UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

UMA ARQUITETURA BASEADA EM WEB SERVICES SEMÂNTICOS PARA AGRUPAMENTO DOS AGENTES NEGOCIANTES NO AMBIENTE ICS DE COMÉRCIO ELETRÔNICO

RICARDO FERRAZ TOMAZ

São Luís
2003

UMA ARQUITETURA BASEADA EM WEB SERVICES SEMÂNTICOS PARA AGRUPAMENTO DOS AGENTES NEGOCIANTES NO AMBIENTE ICS DE COMÉRCIO ELETRÔNICO

Dissertação de Mestrado submetida à Coordenação do curso de Pós-Graduação em Engenharia de Eletricidade da UFMA como parte dos requisitos para obtenção do título de mestre em Ciência da Computação.

Por

RICARDO FERRAZ TOMAZ

Novembro, 2003

UMA ARQUITETURA BASEADA EM WEB SERVICES SEMÂNTICOS PARA AGRUPAMENTO DOS AGENTES NEGOCIANTES NO AMBIENTE ICS DE COMÉRCIO ELETRÔNICO

RICARDO FERRAZ TOMAZ

DISSERTAÇÃO APROVADA EM __ / __ / 2003

Prof. Dr. Sofiane Labidi
(Orientador)

Prof.ª. Dra. Ana Teresa de Castro Martins
(Membro da Banca Examinadora)

Prof. Dr. Edson Nascimento
(Membro da Banca Examinadora)

UMA ARQUITETURA BASEADA EM WEB SERVICES SEMÂNTICOS PARA AGRUPAMENTO DOS AGENTES NEGOCIANTES NO AMBIENTE ICS DE COMÉRCIO ELETRÔNICO

MESTRADO

ÁREA DE CONCENTRAÇÃO: CIÊNCIA DA COMPUTAÇÃO

RICARDO FERRAZ TOMAZ

Orientador: Dr. Sofiane Labidi

Curso de Pós-Graduação em

Engenharia de Eletricidade da

Universidade Federal do Maranhão

AGRADECIMENTOS

Aos meus pais, Nazita e Paulo, pelo apoio e orientação que me conduziram a esta conquista.

Ao professor Sofiane Labidi, pela orientação deste trabalho e pelos muitos ensinamentos que certamente me ajudarão por toda vida.

Aos amigos Bernardo e Roberto Baluz pela ajuda na revisão do texto e pela dedicação com que realizaram esta tarefa. Sem eles o caminho teria sido mais difícil.

Ao grande amigo tio e irmão Eduardo Thomaz, pela conversa inspiradora e boas idéias, que muito influenciaram neste trabalho.

Aos amigos Zeca, Carol, Georgeane e Mauro pela boa conversa e noites agradáveis e relaxantes.

Às minhas queridas irmãs Andréa e Bianca, pela confiança motivadora que depositam em mim.

Finalmente à minha esposa, Viviane, pela tolerância e compreensão das horas em que estive ausente do convívio familiar para dedicar-me a este trabalho.

Dedico este trabalho a

Meus filhos Ana Beatriz e Rômulo, que eles possam espalhar seus sorrisos cativantes por onde passem.

Meus pais, Nazita e Paulo, responsáveis por minha existência no sentido mais amplo da palavra.

“O único lugar onde o sucesso vem antes do trabalho é no dicionário.”

Albert Einsten

RESUMO

Este trabalho é parte do projeto ICS *(Intelligent Commerce System),* atualmente em desenvolvimento na Universidade Federal do Maranhão (UFMA) sob a orientação do Prof. Dr. Sofiane Labidi. O projeto tem como objetivo desenvolver um Sistema de Comércio Eletrônico, na categoria B2B, efetivamente Inteligente. O ICS é baseado na tecnologia de agentes móveis inteligentes e possui três características principais: abordagem em ciclo de vida do comércio eletrônico, modelagem do usuário e ontologias propostas para cada fase do ciclo de vida. Uma das principais fases do ciclo de vida do ICS é o *Matchmaking*. O *Matchmaking* é o processo no qual agentes, representando negociantes (compradores e vendedores) que possuem interesse na troca de valores econômicos, são colocados em contato com seus potenciais parceiros de negócios. Apresentamos aqui a arquitetura do ICS e focamos em um de seus principais componentes: o processo de *Matching* dos agentes negociantes, tarefa atribuída ao Agente *Matchmaker*. Propomos o uso de Web Services como blocos de montagem do Agente *Matchmaker*, aliada à visão da Web Semântica, para permitir o cruzamento semântico entre os requisitos de compra dos clientes e as propagandas dos fornecedores, viabilizando a descoberta automática dos serviços ofertados que realmente interessam aos clientes.

Palavras-Chave: E-Commerce, Agente de Software, Ontologias, WEB Semântica, Web Services.

ABSTRACT

This work is part of the ICS project which is being developed at Federal University of Maranhão under the Coordination of Prof. Phd. Sofiane Labidi. This project is aiming to develop an effective Intelligent B2B e-commerce System. The ICS is based on the mobile and intelligent software agent's technology and has three main characteristics: the electronic commerce lifecycle, the user modeling and the proposed ontologies on each phase of the lifecycle. One of the main phases of the ICS lifecycle is the *matchmaking*. The *matchmaking* is the process in which agents, representing traders that are interested in having exchange of economic value, are put in contact with potential counterparts to negotiate. In this work, we develope the proposed architecture of the ICS and highlight one of its main phases: the *matching* process, attributed to the *Matchmaker* Agent. We propose the use of the Web Services as assembly blocks for the *Matchmaker* agent, allied to the Semantic Web perspective, as an alternative to make possible the semantic crossing between the buyers' requests and the suppliers' advertisements, allowing the automatic discovery of the offered services that really interest the clients.

Keywords: E-Commerce, Software Agents, Ontologies, Semantic Web, Web Services.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

I. INTRODUÇÃO

CONTEXTO DA DISSERTAÇÃO

A busca das empresas pela redução de seus custos operacionais torna-se a cada dia mais indispensável para a sobrevivência em um mercado altamente competitivo, onde cada detalhe pode significar sucesso ou fracasso. As ferramentas de comércio eletrônico estão se firmando como uma das mais fortes aliadas das empresas na busca da otimização de seus processos de negócios.

A convergência do comércio com a Internet simplifica e incrementa a comunicação entre parceiros, reduz custos operacionais, melhora o atendimento ao consumidor e proporciona inovadoras práticas de negócio.

As tecnologias ligadas à Internet evoluem em grande velocidade. Atualmente, a integração de sistemas de plataformas heterogêneas através da infra-estrutura da Internet tem despertado bastante interesse no meio acadêmico e também na indústria de software. O objetivo é reduzir custos operacionais e agilizar os processos de negociação entre as empresas. A visão da Web Semântica tem proporcionado enormes possibilidades nesta direção e algumas pesquisas têm se utilizado dos conceitos, padrões e ferramentas decorrentes do desenvolvimento da Web Semântica para propor soluções inovadoras aplicadas ao comércio eletrônico.

Como as pesquisas sobre a Web Semântica estão ainda em fase embrionária e as arquiteturas e padrões de integração ainda estão sendo investigados, o desenvolvimento de ferramentas que fazem uso da visão da Web Semântica pode conduzir-nos a questões relevantes de ordem prática que contribuam de algum modo para seu desenvolvimento.

Neste sentido, propõe-se o desenvolvimento de uma solução B2B de baixo custo e efetivamente inteligente. Foca-se na prototipação de um dos seus principais componentes, o Agente *Matchmaker*, responsável pela descoberta de oportunidades de negócios entre os agentes negociantes dentro da arquitetura proposta. A perspectiva da Web Semântica é utilizada para resolver o problema de descoberta dos possíveis agentes parceiros de negócios, proporcionando a automação deste processo.

OBJETIVOS DA DISSERTAÇÃO

O objetivo deste trabalho é propor uma arquitetura inovadora e de baixo custo para aplicações B2B e a resolução do problema de descoberta e agrupamento de agentes negociantes dentro da arquitetura proposta.

Especificamente, nós pretendemos:

I. Permitir uma melhor compreensão dos conceitos, padrões e ferramentas decorrentes do desenvolvimento da Web Semântica;

II. Propor uma arquitetura inovadora e de baixo custo para aplicações B2B;

III. Resolver o problema de descoberta e agrupamento de agentes negociantes dentro da arquitetura proposta;

IV. Experimentar e verificar a eficácia dos conceitos, padrões e ferramentas apresentadas quando aplicados a um caso de dimensões e complexidade reais: a prototipação do Agente *Matchmaker*.

JUSTIFICATIVA E RELEVÂNCIA

O Business-to-Bussiness (B2B) (Albertin, 2001), modalidade de comércio eletrônico entre empresas, tem contribuído de maneira significativa na redução dos custos operacionais das empresas. Entretanto, somente empresas de médio e grande porte têm se beneficiado diretamente de sua utilização em função dos altos custos de implantação dos chamados portais B2B.

Como uma alternativa para inclusão de um maior número de empresas no comércio eletrônico, propomos o ICS *(Intelligent Commerce System)*, um sistema de comércio eletrônico de baixo custo, que gerencia desde a aquisição das preferências do usuário até a formação e execução dos contratos de negócios realizados entre compradores e vendedores.

Para garantir maior robustez e agilidade na negociação entre os agentes utilizamos a tecnologia de agentes móveis inteligentes e, para permitir que os agentes negociantes com interesses comuns se descubram em um ambiente altamente distribuído e heterogêneo como a Internet, empregamos padrões e ferramentas decorrentes do desenvolvimento da Web Semântica.

Sendo assim, nosso trabalho se justifica por reduzir custos de operacionalização das soluções B2B, beneficiando um maior número de empresas através da abordagem inovadora de sua arquitetura.

Entendemos também que a utilização da tecnologia de agentes móveis inteligentes e da visão da Web Semântica como ferramentas de suporte à implementação do modelo proposto, nos conduz a uma experimentação das teorias,

padrões e ferramentas ainda em desenvolvimento em duas importantes áreas de pesquisa na Ciência da Computação, que são os Sistemas Multiagentes (SMA) e a Web Semântica, ambas sub-áreas da Inteligência Artificial (IA).

Deste modo, este trabalho presta sua contribuição em quatro momentos: ao reunir de forma estruturada o referencial teórico necessário ao entendimento da arquitetura SMA e da Web Semântica; ao propor uma arquitetura de baixo custo para aplicações B2B; ao propor a abordagem da Web Semântica e dos Web Services para proporcionar descoberta automática de possíveis parceiros de negócios distribuídos pela Web e na experimentação das teorias, técnicas e ferramentas apresentadas, na prototipação do Agente *Matchmaker*.

ORGANIZAÇÃO DA DISSERTAÇÃO

Esta dissertação está organizada em cinco partes, divididas em seis capítulos. A Parte I é constituída por esta introdução, onde contextualizamos nosso trabalho e explicitamos seus objetivos, justificativa e relevância.

Na Parte II, esperamos alcançar o primeiro objetivo de nosso trabalho: fazer a revisão bibliográfica e apresentar o referencial teórico necessário para o entendimento da solução proposta. Esta parte é dividida em três capítulos: O Capítulo 1 apresenta uma visão da Web Semântica - definição, arquitetura e aplicações; Capítulo 2 apresenta uma visão detalhada dos *Metadados Semânticos e Ontologias,* onde discorremos sobre as principais técnicas de representação de conhecimento e a relevância destas técnicas para a Web Semântica e, conseqüentemente, para a compreensão do desenvolvimento do Agente *Matchmaker* e, no Capítulo 3 compilamos os conceitos necessários ao entendimento da *Abordagem Multiagente* e enfatizamos sua adequação à resolução do problema de descoberta automática de mapeamento entre os agentes negociantes no ICS.

Na Parte III, buscamos o segundo objetivo de nosso trabalho: apresentar uma visão de alto nível do ICS *(Intelligent Commerce System)*, abordando cada uma de suas principais características: ciclo de vida do comércio eletrônico, modelagem do usuário e ontologias propostas para cada fase.

Na Parte IV, esperamos alcançar o terceiro e o quarto objetivos de nosso trabalho: Modelar o agente *Matchmaker* e apresentar seus requisitos funcionais e

não funcionais, seu algoritmo genérico, e as ferramentas utilizadas em sua prototipação.

Por fim, Na parte V, apresentamos nossas conclusões sobre este trabalho de pesquisa e as sugestões para trabalhos futuros.

II. REFERENCIAL TEÓRICO

1. WEB SEMÂNTICA E WEB SERVICES SEMÂNTICOS

Apresentamos, neste capítulo, os principais conceitos relacionados à Web Semântica e à tecnologia de Web Services, bem como a contribuição e adequação do uso de cada uma destas tecnologias para o nosso trabalho.

1.1 Introdução

A World Wide Web está crescendo em grande velocidade. O aumento exponencial do número de documentos disponíveis na Web tem gerado grandes dificuldades para os usuários em encontrar a informação desejada.

A Web Semântica, uma evolução da Web atual, proposta por Tim Berners-Lee em sua visão sobre o futuro da Web (Berners-Lee, 1998), proporciona um certo grau de estrutura para os dados disponibilizados na Web, adicionando metadados e ontologias, o que possibilita a recuperação automática de informações por agentes inteligentes de software.

A Web Semântica visa atribuir estrutura aos dados armazenados na Web, juntamente com relações semânticas entre eles que permitam recuperá-los com maior eficiência. A idéia principal da Web Semântica é a de uma Web no qual os recursos disponíveis (dados e informações) são acessíveis não somente por seres humanos, mas também por processos automatizados que podem ser agentes de software. A automação da Web passa pela elevação de seu status de lida automaticamente "*machine-readable"* para entendida automaticamente "*machine understandable"* (Horrocks, 2002).

Apesar de relativamente nova, a idéia da Web Semântica está embasada em pesquisas muito consolidadas na IA que são a Engenharia do Conhecimento (Stanley, 1999) e a tecnologia de Agentes Inteligentes (Jennings et al., 2000). A idéia é modelar e representar o conhecimento disponível na Web e aplicar mecanismos de inferência (raciocínio) fazendo uso de agentes inteligentes para recuperá-lo eficientemente.

1.2 Arquitetura da Web Semântica

A Web Semântica foi idealizada em camadas (Berners–Lee et al., 2000). O W3C (URL 1), entidade certificadora dos padrões e ferramentas desenvolvidas para construção da Web Semântica, propôs uma visão inicial em três camadas como mostra a figura 1.

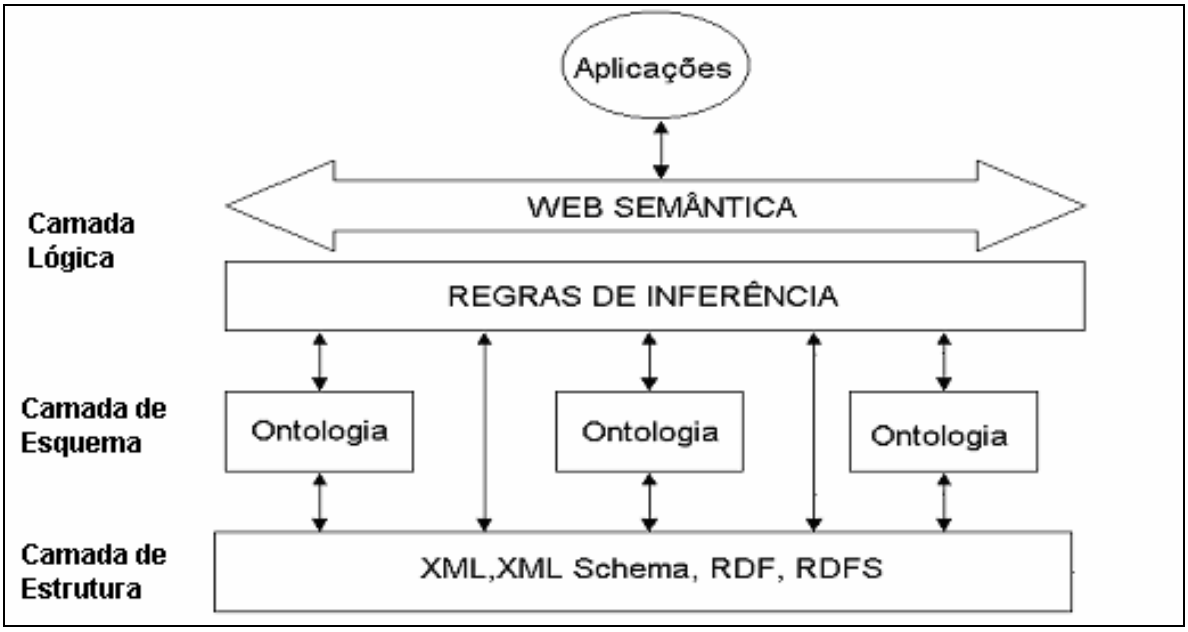

Figura 1: Arquitetura da Web Semântica (Berners – Lee et al., 2000).

- ***Camada de Estrutura*:** a representação do conhecimento é o alicerce da Web Semântica. Nesta camada são definidas as estruturas de dados. Várias linguagens de marcação de conteúdo derivadas de XML(e**X**tensible **M**arkup

Language) (Pitts et al., 2000) tem sido criadas para este fim. RDF e RDF Schema (Abitebul et al., 2000) são atualmente as mais relevantes.

- ***Camada de Esquema:*** também chamada de camada ontológica. O uso de ontologias permite a resolução de problemas de identificação ou conflito de terminologia. O desenvolvedor de uma página XML pode ligá-la a uma ontologia que defina o significado de cada uma das tags utilizadas. A partir deste momento, sua integração com outras páginas XML que possuam também sua própria ontologia definida passa a ser possível através da aplicação de regras de inferência definidas na camada lógica. Algumas linguagens que suportam a construção de ontologias são: OIL(Horrocks et al., 2000), DAML-OIL (Horroks, 2001) e OWL (Bechhofer et al., 2003).

- ***Camada Lógica:*** onde são definidos os mecanismos de inferência sobre os dados. As regras de inferência fornecem aos agentes a possibilidade de *"raciocinar"* sobre os termos e seus significados definidos nas camadas inferiores. Nesta camada, são empregadas linguagens de representação de conhecimento formal, com primitivas comumente empregadas em linguagens de Descrição Lógica *(Description Logics)* (Nardi et al., 2002), de modo a proporcionar suporte ao *"raciocínio"* automático. A linguagem RuleML *(Rule Markup Language)* (URL 5), em desenvolvimento pelo grupo de pesquisa *Rule Markup Initiative,* tem como objetivo expressar regras de inferência baseadas em linguagens como Prolog e Lisp em marcações XML.

A linguagem XML é um dos pilares da Web Semântica. O desenvolvimento desta evolução da Web está sendo baseado sobre o padrão XML.

1.3 Aplicações da Web Semântica

A Web Semântica proporciona uma evolução significativa em muitas áreas, dentre as quais destacam-se: Gestão do Conhecimento (URL 6), Assistentes Pessoais (URL 7), Recuperação de Informação (Stanley, 1999) e o Comércio Eletrônico.

A evolução das ferramentas de Comércio Eletrônico, mais especificamente do Business to Business (B2B), é um dos propósitos de nosso trabalho. Tradicionalmente, o B2B vem sendo realizado através de acordos bilaterais entre as empresas, por exemplo, o padrão EDI[1] *(Eletronic Data Interchange)*, requer que cada parceiro comercial negocie individualmente com o outro em um modelo *"um-para-um"*. A introdução de tecnologias decorrentes do desenvolvimento da Web Semântica torna possível a evolução para um modelo de negociação *"muitos-para-muitos"*, onde cada cliente tem a possibilidade de localizar e negociar com vários fornecedores e cada fornecedor tem a possibilidade de oferecer seus produtos ou serviços para vários clientes. É esta a finalidade do sistema ICS que estamos propondo para suportar o comércio eletrônico na categoria B2B. O ICS promove um modelo de negociação *"muitos-para-muitos"* entre clientes e fornecedores com baixo custo e boa performance.

As arquiteturas baseadas em serviços representam um novo paradigma para modelagem de sistemas. Na próxima seção apresentamos uma visão dos Web Services, aplicações fracamente acopladas que se comunicam através da WEB e que estão tendo um papel crescente nas interações entre as empresas através da

[1] Troca Eletrônica de Dados

Internet, principalmente nas cadeias de fornecimento entre empresas, constituindo redes de negócios do tipo Business-to-Business (B2B).

1.4 Web Services Semânticos

A World Wide Web deixou de ser um repositório de textos e imagens para se transformar em um provedor de serviços. Na última década, muitos sites disponibilizaram para os seus usuários serviços de compras, reservas, consultas, notícias, dentre outros. Entretanto, essas aplicações são, em sua maioria, baseadas em bancos de dados centralizados, pertencentes a um prestador de serviço específico, e, por isso, fornecem informações sobre uma única organização.

A definição de padrões de integração de sistemas de múltiplas organizações que atuam em domínios de negócios comuns, com o objetivo de compilar e reunir informações para seus clientes é ainda uma questão aberta. Muito esforço tem sido despendido para alcançar este objetivo, principalmente em função do crescimento exponencial das fontes de dados heterogêneas espalhadas pela Internet.

Os Web Services são um novo paradigma para implementação de arquiteturas orientadas a serviços na Internet, onde os sites trocam informações dinamicamente e sob demanda. Estas características são relevantes para a comunidade interessada na evolução do comércio eletrônico, pois possibilita uma negociação mais rápida e eficaz entre os parceiros de negócios.

Nas arquiteturas baseadas em Web Services, os componentes são vistos como prestadores de serviços que realizam uma tarefa específica. Uma aplicação é vista como uma composição de serviços que juntos resolvem um problema para um determinado usuário.

Uma aplicação baseada em Web Services possui três entidades: o Provedor de Serviço *(Service Provider)*, o Solicitador de Serviço *(Service Requester)* e o Corretor de Serviços *(Service Broker)*.

Um provedor de Serviço fornece a realização de algum serviço. Para tanto, deve anunciar suas funcionalidades. Um Solicitador de Serviços é uma entidade que irá utilizar-se de um serviço prestado por um Provedor de Serviço (um Web Service), logo, ele deve descobrir o Web Service que melhor atenda às suas necessidades. O Service Broker desempenha o papel de localizar o Provedor de Serviço que melhor se adeque às necessidades do Solicitador de Serviços.

Padrões de descoberta como UDDI *(Universal Description, Discovery and Integration)* e linguagens de descrição de serviços como WSDL *(Web Service Description Language)* são empregadas com este fim (Daum, 2002). Entretanto, eles não fornecem suporte a uma busca semântica.

A utilização da visão da Web Semântica para o desenvolvimento de um Service Broker eficiente tem sido investigada por alguns grupos de pesquisa, como W3C Web Services Activity e CMU[2] Intelligent Software Agents Lab. Neste trabalho, reunimos e estendemos as abordagens investigadas por estes grupos visando atingir melhores resultados no processo de descoberta automática de Web Services.

