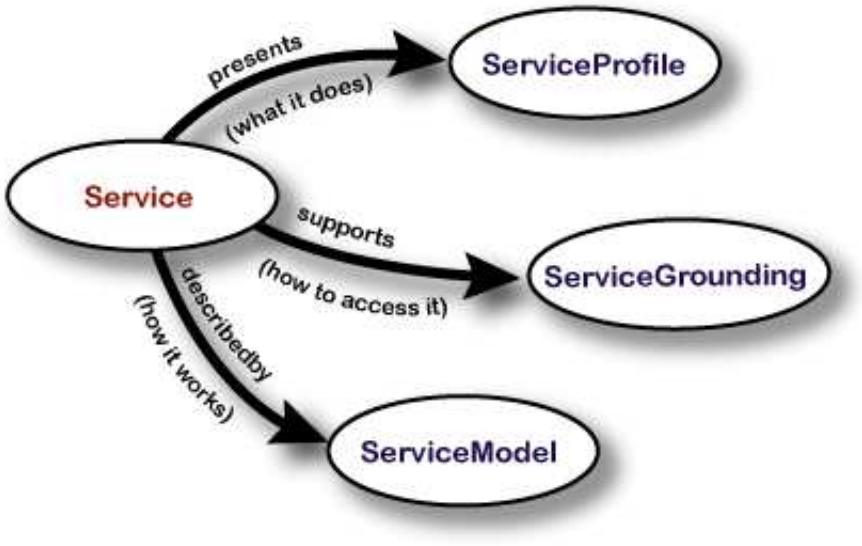

Figura 10. Estrutura proposta de um serviço semântico em OWL-S

informações utilizadas são os tipos de entradas e retorno do serviço, precondições e efeitos da execução do serviço. A primeira vista, essas poucas informações não parecem suficientes para prover toda a interoperabilidade esperada de um serviço semântico, pois só se está utilizando tipos de entrada e saída para avaliações, no entanto, a interoperabilidade semântica é garantida pelo fato de que todos os parâmetros possuem ontologias que os representam no mundo. Suponha por exemplo dois serviços que efetuem a tarefa de um tradutor português-inglês, um sendo descrito em termos puramente sintáticos (*Web Services* tradicionais) e outro sendo descrito em OWL-S. No primeiro caso como a única informação disponível é a indicação que o serviço espera como entrada uma string e tem como retorno outra string, nenhum agente de software será capaz, de maneira automática, de perceber que se trata de um tradutor. Enquanto que, no caso do serviço descrito de maneira semântica, há metadados relacionados a entrada e ao retorno(isto é, ontologias), que possivelmente indicaria que o texto de entrada é uma palavra em língua Portuguesa enquanto que a saída é em língua Inglesa, permitindo que agentes de software realizem raciocínios sobre tal conhecimento.

Diante deste contexto, o perfil foi desenvolvido para representar o primeiro passo no processo de utilização de um serviço semântico, que é a descoberta.

De maneira mais concreta, o perfil do serviço está implementado através de uma classe OWL chamada *Profile*. As informações contidas na classe *Profile* podem ser divididas basicamente em três tipos: *i)* Parâmetros que relacionam o perfil aos demais componentes da descrição do serviço; *ii)* Parâmetros que determinam dados referentes a origem do serviço, desenvolvedor, nome, etc. *iii)* Parâmetros que apresentam informações relacionadas principalmente às entradas, saídas, precondições e efeitos (IOPE's) do serviço.

Na figura 11 pode ser visualizada toda a estrutura da classe *Profile*. Em seguida são discutidas as principais propriedades da classe *Profile*.