1.5 Adequação da Web Semântica e dos Web Services para o desenvolvimento de Ferramentas B2B

As arquiteturas baseadas em serviço proporcionam um modelo de negócio bastante flexível, onde as empresas oferecem e compram bens ou serviços de forma

[2] Carnegie Mellon University.

automática através da utilização da infra-estrutura da Internet. Segundo (Burbeck, 2000), existem três cenários para colaboração de serviços B2B:

- **Acoplamento rígido, estabelecido em tempo de compilação:** A aplicação precisa conhecer os detalhes do serviço com o qual vai colaborar porque o acoplamento é feito durante a concepção da aplicação. Deste modo, a aplicação sabe exatamente como interagir com o serviço.

- **Acoplamento dinâmico a colaborador pré-estabelecido:** A aplicação faz uso de um agenciador *(Broker)* para localizar um serviço específico. Neste cenário a aplicação sabe exatamente como fazer a solicitação para o agenciador, isto porque o programador codificou uma consulta específica a ser feita ao agenciador.

- **Acoplamento dinâmico e escolha dinâmica do colaborador:** A aplicação sabe a semântica e as chamadas da API do serviço a ser usado. Entretanto, consulta um agenciador com um padrão de busca que permite o retorno de uma série de alternativas. A aplicação escolhe um serviço da lista que melhor lhe atenda em tempo de execução.

A Web Semântica representa um grande passo para a elevação das aplicações de comércio eletrônico em direção ao terceiro cenário descrito por Burbeck (Burbeck, 2000). A descoberta de um serviço por uma aplicação é fortemente dependente de sua descrição. Linguagens de descrição semanticamente expressivas e formalmente definidas que possibilitam o processamento automático, por software, das anotações sobre os serviços tem um papel central no sucesso das

aplicações baseadas em Web Services. Burbeck (Burbeck 2000) ainda aponta os requisitos de alto nível destas descrições:

- A descrição dos serviços deve ser feita em XML. Elas precisam ser ao mesmo tempo legíveis por humanos e processadas por programas. Precisam ser extensíveis, uma vez que a evolução dos serviços Web não pode ser prevista em todos os seus detalhes;

- As descrições de serviços devem prover toda a informação semântica necessária aos solicitadores para que estes verifiquem se os requisitos serão satisfeitos e aos agenciadores para estes decidirem sobre a categorização precisa do serviço em sua taxonomia;

- As descrições devem ainda possibilitar a especificação de requisitos não-funcionais, como segurança, autenticação e privacidade, importantes para garantir a troca de informações necessárias para o consumo do serviço e questões contratuais.

O desenvolvimento da Web Semântica fez surgir algumas linguagens e padrões para anotação de Web Services como LARKS (*Language for Advertisement and Request for Knowledge)* (Sycara, 2002) *e DAML-S (DARPA*[3] *Agent Markup Language for Services).*

As aplicações de comércio eletrônico atuais enquadram-se nos dois primeiros cenários descritos por Burbeck: acoplamento rígido e acoplamento dinâmico a colaborador pré-estabelecido. O ICS enquadra-se no terceiro cenário: acoplamento

[3] Defense Advanced Research Project Agency. Agência de Projetos de Pesquisa Avançada de Defesa dos EUA.

dinâmico e escolha de colaborador dinâmica. Neste trabalho além de apresentar a arquitetura do ICS focamos no Agente Matchamaker que, no contexto do ICS, desempenha o papel do agenciador descrito por Burbeck.

1.6 Conclusão

Os paradigmas apresentados neste capítulo são fundamentais para o desenvolvimento de nosso trabalho. O ICS é baseado em uma arquitetura SMA *(Sistema Multiagente)*. Resolvemos o problema da descoberta entre os agentes no ICS com o uso dos Web Services como blocos de montagem de nossa arquitetura e aliamos a visão da Web Semântica como forma de proporcionar um mecanismo de descoberta automática dos Provedores de Serviços *(Services Providers)* pelos Solicitadores de Serviços *(Services Requesters).*

Nos capítulos 2 e 3 apresentamos, respectivamente, uma visão mais detalhada dos *Metadados Semânticos e Ontologias* e da *Abordagem Multiagente* que são os alicerces da Web Semântica e, portanto, de grande relevância para compreensão da arquitetura do ICS e particularmente do Agente *Matchmaker*.

2. METADADOS SEMÂNTICOS E ONTOLOGIAS

No Capítulo 2, apresentamos uma introdução à Web Semântica e aos Web Services. Neste capítulo, abordamos um dos pilares da Web Semântica: os Metadados, que tornam possível atribuir significado aos conteúdos e serviços disponíveis na Web.

2.1 Introdução

A idéia principal da Web Semântica é a de uma Web na qual os recursos disponíveis (Dados e Informações) são acessíveis não somente por seres humanos, mas também por processos automatizados que podem ser agentes de software.

Como dito anteriomente, a automação da Web passa pela elevação de seu status de lida automaticamente *"machine-readable"* para entendida automaticamente *"machine understandable"*. Para atingir este objetivo, é necessária a anotação dos recursos disponíveis na Web através de metadados.

A definição mais simples de metadados é a de *"dados sobre os dados"*, ou seja, metadado é uma estrutura descritiva da informação sobre outro dado.

Os metadados podem ser estruturais ou semânticos. O metadado estrutural descreve a organização e a estrutura dos dados armazenados (formato, tipo de dado e relacionamentos sintáticos). O metadado semântico descreve o significado e os relacionamentos semânticos dos dados armazenados. Por exemplo, dados que descrevem o conteúdo semântico de um valor de um dado: unidade de medida, escala, qualidade, precisão, atualidade, etc.

A modelagem semântica tem tradição na engenharia do conhecimento (Studer, 1998) e na tecnologia de agentes, mas também é aplicada a muitos outros campos na tecnologia da informação, como em projeto de banco de dados, análise orientada a objeto, recuperação de informações (Sparck, 1997), etc.

Em IA, o termo ontologia é usado para indicar um domínio de conhecimento ou o domínio semântico para um agente. Nas seções seguintes discorremos sobre as técnicas de representação de conhecimento com o objetivo de proporcionar uma visão mais ampla do entendimento das ontologias, componente fundamental no contexto da Web Semântica.

2.2 Representação do Conhecimento

Quando as linguagens de alto nível foram desenvolvidas, duas abordagens antagônicas de projetos de sistemas emergiram: abordagem procedural e abordagem declarativa.

O paradigma procedural é mais adequado ao uso da arquitetura de Von Neumann [4] como modelo para representação da solução de um problema a ser resolvido pela máquina. Neste paradigma, programar o computador significa *"dar ordens"* que são executadas seqüencialmente (metáfora do “como fazer”) (Freitas, 2002). Representar a solução de um problema para ser resolvido pelo computador no paradigma procedural consiste em escrever uma série de ações (procedimentos) que, se executadas seqüencialmente, conduzem à solução, ou seja, o programa representa a descrição da solução para o problema.

[4] Arquitetura proposta por John Von Neumann onde a execução de programas é baseada nos fluxos de instruções. Programa e dados encontram-se armazenados em uma memória. As instruções são buscadas na memória e então executadas. Instruções especiais, de desvios condicionais e incondicionais, são utilizadas para mudar o fluxo de controle.

Já na abordagem declarativa, o significado de um programa não é mais dado por uma sucessão de operações elementares que o computador realiza, mas por uma base de conhecimento a respeito de certo domínio e por perguntas feitas a essa base de conhecimento (metáfora do “o quê fazer”) (Freitas, 2002). Na abordagem declarativa há um formalismo para representar o conhecimento a respeito do problema que se quer resolver, de forma declarativa (descritiva). Por trás da base de conhecimento há uma máquina de inferência, em princípio *"escondida"* do programador, responsável por *"encontrar soluções"* para o problema descrito.

A abordagem procedural, por ser mais próxima do modelo de Von Neumann, é mais eficiente em termos de performance e, atualmente, é o paradigma de programação mais utilizado.

O uso de agentes de software como facilitadores e otimizadores da utilização dos recursos disponíveis na Internet ganhou bastante relevância com a revolução provocada pela Web que tornou informações armazenadas em ambientes desestruturados e heterogêneos disponíveis para milhares de pessoas.

Para que os agentes possam processar automaticamente as informações disponíveis na Web é necessário que essas informações estejam codificadas em um formalismo bastante expressivo - para representar toda sua complexidade e, flexível - para permitir sua reusabilidade.

A solução de problemas ligados à Internet baseada em agentes fez ressurgir com vigor a utilização de soluções declarativas baseadas em lógica.

2.3 Formalismos para Representação de Conhecimento

Todo programa de computador contém o conhecimento sobre um determinado problema a ser resolvido. Os sistemas baseados na abordagem declarativa têm como principal característica o fato de o conhecimento está do lado de fora dos programas (procedimentos). Normalmente, armazenado em bases de dados denominadas bases de conhecimento *(Knowledge Base)*.

Existem vários formalismos para representação de conhecimento. A seguir, apresentamos sucintamente alguns dos principais formalismos utilizados na construção de bases de conhecimentos.

2.3.1 Regras de Produção

É uma forma de representação do conhecimento utilizada em Sistemas Baseados em Conhecimento como os Sistemas Especialistas. Regras de produção são regras no formato *"se – então"* que permitem o uso dos conectivos lógicos (E, OU, NÃO, e outros desejados), além do tratamento de incertezas. Uma regra de produção tem um conjunto de premissas e uma conclusão. A figura 2 mostra o exemplo de uma regra de produção.

```
SE aluno tem mais que 75% de faltas
OU aluno tem média abaixo de 7
ENTÃO Situação = reprovado CNF 100%
```

Figura 2: Regra de Produção.

A maior parte dos sistemas especialista emprega este formalismo devido à sua simplicidade e facilidade de utilização.

A utilização de bases de conhecimentos baseadas em regras de produção, compiladas em tempo de execução, torna os sistemas mais flexíveis, uma vez que, o

conhecimento é mantido fora da aplicação, podendo ser ampliado sem nenhuma implicação ao código dos programas. Os motores de inferência com integração a código imperativo permitem uma separção muito clara e eficiente entre os dados e processos.

2.3.2 Redes Semânticas

A Semântica é o estudo do significado de conceitos individuais utilizados na linguagem. Uma *rede* é um conjunto ou um grafo de nodos conectados por ligações. Uma rede semântica é uma estrutura para a representação do conhecimento definida como um padrão de nodos interconectados por arcos rotulados.

A principal idéia por trás das redes semânticas é que o significado de um conceito vem do modo como ele é conectado a outros conceitos (Rich, 1993). A figura 5.2 mostra o exemplo de uma rede semântica.

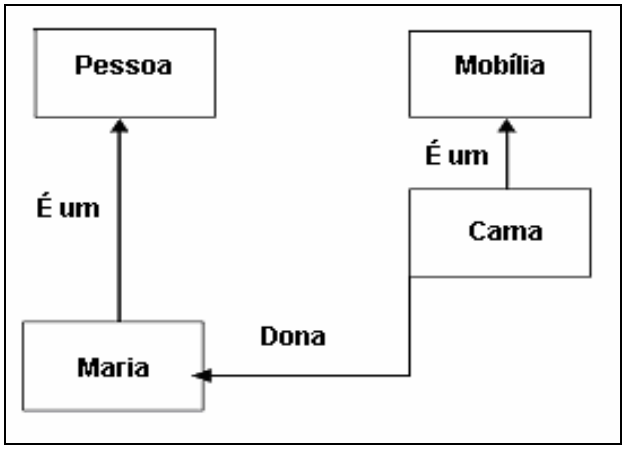

Figura 3: Exemplo de uma Rede Semântica.

Embora seja útil imaginar as redes semânticas utilizando esta notação gráfica característica, obviamente, é necessário que se faça a tradução para um formalismo que possa ser entendido e processado pelo computador. A Figura 3 pode ser expressa da seguinte forma: É-UM(Mobília,Cama); É-UM(Pessoa,Maria);

DONA(Maria,Cama). Esta seqüência de comandos pode ser codificada facilmente em uma linguagem como Prolog ou LISP, onde cada nodo é um átomo, as ligações são as propriedades, e os nodos da outra extremidade são os valores.

2.3.3 Frame

O sistema de representação baseado em Frame, criado por Minsky (Minsky, 1975), surgiu como uma evolução das redes semânticas. Os Frames foram inspirados na forma como os humanos resolvem os problemas. Os sistemas de Frames raciocinam sobre classes de objetos. A idéia é ter uma estrutura de dados para armazenar todo o conhecimento sobre uma classe de objetos, em vez de ter o conhecimento distribuído em forma de regras e fórmulas lógicas.

O Frame permite representar o conhecimento declarativo e procedural em um registro baseado em slots (atributos do objeto) e Facetas (restrições sobre os valores associados ao slot). Os Frames são dispostos em uma hierarquia, onde os Frames de baixo podem herdar os valores dos slots dos Frames de cima. Além disso, um slot pode armazenar valores atômicos ou multivalorados. A Figura 4 mostra um exemplo de representação de conhecimento baseado em Frames.

```
(frame duto-sanguineo  (forma tubular)
                       (contem sangue))
(frame arteria                (é-um duto-sanguineo)
                       (localizacao {braco, cabeca, perna, tronco})
                       (sangue rico-em-oxigenio)
                       (parede muscular))
(frame veia            (é-um duto-sanguineo)
                       (parede fibrosa))
(frame aorta           (é-um arteria)
                       (localizacao tronco) )
```

Figura 4: Representação de Conhecimento utilizando Frames (Morales, 1999).

2.3.4 Lógica de Descrição

As lógicas de descrições *(Description Logics)* (URL 9) originaram-se a partir das R*edes Semânticas* e dos *Frames.* Elas tratam de representações estruturadas de conceitos e raciocinam sobre elas. A estrutura de um conceito é descrita usando uma linguagem, chamada linguagem de conceitos, que inclui os operadores booleanos (conjunções, disjunções e negação) e várias formas de quantificação sobre os atributos (ou papéis) de um dado conceito.

A linguagem de conceitos, peça fundamental nos sistemas de representação de conhecimento baseados em lógica de descrição, é formada por conjuntos de construtores para denotar classes e relacionamentos entre classes. Os conceitos são utilizados para representar as classes e as regras são relações binárias usadas para especificar as propriedades ou atributos das classes.

Nos sistemas baseados em lógica de descrição, a base de conhecimento é composta de dois componentes: TBox *(terminological Box)* que define a estrutura dos conceitos e, ABox *(assertional Box)* uma instanciação da estrutura de conceitos definida na TBox.

A TBox é um esquema geral da base de conhecimento composta por classes, relacionamentos entre as classes e propriedades de cada classe. Uma entrada na TBox poderia ser: $Humano \subseteq Animal \cap Bípede$, denota que a classe Humano está contida (é subclasse de) na interseção entre as classes Animal e Bípede. Na realidade trata-se de um relacionamento do tipo "*é-um*", ou seja, um humano é um animal ***e*** é um bípede.

A ABox é composta por um conjunto de axiomas que descrevem situações concretas. Uma entrada na ABox poderia ser: Pedro:Humano, denota que Pedro é uma instância da classe Humano definida anteriormente na TBox, logo, Pedro é um animal e Pedro é bípede.

Uma base de conhecimento, em sistemas baseados em lógicas de descrição, é composta pelo par <TBox,ABox>. A arquitetura genérica de um sistema baseado em lógica de descrição é apresentada na figura 5.

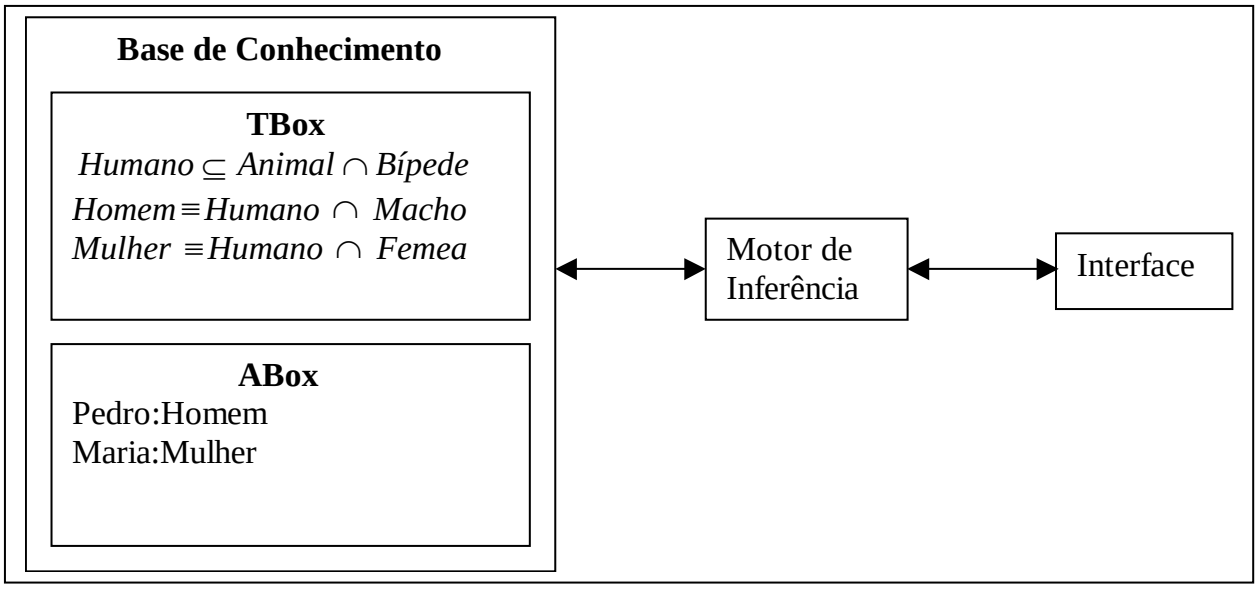

Figura 5: Arquitetura de sistemas baseados em lógica de descrição.

O raciocínio sobre uma base de conhecimento codificada em lógica de descrição é um processo que envolve esquema (TBox) e instanciação (ABox). Dada uma base de conhecimento $\sum$ = (T,A), onde T é a TBox e A é a ABox expressada em uma linguagem de conceito $\mathcal{L}$ e dois conceitos C e D, podemos descrever os principais serviços de raciocínio provido por um motor de inferência baseado em lógica de descrição como segue:

- ***Satisfatibilidade de conceitos:*** Consiste em testar se o conceito C é satisfatível, ou seja, se existe uma interpretação para C diferente de vazio. Em outras palavras, se há um modelo para T e para A que satisfaça o conceito C.

- ***Classificação (subsumption):*** consiste em testar se um conceito C está incluído no conceito D ($C \subseteq D$), ou seja, se C é subClasse de D.

- ***Consistência:*** Objetiva verificar se a base $\sum$ é satisfatível, ou seja, se a base de conhecimento tem um modelo.

- ***Teste de instância:*** verifica se a fórmula C(a) (lê-se,"a pertence ao conceito C") é satisfeita pelo modelo de $\sum$.

2.4 Ontologias

Os formalismos de representação de conhecimento possibilitam a estruturação de conhecimentos em ontologias, com conceitos e termos pertencentes a determinado domínio.

O termo ontologia é usado para indicar um domínio de conhecimento ou um domínio semântico para um agente.

Ontologias são acordos a respeito de conceitualizações compartilhadas (Gruber, 1993). Usam-se ontologias comuns para descrever a representação de uma base de conhecimento para um conjunto de agentes. Desta forma, os agentes podem passar informações sobre o domínio do discurso, sem necessariamente operar sobre uma teoria globalmente compartilhada (Bonifácio, 2002).

Até meados de 90 o conhecimento de um sistema especialista não podia ser reusado ou compartilhado, organizando-se em bases de conhecimentos monolíticas e isoladas, sem interface de acoplamento, e, portanto, sem interoperabilidade (Freitas, 2002).

As ontologias proporcionam um avanço significativo na representação de conhecimento, na medida em que são reusáveis, extensíveis, especializáveis e interoperáveis. Ou seja, as ontologias possibilitam a representação de conhecimento de uma maneira evolutiva. Podemos dizer que as ontologias representam para a abordagem declarativa de representação de conhecimento o que o paradigma da orientação à objetos representa para a abordagem procedural.

Guarino (Guarino, 1998) Classifica as ontologias em quatro tipos, de acordo com sua dependência em relação a uma tarefa específica ou a um ponto de vista:

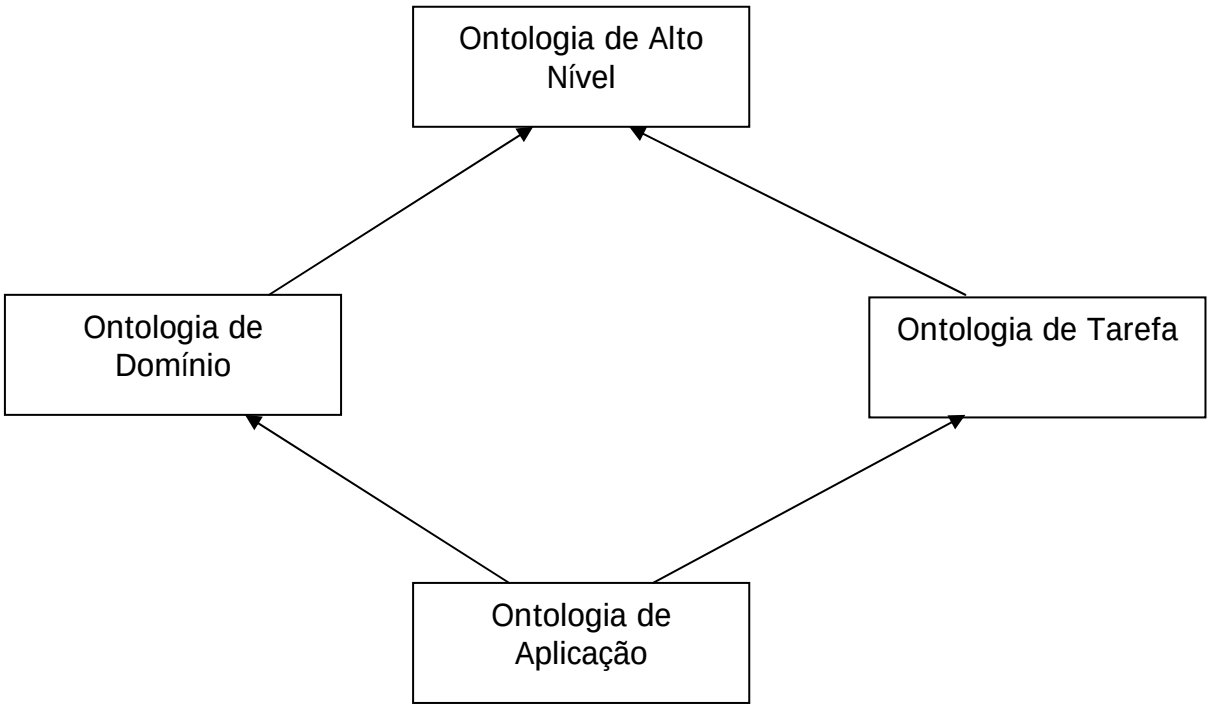

Figura 6: Tipos de Ontologias e seus relacionamentos (Guarino, 1998).

- ***Ontologia de Alto Nível:*** também chamada de ontologia de nível superior, descreve conceitos gerais como espaço, tempo, objeto, etc. Estes conceitos são dependentes de um problema específico.

- ***Ontologia de Domínio:*** descreve um vocabulário relacionado a um domínio genérico, como automóveis, medicamentos, livros, etc.

- ***Ontologias de Tarefa:*** descreve uma tarefa ou atividade dentro de um domínio, como venda de carros, compra de livros, vendas de medicamentos, etc.

- ***Ontologias de Aplicação:*** descreve conceitos que dependem tanto de um domínio específico como de uma tarefa específica.

Uma ontologia contém, tipicamente, um vocabulário de conceitos ou classes, relacionamento entre conceitos, atributos de conceitos e um conjunto de axiomas que definem as regras do negócio (Erdmann Apud Arruda, 2000), servindo para recuperção semântica de informações em ambientes heterogêneos de fontes de dados semi-estruturadas como ocorre na Web.

Os dados disponíveis na Web são ditos semi-estruturados, pois possuem metadados implicitamente armazenados juntamente com os dados, sendo também chamados de auto-descritivos. Pode-se ainda dizer que estes dados não são nem totalmente não-estruturados nem estritamente tipados (Abitebul 2000).

2.4.1 Ontologias e a Web Semântica

A Web pode ser vista como um enorme banco de dados de documentos disponibilizados para leitura. É natural que a vejamos desta forma, no entanto, a

aplicação das técnicas de projeto e a utilização das ferramentas de gerenciamento de banco de dados convencionais para gerenciar e integrar dados na Web não é trivial. O rigor dos Sistemas Gerenciadores de Bancos de Dados *(SGBD's)* tradicionais não é aplicável ao formato dos dados encontrados na Web.

Na Web os dados são semi-estruturados. Uma ontologia pode ser entendida como um modelo conceitual cujo objetivo é permitir o acesso aos dados semi-estruturados, exatamente como acontece nos bancos de dados convencionais. Existe, entretanto, uma sutil diferença entre uma ontologia e um modelo conceitual. Um modelo conceitual descreve, dentre outras coisas, a estrutura dos dados em um banco de dados em alto nível de abstração. Uma ontologia não representa a estrutura das fontes de dados associadas a ela, apenas propõe uma estrutura de consenso para conceitos e relações que são úteis para grupos de especialistas (Arruda, 2002).

A utilização de ontologias como suporte para o desenvolvimento de soluções sob a perspectiva da Web Semântica tem dois propósitos:

i) Garantir um entendimento conceitual entre os agentes acerca de um determinado domínio a partir da definição de um vocabulário comum e compartilhado por todos os agentes de uma comunidade;

ii) Atribuir significado aos recursos disponíveis na Web para possibilitar que os agentes possam fazer uma recuperação semântica dos conteúdos publicados

A linguagem HTML *(HyperText Markup Language)* se preocupa apenas com a apresentação de conteúdo e não atribui qualquer significado aos documentos, o que

limita os agentes à recuperação sintática das informações contidas nos documentos publicados atualmente na Web.

A linguagem XML, padrão para publicação, combinação e intercâmbio de documentos, desenvolvido pelo consórcio W3C *(World Wide Web Consortium)*, tem servido como plataforma para definição de algumas linguagens para representação de ontologias adequadas para uso no âmbito da Internet. Apresentamos, na próxima seção, algumas das principais linguagens desenvolvidas sobre a plataforma XML para representação de ontologias.

2.4.2 Linguagens para Representação de Ontologias

Para representar uma ontologia é necessária uma linguagem específica. Muitas são as linguagens de representação definidas. Entretanto, para aplicações Web é necessário que a linguagem de anotação ontológica possua uma sintaxe no padrão XML. As linguagens para representação de ontologias que nos interessam são as utilizadas para construir ontologias compartilhadas por agentes de software no âmbito da Web.

A seguir apresentamos as principais linguagens utilizadas para a representação de conhecimento no domínio da Web Semântica.

- **RDF e RDF Schema (RDFS)**

RDF *(Resource Description Framework)* é uma linguagem para representação de metadados baseada em XML.

RDF capacita a modelagem semântica, pois, sobre a sua especificação foram definidas outras linguagens com primitivas suficientes para a completa modelagem e raciocínio ontológico.

A linguagem RDF permite fazer declarações a respeito de recursos Web, não apenas páginas, mas também a qualquer componente de software ou de hardware acessível pela Web e, também a recursos off-line, que não podem ser acessados diretamente pela Web.

As declarações RDF possuem a forma de um trio (tripla), que consiste de um predicado, um sujeito e um objeto. Por exemplo, na frase “Aorta é uma artéria localizada no tronco”, extraída da figura 4, “Aorta” é o sujeito, “é uma artéria localizada” é o predicado, e “tronco” é o objeto.

Podemos ver essa frase por uma perspectiva diferente. Podemos dizer que localização é uma propriedade de “Aorta”. Esta propriedade tem o nome “localização” e o valor da propriedade é “tronco”. Podemos inferir, a partir do exposto, que RDF tem sua notação baseada em Frames. Onde, o sujeito é o recurso que se quer descrever, o predicado é cada propriedade do recurso e, o objeto os valores das propriedades. Podemos representar a frase acima em notação RDF como segue na Figura 7.

```
<rdf:RDF>
    <rdf:Description about=”Aorta”>
        <localizacao>tronco</localização>
    </rdf:Description>
</rdf:RDF>
```

Figura 7: Representação RDF da Frase: “Aorta é uma artéria localizada no tronco”.

O modelo RDF descreve recursos sempre no nível de instância de recursos. RDF Schema (RDFS) permite definir uma taxonomia de recursos em termos de classes, subclasses e superclasses de recursos.

RDFS provê informações sobre a interpretação das declarações formuladas em um modelo de dados RDF. O uso de RDFS é opcional, e seria equivalente a uma TBox das lógicas de descrição.

A expressividade das primitivas de RDF e RDFS não é suficiente para a completa modelagem e raciocínio ontológico. RDF e RDFS possuem capacidade de representação de restrições e de raciocínio sobre suas declarações bastante limitadas. Entretanto, RDF e RDFS serviram como um ponto de partida para construção de linguagens que permitem expressar o completo raciocínio ontológico.

RDF pode ser vista como uma tecnologia de capacitação para a modelagem semântica, como uma linguagem montadora genérica, sobre a qual podem ser criadas linguagens específicas do domínio e da tarefa (DAUM, 2002).

- **DAML-OIL**

O Joint Commitee US/EU foi criado em outubro de 2000 por Jim Hendler do US Defense Advanced Research Projects Agency *(DARPA)* e Hans-Georg Stork da European Union Informations Society *(IST)* com o objetivo de desenvolver uma linguagem semanticamente expressiva para representar o completo raciocínio ontológico.

Inicialmente uma linguagem chamada DAML-ONT *(DARPA Agent Markup Language)* foi desenvolvida pelo DARPA para que entendesse RDF com

construtores das linguagens de representação de conhecimento baseada em Frame. Como RDFS, DAML-ONT tinha uma especificação semântica fraca.

Paralelamente aos esforços do DARPA, na construção de DAML-ONT, um grupo de pesquisa europeu ligado ao *IST* desenvolveu uma linguagem de descrição de ontologias voltada para a Web chamada OIL *(Ontology Inference Layer)*. Do mesmo modo que DAML-ONT, OIL possuía uma sintaxe baseada em RDFS e um conjunto de construtores baseados em Frames. Entretanto, um forte rigor formal foi estabelecido e a linguagem foi explicitamente definida. OIL possui uma semântica formal e suporte a raciocínio providos pelas linguagens de lógica de descrição.

Os grupos de pesquisa resolveram unir seus esforços e unificaram DAML-ONT a OIL, produzindo deste modo, DAML-OIL.

Um conjunto de declarações DAML-OIL pode permitir a conclusão de uma outra declaração DAML-OIL. Por exemplo, podemos expressar o fato de que "*paternidade*" é um relacionamento mais genérico que "*maternidade*" e que "Maria é mãe de William" juntas, as definições permitem a conclusão de que "Maria é um dos pais de William". Desta forma, se um usuário perguntar "*quem são os pais de William?*", o sistema pode responder que "*Maria é um dos pais de William*", mesmo que o fato não tenha sido declarado em nenhum lugar.

De um ponto de vista formal, DAML-OIL é equivalente a uma linguagem de lógica de descrição bastante expressiva, com a ontologia DAML-OIL correspondendo a TBox, e as declarações de tipos e propriedades RDF correspondendo a ABox (Horrocks, 2002). Esta equivalência permite a DAML-OIL

aproveitar de todos os desenvolvimentos conquistados em anos de pesquisas realizadas sobre as lógicas de descrição.

A seguir, apresentamos DAML-S uma ontologia de alto nível utilizada para descrever Web Services. A ontologia DAML-S representa uma padronização na forma de descrever Web Services utilizando notação DAML-OIL. Utilizamos DAML-S para fazer as anotações das propagandas e dos requisitos de compras no ICS. Por isso achamos conveniente apresentá-la neste momento.

- **DAML-S**

Para fazer uso de um Web Service um agente de software necessita de uma descrição "*machine understandable*" do serviço que ele realiza e da forma como ele pode ser acessado. Com o objetivo de criar uma linguagem semanticamente expressiva para descrever as capacidades de um Web Service está sendo desenvolvida a linguagem de marcação DAML-S *(DARPA Agent Markup Language for Services)*, uma ontologia DAML-OIL de alto nível para descrição de propriedades e capacidades de Web Services

DAML-S provê uma padronização de como descrever as funcionalidades de um Web Service *(ServiceProfile)*, como o Web service realiza sua tarefa *(ServiceModel)* e como podemos acessá-lo *(ServiceGrouding)*. Dentre os benefícios proporcionados por DAML-S destacamos a automação de Web Services por meio da utilização de agentes e, a possibilidade de raciocínio *(reasoning)* sobre as propriedades e capacidades dos Web Services. Como podemos observar na figura 8, o modelo DAML-S possui três componentes: *ServiceProfile, ServiceModel* e *ServiceGrouding.*

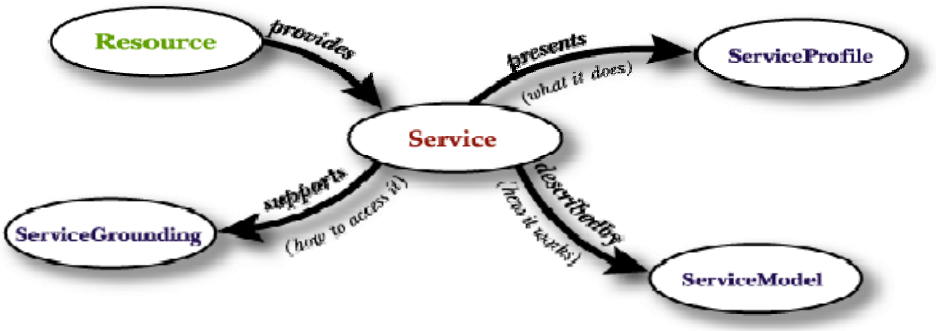

Figura 8: Ontologia de Serviço DAML-S (Anupriya, 2002).

Cada seção da ontologia DAML-S possui objetivos e funcionalidades particulares:

- ServiceProfile
  - Objetivo: proporcionar uma descrição de alto nível do Web Service.
  - Utilidade: povoar serviços de registro, descoberta automática de serviços e *Matchmaking***.**
- ServiceModel
  - Objetivo: descrever como o Web Service trabalha.
  - Utilidade: composição, interoperação e monitoramento de Web Services.
- ServiceGrouding
  - Objetivo: especificação da forma de acesso para o Web Service.
  - Utilidade: descrição de protocolos de comunicação e mecanismos de transporte. Ex. CORBA, http, RMI, etc.

Nós temos particular interesse pelas informações contidas na *ServiceProfile*. Faremos, portanto, maior detalhamento desta seção.

- **DAML-S ServiceProfile**

O objetivo do *ServiceProfile* é descrever as funcionalidades do Web Service, ou seja, o que o Web Service provê.

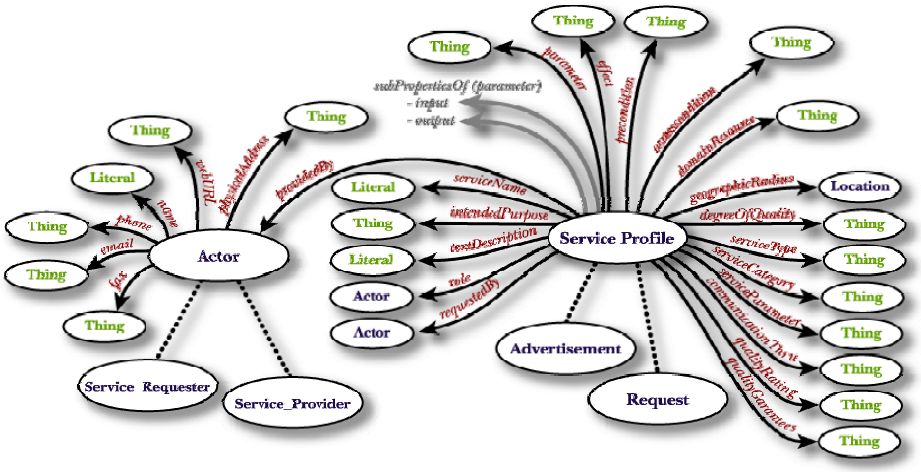

Figura 9: Service Profile (Anupriya, 2002).

Como podemos observar na figura 9, a ServiceProfile provê uma superclasse para descrição de alto nível dos serviços. No contexto ICS a ServiceProfile pode descrever tanto os requisitos de um comprador quanto a propaganda de um fornecedor. A propriedade *presents*, como podemos observar na figura 8, relaciona a classe *ServiceProfile* a classe *service*. A propriedade *presentedBy* é o inverso de presents e especifica que um dado profile descreve um serviço. ServiceProfile fornece também informações *"human-readable"* como: *serviceName*, *textDescription* e *contactInformation*. A classe *Actor* provê informações sobre o cliente ou fornecedor do serviço. A classe *ServiceProfile* ainda fornece a descrição funcional do serviço

que é especificada em termos de *entradas (inputs)*, *saídas (outputs)*, pré-*condições (pre-conditions)* e *efeitos (effects)*.

- Uma entrada é o que o Web Service requer para produzir a saída desejada.
- Uma saída é o que o Web Service fornece como resposta a uma solicitação, é a confirmação de que uma solicitação foi recebida e devidamente processada.
- Pré-condições representam as condições do mundo real que devem ser verdadeiras para que o serviço seja executado, por exemplo, possuir cartão de crédito de determinada bandeira (Visa, Mastercard, etc.) e possui um limite disponível de R$1000,00.
- Os efeitos são as mudanças ou ações efetivadas pelo Web Service, por exemplo, a venda de um produto.

DAML-S foi escolhida como linguagem para descrição das propagandas e dos requisitos de compras no ambiente ICS. A principal motivação para a nossa escolha é que DAML-S é uma ontologia de alto nível que utiliza a notação DAML-OIL.

2.5 Conclusão

A visão da Web Semântica, em particular a utilização de uma ontologia compartilhada pelos agentes em um domínio de aplicação, proporciona uma solução para problema de descoberta e emparceiramento automáticos de Web Services. Para tanto, a ontologia tem que ser descrita utilizando-se uma linguagem formal - para permitir o tratamento automático por agentes de software e, semanticamente

expressiva - para descrever com precisão as especificações dos clientes e as propagandas dos fornecedores.

O fato de DAML-OIL possuir primitivas que derivam das lógicas de descrição *(Description Logics)* possibilita-lhe fornecer os serviços de raciocínio providos por estas: *satisfatibilidade, subsumption, consistência e teste de instância.* Serviços fundamentais no contexto de nosso trabalho. Por esta razão, DAML-S foi escolhida como linguagem padrão para representação de propagandas e requisitos de compras, que serão validadas (teste de satisfatibilidade) e posteriormente comparadas (subsumption e teste de instância) para verificar a possibilidade de agrupamento de agentes fornecedores e compradores no contexto do ICS.

3. A ABORDAGEM MULTIAGENTE

No Capítulo 3, apresentamos os metadados semânticos e as ontologias, peças fundamentais para atribuir significado aos dados e serviços disponíveis na Web. Neste capítulo, apresentamos a tecnologia de agentes - componentes "Inteligentes" de software capazes de entender e processar os dados disponíveis na Web quando anotados em uma linguagem ontológica de representação de conhecimento como as apresentadas no capítulo anterior. Deste modo, os agentes constituem-se, juntamente com os metadados semânticos, componentes essências para o desenvolvimento da Web Semântica.

3.1 Introdução

A Internet, em especial a WWW (World Wide Web), criada no início da década de 90 por Tim Berners-Lee, é considerada a maior fonte de disseminação de informações nas principais áreas do conhecimento. A Web revolucionou os meios de comunicação, pois pode ser acessada de qualquer dispositivo conectado à Internet.

Com a facilidade de criação e publicação de páginas na Web um grande repositório de conteúdo foi criado. Encontrar a informação desejada passou a ser tarefa complexa. Sentiu-se a necessidade de melhorar as formas de busca e acesso à informação.

Os mecanismos de busca atuais são programas que varrem a Web capturando meta-informações, que posteriormente são armazenadas em enormes bancos de dados indexados por palavras-chaves. Esta abordagem tem se mostrado inadequada, uma vez que permite apenas buscas sintáticas, não considerando o

contexto em que as palavras ocorrem dentro do documento nem os relacionamentos existentes com outros termos.

A Web semântica surge como uma alternativa para superar este problema. Os agentes inteligentes de software são componentes chave desta evolução da Web. Nas próximas seções discorremos sobre os principais conceitos e aspectos relacionados à tecnologia de agentes buscando contextualizá-los em nosso trabalho.

3.2 Agentes Inteligentes

A noção de agente inteligente teve início na comunidade de IA como um paradigma para construção de sistemas inteligentes. A tecnologia de agentes extrapolou os domínios da IA, sendo empregada nas mais diversas áreas da Ciência da Computação, notadamente na solução de problemas ligados à Internet.

Mesmo sendo bastante antiga, não há uma definição consensual sobre a noção de agentes. A dificuldade de uma definição unificada consiste no fato de que as características dos agentes assumem diferentes graus de relevância em ambientes distintos. Dessa forma, o que tem ocorrido é que cada pesquisador define agente da maneira que melhor se adapte ao contexto de seu trabalho.

Uma característica dos agentes em que há consenso é a sua autonomia, ou seja, o agente deve satisfazer suas metas de forma autônoma, decidir que ações tomar baseado na percepção do ambiente em que está inserido, e provocar mudanças neste ambiente.

Uma definição bastante popular de agentes é dada por (Wooldridge, 1998): *“Um agente é um sistema computacional que está situado em algum* ***ambiente*** *e que é capaz de executar* ações ***autônomas*** *de forma* ***flexível*** *neste ambiente, a fim*

de satisfazer seus objetivos de projeto". As palavras chaves nesta definição são: *ambiente, autonomia* e *flexibilidade*.

Estar situado em um *ambiente*, no contexto da definição de Wooldridge, significa que o agente recebe entradas através de sensores de ambiente e que executa ações que mudam o ambiente de alguma forma.

Autonomia significa que o agente deve decidir que ações tomar para alcançar suas metas com base nas informações captadas do ambiente.

A *flexibilidade*, citada na conceituação acima, significa que o agente deve ser:

- Reativo, ou seja, deve perceber seu ambiente e responder oportunamente às mudanças que ocorrem nele;
- *Pró-ativo*, ou seja, n*ão* deve simplesmente atuar em resposta ao ambiente, deve exibir um comportamento oportunista e direcionado ao seu objetivo e tomar a iniciativa quando apropriado; e
- Social, ou seja, deve interagir, quando apropriado, com outros agentes artificiais ou humanos para completar suas próprias soluções de problemas ou ajudar os outros com suas atividades.

As características mencionadas acima estão ligadas ao que a comunidade científica convencionou chamar de noção fraca de agentes, que seriam as características mínimas requeridas.

Entretanto, para alguns pesquisadores, o termo agente tem significado bem mais forte que o apresentado anteriormente e, para eles, além das características

supramencionadas, os agentes devem possuir características mais próximas dos seres humanos como:

- **Benevolência:** é a suposição que os agentes não têm metas contraditórias, e que todo agente tentará sempre fazer o que lhe for solicitado. Um agente é benevolente se ele adota metas de outros agentes, quando solicitado, desde que estas não sejam contraditórias às suas;

- **Racionalidade:** um agente deve agir para alcançar suas metas e, não deve aceitar objetivos que sejam contraditórios com os seus, ou ainda que não sejam compensadores em termos de risco, custo e/ou esforço envolvido;

- **Adaptatividade:** um agente deve ajustar-se aos hábitos, métodos e preferências de seu usuário;

- **Mobilidade:** a habilidade de um agente para migrar de um ponto a outro da rede, de acordo com sua conveniência.

Outras características são atribuídas aos agentes: *veracidade*, *aprendizagem*, *intencionalidade*, etc. O número de características cresce à medida que novas aplicações vão sendo desenvolvidas e novos requisitos surgem.

No ICS, por exemplo, a mobilidade é uma característica fundamental para os agentes negociantes (compradores e vendedores), já que pretendemos agrupá-los localmente em mercados virtuais para reduzir os custos de comunicação e permitir estratégias de negociação mais robustas. Em outras aplicações a mobilidade pode

não ser útil e características como aprendizagem e veracidade serem mais relevantes o que provê à tecnologia de agentes um grande espectro de definições.

3.3 Inteligência Artificial Distribuída – IAD

A Inteligência Artificial Distribuída (IAD) é uma sub-área da inteligência artificial que tem investigado modelos de conhecimento e técnicas de comunicação e raciocínio que os agentes computacionais necessitam para participar em *"sociedades"* compostas por computadores e pessoas.

A IAD pode ser vista também como uma abordagem de solução de problemas complexos baseada em sociedades de agentes, onde as soluções para os problemas emergem das ações e interações entre os agentes membros de uma sociedade.

A IAD pode ser dividida em duas áreas, de acordo com o modelo usado para projetar a sociedade de agentes: *Resolução Distribuída de Problemas* e *Sistemas Multiagentes* (Freitas, 2002).

3.3.1 Resolução Distribuída de Problemas

Em uma abordagem DPS *(Distributed Problem-Solving)* há uma clara divisão de tarefas entre os agentes. Cada agente é projetado para resolver uma parte específica do problema e não pode desempenhar outro papel dentro da sociedade de agentes, uma vez que, cada agente possui uma visão particular e incompleta do problema. Deste modo, para resolver o problema os agentes têm que cooperar entre si e compartilhar informações sobre o problema e sobre os meios de se obter uma solução.