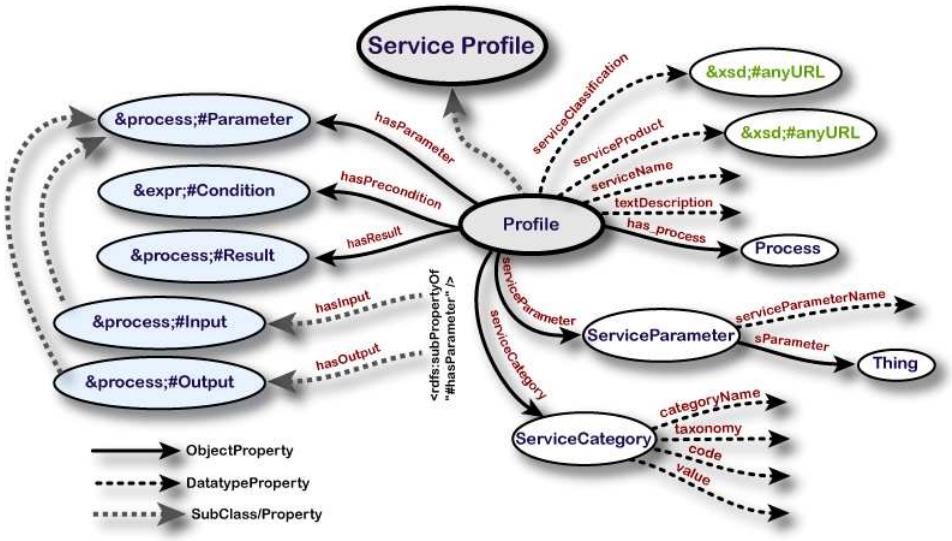

Figura 11. Classe Profile

1. Possui conjunto de propriedades que interligam o profile ao serviço que ele apresenta;
    - *presents* especifica qual serviço está sendo apresentado;
    - *presentedBy* propriedade inversa da anterior.
2. Possui um conjunto de propriedades que dão uma descrição geral do serviço;
    - *serviceName* especifica o nome do serviço;
    - *textDescription* dá uma descrição do serviço. Essa descrição é utilizada apenas para usuários humanos, não sendo possível efetuar qualquer inferência sobre esta propriedade;
    - *contactInformation* é uma propriedade que especifica informações sobre o provedor do serviço. Pode ser tanto um texto, ou um caminho para alguma ontologia.
3. Um conjunto de descrições funcionais do serviço. Basicamente, apresenta as propriedades de entrada, saída, pré-condições e efeitos (IOPE's), sendo as principais propriedades que são utilizadas no processo de descoberta, nas ferramentas de busca. Normalmente, os valores presentes nesses parâmetros são meramentes apontadores para os parâmetros semelhantes, presentes na classe *Process*(a ser explicado a seguir), de forma que se mantenha uma maior coesão na descrição do serviço;
    - *hasParameter* é o conjunto de todas as entradas e saídas esperadas do serviço;
    - *hasInput* especifica apenas as entradas que o serviço necessita para operar. Os tipos permitidos estão dentro do conjunto de instâncias da classe *Input* da ontologia *Process*
    - *hasOutput* especifica todas as saídas que serão produzidas após a execução do serviço. Assim como a propriedade *hasInput*, os tipos aceitos seguem alguma instância presente na ontologia *Process*, mas nesse caso, da classe *Output*;
    - *hasPrecondition* espefica as pré-condições que devem ser satisfeitas para que o serviço possa operar. Um exemplo de uma pré-condição

seria um serviço de venda de livros, o usuário precisa de um cartão de crédito ainda válido. Pode variar de acordo com as instâncias definidas da classe *Precondition*;

- *hasResult* especifica um dos resultados da execução do serviço, assim como definida pela classe *Result*. Ou seja especifica sobre quais condições a saída é gerada. Além disso, especifica quais serão as mudanças no domínio dada a execução do serviço.

Modelo do Serviço O objetivo do *ServiceModel* é dar uma visão bem mais detalhada do serviço em relação ao *serviceProfile*. Este elemento da descrição terá sua utilização iniciada após o momento da descoberta e da seleção do serviço desejado. Isto é, quando o agente de software necessitar de informações mais concretas do funcionamento do serviço para poder efetuar interoperações (como composição de serviços para efetuar uma tarefa mais complexa, criação de um plano de execução, etc.), e posteriormente sua devida invocação, à partir do terceiro elemento serviço semâtico em OWL-S, que é o *Grounding*.

Para modelar esse conceito do serviço semâtico, a OWL-S disponibiliza uma classe chamada (*Process*). A partir dessa classe será possível criar um plano de execução. É importante notar que um processo não é um programa a ser executado, mas sim, um plano de execução, isto é, um conjunto de etapas que o cliente deverá utilizar para interagir com o serviço desejado.

Como já foi explicitado, muitas das propriedaes presentes no perfil do serviço, também apresenta-se no modelo do serviço, como IOPE's. E são à partir dessas propriedade que no *Process* é possível criar estruturas que descrevem como serviços diferentes podem ser composto para uma tarefa complexa possa ser executada.

A classe *Process* não é utilizada de maneira direta, mas sim pela utilização de uma das três classes derivadas dela, que são:

1. *AtomicProcess* são diretamente invocáveis (passando as devidas mensagens de entrada). O fluxo de execução de um processo atômico é análogo à de uma invocação de um método. Sendo assim, um processo atômico não possui nenhum subprocesso. Uma outra informação, que faz-se importante notar, é o fato que um processo atômico não apresenta o conceito de estado de execução, como ocorre com o serviço composto e portanto não efetua a monitoração durante a execução, pois não há o conceito de estado interno da execução;
2. *SimpleProcess* pode ser visto como uma abstração de um processo. Enquanto um *AtomicProcess* e um *CompositeProcess* tentam dar uma visão de uma caixa-de-vidro ao processo, um *SimpleProcess* mais se assemelha a uma caixa-preta. *SimpleProcess* são usados como elementos de abstração, no qual pode fornecer uma visão mais especializada de um *AtomicProcess* ou uma visão simplificada de um *compositeProcess* para efeitos de inferência e contrução de planejamento. Um *SimpleProcess* se relaciona com os outros tipos de processo no fato de que um *AtomicProcess* realiza um *Simpleprocess* (pela propriedade *realize* e pela sua inversa, *realizedBy*) e um *compositeProcess* é uma expansão de um *SimpleProcess* (pela propriedade *expandTo*, havendo também sua propriedade inversa, *collapsesTo*);

3. *CompositeProcess* são processos que podem ser decompostos em outros processo (atômicos ou complexos). Esses subprocessos serão executados de acordo com um conjunto de regras definidas em um plano de execução e na qual a entrada de dados para alimentar os subprocessos pode ser proveniente tanto do usuário do serviço, quanto de uma saída de um processo anterior.
   As regras de controle de construção (*control construct*) do plano de execução (ou *workflow*) irão definir de que maneira os serviços serão executados, a ordem de execução, se a execução deve ser síncrona, etc. Muitos desses controles de contrução assemelham-se aos controle de fluxos de linguagens estruturadas, como controle If-Then-Else, que efetua a construção de um plano de execução com seleção, ou Repeate-While, que efetua loop na execução do processo até que uma dada informação mostre-se falsa.