3.3.2 Sistemas Multiagentes (SMA)

Os Sistemas Multiagentes (*Multi-agent Systems – MAS*) possuem uma complexidade maior, uma vez que, a distribuição das tarefas é feita dinamicamente entre os agentes que formam a sociedade, ou seja, não existe uma definição *a priori* sobre quais tarefas serão executadas por qual agente.

Na abordagem SMA os agentes são entidades autônomas que têm conhecimento da sua própria existência e da existência de outros agentes e, portanto, colaboram uns com os outros para atingirem um objetivo comum dentro de um ambiente.

Os membros de uma sociedade de agentes são classificados como *reativos* ou *cognitivos* de acordo com o grau de complexidade e a forma que o conhecimento é representado.

Os agentes *reativos* possuem uma estrutura simples. O modelo de funcionamento de um agente reativo é o de *estímulo-resposta.* Em geral, estes agentes não apresentam memória, não planejam suas ações futuras e não se comunicam com outros agentes. Cada agente toma conhecimento das ações e do comportamento dos demais agentes através de modificações no ambiente.

Os agentes *reativos* são baseados em modelos de organização etológica/biológica (Freitas, 2002), como, por exemplo, as colônias de formigas ou abelhas. A idéia desta abordagem é que uma formiga não é capaz de realizar tarefas inteligentes, mas uma colônia de formigas o é. Assim, a inteligência se encontra no agrupamento de várias entidades simples.

Os agentes *cognitivos* são baseados em organizações sociais humanas e possuem:

- Conhecimento explicito, codificados em um formalismo lógico de representação de conhecimento;
- Mecanismo de comunicação direta, baseado na teoria de Atos da Fala (*speech act theory)* (Austin, 1962); e
- Dispõem de memória e, por meio dela, são capazes de planejar suas ações futuras.

Os agentes *cognitivos* que possuem uma representação simbólica formal do ambiente em que estão inseridos e, sobre as quais é possível fazer deduções de novos fatos são chamados *agentes deliberativos.* Os agentes *deliberativos* são de grande relevância no contexto da Web Semântica. Uma vez atribuído o significado aos dados disponíveis na Web, através dos formalismos e linguagens ontológicas apresentadas no capítulo anterior, os agentes deliberativos tornam-se capazes de entendê-los *(understandable)* e de inferir novos conhecimentos a partir deles.

3.4 Comunicação entre Agentes

Para que uma sociedade de agentes coopere a fim de atingir um determinado objetivo, é necessário que seja definida uma arquitetura que possibilite a interação entre eles. Nestas interações, ocorre troca de conhecimento, objetivos, planos ou escolhas através da comunicação, que pode acontecer de forma direta - empregada quando os agentes têm conhecimento um do outro ou indireta - utilizada quando os agentes não se conhecem.

A principal forma de comunicação indireta é o *blackboard*, uma estrutura de dados intermediária compartilhada por todos os agentes onde as informações serão escritas e lidas durante o desenvolvimento das tarefas.

A comunicação direta entre agentes abrange dois modelos: Cliente-Servidor e Peer-to-Peer.

3.4.1 Modelo Cliente-Servidor

O modelo Cliente-Servidor é baseado em RPC *(Remote Procedure Call)* para efetuar comunicação passada como parâmetros para os procedimentos solicitados.

O modelo cliente-servidor é derivado do paradigma de programação procedural e é baseado na chamada de métodos remotos (Freitas et al., 2002). O modelo cliente-servidor efetua internamente uma comunicação do tipo *pedido-resposta*. A maior limitação deste modelo é a falta de flexibilidade, pois os métodos possuem uma interface estática e o modelo de comunicação é pré-definido.

3.4.2 Modelo Peer-to-Peer

O modelo *Peer-to-Peer* é baseado na teoria de Atos de Fala. Esta teoria considera que a linguagem falada engendra ações e provoca mudanças no ambiente. A comunicação entre agentes cognitivos utiliza o modelo *Peer-to-Peer* que é semanticamente mais precisa e abrangente que o modelo Cliente-Servidor.

A teoria de atos de fala propõe uma categorização de primitivas de comunicação como, por exemplo, *inform, ask-to-do, answer, propose,* etc. As primitivas são associadas às suas conseqüências. Os atos de fala podem ser classificados segundo três aspectos:

Ato Locucionário - emissão de palavras e sentenças com algum significado;

Ato Ilocucionário - corresponde à intenção da emissão: *request, order, apologize, assert*; etc.

Ato Perlocucionário - corresponde ao resultado desejado da emissão: *convince, insult, frighte, etc.*

No modelo *Peer-to-Peer*, qualquer agente pode iniciar um diálogo baseado em atos de fala (que expressa a intenção do agente) e com conteúdo baseado em conhecimento declarativo codificado em algum formalismo de representação. As mensagens devem limitar-se a um contexto e vocabulário comuns, onde as trocas de mensagens sejam efetuadas dentro de uma semântica bem definida e sem ambigüidade. Este contexto é normalmente fornecido por Ontologias compartilhadas.

3.5 Linguagens de Comunicação entre Agentes

As duas principais características dos agentes são perceber e agir. A habilidade de comunicar-se é fundamental à percepção (quando recebe mensagens) e para ação (quando envia mensagens). Existem linguagens que são destinadas exclusivamente à comunicação entre agentes. Estas linguagens definem padrões para a troca de mensagens. As principais linguagens de comunicação entre agentes são KQML e FIPA-ACL.

3.5.1 KQML

KQML(*Knowledge Query and Manupulation Language)* é uma linguagem e um protocolo de comunicação, de alto nível, para troca de mensagens independentemente da sintaxe, do conteúdo e da ontologia aplicável. Foi

desenvolvida dentro do *“Knowledge Sharing Effort”* patrocinado pelo DARPA. KQML é baseada na teoria de atos de fala.

KQML trata os agentes como bases de conhecimento virtuais e associa estados cognitivos dos agentes com primitivas da linguagem, chamadas performativas. Uma mensagem é enviada junto a uma performativa que indica o que se espera que o receptor faça com a mensagem. Por exemplo, *"ask"* informa ao receptor que é esperado dele uma resposta a uma pergunta e *"tell"* indica que do receptor é esperado que acredite nos fatos comunicados. Na figura 10 mostramos um exemplo de comunicação entre dois agentes *“joe”* e *“stock-server”*.

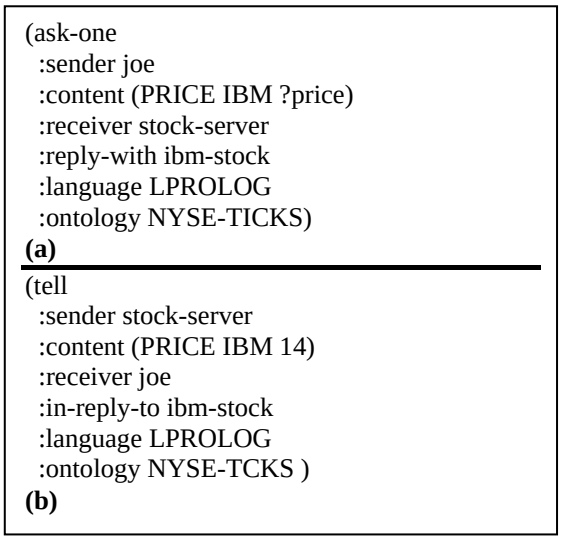

Figura 10: Troca de mensagens KQML (Labrou, 1999).

Na figura 10 (a) o agente “joe” solicita ao agente *“stock-server”* o preço das ações da IBM. Na figura 10 (b) o agente *“stock-server”* devolve ao agente *“joe”* o valor das ações da IBM.

KQML possui um conjunto de performativas reservadas. O conjunto de performativas KQML é extensível. Uma comunidade de agentes pode escolher utilizar performativas adicionais. No entanto, se uma performativa é reservada, ela

deve ser implementada da forma padrão. Na tabela 1 listamos as principais performativas KQML.

Categorias	Performativas
Serviços básicos de informação	tell, deny e untell
Consultas básicas	evaluate, reply, ask-if, ask-about, ask-one, ask-all
Consultas multi-resposta	stream-about, stream-all, eos
Rede	register, unregister, foward, broadcast, pipe, break, transport-address
Respostas básicas	reply, error, sorry
Convencimento	achieve, unachieve
Definição de capacidades	advertise
Notificação	subscribe, monitor
Banco de Dados	insert, delete, delete-one, delete-all
Auxílio	Broker-one, broker-all, recommend-one, recommend-all, recruit-one, recruit-all

Tabela 1: Principais performativas reservadas KQML.

Uma mensagem KQML é organizada em três partes: comunicação, conteúdo e mensagem. A sintaxe de uma mensagem KQML é mostrada na figura 11.

```
( performativa
        :sender         <word>
        :receiver       <word>
        :reply-with     <word>
        :in-reply-to    <word>
        :language       <word>
        :ontology       <word>
        :content        <expression>
)
```

Figura 11: Sintaxe mensagem KQML.

3.5.2 FIPA-ACL

A FIPA (*Foundation for Intelligent Physical Agents*) é uma fundação sem fins lucrativos, ligada ao movimento de software livre, que tem como objetivo a disseminação do uso de agentes, especialmente no desenvolvimento de soluções de comércio eletrônico.

A FIPA-ACL é uma linguagem, como a KQML, baseada na teoria de atos de fala. Sua especificação consiste de um conjunto de tipos de mensagem e descrições dos efeitos da mensagem sobre os agentes que a enviam e sobre os que a

recebem. FIPA-ACL é bastante parecida a KQML. Sua sintaxe é idêntica, exceto por alguns nomes diferentes para primitivas.

KQML utiliza o termo “performativa” para se referir às primitivas de comunicação. Em FIPA-ACL, as primitivas de comunicação são chamadas de *ações comunicativas*. Apesar dos nomes diferentes, as *performatives* e ações comunicativas significam a mesma coisa.

A principal diferença entre FIPA-ACL e KQML é que o padrão FIPA-ACL tenta identificar os mecanismos de comunicação e cooperação entre agentes através da definição de uma linguagem com semântica concisa e formal.

A especificação FIPA para comunicação entre agentes – que inclui primitivas de comunicação *(performativas)*, um modelo formal, e uma linguagem de conteúdo *(Semantic Language SL)* – Foi extraído da linguagem ARCOL projetada pela *France Télecom*, enquanto que KQML inspirou a estrutura de suas mensagens (Montesco, 2001).

3.6 Agentes Intermediários

Um dos principais problemas no projeto de sistemas multiagentes em ambientes abertos, como a Internet, é o *problema da descoberta*. Encontrar outros agentes que tenham informações ou capacidades que um determinado agente está precisando não é uma tarefa fácil. Para resolver este problema, normalmente utiliza-se agentes intermediários.

Existem basicamente três tipos de agentes intermediários: *Blackboard*, *Matchmaker* e *Broker*.

3.6.1 Agente Blackboard

Um agente *Blackboard* tem como principal atribuição fazer o gerenciamento de consultas. Os usuários enviam seus problemas para o *Blackboard* e os fornecedores consultam o *Blackboard* para saber se há algum problema que ele seja capaz de resolver, ou seja, nesta arquitetura, somente os fornecedores conhecem suas próprias capacidades.

A Figura 12 mostra o funcionamento de um *Blackboard*. Um problema é enviado ao *Blackboard* por um agente usuário (AU). Um agente fornecedor (AF) consulta o *Blackboard* para encontrar um problema que possa resolver. O *Blackboard* então envia o problema ao AF. Este o resolve e envia a resposta diretamente para o AU que formulou o problema.

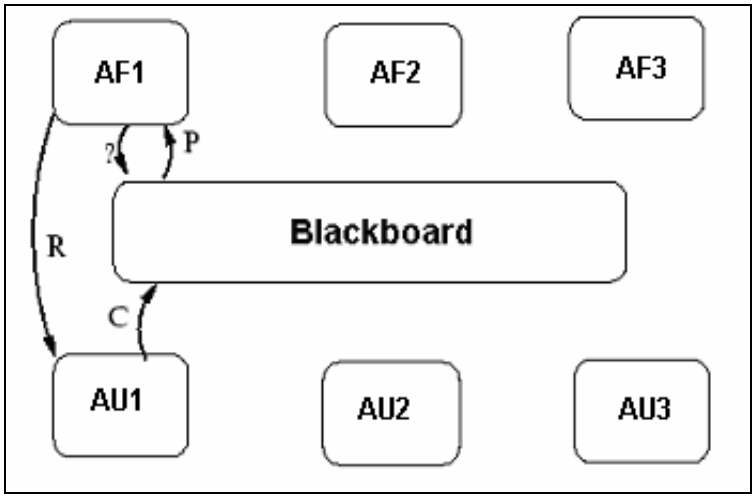

Figura 12: Funcionamento do Agente Blackboard (Eudenia 2001).

3.6.2 Agente Broker

Um *Broker* é um agente que armazena as capacidades dos agentes fornecedores e possivelmente, as preferências dos agentes usuários. O *Broker* recebe uma consulta do usuário, escolhe um agente fornecedor que possa respondê-la e a envia. Recebe a resposta do fornecedor e a repassa para o agente

usuário que formulou a consulta. Deste modo, apenas o *Broker* sabe que agente fornecedor possui determinada informação ou capacidade. A figura 13 mostra o funcionamento do agente *Broker*.

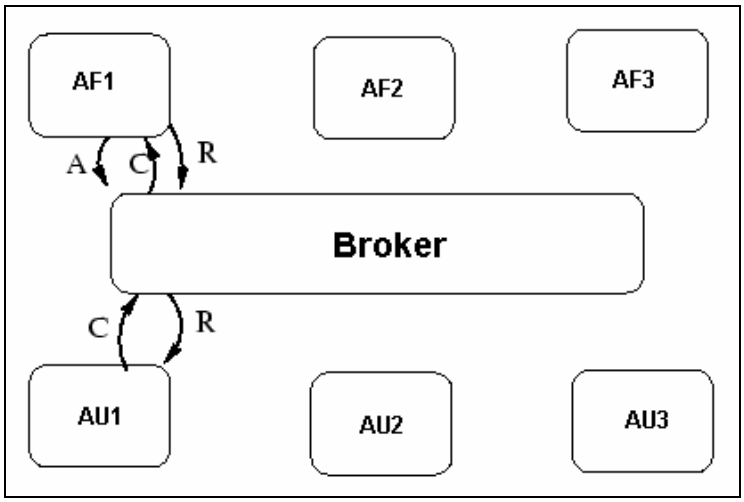

Figura 13: Funcionamento do Agente Broker (Eudenia, 2001).

Nesta abordagem os agentes usuários (AU) e fornecedores (AF) não precisam saber planejar consultas, formular consultas e traduzir linguagens, devendo apenas conhecer a linguagem do *Broker* que se encarrega de realizar todas as tarefas mencionadas. Outra característica dos *Brokers* é a privacidade, pois somente o *broker* sabe das capacidades de cada agente fornecedor.

3.6.3 Agente Matchmaker

Um *Matchmaker* é um agente que armazena as capacidades dos fornecedores, recebe consultas dos usuários e responde que agentes fornecedores podem resolver a consulta. O agente usuário escolhe e entra em contato diretamente com o fornecedor que deseja. A figura 14 mostra o esquema de funcionamento descrito.

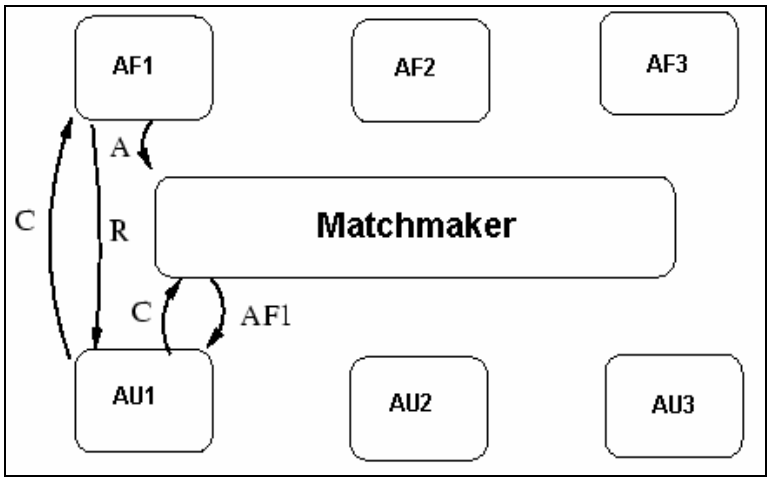

Figura 14: Funcionamento do Agente Matchmaker (Eudenia, 2001).

Note que diferentemente do *Blackboard*, o *Matchmaker* informa para o agente usuário (AU1) que o agente fornecedor (AF1) pode resolver sua solicitação. A partir deste momento os agentes (AU1) e (AF1) passam a estabelecer uma comunicação direta sem intermédio do agente *Matchmaker.*

As principais características dos agentes *Matchmakers* são:

- **Falta de compromisso*:*** um agente usuário pode fazer consultas sem firmar nenhum compromisso. Isto significa que um agente pode descobrir quem pode responder uma determinada consulta e, no entanto, não fazê-la.

- **Rapidez*:*** o *Matchmaker* tem processamento mais rápido que um *Broker* ou um *Blackboard*, pois faz *"apenas'' um casamento (Matching) de padrões".*

- **Comunicação direta*:*** não existem *"gargalos"* de comunicação entre os agentes usuários e fornecedores. Uma vez escolhido que agente vai ser consultado, a consulta é feita diretamente. Caso mais de um

agente precise ser consultado, o planejamento da consulta é feito pelo próprio agente usuário.

Estas características fizeram com que a abordagem de *Matchmaker* fosse a escolhida para aproximar os agentes vendedores dos compradores na arquitetura do ICS.

3.7 Conclusão

O ICS *(Intelligent Commerce System)* é uma arquitetura baseada em SMA, projetada para rodar em um ambiente aberto como a Internet. Para proporcionar uma negociação mais eficiente utilizamos a tecnologia de agentes móveis, o que permite que após se descubram, os agentes possam migrar para um host específico e negociarem localmente.

Considerando que os agentes podem estar espalhados por *hosts* em todo mundo e que atualmente a largura de banda em uso na Internet é bastante limitada, um processo de negociação robusto torna-se inviável. A idéia é aproximar os agentes negociantes em regiões de negociação, instanciada em um *host*, onde todos os agentes inseridos nesta região possuam interesses de negócio comuns.

Antes que os agentes possam migrar para um *host* específico, eles precisam se descobrir na Web. Para resolver o problema de descoberta entre os agentes compradores e os agentes vendedores, quaisquer das arquiteturas de agentes intermediários apresentadas na seção 3.6 podem ser usadas. Para promover a aproximação entre compradores e vendedores, no contexto do ICS, escolhemos a abordagem de *Matchmakers* por serem mais ágeis, permitirem comunicação direta

entre os agentes negociantes e por possuírem uma implementação mais simplificada.

III. INTELLIGENT COMMERCE SYSTEM - ICS

4. ICS

Neste capítulo, descrevemos o ambiente ICS (Labidi et al., 2003) de suporte ao comércio eletrônico na categoria B2B. O objetivo é fornecer uma visão clara do sistema, de modo que possamos delimitar e justificar nossa contribuição dentro do ICS.

4.1 Introdução

Os portais B2B, sites utilizados por empresas de um determinado ramo de negócio para comercialização de bens ou serviços, além de muito caros, são também muito antiquados. Quando uma empresa deseja comprar determinado produto ou serviço nestes portais, um operador humano vasculha o portal em busca de oportunidades de negócios, em seguida, realiza uma espécie de barganha, comparando cada oferta e, finalmente fecha um contrato com determinado fornecedor.

Como uma alternativa para esta modalidade de B2B propomos o ICS - sistema inteligente de comércio eletrônico em que agentes de software negociam a compra e a venda de produtos ou serviços em nome das empresas usuárias.

A principal finalidade do ICS é automatizar as tarefas de encontrar parceiros comerciais, negociar e fechar contratos, atualmente realizadas por operadores humanos nos portais B2B.

O ICS é baseado em agentes móveis inteligentes e visa automatizar o processo de compra e venda entre as empresas, reduzindo seus custos operacionais e agilizando seus processos de negócios. Os agentes fazem uso da

infra-estrutura da Internet para se comunicar, negociar e fechar contratos. Nas seções seguintes apresentamos em mais detalhes cada componente do ICS.

4.2 Ciclo de Vida do Comércio Eletrônico no ICS

O ICS utiliza uma extensão do modelo de ciclo de vida proposto por (Jennings et al., 1996) e (Bartolini, 2001). Foram introduzidos: a fase de modelagem do usuário, o conceito de feedback de informações e ontologias específicas para cada fase. Ficando o ciclo de vida do comércio eletrônico no ICS como ilustrado na figura 15.

Figura 15. Ciclo de vida do comércio eletrônico no ICS.

- ***Modelagem do usuário:*** É uma característica importante do ICS que permite um tratamento personalizado a cada agente negociador dentro do mercado virtual com base em informações coletadas do próprio ambiente de negociação.

- ***Matchmaking:*** É um dos principais aspectos em ambientes de comércio eletrônico e consiste basicamente em aproximar fornecedores e compradores com interesses complementares. No ICS, o *matchmaking* é realizado fazendo-se a comparação sintática e semântica das características que são requeridas por uma das partes negociantes (o comprador) e oferecidas pelas outras partes negociantes (os vendedores).

- ***Negociação:*** Uma vez agrupados os possíveis parceiros de negócio, os agentes negociantes iniciam o processo de negociação para concordar ou discordar os temas mutuamente aceitáveis, ou seja, cada agente busca a compatibilidade dos interesses da empresa que representa com os interesses das demais empresas.

- ***Formação do Contrato:*** Consiste da formalização em um contrato eletrônico com validade jurídica dos termos que regem a negociação como: preço, prazo, forma de pagamento, meio de entrega etc. Para tanto, o contrato eletrônico deve ser assinado por cada uma das partes e certificado por uma entidade certificadora como *Verisign* a fim de garantir sua privacidade, autenticidade, integridade e o não repúdio por alguma das partes.

- ***Execução de Contratos:*** Os contratos trabalham com datas limites e sua execução segue um protocolo. Um contrato pode ser visto como um protocolo de recompensa ou punição, onde os agentes são recompensados quando cumprem o contrato e punidos em caso contrário. Propomos a utilização da tecnologia de *Workflow* temporal (Labidi et al., 2000) para controlar o fluxo de execução dos contratos firmados dentro do ICS.

4.3 Arquitetura do ICS

O ICS foi idealizado para proporcionar uma arquitetura aberta, flexível e evolutiva. Com esta finalidade foram empregadas camadas de abstração de alto nível. O ICS é baseado na arquitetura de agentes. Utilizamos agentes móveis para

representar os negociantes (vendedores e compradores). O objetivo desta abordagem é proporcionar maior robustez e agilidade no processo de negociação, pois deste modo, os agentes negociantes podem migrar através da rede para um lugar comum (um *host*) e somente depois de estarem fisicamente próximos iniciarem um processo de negociação.

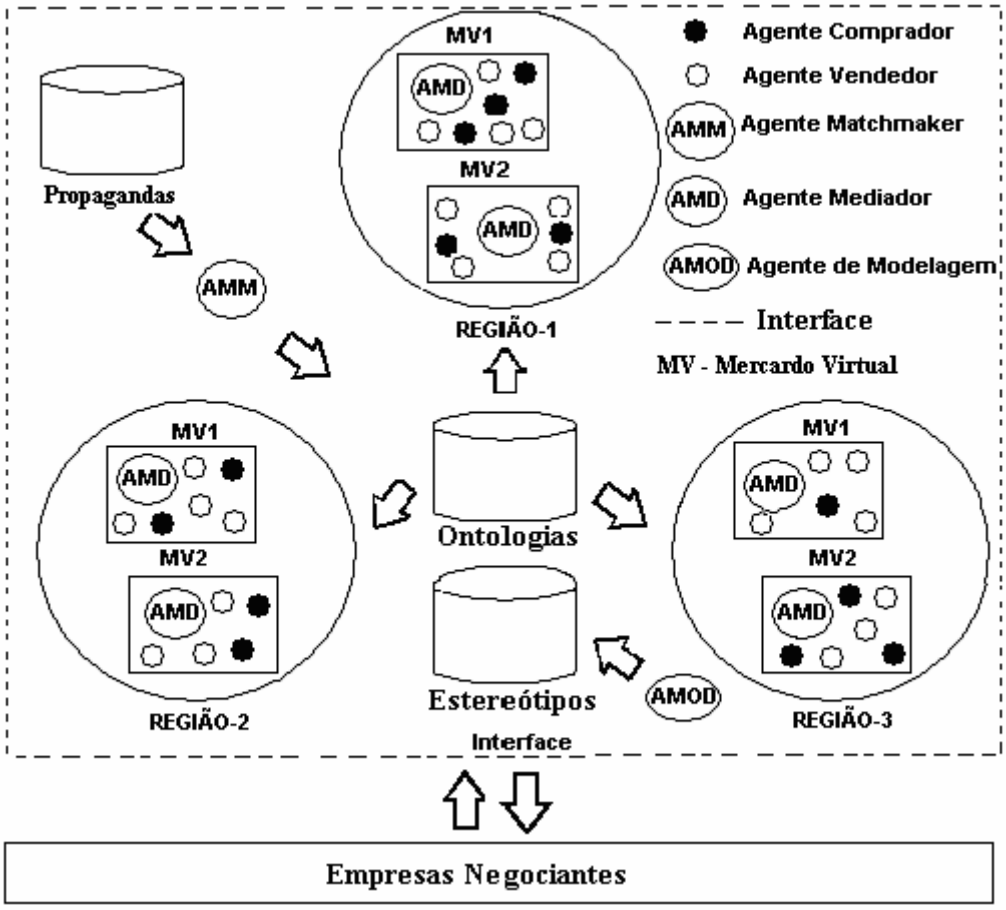

Figura 16: Arquitetura do ICS.

A arquitetura do ICS é baseada em duas abstrações: *Mercado Virtual e Região,* quatro classes de agentes: *Agente Matchmaker, Agente Mediador, Agente Negociante e Agente de Modelagem*, e três repositórios de dados semi-estruturados: *Repositório de Ontologias, Base de Estereótipos e Base de Propagandas* (Tomaz, 2003). A figura 16 ilustra uma visão de alto nível da arquitetura do ICS.

A seguir, detalhamos cada componente do ICS. Os componentes estão dispostos em uma seqüência didática que permite uma melhor compreensão de cada componente bem como as interações existentes entre eles.

4.3.1 Mercado Virtual (MV)

É o local onde os agentes realizam a negociação e o fechamento do contrato. O Mercado Virtual provê um contexto dentro do qual os agentes podem trabalhar, ou seja, protocolo de comunicação, identificação dos agentes negociantes, uniformização para a representação dos requisitos de compras dos clientes e propagandas dos fornecedores, ontologia de domínio do negócio e protocolo de negociação utilizado.

Um mercado virtual é instanciado em uma máquina (*host*), ou seja, os agentes operam localmente. Entretanto, nada impede que em uma máquina seja instanciado mais de um mercado virtual.

4.3.2 Região

É uma abstração de mais alto nível que o Mercado Virtual (MV) que proporciona uma visão de agrupamento de mercados virtuais que estão operando sobre a mesma ontologia de domínio, ou seja, atuando na mesma área de negócio (venda de veículos, medicamentos, livros, etc). Isto possibilita a comunicação entre mercados virtuais instanciados na mesma máquina ou em máquinas distintas.

Apesar de uma região proporcionar uma visão de agrupamento de mercados virtuais eles podem estar fisicamente separados, rodando em máquinas distintas.

Sempre que um novo mercado virtual é criado ele é identificado como membro de uma região. Somente mercados virtuais de mesma região podem se comunicar. O objetivo é impedir que agentes negociantes que estejam representando empresas de ramos diferentes, mas que possuam conceitos homônimos, porém com significados diferentes, se confundam com os termos empregados em função do contexto em que está sendo utilizado. Por exemplo, uma loja de confecções anuncia uma camisa com determinado modelo de "**manga"** e um atacadista que opera com frutas tropicais deseja comprar uma tonelada de "**manga**". Para seres humanos isso não representa nenhum problema, pois nós somos capazes de percorrer vários contextos com relativa facilidade, já que não seguimos modelos lógicos formalizados, por outro lado, para os agentes isso causaria uma confusão semântica muito grande.

4.3.3 Repositório de Ontologias (modelos de domínios)

É uma base de dados semi-estruturada que permite armazenar as ontologias de domínio. O ICS não se limita a operar em um domínio de negócio especifico, e pode evoluir para quantos domínios se queira. Para tanto, basta adicionar uma nova ontologia de domínio no repositório de ontologias. Cada Ontologia armazenada no repositório está associada a uma região específica.

4.3.4 Base de Estereótipos

É uma base de dados semi-estruturada que permite descrever o perfil de cada usuário. É um protótipo do modelo do usuário. É utilizada pelos Agentes negociantes para deliberarem sobre as preferências das empresas que eles estão representando. Por exemplo, determinada empresa pode preferir prazo ou qualidade

à preço. Deste modo, podemos dizer que a Base de Estereótipos dá suporte às negociações entre os Agentes Compradores e Vendedores.

4.3.5 Base de Propagandas

É uma base de dados semi-estruturada que armazena todas as propagandas anotadas em DAML-S. Após um comprador especificar seus requisitos de compra o agente *Matchmaker* percorre este repositório em busca de propagandas que satisfaçam aos requisitos do comprador.

4.3.6 Agente Matchmaker

O agente *Matchmaker* tem por objetivo aproximar agentes negociantes com objetivos complementares, ou seja, ele é o responsável pela composição dos Mercados Virtuais.

Para realizar sua tarefa o Agente *Matchmaker* precisa descobrir os identificadores dos agentes com objetivos complementares, instanciar um MV e instanciar um Agente Mediador que controlará o processo de negociação entre os agentes negociantes dentro do Mercado Virtual.

4.3.7 Agente Mediador

O Agente Mediador opera como um árbitro dentro de um MV. Ele acompanha cada transação realizada e intervém quando necessário com o objetivo de resolver problemas de negociação, formação e execução de contratos. Para cada Mercado Virtual há uma instância do Agente Mediador.

Uma vez aproximados pelo *Matchmaker*, os agentes negociantes iniciam a negociação sob a supervisão do mediador, que pode desclassificar ou punir agentes que infringirem regras básicas de negociação. Por exemplo, um agente pode intencionalmente reduzir seu preço a valores inexeqüíveis para ganhar uma concorrência e depois não firmar o contrato de compra e venda.

De acordo com a conveniência das empresas que operam em determinado domínio, o mediador pode fazer composição de produtos ou serviços ofertados. Por exemplo, em um domínio de vendas de computadores e periféricos, digamos que um agente comprador esteja à procura de um computador com gravador de CD, mas que não haja nenhuma oferta de computador com gravador de CD no mercado virtual. Entretanto, existam várias ofertas de computadores e algumas de gravadores de CD separadamente. Se for informado pelo agente de modelagem que determinados agentes negociantes têm interesse em formar parcerias, o mediador pode formar um contrato de parceria entre as empresas representadas por estes agentes negociantes para que elas forneçam conjuntamente o computador com o gravador de CD.

Outra característica relevante do agente mediador é que ele registra todas as negociações realizadas dentro do MV para fins de auditoria. Novos agentes negociantes, ao entrarem no mercado, podem fazer uso das trilhas de auditoria para se situarem. Por exemplo, verificar negociações anteriores à sua entrada no MV e assim aumentar ou diminuir o preço de seu produto para se tornar mais competitivo.

4.3.8 Agente Negociante

Um agente negociante pode ser um comprador ou um vendedor. Estes agentes são instanciados pelas empresas negociantes (compradoras ou vendedoras) usuárias do ICS a partir de sua interface Web ou através de uma chamada remota aos métodos do serviço de Match, já que o *Matchmaker* é ele próprio um Web Service. Ao Instanciar um agente negociante é necessário inicialmente saber se ele deseja comprar ou se ele deseja vender um produto ou serviço. O ICS fornece um formulário de cadastro de propaganda caso seja um agente vendedor ou um formulário de cadastro de consulta caso seja um agente comprador.

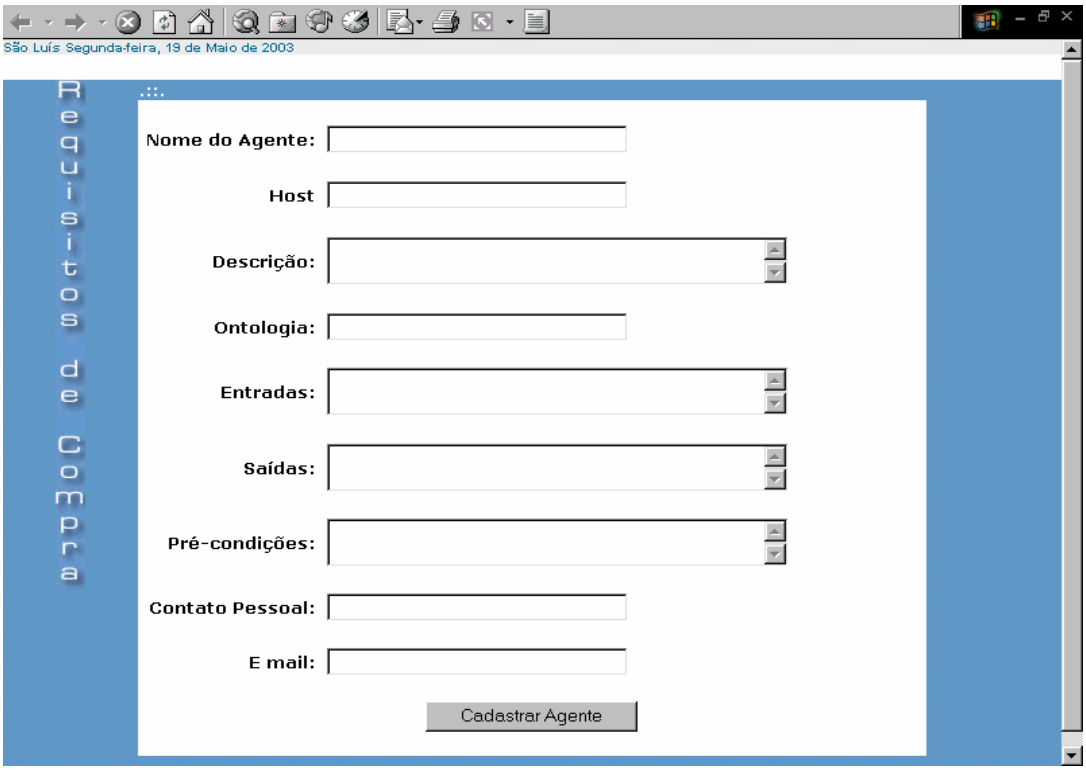

Figura 17: Formulário de Cadastro de Agentes Compradores no ICS.

Na figura 17 é mostrado o formulário de cadastro de propagandas. O formulário para cadastro de consultas é bastante semelhante ao apresentado na figura 17.

Neste formulário devem ser informados dados de identificação do agente como nome do agente, *host* onde está instanciado, porta em que está escutando, e-mail para contato com quem o disparou (humano), a ontologia de domínio a qual deseja se associar (veículos, medicamentos, livros, etc.) - para que o agente possa ser enquadrado em uma região e finalmente, as especificações do produto ou serviço que o agente deseja comprar ou vender em termos de entradas, saídas e pré-condições.

4.3.9 Agente de Modelagem

O agente de modelagem tem por finalidade informar ao agente mediador das preferências de cada agente negociante para que o agente mediador possa interagir de maneira personalizada com cada um dos agentes negociantes.

Para inferir o perfil de cada agente negociante o agente de modelagem reúne informações provenientes da interface, do modelo de domínio (Repositório de Ontologias) e da Base de estereótipos.

Para melhor explicar o papel do agente de modelagem, segue dois exemplos de atividades desempenhadas por este agente:

i) É o agente de modelagem quem informa ao agente mediador se um determinado agente vendedor está ou não interessado em fazer parcerias com outros agentes vendedores para fornecer determinado produto ou serviço para um agente comprador.

ii) É o agente de modelagem responsável por informar ao agente mediador se, para um agente comprador, o critério mais relevante na

definição de um fornecedor é o preço, a qualidade, a forma de pagamento, o prazo de entrega, ou qualquer outra característica do produto ou serviço ofertados pelos agentes vendedores.

4.4 Conclusão

O ICS diferencia-se das aplicações B2B convencionais principalmente por não ser limitado a um único domínio de negócio, podendo evoluir para um número ilimitado de domínios.

Outro importante diferencial é a utilização da tecnologia de agentes móveis que possibilita um processo de negociação mais eficiente, uma vez que os agentes com interesses complementares interagem localmente em um *host*.

A abordagem da Web Semântica para resolver o problema de descoberta dos possíveis parceiros de negócios também é outro diferencial, que faz com que a descoberta seja muito mais eficiente e flexível, pois a busca não se restringe a aspectos sintáticos, mas também a aspectos semânticos e contextualizados.

A arquitetura baseada em um ciclo de vida bem definido possibilita a componentização da implementação do ICS, proporcionando maior qualidade do produto final.

No capítulo seguinte, abordamos em mais detalhes o processo de *matchmaking* dentro do ICS. Este processo é realizado por um agente especifico denominado Agente matchmaker.

.

IV. AGRUPAMENTO DE AGENTES NEGOCIANTES NO ICS

5. MATCHMAKING NO ICS

Neste capítulo, apresentamos o conceito de matchmaking e o processo de modelagem e implementação do Agente Matchmaker - componente do ICS responsável pela composição dos agrupamentos (Clusters) de agentes negociantes com interesses complementares dentro de um domínio de negócio.

5.1 Introdução

O objetivo do processo de *matchmaking* é o de aproximar compradores e fornecedores com potencialidade de realização de negócios entre si. Em um Mercado Virtual, onde os agentes de software negociam em favor de seus usuários humanos, comumente são empregados agentes intermediários para realizar a descoberta de oportunidade de negócios entre eles.

Somente é possível a descoberta de agentes com interesses afins comparando-se os requisitos de compras dos agentes compradores com as propagandas dos agentes vendedores. Para permitir o cruzamento semântico, automático por agentes de software, entre os requisitos de compras e as ofertas disponíveis no Mercado Virtual são necessárias a utilização de um formalismo bastante expressivo - para representar os requisitos e as ofertas e, um algoritmo capaz de realizar o cruzamento semântico das descrições.

Padrões de mercado como UDDI *(Universal Description, Discovery, and Integration)* foram criados com esta finalidade, entretanto a linguagem utilizada WSDL *(Web Services Description Language)* é muito pouco expressiva e não permite uma descrição mais elaborada dos produtos ou serviços oferecidos.

Neste capítulo, apresentamos a última colaboração deste trabalho que é a prototipação do agente *Matchmaker* utilizando uma abordagem baseada na Web Semântica e na tecnologia de Web Services.

5.2 Trabalhos Relacionados

Existem trabalhos relacionados à descoberta automática de Serviços Web sendo desenvolvido em alguns centros de pesquisa. Dentre os trabalhos existentes o que mais se aproxima de nossa pesquisa é o *"Semantic Matchmaker"* em desenvolvimento no *Robotics Institute of Carnegie Mellon University-CMU* (URL 2).

O *"Semantic Matchmaker"* possui vários pontos em comum com o nosso trabalho e podemos destacar, particularmente, dois aspectos:

i) O fato de utilizar DAML-S como linguagem de descrição de serviços; e

ii) O fato de empregar um motor de inferência baseado em lógica de descrição, *Jess* (URL 14), para fazer o cruzamento semântico entre os requisitos de compras dos clientes e as propagandas dos fornecedores.

Entretanto, alguns pontos são fundamentais na diferenciação de nosso trabalho em relação ao trabalho em desenvolvimento no CMU:

i) Optamos por armazenar requisitos de compras e propagandas em uma base de dados semi-estrturada *(Tamino Server)* enquanto que no *"Semantic Matchmaker"* desenvolvido pelo *CMU* eles optaram por utilizar o padrão UDDI enriquecido com uma função adicional de busca semântica (URL 3). A vantagem de nossa escolha é que ela nos

permite aplicar um filtro de contexto para trazer apenas as propagandas que estiverem relacionadas ao domínio especificado pelo comprador, utilizando uma linguagem de consulta como *XQuery* (URL 8), o que restringe significativamente o espaço de busca no repositório de propagandas e reduz o custo computacional do cruzamento semântico dos requisitos de compras com as propagandas;

ii) Utilizamos um padrão aberto de comunicação com os *reasoners DL*[5], o formato DIG[6] (URL 9), que em breve promete tornar-se um padrão de comunicação com *reasoners DL* em aplicações baseadas na Web Semântica. Esta característica permite que o nosso *Matchmaker* torne-se independente de motor de inferência, permitindo, inclusive, trabalharmos com múltiplos motores de inferência simultaneamente;

iii) Nosso *Matchmaker* enxerga cada Web Service como um agente móvel que representa o interesse de uma empresa, portanto, após ele identificar os agentes negociantes com interesses complementares ele envia uma mensagem para cada agente convidando-o para deslocar-se para um Mercado Virtual onde eles prosseguirão o ciclo de comercialização, passando à fase de negociação.

iv) Por fim, nosso *Matchmaker* é ele próprio um Web Service o que permite sua utilização por aplicações de forma distribuída e não

[5] Ferramenta que faz inferências a uma base de conhecimento descrita em um formalismo de representação baseado em lógica de descrição.

[6] DL Implementation Group (DIG) é um grupo formado por pesquisadores e desenvolvedores ligados à implementação de sistemas baseados em lógica de descrição (Description Logic Systems). O padrão DIG de comunicação com reasoners DL foi especificado por Sean Bechhofer da University of Manchester, para maiores detalhes sobre a linguagem consulte o Anexo I. Este padrão encontra-se atualmente em desenvolvimento. O release atual é o 1.1 lançado no inicio de 2003.

apenas a partir de uma interface Web. Característica original de nosso trabalho, pois até o fechamento desta dissertação não havia nenhuma proposta de trabalho com esta propriedade.

Outro importante trabalho, também desenvolvido no CMU e que possui relação com o nossa pesquisa é o projeto LARKS *(Language for Advertisement and Request for Knowladge Sharing)*, idealizado por Katia Sycara (URL 4). A LARKS é uma linguagem que permite a descrição das capacidades de agentes de software de modo a permitir a descoberta automática e a aproximação de agentes com interesses afins. LARKS é considerada bastante expressiva e fácil e assim como DAML-S também é capaz de permitir inferências. Para maiores informações sobre LARKS consulte (Sycara, 2002).

5.3 Arquitetura do Agente Matchmaker

Como dito anteriormente, modelamos o Agente *Matchmaker* utilizando a tecnologia de Web Services como blocos de montagem. Entretanto, como os padrões de descoberta de Web Services sofrem da falta de semântica, empregamos a visão da Web semântica para superar esta deficiência e, deste modo, assegurar uma descoberta automática e eficiente de serviços na Web.

Propomos a utilização de ferramentas e padrões decorrentes do desenvolvimento da Web Semântica como uma alternativa para um *matchmaking* semântico.

No ICS os agentes vendedores e compradores podem ser vistos como Web Services, logo precisam publicar respectivamente os serviços que oferecem e os requisitos de compras de modo que o Agente *Matchmaker* possa aproximá-los.

Como podemos observar na figura 18 os usuários do ICS (compradores e vendedores) se comunicam com o ICS a partir de uma interface Web, representada na figura acima pela linha pontilhada. A interface Web permite aos fornecedores cadastrarem suas propagandas em um repositório de propagandas e aos agentes compradores localizarem dentro deste repositório as propagandas que satisfaçam de algum modo seus requisitos de compras.

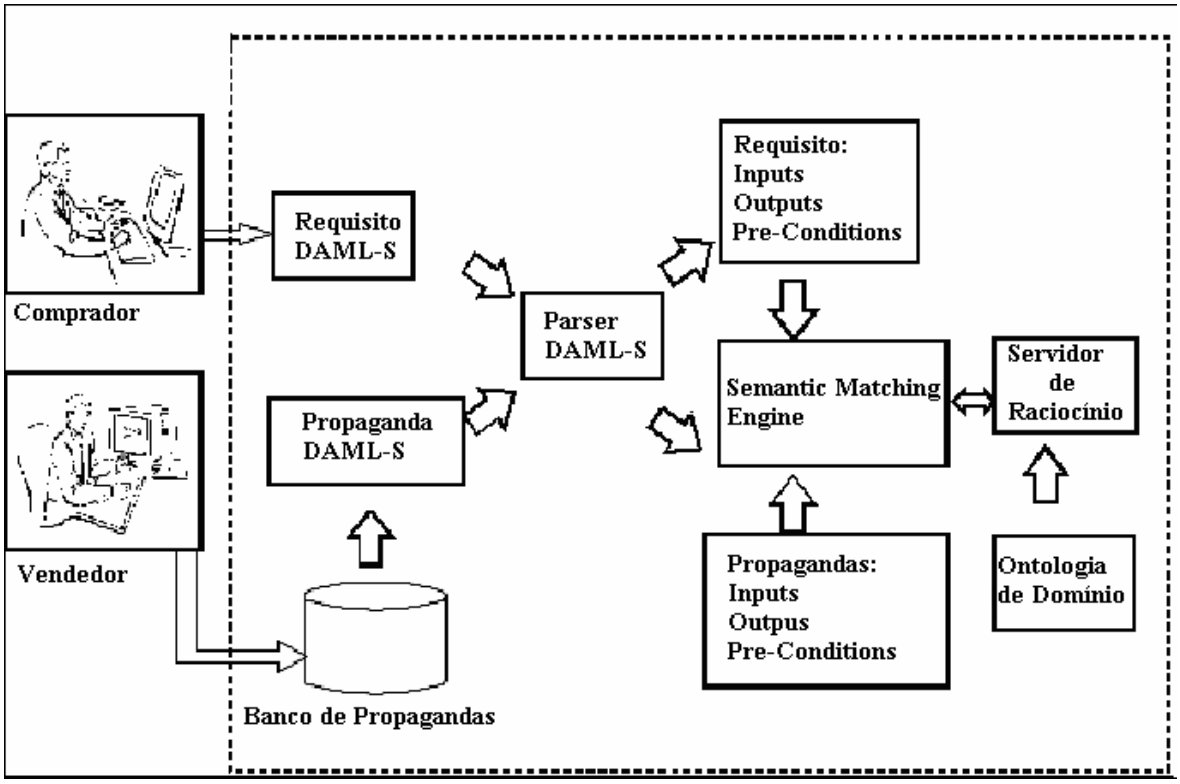

Figura 18: Arquitetura do Agente Matchmaker.