Grounding O *Grounding* apresenta-se como último elemento a ser utilizado ao se trabalhar com *Semantic Web Services* , pois é justamente nele que está descrita a maneira como um serviço semântico pode ser invocado.

Ele pode ser considerado como o elemento mais próximo do mundo sintático se comparado ao perfil e ao modelo, já que ele pode ser visto como uma interface, ou "cola" entre as descrições puramente semânticas do perfil e do modelo e a descrição sintática do WSDL[Kopeck et al.], permitindo que o consumidor do serviço obtenha informações, como por exemplo, a maneira como serializar uma requisição, presente nos arquivos de descrição WSDL.

4.4. Ferramentas

4.4.1. Mindswap

Mindswap [Mindswap] é uma OWL-S API que foi desenvolvida em Java para a manipulação de ontologias baseadas em OWL-S através de objetos. Mindswap provê mecanismos para ler, escrever e executar serviços de ontologias. Além disso, possui suporte para a execução de serviços cujos "groundings"são baseados em WSDL ou UPnP.

Objetos definidos pela API seguem a estrutura da especificação OWL-S. Além disso, relações entre objetos podem ser definidas. Em instância, a classe Service apresenta um atributo do tipo Profile que segue a mesma especificação da descrição do serviço.

Outro ponto importante é que a API é focada nas facilidades para composição de serviços utlizando uma base de conhecimento para armazenar os serviços lidos de cada ontologia.

5. Agentes

Neste capítulo serão apresentados os principais fundamentos relacionados ao uso das abordagens utilizadas na plataforma proposta. Isso inclui também as tecnologias utilizadas ao longo do desenvolvimento da plataforma que foram: ontologias, agentes inteligentes e mecanismos de inferência.

5.1. Agentes Inteligentes

O uso de agentes tem sido bastante explorado pela comunidade de Inteligência Artificial. Segundo [Russel and Norvig 2004], um agente é uma entidade que possui a capacidade de perceber um ambiente, através de sensores, e agir no ambiente, através de atuadores. Uma arquitetura abstrata de um agente é mostrada na Figura 12

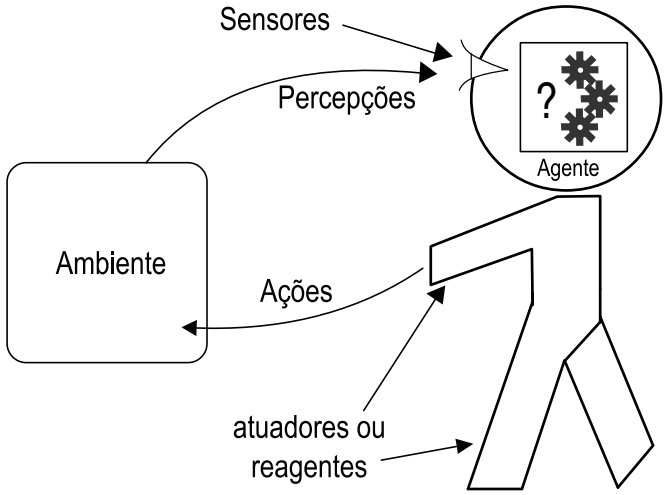

Figura 12. Arquitetura Abstrata de Agentes
Fonte: Extraída e adaptada de [Russel and Norvig 2004].

A abordagem de agentes é normalmente motivada pelas seguintes características:

- Racionalidade: os dados de entrada do agente equivalem a informações que são tratadas de forma correta, não necessitando de interferências de terceiros. Além disso, a racionalidade pode ser medida através de fatores:
    1. Medida de desempenho do agente;
    2. Conhecimento do agente sobre o ambiente;
    3. Capacidade de atualização dos agentes, ou seja, pertinência das açõs de cada agente;
    4. Percepções dos agentes com relação ao ambiente que está sendo observado.
- Aprendizado: capacidade do agente de aprender em suas interações;
- Autonomia: característica essencial para garantir o ciclo de vida de um agente, onde as percepções e ações dos agentes garantem sua capacidade de autonomia.

Os agentes podem ser caracterizados de diversas formas (agente reativo, baseado em objetivos, aprendizagem, etc), porém, a abordagem de agentes inteligentes tem sido também, explorada pela comunidade de inteligência artificial em educação. Tais agentes fazem uso de sistemas baseados em conhecimento para poder garantir sua inteligência no processo de resolução de problemas.

Porém, a complexidade envolvida na resolução de problemas motiva uma abordagem multiagente, que será o foco da subseção seguinte.