Propagandas e requisitos de compras são descritos em DAML-S como forma de padronizar as especificações dos requisitos de compras e dos produtos ou serviços em termos de entradas, saídas e pré-condições. Um *parser* DAML-S extrai as entradas, saídas e pré-condições da *ServiceProfile* das propagandas e requisitos de compras.

O *Semantic Matching Engine* (SME) envia a um *reasoner DL* as entradas dos requisitos juntamente com as entradas de cada uma das propagandas para serem classificadas e comparadas, com base na ontologia de domínio, para encontrar alguma relação semântica entre elas. Este mesmo procedimento é repetido para as saídas e para as pré-condições.

De acordo com a distância semântica existente entre os conceitos presentes nos pares de entradas, saídas e pré-condições de requisitos e de cada propaganda apontado pelo *reasoner DL*, o *Matchmaker* monta uma lista ordenada de emparceiramentos, onde os agentes com maior possibilidade de se tornarem parceiros vêm na frente.

A descrição feita nesta seção dá uma visão muito superficial do real funcionamento do Agente *Matchmaker*. Com o objetivo de detalhar melhor as funcionalidades do agente *Matchmaker* apresentamos na seção abaixo o diagrama de classes de sua arquitetura.

5.4 Projeto e Implementação

O protótipo foi projetado de forma iterativa e incremental e implementado na linguagem JAVA (Flanagan 1997). A figura 19 apresenta um diagrama de classes em notação UML *(Unified Modeling Language)*.

A classe principal do sistema é a classe *SME (Semantic Matching Engine).* A Classe SME possui os seguintes métodos:

SendDigQuery: este método envia uma mensagem no formato DIG contendo consultas enviadas ao *reasoner DL*.

ReadDigResponse: este método ler a resposta fornecida pelo *reasoner DL* às consultas enviadas pelo método *SendDigQuery.* As respostas devolvidas pelo *reasoner* também estão no padrão DIG.

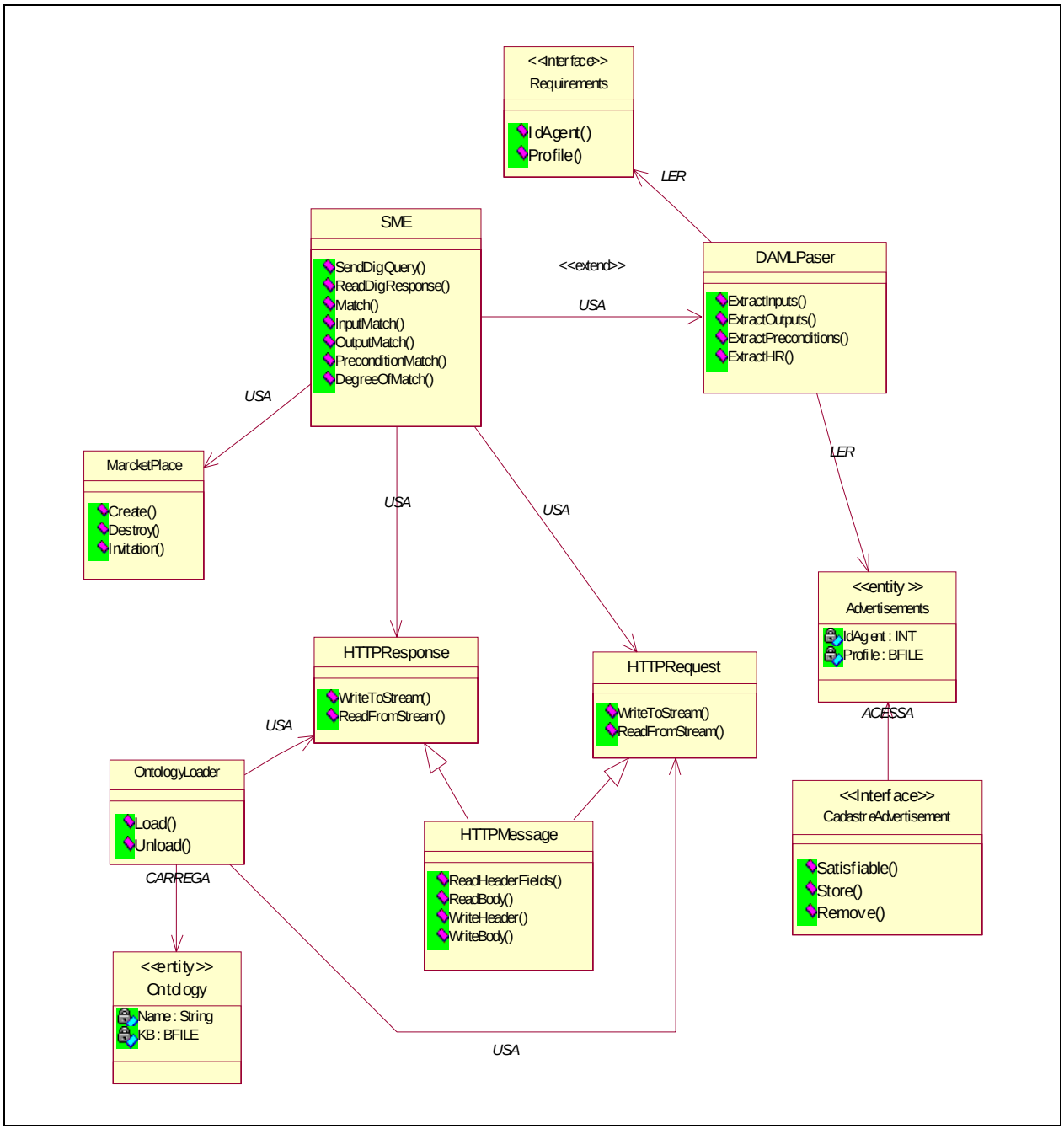

Figura 19: Diagrama de Classes.

Match: método que devolve um lista ordenada contendo os identificados dos agentes compradores e vendedores *(AgentID)* que possuem interesses

complementares. Este método faz uso dos demais métodos privados descritos abaixo para alcançar seu objetivo.

InputMatch: método que compara as entradas do requisito de compra de um usuário às entradas de cada propaganda armazenada no repositório de propagandas *(advertisement)*.

OutputMatch: método que compara as saídas do requisito de compra de um usuário às saídas de cada propaganda armazenada no repositório de propagandas.

PreconditionMatch: método que compara as pré-condições do requisito de compra de um usuário às pré-condições de cada propaganda armazenada no repositório de propagandas.

*DegreeOfMatch**:*** Método que define o grau de combinação entre os pares de entradas, saídas e pré-condições presentes no requisito de compra e em cada uma das propagandas.

Uma descrição em pseudo-código dos algoritmos dos métodos da classe SME são apresentados na seção 5.6. Não apresentamos o algoritmo dos métodos *SendDigQuery* e *ReadDigResponse**,*** pois estes métodos consistem basicamente de uma conexão via TCP *Sockets* ou *http Steams* com o *reasoner DL* onde mensagens no formato DIG são enviadas e recebidas.

A classe *OntologyLoader* possui apenas dois métodos:

Load: método que submete uma base de conhecimento ao *reasoner* e devolve uma resposta do *reasoner* quanto a satisfatibilidade ou não da ontologia.

Unload: método que descarrega (limpa da memória) uma base de conhecimento previamente carregada pelo método *load*.

As classes *HTTPResponse* e *HTTPRequest* são especializações da classe *HTTPMessage.* Estas classes fornecem métodos que permitem a manipulação de mensagens no formato *http* e servem como uma infra-estrutura de comunicação com os *reasoners DL* que utilizam o padrão DIG de comunicação. Esta infra-estrutura de comunicação foi desenvolvida por *Christian Finckler da University of Applied Sciences, Wedel*, a quem somos gratos.

As Classes *Ontology* e *Advertisement* são repositórios semi-estruturados em notação XML. *Ontology* possui um atributo *name* que é o identificador de cada ontologia e um atributo *kb (Knowledge Base)* onde fica armazenada as sentenças da ontologia em marcação XML. A classe *Advertisement* possui o atributo *ID* que identifica cada propaganda dentro do repositório e o atributo *Profile* que armazena a propaganda no formato DAML-S.

A classe *DAMLSParser* possui métodos que permitem a extração das entradas *(inputs)*, saídas *(outputs)* e pré-condições *(preconditions)* das propagandas armazenadas na entidade *Advertisement* e também para extrair as entradas, saídas e pré-condições dos requisitos de compra que são fornecidos na interface *Requeriments.*

A interface *CadastreAdvertisement* acessa o repositório de propagandas e permite armazenar, remover e testar a satisfatibilidade das propagandas armazenadas na entidade *Advertisement.*

A seguir apresentamos o modelo de caso de uso do agente *Matchmaker* com a finalidade de apresentá-lo sob a perspectiva dos usuários (compradores e vendedores).

5.5 Modelo de Caso de Uso do Agente Matchmaker

Com o objetivo de proporcionar uma visão mais clara do funcionamento do Agente *Matchmaker* sob a perspectiva de seus usuários, foi elaborado um modelo de caso de uso, onde são explorados em mais detalhes os principais eventos e interações entre os componentes do *Matchmaker* e os usuários.

Apresentamos os casos de uso na ordem em que ocorrem, entretanto, alguns casos de uso precisam executar outros processos, ou seja, possuem outros casos de uso embutidos.

Caso de Uso	Anunciar_Produto_ou_Servico
Ator	Usuário Web
Propósito	Armazenar uma propaganda de um produto ou serviço no repositório de Anúncios.
Resumo	O Usuário acessa a URL do ICS na Web seleciona a opção propaganda e em seguida preenche os campos disponíveis na interface de cadastro de propagandas (Nome do Agente, Endereço do Host, Ontologia de Domínio,..., Descrição do Serviço ou produto DAML-S).

Tabela 2: Caso de Uso Anunciar Produto ou Serviço.

Para que o caso de uso "*Anunciar Produto ou Serviço"* cadastre uma propaganda no repositório de propagandas é necessário antes a realização do caso de uso "*Validar Anuncio"*, mostrado na tabela 3.

Caso de Uso	Validar_Anuncio
Ator	Agente Matchmaker
Propósito	Verificar a consistência da propaganda com a ontologia do domínio especificada na interface de cadastro de propagandas
Resumo	Depois de preencher o formulário no use case Anunciar_Produto_ou_Serviço, o usuário Web deve acionar o botão confirmar para que o agente Matchmaker valide os conceitos presentes na propaganda baseado na ontologia de domínio selecionada. Caso a propaganda seja válida o Matchmaker deve armazená-la no repositório de propagandas, em caso contrário, notificar o usuário da inconsistência na propaganda.

Tabela 3: Caso de Uso Validar Anúncio.

Cada usuário pode cadastrar quantos agentes desejar no repositório de propagandas. Outra ação disparada pelo usuário é o *matching* dos agentes negociantes como mostra o caso de uso da tabela 4.

Caso de Uso	Matching_Agentes
Ator	Agente Matchmaker
Propósito	Montar agrupamento de agentes com interesses afins e com possibilidade de negociação
Resumo	O usuário Web acessa o ICS a partir de sua URL e aciona a opção Matching, preenche os campos disponíveis na interface (Nome do Agente, Endereço do Host, Ontologia de Domínio, ..., Descrição dos requisitos de compra em DAML-S) e pressiona o botão confirmar. O agente Matchmaker fará a comparação semântica dos requisitos de compras especificados no formulário com todas as propagadas cadastradas no repositório de propagandas sob a mesma ontologia de domínio que o agente comprador recém instanciado.

Tabela 4: Caso de Uso Matching Agentes.

Para realizar o caso de uso "*Matching Agentes"* é preciso realizar outros dois processo: "*Parser Requisito Propaganda"* e "*Compara Conceitos"* respectivamente apresentados nas tabelas 5 e 6.

Caso de Uso	Parser_Requisito_Propaganda
Ator	Agente Matchmaker
Propósito	Extrair entradas, saídas e pré-condições dos requisitos dos compradores e das propagandas dos fornecedores.
Resumo	Para permitir uma comparação entre as entradas, saídas e pré-condições do requisito com as entradas, saídas e pré-condições das propagandas é necessário fazer-se um parser nas anotações DAML-S para extrair estes parâmetros.

Tabela 5: Caso de Uso Parser Requisito Propaganda.

O Use Case "*Compara Conceitos"* submete a um *reasoner DL* os pares de entradas, saídas e pré-condições extraídos dos requisitos e das propagandas, respectivamente, para serem comparados.

Caso de Uso	Compara_Conceitos
Ator	Agente Matchmaker
Propósito	Verificar o grau de similaridade entre as propagandas armazenadas no repositório de propagandas e o requisito de compra de um agente
Resumo	Depois de extraídas pelo parser DAML-S as entradas, saídas e as pré-condiçoes das propagandas e dos requisitos de compras estes parâmetros são enviados para um reasoner DL apontar o grau de similaridade (Distância Semântica) dos conceitos neles embutidos com base na ontologia de domínio.

Tabela 6: Caso de Uso Compara Conceitos.

Depois de cruzados os parâmetros das propagandas com os requisitos de compra, o agente o *Matchmaker* pode montar o *Cluster* de Agentes negociantes com

interesses complementares. Isto é realizado no use case "*Montar Cluster"* descrito na tabela 7.

Caso de Uso	Montar_Cluster
Ator	Agente Matchmaker
Propósito	Montar uma lista contendo os identificadores dos possíveis agentes parceiros
Resumo	Depois de realizada a comparação o Matchmaker deve montar uma lista contendo todos os agentes que devem deslocar-se ao novo Mercado Virtual.

Tabela 7: Caso de Uso Montar Cluster.

Agora o *Matchmaker* precisar instanciar um novo Mercado Virtual e convidar os agentes contidos na lista de *cluster* para se deslocarem para o contexto deste mercado, como descrito nos Use Cases "*Instanciar Mercado"* e "*Convidar Agentes"* nas tabelas 8 e 9 respectivamente.

Caso de Uso	Instanciar_Mercado
Ator	Agente Matchmaker
Propósito	Instanciar um mercado virtual contendo os possíveis parceiros de negócio
Resumo	Depois de montado o cluster de agentes, o matchamker deve verificar se não há um Mercado Virtual com características semelhantes já instanciado em algum host, caso não haja nenhum deverá instanciar um novo Mercado Virtual

Tabela 8: Caso de Uso Instanciar Mercado.

Caso de Uso	Convidar_Agentes
Ator	Agente Matchmaker
Propósito	Convidar os agentes negociantes contidos na lista de cluster para se deslocarem para o host onde foi instanciado o Mercado Virtual
Resumo	O Matchmaker deve enviar uma mensagem para cada um dos agentes negociantes da lista de cluster solicitando que se desloquem para dentro do contexto do Mercado Virtual recém-instanciado.

Tabela 9: Caso de Uso Convidar Agentes.

Nas próximas seções abordaremos as contribuições da Web Semântica e da tecnologia de Web Services para a modelagem do agente *Matchmaker*.

5.6 Contribuições da Web Semântica

A Web Semântica é uma combinação das técnicas e formalismos de representação de conhecimento com a tecnologia de agentes. O objetivo é proporcionar que os dados distribuídos na Web, chamados dados semi-estruturados,

possam ser lidos, entendidos e processados automaticamente por agentes de software.

Para permitir uma descrição semanticamente expressiva dos requisitos de compra e das propagandas dentro do ICS utilizamos DAML-S, uma ontologia DAML-OIL de nível superior que padroniza a forma como descrevemos os serviços na Web.

Empregamos a tecnologia de agentes móveis para dar maior robustez e velocidade ao processo de negociação entre os agentes. Uma vez descritas em DAML-S, as propagandas e requisitos são comparados por um *reasoner DL*, capaz de fazer uma classificação de taxonomia dos conceitos presentes tanto nas propagandas quanto nos requisitos de compras e, posteriormente, calcular a distância semântica existente entre estes conceitos dentro da árvore taxonômica.

A Web Semântica permite a aproximação de agentes que não se complementam perfeitamente, mas que podem, de algum modo, se ajudar em um processo posterior de negociação. Por exemplo, uma floricultura, representada por um agente negociante, oferece flores e um comprador, representado por um agente negociante que deseja comprar orquídeas. Caso faça apenas uma comparação sintática dos conceitos *flor* e *orquídea*, o Agente *Matchmaker* não faz a aproximação dos agentes negociantes. Entretanto, consultando uma ontologia de domínio, o *Matchmaker* percebe que orquídea é um tipo de flor, ou seja, Orquídea $\subset$ Flor e conseqüentemente infere que deve promover a aproximação dos agentes.

5.7 Contribuições da Tecnologia de Web Services

Os Web Services oferecem uma nova abordagem para modelar aplicações na Web, onde os sites se comunicam dinamicamente e sob demanda.

Nas aplicações convencionais de B2B, as negociações são feitas em um modelo "*um-para-um"*, ou seja, cada comprador negocia individualmente com um fornecedor. A utilização de uma arquitetura baseada em serviços permite um modelo de negociação "*muito-para-muitos".*

A aplicação da abordagem dos Web Services como bloco de montagem do Agente *Matchamaker* possibilita a utilização de padrões de descoberta de Web Services como UDDI. Entretanto, optamos por utilizar uma base de dados semi-estruturada que permite a aplicação de filtros de contexto a partir de uma linguagem de consulta como *XQuery*.

5.8 Algoritmos de Matching

Os principais algoritmos utilizados para a implementação do Agente *Matchmaker* são os algoritmos de *matching*. Estes algoritmos são utilizados para fazer o cruzamento semântico entre os conceitos presentes nos requisitos de compra dos compradores e os conceitos presentes nas propagandas dos vendedores. Baseamos nossa implementação nos algoritmos de *matching* semântico propostos por (Paolucci, 2002). A idéia principal destes algoritmos é a de que uma propaganda satisfaz um requisito de compra quando:

i) Todas as saídas da propaganda combinam com todas as saídas do requisito; e

ii) Todas as entradas da propaganda combinam com todas as entradas do requisito.

A primeira assertiva garante que o vendedor oferece o que o comprador deseja comprar. Já a segunda assertiva assegura que o comprador fornece todas as

informações que o vendedor necessita para oferecer-lhe o produto ou serviço adequado.

Os algoritmos originalmente propostos por Paolucci levam em consideração apenas as entradas e as saídas, ignorando as pré-condições, fundamentais para um processo de negociação mais elaborado. Questões como forma de pagamento, prazo de entrega, meio de transporte, etc. são tratados nas pré-condições, enriquecendo sobremaneira o processo de negociação.

O algoritmo principal do *Matchmaker* (método *match* da Classe SME) é apresentado abaixo:

```
Match(request){
  recordMatch = empty list
     forall adv in advertisement do
       {  if match(request, adv) then
            recordMatch.append(request,adv)
     return sort(recordMatch); }}
```

A função *Match* tem por objetivo retornar uma lista ordenada contendo os pares de identificadores de agentes compradores e fornecedores que possuem alguma afinidade.

Para cada requisito de um agente comprador *(request)*, passado como parâmetro para a função *Match*, será percorrido todo o banco de propagandas *(advertisement)* em busca de ofertas que o satisfaçam. Sempre que uma propaganda satisfaz o requisito do agente comprador o identificador do agente vendedor, proprietário da propaganda, é armazenado dentro de uma lista *(recordMatch)* juntamente com o identificador do agente comprador.

Neste algoritmo, a função *Match* é composta de duas partes: a primeira parte compara todas as saídas dos requisitos *(outR)* com cada uma das saídas das

propagandas *(outA)* armazenadas no banco de propagandas. A segunda parte compara as entradas das propagandas *(InA)* armazenadas no banco de propagandas com as entradas do requisito *(InR)* passado como parâmetro para a função *Match*.

A primeira parte da função *Match* pode ser vista no fragmento de código abaixo. A segunda parte é semelhante à exibida abaixo. Somente os parâmetros para a função são alterados, uma vez que, na segunda parte, compara-se as entradas das propagandas *(InA)* com as entradas do requisito *(InR)*.

```
OutputMatch(outputRequest, outputAdvertisement)
{   globalDegreeMatch=Exact
    forall outR in outputRequest do {
        find outA in outputAdvetisement such as that
            degreeMatch=maxDegreeMatch(outR,outA)
             if (degreeMatch=fail) return fail
             if (degreeMatch<globalDegreeMatch)
                globalDegreeMatch=degreeMatch }}
```

Como podemos observar, para cada uma das saídas do requisito *(outR)* será feita uma busca por propagandas cuja saída *(outA)* a complemente com algum grau de satisfação. O grau de combinação entre duas saídas ou duas entradas depende do tipo de conexão semântica existente entre conceitos embutidos nas entradas e saídas, ou seja, pela distância entre os conceitos dentro da árvore de taxonomia da ontologia de domínio.

O algoritmo utilizado distingue quatro graus de combinação: *"**exact**"*, *"**plugin**"*, *"**subsumes**"* e *"**fail**"* como mostra o fragmento de código abaixo:

```
DegreeOfMatch(outR,outA)
    If outA=outR then return exact
    If outA subsumes outR then return plugin
    If outR subsumes outA then return subsumes
Otherwise  fail
```

Se a saída da propaganda *(outA)* for igual à saída do requisito *(outR)* dentro da ontologia de domínio, então o grau de combinação é *“exact”*.

Se a saída da propaganda *(outA)* é um conjunto que contém a saída do requisito *(outR)*, ou, em outras palavras, a saída da propaganda *(outA)* pode ser *“plugada”* no lugar da saída do requisito *(outR)*, a combinação é dita *“plugin”*. Por exemplo, um serviço que oferece flores *(outA)* pode ser usado por clientes que desejam comprar orquídeas *(outR)*. Neste caso, pode-se assegurar que o vendedor atende aos requisitos do comprador, mesmo que parcialmente. Entretanto, quando ocorre o contrário, a saída do requisito *(outR)* é que contém a saída da propaganda *(outA)*, não se pode assegurar que o vendedor atende aos requisitos de compras do comprador mas há uma possibilidade de que isso ocorra. Neste caso, diz-se que o grau de combinação é *“subsumes”.*

Quando não existe nenhuma relação semântica entre os conceitos, a função *DegreeOfMatch* devolve *“fail”*.