5.2. Sistemas Multiagentes

A construção de software robusto e de qualidade para domínios como telecomunicações, controle industrial, e gerenciamento de processos de negócios representam algumas das mais complexas tarefas humanas. Neste contexto, a função do engenheiro de software é disponibilizar estruturas e técnicas que permitam o fácil gerenciamento desta complexidade.

Sistemas complexos, como mostra a Figura13, apresentam algumas regularidades:

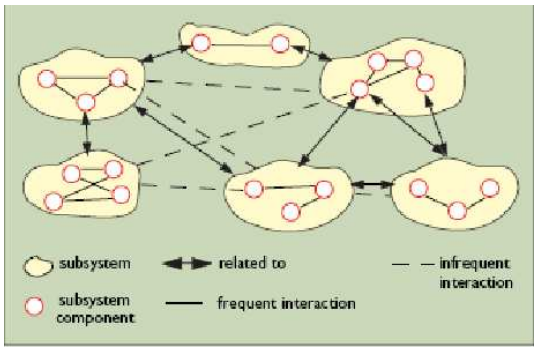

Figura 13. Visão de Sistema Complexo

- Frequentemente tomam a forma de uma hierarquia. Ou seja, um sistema composto de subsitemas interrelacionados, e assim sucessivamente até atingir um subsistema elementar. Diante deste cenário, observamos que tais relacionamentos não são estáticos (variam com o tempo)
- A escolha de qual susbsistema é a unidade elementar dependerá dos objetivos almejados.
- Sistemas hierárquicos evoluem mais rapidamente do que sistemas não hierárquicos com igual tamanho.
- É possível distinguir interações entre subsistemas e aquelas presentes dentro dos subsistemas.

Desta forma, o engenheiro de software têm alguns conceitos fundamentais que o auxiliam no gerenciamento da complexidade:

- Decomposição
  Recebe grandes problemas como entrada, e o subdivide em pedaços menores. Com isso, limitando o domínio de atuação.
- Abstração
  Processo de definir um modelo simplificado do sistema que represente algumas dos detalhes e propriedades, e omite outras. Desta forma, limitando o escopo de interesse em um determinado instante.
- Organização
  Processo de definir e manter os relacionamentos entres os componentes. Desta forma, permitindo uma visualização de um conjunto de pequenos componentes como uma unidade unificada.

Tendo apresentado visões e conceitos relacionados a sistemas complexos, torna-se visível que a maioria dos problemas considerados anteriormente requerem ou envolvem vários agentes, seja para representar a natureza descentralizada do problema, múltiplas atividades de controle e perspectiva ou até mesmo os interesses competitivos. Logo, os agentes precisarão interagir entre eles almejando atingir seus objetivos particulares ou gerenciar dependências inatas de ambientes complexos. Tais interações podem variar de simples operações semânticas, que ocorrem através de interações cliente-servidor, para ricas interações sociais, onde encontramos a habilidade de cooperar, coordenar e negociar sobre um plano de ações.

Um sistema multiagente geralmente atua em um contexto organizacional, tal contexto define a natureza das interações, por exemplo: um agente pode ser uma pessoa trabalhando em um time ou ainda ser um gerente em uma determinada organização. Logo, assim como os contextos são dinâmicos na vida real, os sistemas multiagentes também apresentam suporte a dinamicidade no estabelecimento de suas relações.

Por fim, como mostra a Figura14, verificamos que adotar uma abordagem orientada a agentes para engenheiros de software significa decompor os problemas em múltiplos componentes autônomos que podem atuar e relacionar-se em um caminho flexível almejando atingir seu conjunto de objetivos, modelar abstratamente uma parte de interesse do problema em agentes, interações e unidades organizacionais, e finalmente, desenvolver estruturas e mecanismos que freqüentemente são usados para descrever e gerenciar a complexidade e mudanças ocorridas nos relacionamentos organizacionais existentes entre os agentes.

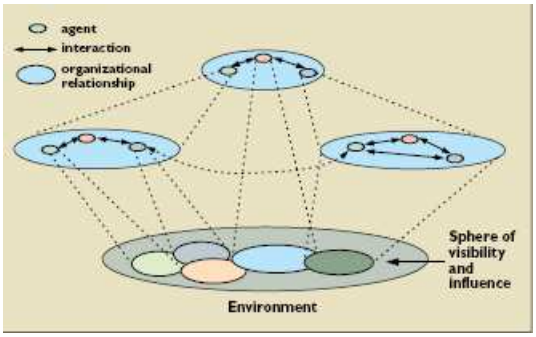

Figura 14. Visão de Sistema Multiagente

5.3. Ferramentas

5.3.1. Jade

O JADE (*Java Agent Development Framework*) é uma plataforma *open source* para desenvolvimento de sistemas multiagentes, possui licença LGPL (*Library General Public License*), é implementado totalmente em Java e segue as especificações propostas pela FIPA (*Foundation for Intelligent Physical Agents*)[9]. Por essas características, o JADE foi escolhido para o projeto, além de ser uma plataforma de fácil acesso e largamente utilizada no meio acadêmico.

[9]Entidade responsável pela padronização em sistemas baseados em agentes.

Arquitetura Um modelo abstrato para uma plataforma de agentes pode ser visto na Figura 15, onde alguns ítens importantes, definidos pela FIPA, estão identificados:

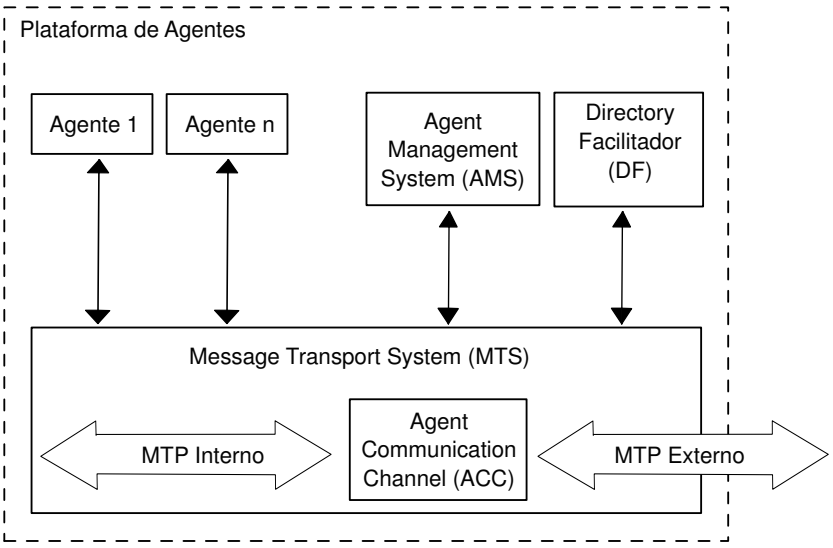

Figura 15. Modelo de referência da FIPA para plataformas de agentes.

- *Agent Management System (AMS)*: responsável por controlar todo o acesso e uso de uma instância da plataforma. Conhecido como "serviço de páginas brancas", existe apenas um AMS para cada instância da plataforma. Além disso, é esse agente que gerencia o ciclo de vida de todos os agentes existentes. No Serviço de páginas brancas, que estão relacionado ao serviço de endereçamento, cada agente recebe uma identificação (*AID*) única no ambiente.
- *Directory Facilitator (DF)*: esse agente fornece um serviço de "páginas amarelas", que é a catalogação dos serviços oferecidos pelos agentes na plataforma.
- *Message Transport System (MTS)*: componente para o transporte das mensagens entre os agentes, onde se pode encontrar:
  - *Agent Communication Channel (ACC)*: provê a comunicação entre os agentes, e é responsável pelas devidas traduções em trocas de mensagens entre agentes distribuídos e localizados, em ambientes heterogêneos.
  - *Message Transport Protocol (MTP)*: que utiliza-se do RMI[10] para troca de mensagens entre as máquinas virtuais Java (com IIOP[11] ou HTTP[12] quando entre plataformas diferentes)

O JADE utiliza-se dessa mesma arquitetura especificada pela FIPA, e um exemplo de um sistema multiagente em JADE pode ser visto na Figura 16. Cada *container* é uma máquina virtual Java diferente, e agrupa um conjunto de agentes. As máquinas virtuais podem estar em hospedeiros (*hosts*) diferentes e heterogêneos.

[10] *Remote Method Invocation* (Java RMI)
[11] *Internet Inter ORB Protocol*
[12] *Hypertext Transfer Protocol*

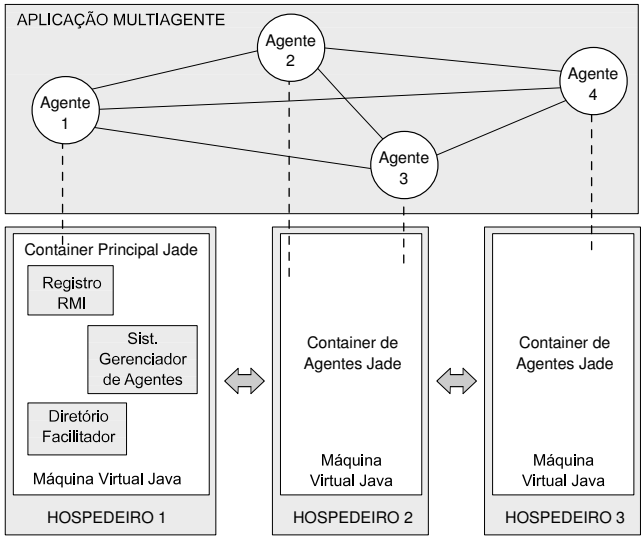

Figura 16. Arquitetura JADE.

Agentes JADE A Plataforma JADE oferece um arcabouço para construção de agentes, onde a tarefa de trocar mensagens, registro nas páginas amarelas, endereçamento dos agentes estão disponíveis para os usuários do arcabouço. Para que um agente JADE possa ser construído, basta que o usuário do arcabouço implemente a classe abstrata *jade.core.Agent*, que possui um único método abstrato chamado *setup()*, conforme pode mostrar a Figura 17.

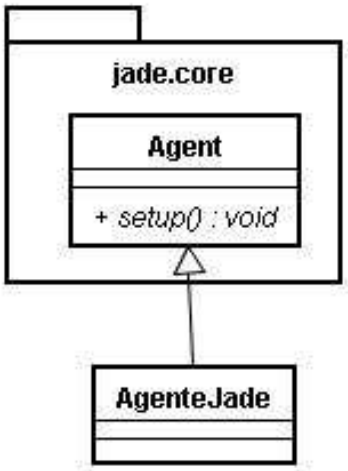

Figura 17. Estendendo um agente JADE.

Por fim, é importante frisar que um agente JADE deve possuir conhecimento sobre três aspectos:

1. Comportamentos: cada agente pode executar diversos tipos de atividades diferentes através dos comportamentos;
2. Serviços: cada agente deve ter a capacidade de i) cadastrar os seus serviços e/ou ii) buscar serviços de outros agentes. O objetivo deste é devido a necessidade de comunicação entre os agentes;

3. Mensagens: para que um agente possa se comunicar com outro, ele deve ter a capacidade de receber e/ou enviar mensagens para outros agentes.

6. Construindo Sistemas Inteligentes com Web Semântica

Esta seção objetiva especificar o domínio de projeto de pesquisa. O esforço despendido por pesquisadores e instituições financiadoras, tanto na construção de propostas de projeto quanto nas avaliações das mesmas é, normalmente, uma tarefa árdua. O processo é árduo tanto para os pesquisadores que definem o projeto de pesquisa quanto para as instituições financiadoras.

No lado dos pesquisadores, além da definição de cronograma de execução, desembolso, recursos humanos, avaliação, impacto do projeto, entre outros, o levantamento bibiográficos e de parcerias para o projeto de pesquisa é uma tarefa que demanda bastante tempo.

Já no lado das Instituições financiadoras, há a preocupação com a avaliação dos projetos, verificando as competências do coordenador/pesquisadores envolvidos além da qualidade da proposta.

Com isso, esta seção especifica uma ontologia para projetos de pesquisa que objetiva ser utilizada por pesquisadores e instituições financiadoras. Com isso, os pesquisadores irão construir o projeto de pesquisa e terão um suporte de agentes e serviços para tal atividade. Já as instituições financiadoras serão auxiliadas por agentes de software e serviços para avaliar os projetos de pesquisa.

6.1. Domínio

Diversas caracaterísticas estão relacionadas na construção de um projeto de pesquisa, onde algumas delas são citadas abaixo:

- Indexação: por se tratar de uma base de conhecimento fundamentada em texto, um *Projeto de Pesquisa* é composto por elementos indexáveis e não indexáveis. Os documentos são indexados em *palavras* e estas se inter-relacionam através de *antônimos* e *sinônimos*;
- Objetivo: é definido por *Objetivo Geral* e/ou *Objetivo Específico*;
- Limitação do Projeto: é composta por uma ou mais *Limitações* e o *Resultado Esperado* por um ou mais *Resultados*
- Cronograma: é composto por um ou mais cronogramas de *Execução* e *Desembolso*. O cronograma de *Execução* está relacionado com o *Método*, isto é, para cada *Método* é estabelecido um cronograma de *Execução* sendo este composto por uma ou mais *Atividades* e estas estão associadas a um ou mais *Períodos*;
- Recursos: é composto por recursos humanos e materiais.

6.2. Modelagem e Implementação

Na Figura 18 é apresentada a arquitetura do sistema para projetos de pesquisa. Os componentes providos na arquitetura são:

- Ontologia do projeto de pesquisa armazenado em um repositório, onde tanto os pesquisadores quanto as instituições usam. Tal ontologia foi desenvolvida

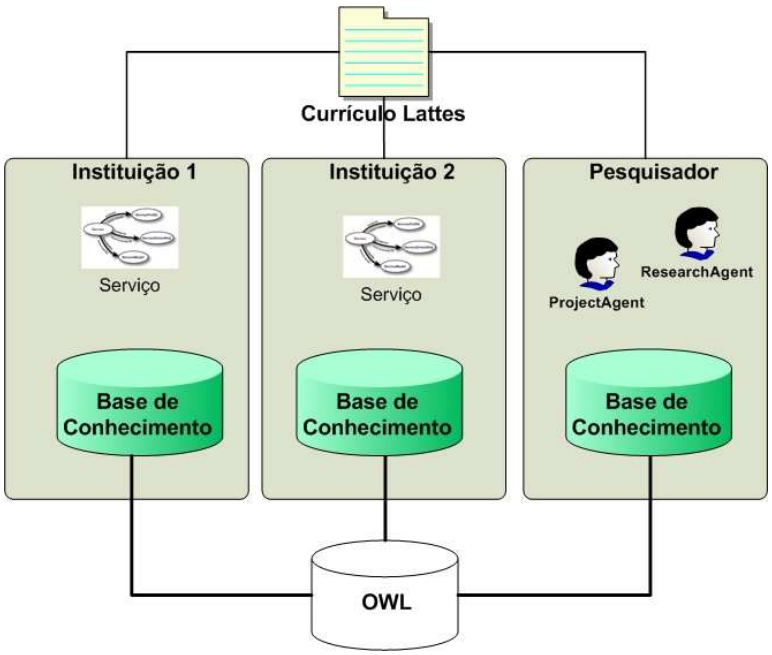

Figura 18. Arquitetura do Sistema.

utilizando OWL e regras de inferência com SWRL, como mostram as Figuras 19 e 20;

- Currículo Lattes: o Lattes é utilizado objetivando avaliar o currículo dos pesquisadores envolvidos em determinado projeto de pesquisa;
- Serviços são utilizados para fazer consultas e análises no currículo lattes e na ontologia de projeto de pesquisa. Algumas característica do OWL-S são descritos abaixo:
  - Serviço *retrieveLattesResumeService*: O objetivo deste serviço é a criação e o retorno é uma instância da classe Researcher. Será efetuada uma busca na base de currículos Lattes;
  - Entrada: CPF, objeto do tipo string representando o CPF do pesquisador Saída: lattesResume, instância da classe OWL Researcher, representando as informações de um pesquisador;
  - Serviço *retrieveResearchProjectService*: Este serviço tem como objetivo efetuar uma recuperação dos projetos de pesquisas, dado um pesquisador. Esse serviço será utilizado por fontes de fomentos para efetuar busca por projetos de pesquisas executados por pesquisadores, como forma de avaliar o desempenho do pesquisador
  - Entrada: instância da classe OWL Researcher, representando informações do pesquisador; saída: instâncias da classe OWL ResearchProject, representando as informações dos projetos no qual o pesquisador em questão está envolvido.
- Agentes: os agentes são utilizados para inferir e ajudar os pesquisadores na construção do projeto. Podemos verificar que dois agentes fazem parte do sistema.
  - ProjectAgent

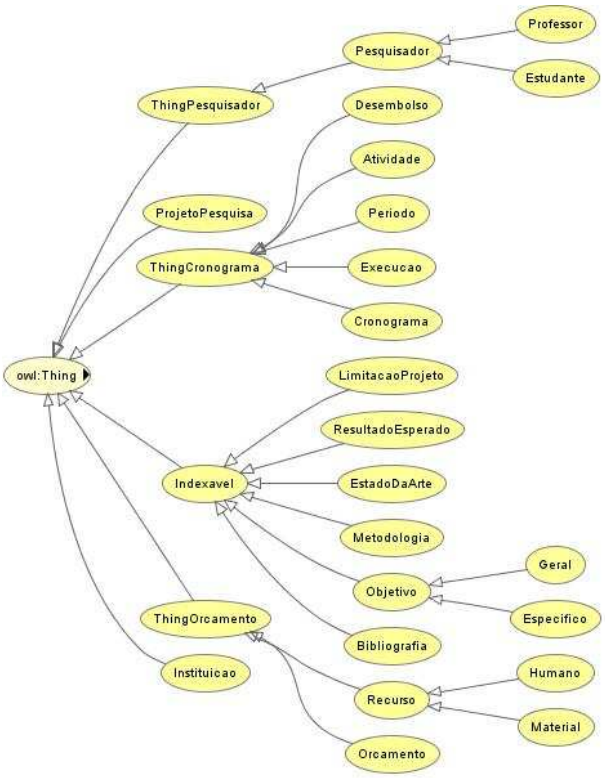

Figura 19. Ontologia do projeto de pesquisa.

Objetiva verificar, através do serviço SimilarProjects, similaridades de projetos, almejando auxiliar os pesquisadores durante a elaboração das suas propostas de pesquisa. Além disso, este agente utiliza conceitos inerentes ao Raciocínio Baseado em Casos.

– ResearchAgent
Almeja verificar, através do serviço SimilarResearch, o currículo dos pesquisadores, objetivando contribuir positivamente nas decisões tomadas pelas instituições financiadoras. Contando com a ajuda de algoritmos de recomendação colaborativa.

7. Conclusão

O texto apresentou o tema de Sistemas Inteligentes enfatizando a utilização de agentes e serviços e suas potencialidades sob perspectiva da Web Semântica. Com isso, foi mostrada uma revisão sobre sistemas inteligentes, focando na adoção de Web Semântica e na utilização de ferramentas para a construção de tais sistemas.

O objetivo deste material foi de provê um suporte para o tutorial do evento WebMedia 2007 com o título de Desenvolvimento de Sistemas Inteligentes sob a perspectiva da Web Semântica.

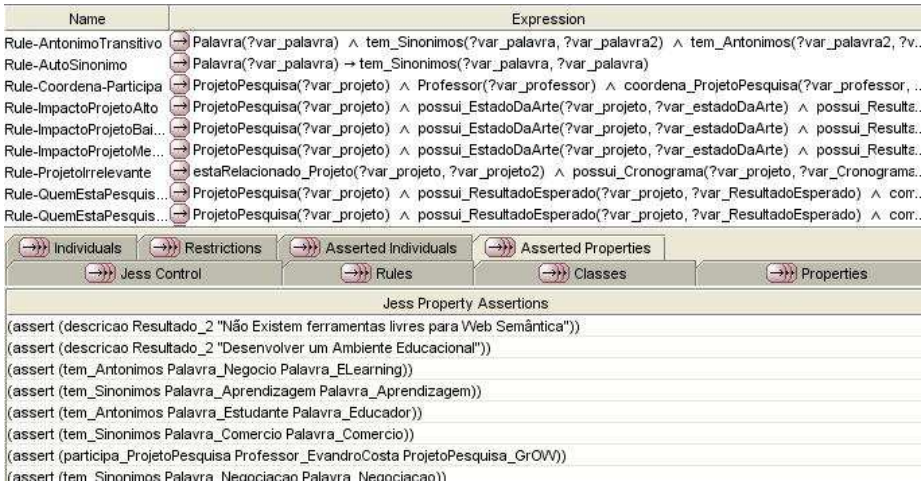

Figura 20. Regras construídas em SWRl.

Referências

Ankolekar, A., Burstein, M., Hobbs, J. R., Lassila, O., Martin, D. L., McIlraith, S. A., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., and Zeng, H. (2004). Daml-s: Semantic markup for web services.

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

Berners-Lee, T., Fielding, R. T., and Masinter, L. (2005). Unified resource identifiers. http://gbiv.com/protocols/uri/rfc/rfc3986.html. Acess: July, 2007.

Berners-Lee, T., Lassila, O., and Hendler, J. (2001). The semantic web. *Scientific American*, 254(5):34–43.

Bittencourt, I. I., Costa, E., Ferneda, E., and Alvarenga, R. (2006). Sistemas de informação e apoio a decisão baseado em conhecimento. In *Escola Regional de Informática de Minas Gerais*.

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

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

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web services description language (wsdl). http://www.w3.org/TR/wsdl. Ultimo acesso em Agosto de 2007.

David Nadler Prata, E. C. L. (2006). Projeto ontoinvpol: Uma ontologia de domínio para auxílio na descoberta de conhecimento em investigação policial. Projeto desenvolvido na disciplina de inteligência artificial, Universidade Federal de Campina Grande. Departamento de Sistemas e Computação. Coordenação de Pós-Graduação em Informática.

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

do Carmo Costa, M. M. (1992). *Introdução a Lógica Modal Aplicada à Computação*.

Durkin, J. (1994). *Expert System: design and development*. Prentice-Hall. ISBN:0023309709.

GmbH, R. S. and KG, C. (2005). *RacerPro User's Guide*. Racer Systems GmbH and Co. KG.

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

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). Swrl: A semantic web rule language combining owl and ruleml. Disponível em: http://www.w3.org/Submission/SWRL/. Acesso em: 13 Abr. 2007.

HP (2006). Jena - a semantic web framework for java. http://jena.sourceforge.net.

Júnior, C. L. N. and YONEYAMA, T. (2000). *Inteligência artificial em controle e automação*. Brochura, 1 edition. ISBN: 8521203101.

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

Kopeck, J., Roman, D., Moran, M., and Fensel, D. Semantic web services grounding. paper.

Kreger, H. (2001). Web service conceptual architecture (wsca 1.0). Disponivel em www-3.ibm.comsoftwaresolutionswebservicespdfWSCA.pdf.

lex Sakharov (2003). Horn clause in mathword. Disponível em: http://mathworld.wolfram.com/HornClause.html. Acesso em: 10 Mai. 2007.

Li, H., Zhang, X., Wu, H., and Qu, Y. (2005). Design and application of rule based access control policies. In *Semantic Web and Policy Workshop. In 4th International Semantic Web Conference:*.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2004). Owl-s: Semantic markup for web services. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/. W3C Member Submission.

Mastella, L. S. (2005). Um modelo de conhecimento baseado em eventos para aquisição e representação de seqüências temporais. Master's thesis, Instituto de Informática. Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande do Sul.

McIlraith, S. A., Son, T. C., and Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems*.

Mindswap. Owl-s api. http://www.mindswap.org/2004/owl-s/api/.

Minsky, M. (1975). *A Framework for Representing Knowledge*. Branchman and Levesques (1985).

OKBC (1995). Open knowledge base connectivity. Disponível em: http://www.ai.sri.com/ okbc. Acesso em: 27 Set. 2006.

OWL (2006). Owl web ontology language overview. Disponível em: http://www.w3.org/TR/owl-features/. Acesso em: 22 de Setembro de 2006.

Payne, T. and Lassila, O. (2004). Semantic web services. *IEEE Intelligent Systems*.

Protégé (2006). The protégé ontology editor and knowledge acquisition system. Disponível em: http://protege.stanford.edu/. Acesso em: 22 Set. de 2006.

RacerPro (2000). Racerpro. http://www.franz.com/products/racer.

RIESBECK, C. K. and SCHANK, R. C. (1989). *Inside case-based reasoning*. Hillsdale, New Jersey: LEA - Lawrence Erlbaum Associates. ISBN: 0898597676.

Russel, S. and Norvig, P. (2004). *Artificial Intelligence: A Modern Approach*. Elsevier.

Schank, R. C. (1982). *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, New York, EUA.

Shadbolt, N., Lee, T. B., and Hall, W. (2006). The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101.

Silva, R. P. (2005). Modelo de apoio ao diagnóstico no domínio médico aplicando raciocínio baseado em casos. Master's thesis, Universidade Católica de Brasília, Brasília (DF).

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

SMI (2006). Stanford medical informatics. Disponível em: http://www.smi.stanford.edu. Acesso em: 27 Set. 2006.

SPARQL (2007). Sparql query language for rdf. Technical report, W3C Working Draft.

Zhou, J., Koivisto, J.-P., and Niemela, E. (2006). A survey on semantic web services and a case study. *10th International Conference on Computer Supported Cooperative Work in Design*.