Além das entradas e das saídas o algoritmo do Agente *Matchmaker* aplica também as mesmas regras sobre as pré-condições. Obviamente, as saídas têm um peso maior que as entradas e as pré-condições, pois refletem exatamente o desejo do comprador e a oferta do vendedor. Se os agentes não se complementarem com grau de satisfação *“exact”*, *“plugin”* ou *“subsumes”*, não há porque submeter as entradas e pré-condições para comparação, deve-se passar logo para uma outra propaganda no banco de propagandas.

5.9 Ferramentas

Como pode ser observado na figura 18, os principais componentes do Agente Matchmaker são*: Parser DAML-S, Semantic Matching Engine, Reasoner DL e Banco de Dados de propagandas*. Nas subseções seguintes, comentamos sobre cada uma das ferramentas utilizadas na implementação dos componentes do Agente Matchamker.

5.9.1 Jena

Jena é um *"toolkit"* para o desenvolvimento de aplicações dentro do contexto da Web Semântica. Jena possui uma API Java que permite a manipulação de modelos RDF e, conseqüentemente, todas as linguagens construídas sobre este padrão, inclusive DAML-OIL. No pacote Jena destacamos as seguintes características:

i) Suporte para carga de ontologias DAML-OIL em um modelo Jena;

ii) Parser RDF integrado *(ARP- Another RDF Parser)*;

iii) Módulo de armazenamento persistente em banco de dados padrão XML *(Berkley DB)*;

iv) Linguagem de consulta integrada *(RDQL)*; e

v) Arquitetura aberta para suporte a outras implementações de armazenamento.

Jena é utilizada, principalmente, na implementação do *parser* DAML-S, responsável por extrair as entradas, saídas e pré-condições dos requisitos dos

compradores e das propagandas dos vendedores. Como DAML-S é uma ontologia DAML-OIL que até o momento, ainda está sendo especificada e, portanto algo muito novo, não encontramos nenhum *parser* DAML-S implementado. Utilizamos JENA como ferramenta para implementação do *parser* DAML-S.

Jena foi desenvolvida nos laboratórios da HP. Não é objetivo deste trabalho aprofundar em sua estrutura e funcionalidades. Para maiores informações consulte (URL 13).

5.9.2 RACER

O RACER *(Renamed ABox and Concept Expression Reasoner)* é um sistema de representação de conhecimento baseado em *tableux*[7] para linguagens de lógica de descrição (DL). Este sistema permite serviço de raciocínio para múltiplos *ABoxes* e múltiplos *TBoxes*. O RACER implementa um sistema de descrição lógica conhecido como SHIQ (URL 14).

Utilizamos o RACER para fazer o teste de satisfatibilidade das ontologias de domínio e das propagandas antes de armazená-las no banco de dados de ontologias e de propagandas respectivamente e para realizar a classificação dos conceitos presentes nas propagandas e nos requisitos de compras com o objetivo de encontrar algum grau de relacionamento semântico entre eles.

O RACER funciona como um servidor de raciocínio. Para utilizá-lo basta instanciar o RACER Server em um *host*. Uma vez instanciado, o RACER pode ser acessado por aplicações clientes via uma interface baseada em TCP *Sockets* ou

[7] Um tableux é uma sequência de fórmulas construídas a partir de regras e apresentadas em forma de árvore.

http. A figura 20 mostra o RACER Server informando que está *"escutando"* o protocolo http na porta 8080 e TCP na porta 8088. As portas podem ser configuradas a critério do usuário.

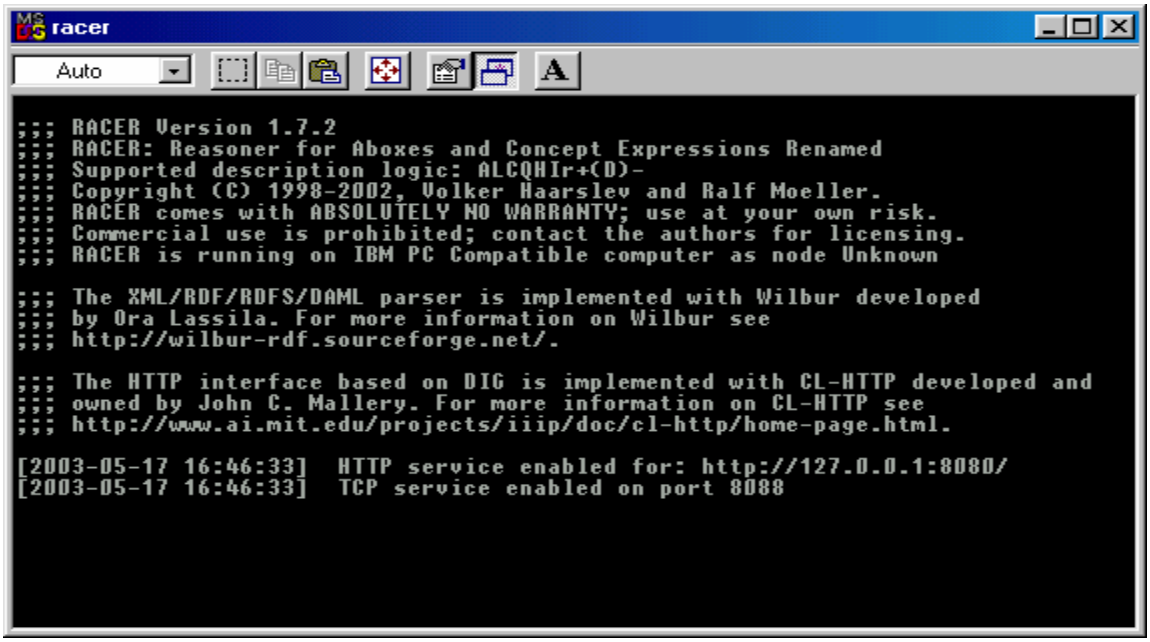

Figura 20: Servidor RACER.

Para ilustrar a utilização do RACER fazemos uso de uma interface gráfica denominada RICE *(RACER Interactive Client Enviroment*, desenvolvida por *Ronald Cornet* da Academic Medical Center em Amsterdam.

Note que ao abrir a base de conhecimento *SalesVehicle*, o RACER respondeu que todos os conceitos presentes nesta ontologia são coerentes (Satisfatíveis). O RICE permite enviar consultas ao RACER através do botão *Submit*. Na figura 21 preparamos uma consulta que pergunta se o conceito *"SEDAM"* está contido (subsumes) no conceito *"CAR"*. A sintaxe deste comando é: *"concept-subsumes c1 c2"*, onde c1 é a superclasse e c2 a subclasse. A resposta fornecida para esta consulta pelo RACER, como pode ser inferida da árvore da ontologia *SalesVehicle*, será *"TRUE"*.

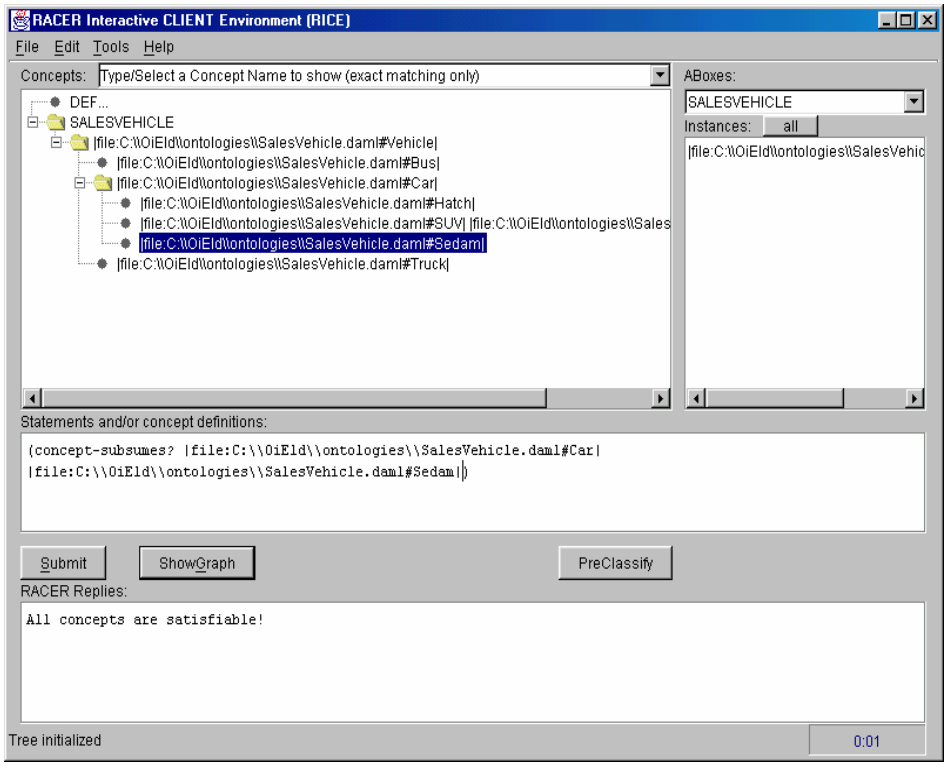

Figura 21: Interface RICE.

5.9.3 TAMINO

O Banco de Dados de Propagandas armazena as anotações das propagandas em DAML-S. Se optássemos pela utilização de um banco de dados convencional, por exemplo, um SGBDR *(Sistema Gerenciador de Bases de Dados Relacionais)*, o *Semantic Matchmaking Engine* ficaria limitado a fazer o emparceiramento dos agentes negociantes percorrendo seqüencialmente o banco de dados e comparando cada uma das propagandas armazenadas com cada requisito. Ao invés disto, optamos pelo TAMINO, um Banco de Dados semi-estruturado que segue o padrão XML desenvolvido pela SoftwareAG. Deste modo, podemos armazenar as propagandas em DAML-S diretamente no banco de dados e gerar visões não materializadas das regiões, utilizando linguagens de consulta padrão XML como *XQuery* e, deste modo, reduzir consideravelmente o espaço de busca.

Como o ICS é extensível a múltiplos domínios de aplicações e o banco de propagandas é comum a todos os domínios, quando um agente comprador realiza a tarefa de encontrar parceiros, o ideal é restringir dentro do banco de dados o espaço de busca do agente comprador para que a busca seja mais eficiente. Uma alternativa é proporcionar uma visão não materializada da parte do banco de dados de propagandas que interessa ao agente comprador.

Para ilustrar como é possível gerar uma visão de uma base de dados XML, fazemos uso de um protótipo da linguagem *XQuery* implementado pela SoftwareAG chamado QUIP. O QUIP possui uma interface gráfica que permite a realização de consultas sobre o banco de dados XML Tamino ou sobre um arquivo XML armazenado no *"file system"* do sistema operacional. Para permitir uma visão mais objetiva das considerações feitas acima mostramos na figura 22 um exemplo de consulta *XQuery*. A idéia é restringir o espaço de busca dos agentes. Aplicamos uma seleção e uma projeção[8] sobre um arquivo XML que armazena dados sobre livros (título, autor, editora, ano publicação e preço) de modo a gerar um outro arquivo XML contendo apenas o ano de publicação e o título dos livros publicados pela editora *"Addison-Wesley"* com ano de publicação superior a 1991.

A GUI *(Graphical User Interface)* do QUIP possui duas grandes caixas de texto. Na caixa de texto superior inserimos o consulta *XQuery*. Na caixa de texto inferior é apresentado o resultado da consulta submetida.

Como é possível observar, o arquivo XML resultante possui apenas dois livros, ou seja, é um fragmento do arquivo original.

[8] Seleção e Projeção são operações da Álgebra Relacional implementadas pela maioria das linguagens de consulta a base de dados relacionais.

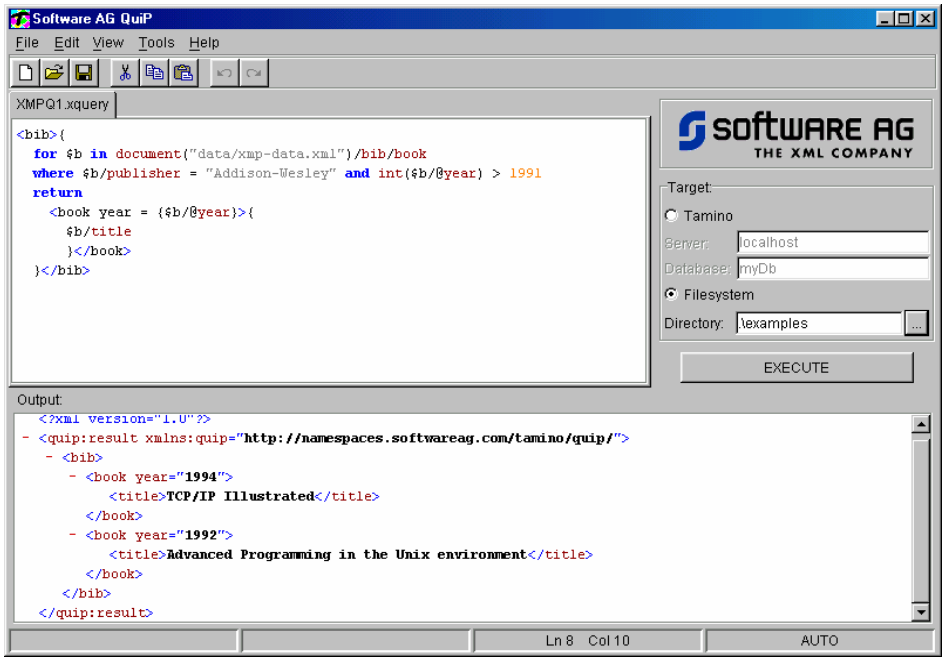

Figura 22: Consulta XQuery realizada a partir da GUI QUIP.

Como as propagandas armazenadas estão anotadas em notação XML é possível realizar-se operação semelhante à demonstrada no exemplo anterior com a finalidade de reduzir o espaço de busca dos agentes compradores.

Para maiores informações sobre *XQuery* acesse (URL 10) e para maiores informações sobre o Tamino Server consulte (URL 11).

5.9.4 Oiled

Uma ontologia é construída por um grupo de especialistas no domínio da aplicação a que ela se destina. Normalmente emprega-se editores de ontologias para facilitar esta tarefa.

Para exemplificar a utilização do ICS desenvolvemos uma ontologia do domínio de comercialização de veículos automotivos. Utilizamos o Oiled (URL 12), um editor de ontologias desenvolvido na Universidade de Manchester, para criar a ontologia *AutoSales* empregada para ilustrar o funcionamento do protótipo.

O Oiled, a partir de sua versão 3.5, permite comunicação com *reasoners DL*, como RACER e FACT (URL 10), via padrão DIG de comunicação. Esta facilidade oferecida pelo Oiled permite-nos criar graficamente TBoxes (ou ontologias) e ABoxes e checar imediatamente a satisfatibilidade dos conceitos definidos. Outra característica importante é que Oiled permite exportar a ontologia criada diretamente para uma marcação XML seguindo o padrão DIG.

É possível também, a partir do Oiled, visualizar graficamente a hierarquia de classes da ontologia definida, o que permite um entendimento mais natural dos conceitos e seus relacionamentos dentro da ontologia.

5.10 Estudo de Caso

Com o objetivo de deixar mais claro o funcionamento da arquitetura proposta, apresentamos um exemplo prático, porém, de complexidade reduzida do funcionamento de nosso *Matchmaker*.

Para fazer uso do *ICS-Matchmaker*, inicialmente é necessário que haja pelo menos uma ontologia de domínio armazenada no banco de ontologias. Como dito anteriormente, criamos uma ontologia DAML-OIL bastante simplificada de um domínio de comercialização de veículos, denominada *AutoSales*, com a finalidade de demonstrar de forma empírica as funcionalidades do Agente *Matchmaker*. A seguir apresentamos a ontologia *AutoSales* em notação DIG.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tells xmlns="http://dl.kr.org/dig/lang"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://dl.kr.org/dig/lang dig.xsd">
   <clearKB/>
   <defconcept name="Hatch"/>
   <impliesc>
      <catom name="Hatch"/>
```

```
    <and>
      <catom name="Car"/>
    </and>
  </impliesc>
  <defconcept name="Sedam"/>
  <impliesc>
    <catom name="Sedam"/>
    <and>
      <catom name="Car"/>
    </and>
  </impliesc>
  <defconcept name="Truck"/>
  <impliesc>
    <catom name="Truck"/>
    <and>
      <catom name="Automobile"/>
      <catom name="Automotivo"/>
      <catom name="Automovel"/>
    </and>
  </impliesc>
  <defconcept name="Automotivo"/>
  <equalc>
    <catom name="Automotivo"/>
    <and>
      <catom name="Automovel"/>
    </and>
  </equalc>
  <defconcept name="Automobile"/>
  <defconcept name="Bus"/>
  <impliesc>
    <catom name="Bus"/>
    <and>
      <catom name="Automobile"/>
      <catom name="Automotivo"/>
      <catom name="Automovel"/>
    </and>
  </impliesc>
  <defconcept name="Car"/>
  <impliesc>
    <catom name="Car"/>
    <and>
      <catom name="Automobile"/>
      <catom name="Automotivo"/>
      <catom name="Automovel"/>
    </and>
  </impliesc>
  <defconcept name="SUV"/>
  <equalc>
    <catom name="SUV"/>
    <and>
      <catom name="Utility"/>
      <catom name="Car"/>
    </and>
  </equalc>
  <defconcept name="Utility"/>
  <impliesc>
```

```
      <catom name="Utility"/>
      <and>
        <catom name="Car"/>
      </and>
    </impliesc>
    <defconcept name="Automovel"/>
    <equalc>
      <catom name="Automovel"/>
      <and>
        <catom name="Automobile"/>
      </and>
    </equalc>
    <defindividual name="Gol"/>
    <instanceof>
      <individual name="Gol"/>
      <and>
        <catom name="Hatch"/>
      </and>
    </instanceof>
    <defindividual name="Blazer"/>
    <instanceof>
      <individual name="Blazer"/>
      <and>
        <catom name="SUV"/>
      </and>
    </instanceof>
    <defindividual name="Vectra"/>
    <instanceof>
      <individual name="Vectra"/>
      <and>
        <catom name="Sedam"/>
      </and>
    </instanceof>
    <impliesc>
      <catom name="SUV"/>
      <or>
        <catom name="Car"/>
      </or>
    </impliesc>
    <disjoint>
      <catom name="SUV"/>
      <catom name="Hatch"/>
    </disjoint>
    <disjoint>
      <catom name="SUV"/>
      <catom name="Sedam"/>
    </disjoint>
    <disjoint>
      <catom name="Hatch"/>
      <catom name="Sedam"/>
    </disjoint>
  </tells>
```

Para permitir uma melhor compreensão da ontologia descrita anteriormente apresentamos a hierarquia de conceitos (taxonomia) criada no editor de ontologias Oiled.

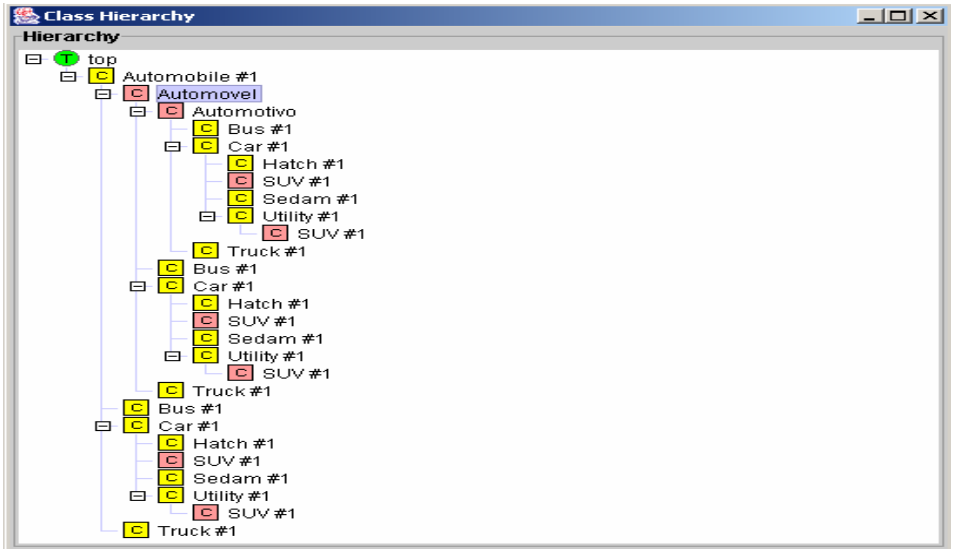

Figura 23: Hierarquia de Classes da Ontologia AutoSales.

Digamos que uma empresa, que opere no ramo de comercialização de veículos, deseje cadastrar um agente para representá-la dentro do ICS. Um funcionário da empresa deve acessar a interface do Agente *Matchmaker*, chamada de *ICS-Match*, a partir da Web. A Interface *ICS-Match* é mostrada na figura 24:

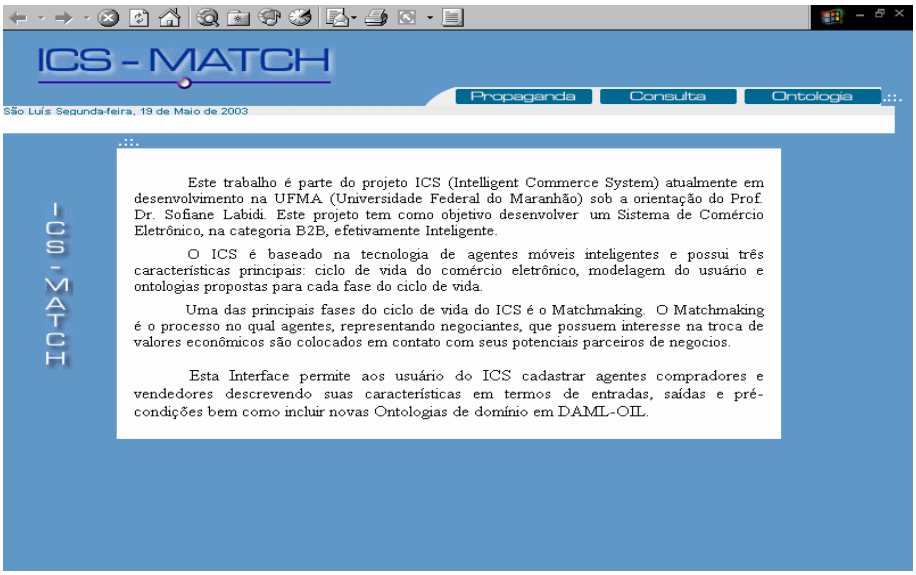

Figura 24: Formulário Inicial do ICS-Match.

Na área central desta janela há uma descrição resumida do projeto ICS e suas funcionalidades. Na parte superior direita o menu de opções apresenta três possibilidades: Propaganda, Consulta e Ontologias.

- **Propaganda:** esta opção conduz o usuário ao formulário de cadastramento de agentes vendedores. Neste formulário o usuário deve cadastrar o nome do agente, o nome ou o endereço IP do *host* onde o agente será instanciado, a porta em que o agente ficará *"escutando"* para comunicação com outros agentes, uma descrição "*human-readable"* das capacidades do agente, o domínio de negócio em que ele opera (Ontologia de domínio), a interface do agente em termos de entradas, saídas e pré-condições e finalmente seu nome e e-mail para o caso de alguma notificação como mostra a figura 25.

- **Consulta:** leva o usuário a um formulário similar ao apresentado na figura 25 que permite o cadastro de agentes compradores do mesmo modo que o cadastro de agentes vendedores.

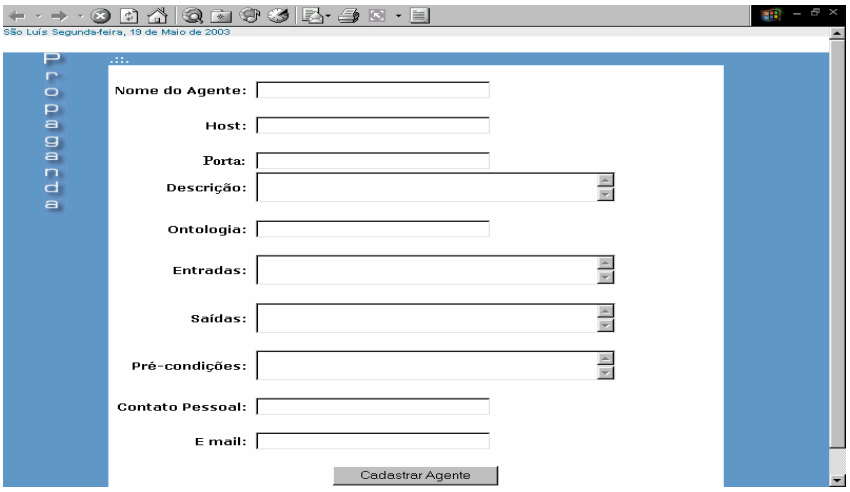

Figura 25: Formulário de Cadastro de Agentes Vendedores.

- **Ontologias:** conduz o usuário a um formulário de cadastro de ontologias que permite acrescentar domínios de negócios onde o ICS passa a poder operar. A figura 26 mostra o formulário de cadastro de ontologias.

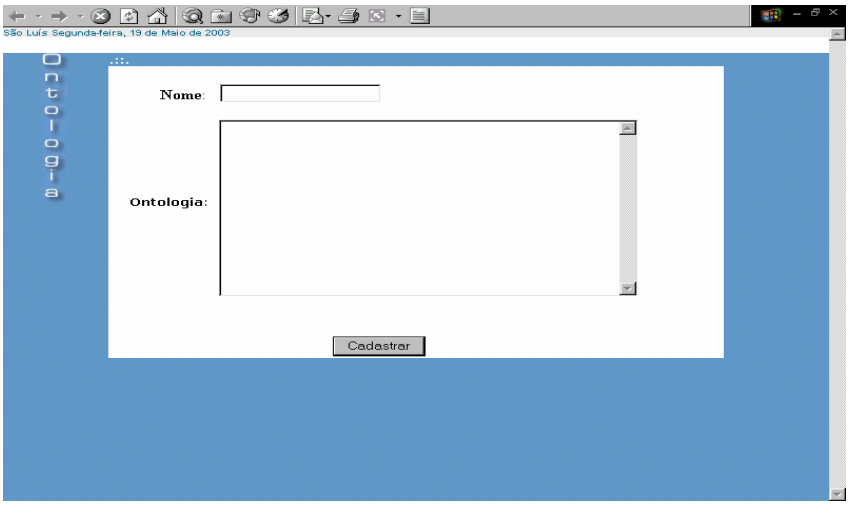

Figura 26: Formulário de Cadastro de Ontologias.

Neste formulário o usuário necessita apenas dar um nome à ontologia e em seguida digitá-la ou copiá-la de um arquivo gerado por um editor de ontologias no formato DIG.

Note que para cadastrar um agente negociante (comprador ou vendedor) no ICS é necessário especificar o nome da ontologia em que ele opera. Esta informação é imprescindível para a formação das Regiões e é o que define o contexto para cada agente negociante dentro do ICS.

Uma vez cadastrados os agentes negociantes, o Agente *Matchmaker* inicia a formação dos clusters. Em seguida, o *Matchmaker* convida os agentes contidos nos clusters para deslocarem-se para um *host* específico onde serão instanciados os Mercados Virtuais.

Suponhamos também que um par de usuários tenha escolhido a ontologia *AutoSales* para cadastrar um agente comprador *(AC)* e um agente vendedor *(AV)*, respectivamente, e que o usuário da empresa vendedora tenha cadastrado a seguinte propaganda no formato DAML-S:

```
<profile:Profile rdf:ID=”GM”>
   <profile:NomeServico >GM</profile:NomeServico>
   <profile:ContatoPessoal>Dr. Gutemberg Farias <profile:ContatoPessoal>
   <profile:EMAIL>Gutemberg@gm.com.br</profile:EMAIL>
   <input>
      <profile:ParameterDescription rdf: ID=”Preco_Input”>
         <profile:ParameterName>Preco</profile:ParameterName>
      </profile:ParameterDescription>
    </input>
   <output>
      <profile:ParameterDescriptio rdf:ID=”Automobile_Output”>
          <profile:parameterName>Automobile</profile:parameterName>
      </profile:ParameterDescription>
   <output>
</profile:Profile>
```

Figura 27: Propaganda no formato DAML-S.

Como é possível observar, para simplificar o nosso exemplo, especificamos o serviço apenas em termos de suas entradas *(Inputs)* e saídas *(Outputs)*. A única entrada descrita na propaganda *(InA)* apresentada na figura 27 é *“Preço”*. Do mesmo modo a única saída descrita na propaganda *(OutA)* é *“Automobile”.*

Considere ainda, que o usuário da empresa compradora tenha especificado o seguinte requisito de compra, também anotado em DAML-S:

```
<profile:Profile rdf:ID=”DALCAR”>
   <profile:NomeServico >DALCAR</profile:NomeServico>
   <profile:ContatoPessoal>Alessandro Martins<profile:ContatoPessoal>
   <profile:EMAIL>martins@dalcar.com.br</profile:EMAIL>
   <input>
      <profile:ParameterDescription rdf: ID=”Preco_Input”>
         <profile:ParameterName>Preco</profile:ParameterName>
      </profile:ParameterDescription>
   </input>
   <output>
      <profile:ParameterDescriptio rdf:ID=”Car_Output”>
          <profile:parameterName>Car</profile:parameterName>
      </profile:ParameterDescription>
   <output>
</profile:Profile>
```

Figura 28: Requisito de compra anotado em DAML-S.

A exemplo da propaganda, o requisito de compra descrito na figura 28 também foi especificado apenas em termos de suas entradas e saídas. As entradas do requisito *(InR)* e as saídas do requisito *(OutR)* possuem apenas um parâmetro cada uma: *"Preco"* e *"Car"*.

O *Semântic Matching Engine (SME)* faz inicialmente a comparação das saídas do requisito de compra *(OutR)* com as saídas da propaganda *(OutA)*, e somente em caso de não identificar nenhuma proximidade semântica entre os conceitos descritos em *OutR* e *OutA* passará à comparação dos conceitos presentes em *InR* e *InA*. Para definir o grau de combinação entre os conceitos presentes nas saídas (*OutR*, *OutA*) o SME utiliza o seguinte algoritmo:

```
DegreeOfMatch(outR,outA)
   If outA=outR then return exact
   If outA subsumes outR then return plugin
   If outR subsumes outA then return subsumes
Otherwise  fail
```

Para fazer as comparações entre os conceitos o *SME* envia algumas consultas ao *reasoner* RACER. Estas consultas são respondidas pelo RACER com base em uma ontologia de domínio previamente carregada, ou seja, o RACER vai inferir as respostas buscando alguma relação semântica entre os conceitos presentes em *OutR* e os conceitos presentes em *OutA* com base na taxonomia da ontologia de domínio.

Para ilustrar, apresentamos na figura 29 algumas consultas no formato DIG que poderiam ser enviadas ao RACER pelo SME para resolver o grau de similaridade entre os conceitos presentes em *OutR* e *OutA*.

```
<?xml version="1.0"?>
<asks
  xmlns="http://dl.kr.org/dig/lang">
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/lang"
  http://potato.cs.man.ac.uk/dig/level0/ask.xsd" >
//Consulta 1
  <equivalents id="q1">
    <catom name="Automobile"/>
  </equivalents>
//Consulta 2
  <subsumes id="q2 ">
    <catom name="Automobile"/>
    <catom name="Car"/>
  </subsumes>
//Consulta 3
  <subsumes id="q3">
    <catom name="Car"/>
    <catom name="Automobile"/>
  </subsumes>
```

Figura 29: Exemplo de consultas enviadas ao RACER.

Como é possível observar uma mensagem DIG pode conter em seu corpo várias pedidos. Existem dois tipos de mensagens no padrão DIG: *Tells* e *Asks*.

Mensagens do tipo *Tell* permitem a definição da base de conhecimento. Mensagens do tipo *Ask* permitem inferências sobre a base.

Cada sentença contida no corpo de uma mensagem *Ask* possui um atributo identificado *(Id)* que identifica cada consulta e suas respectivas respostas.

A primeira consulta, identificada pelo *Id*=“q1”, solicita ao RACER todos os termos Equivalentes a *“Automobile” (OutA).* Baseado na ontologia *AutoSales,* apresentada anteriormente, o RACER fornece a seguinte resposta:

```
<conceptSet id="q1">
<synonyms> <catom name="Automotivo"/> <catom name="Automovel"/>
<catom name="Automobile"/> </synonyms>
</conceptSet>
```

Figura 30: Resposta à Consulta 1.

O RACER informa que os sinônimos de “*Automobile”* são: “*Automotivo”*, “*Automóvel”* e o próprio “*Automobile”*. Logo, a primeira condição do algoritmo *“If outA=outR then”* não é satisfeita, uma vez que *“Car”* não aparece na relação de termos equivalentes à *“Automobile”*.

A consulta 2, que possui *id*=”q2”, pergunta ao RACER se *“Automobile”* (OutA) subsumes “*Car”* (OutR). O RACER fornece a seguinte resposta:

```
<true id="q2"/>
```

Figura 31: Resposta à Consulta 2.

O RACER devolve uma reposta afirmativa. Logo, a segunda condição do algoritmo *“If outA subsumes outR then”* é satisfeita. Neste caso, os agentes são emparceirados com grau de combinação “*Plugin”.*

A terceira consulta não seria mais necessária. Entretanto, caso SME enviasse a consulta 3, cujo *id*=”q3”, para perguntar ao RACER se *“Car”* (OutR) subsumes *“Automobile”* (OutA). O RACER forneceria a seguinte resposta:

```
<false id="q3"/>
```

Figura 32: Resposta à Consulta 3.

Deste modo, a terceira condição do algoritmo *“If outR subsumes outA then”,* a exemplo da primeira*,* não é satisfeita.

Estas são apenas algumas das perguntas possíveis de serem realizadas pelo SME ao RACER. Com o objetivo de simplificar a explanação apresentamos as inferências feitas somente sobre a TBox. Na realidade, o SME ainda envia outras consultas sobre a ABox como testes de instanciação, por exemplo, que visa testar se um conceito é ou não uma instância de uma classe.

Importante salientar que *ICS-Match* é um Web Service, logo pode ser acessado através da evocação remota de seus métodos através da infra-estrutura da Internet. Deste modo, aplicações legadas de cadeia de suprimento *(Supply Chain)* podem fazer uso do *ICS-Match* para automatizar o processo de descoberta de parceiros de negócio.

5.11 Conclusão

Os padrões e ferramentas utilizados neste trabalho encontram-se ainda em fase inicial de desenvolvimento e há muito por ser feito. Isto dificultou bastante a implementação do protótipo. No fechamento desta dissertação, o protótipo demonstrou a viabilidade de tornar-se um produto.

Realizamos diversos testes de funcionalidade e performance do *ICS-Match* que demonstrou funcionamento adequado e tempo de resposta satisfatório. O *ICS-Match* não apresentou erros de classificação e inferências sobre as bases de conhecimento.

Os testes foram realizados fazendo-se a definição das ontologias e posteriormente realizando-se inferências sobre elas. Iniciamos com a definição e posterior inferência sobre uma ontologia com 500 regras e ampliamos este número até10.000 regras. A tabela abaixo demonstra o tempo de resposta médio, em segundos, de acordo com o número de sentenças submetidas. Como o objetivo do teste é avaliar a performance e não a riqueza de inferências possíveis, submetemos apenas dois tipos de sentenças: definição de conceitos *(<defconcept name="conceptname"/>*) e teste de satisfatibilidade de conceitos *(<satisfiable id="q1"><catom name="conceptname"/></satisfiable>*).

Nº Sentencas	Definição KB	Satisfatibilidade
500	0,05s	0,02s
1000	0,07s	0,05
5000	0,1s	0,17s
10000	0,16s	0,3s

Tabela 10: Resultado testes ICS-Match.

Os números demonstram a boa performance do protótipo. Entretanto, o padrão DIG ainda está em desenvolvimento e algumas características do RACER não foram padronizadas na especificação da linguagem DIG. Isto impõe restrições de uso do RACER via interface DIG. Com o final da especificação DIG esperamos sua completa implementação pelos desenvolvedores do RACER e do FACT os dois principais *reasoners DL.*

O padrão DAML-S é muito recente e ainda está em desenvolvimento. No momento da implementação de nosso protótipo não existia nenhum *parser* DAML-S implementado, isto nos levou a desenvolver o nosso próprio *parser* que se encontra inacabado e necessita ainda de alguns testes e melhorias.

A persistência das ontologias e das propagandas no banco de dados Tamino está em fase final de implementação. Entretanto, na versão atual do *Matchmaker* é feita uma varredura completa no repositório de propagandas em busca dos potenciais parceiros de negócio. Estamos trabalhando em um processador de consultas *XQurery* para permitir a recuperação apenas das propagandas que estejam no mesmo contexto do requisito de compra. Isto reduzirá o espaço de busca e dará melhor performance ao *Matchmaker*.

V. CONCLUSÃO E TRABALHOS FUTUROS

Nesta dissertação apresentamos uma aplicação prática dos conceitos padrões e ferramentas decorrentes do desenvolvimento da Web Semântica e da tecnologia de Web Services como uma alternativa para melhorar os atuais mecanismos de comercialização eletrônica entre empresas.

A perspectiva da Web Semântica aliada às arquiteturas baseadas em serviços possibilitam uma grande evolução das ferramentas de comércio eletrônico, em especial do B2B. O uso de agentes de software representando negociantes (compradores e vendedores) com autonomia para agir em nome de seus representados agiliza e reduz os custos do processo de negociação entre as empresas, uma vez que a intervenção humana é bastante reduzida.

A utilização da tecnologia de agentes móveis para implementação dos agentes negociantes é muito adequada, pois permite um processo de negociação mais robusto e sofisticado, já que os agentes se deslocam para um local comum *(Virtual MarketPlace)* e interagem localmente. Além disso, o sistema ganha maior tolerância à falhas, podendo funcionar em condições de rede onde as conexões são pouco confiáveis como a Internet.

Este trabalho nos proporcionou uma experimentação das teorias e padrões em desenvolvimento em duas importantes frentes de pesquisa: *Web Semântica* e *Web Services*, ambas fundamentais para a construção da próxima geração de ferramentas para a Web e foi objeto de três publicações internacionais:

I. R. F. Tomaz and S. Labidi, ***"A Semantic Web Approach for Matching Traders in B2B Systems".*** 3rd. IEEE/IFIP Conference on eCommerce, eBusiness and eGovernment, São Paulo, Brazil, Sep 21-24, 2003.

II. R. F. Tomaz and S. Labidi, ***"Increasing Matchmaking Semantics in Intelligent Commerce System".*** IEEE/WIC International conference on Web Intelligence, Halifax, Canada, Oct 13-17, 2003.

III. R. F, Tomaz, S. Labidi and B. Wanghon, ***"A Semantic Matching Method for Clustering Traders in B2B Systems"***. IEEE/LA-Web First Latin American Web Congress, Santiago, Chile, Nov 10-12, 2003.

A abordagem de construção do Matchmaker como um Web Service é inovadora. Até o fechamento deste trabalho não havia outro serviço de *Matching* com esta característica. A abordagem baseada em Web Services possibilita o acoplamento a aplicações legadas. Deste modo, o nosso Matchmaker pode servir como uma plataforma integradora de aplicações B2B legadas.

Como proposta para trabalhos futuros, sugerimos o desenvolvimento de um *framework*, implementado em forma de uma API, que possibilite a aproximação de agentes independentemente do domínio da aplicação. O problema de descoberta e de aproximação de agentes em ambientes distribuídos é um problema recorrente, surge em muitos outros domínios como *e-learning* (formação de grupos de estudantes) e assitentes pessoais (planejamento de reuniões). O desenvolvimento de uma API reduziria a complexidade e o custo do desenvolvimento de aplicações que fazem uso da tecnologia de agentes em ambientes distribuídos.

Propomos também, o desenvolvimento de ferramentas que possibilitem aos usuários finais especificarem os serviços que eles estão disponibilizando sem a necessidade de conhecimento técnico prévio sobre as linguagens empregadas para descrever os serviços *(DAML-OIL,DAML-S,LARKS, DIG, etc)*.

Propomos ainda, a extensão das ferramentas de *Description Logics* com uso de lógicas não clássicas: não-monotônica – para permitir o raciocício sobre informações incompletas; paraconsistente – para permitir o raciocínio sobre informações contraditórias (Martins, 2002) e *fuzzy* – para converter os graus de matching em um formato numérico e permitir comparações mais precisas (Steven, 1998).

As informações disponíveis na Web são por natureza semi-estruturadas, incompletas e, em grande parte, contraditórias. O uso de raciocínio não-monotônico aliado à lógica paraconsistente pode ajudar a superar estas dificuldades.

A utilização da lógica *fuzzy* pode ajudar a resolver questões como: um cliente deseja comprar um sapato preto tamanho 38. O vendedor “X” oferece um sapato marrom tamanho 38 e um segundo vendedor “Y” oferece um sapato preto de tamanho 32. Qual dos vendedores possui oferta mais aproximada do requisito de compra do cliente?

A utilização das lógicas não clássicas pode enriquecer sobremaneira o processo de *matching*, permitindo comparações mais refinadas.

Concluímos este trabalho com a expectativa de ter contribuído para melhor entendimento e divulgação dos assuntos tratados. Somos conscientes do desafio que ainda temos pela frente. Entretanto, o desenvolvimento do protótipo e a análise dos resultados, até agora obtidos, nos encorajam a aprofundar nossos estudos nesta desafiadora linha de pesquisa. Esperamos também ter despertado o interesse por essa linha de pesquisa naqueles que compartilham conosco o desejo de uma Web mais Inteligente e amigável.

ANEXO I - ESPECIFICAÇÃO XML SCHEMA DA LINGUAGEM DIG.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This schema describes the DIG Language. There are a number of parts to
this.
1) Concept Expression Language
2) Tell format
3) Ask format
4) Response format
These were originally held in separate schemas. These have now been
combined into one single schema. -->
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Karon Mee (The
University of Manchester) -->
<!--W3C Schema generated by XML Spy v4.3 U (http://www.xmlspy.com)-->
<xs:schema targetNamespace="http://dl.kr.org/dig/lang"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://dl.kr.org/dig/lang" elementFormDefault="qualified"
attributeFormDefault="unqualified">
      <xs:annotation>
            <xs:documentation>Basic concept types</xs:documentation>
      </xs:annotation>
      <xs:element name="bottom">
            <xs:annotation>
                  <xs:documentation>Bottom Concept</xs:documentation>
            </xs:annotation>
            <xs:complexType/>
      </xs:element>
      <xs:complexType name="chainType">
            <xs:annotation>
                  <xs:documentation>Feature Chain</xs:documentation>
            </xs:annotation>
            <xs:sequence minOccurs="0">
                  <xs:element name="feature" type="atomType"
maxOccurs="unbounded"/>
                  <xs:element name="attribute" type="atomType"/>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string"/>
      </xs:complexType>
      <xs:complexType name="attType">
            <xs:annotation>
                  <xs:documentation>Max (concrete
domain)</xs:documentation>
            </xs:annotation>
            <xs:choice>
                  <xs:element name="attribute" type="atomType"/>
                  <xs:element name="chain" type="chainType"/>
            </xs:choice>
      </xs:complexType>
      <xs:element name="object">
            <xs:annotation>
                  <xs:documentation>Concrete Domain
Object</xs:documentation>
            </xs:annotation>
            <xs:complexType>
```

```
                  <xs:attribute name="name" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="top">
            <xs:annotation>
                  <xs:documentation>Top Concept</xs:documentation>
            </xs:annotation>
            <xs:complexType/>
      </xs:element>
      <xs:group name="conceptGroup">
            <xs:annotation>
                  <xs:documentation>The various possible concept
descriptions</xs:documentation>
            </xs:annotation>
            <xs:choice>
                  <xs:element name="defined" type="attType">
                        <xs:annotation>
                              <xs:documentation>The class of things for
which the given attribute is defined.</xs:documentation>
                        </xs:annotation>
                  </xs:element>
                  <xs:element name="stringmin">
                        <xs:annotation>
                              <xs:documentation>Min value on a string valued
attribute</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                              <xs:complexContent>
                                    <xs:extension base="attType">
                                          <xs:attribute name="val"
type="xs:string" use="required"/>
                                    </xs:extension>
                              </xs:complexContent>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="stringmax">
                        <xs:annotation>
                              <xs:documentation>Max value on a string valued
attribute</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                              <xs:complexContent>
                                    <xs:extension base="attType">
                                          <xs:attribute name="val"
type="xs:string" use="required"/>
                                    </xs:extension>
                              </xs:complexContent>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="stringequals">
                        <xs:annotation>
                              <xs:documentation>Exact string
value</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                              <xs:complexContent>
                                    <xs:extension base="attType">
                                          <xs:attribute name="val"
type="xs:string" use="required"/>
```

```
                                    </xs:extension>
                              </xs:complexContent>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="stringrange">
                        <xs:annotation>
                              <xs:documentation>String
range</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                              <xs:complexContent>
                                    <xs:extension base="attType">
                                          <xs:attribute name="min"
type="xs:string" use="required"/>
                                          <xs:attribute name="max"
type="xs:string" use="required"/>
                                    </xs:extension>
                              </xs:complexContent>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="intmin">
                        <xs:annotation>
                              <xs:documentation>minimum integer
value</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                              <xs:complexContent>
                                    <xs:extension base="attType">
                                          <xs:attribute name="min"
type="xs:integer" use="required"/>
                                    </xs:extension>
                              </xs:complexContent>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="intmax">
                        <xs:annotation>
                              <xs:documentation>maximum integer
value</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                              <xs:complexContent>
                                    <xs:extension base="attType">
                                          <xs:attribute name="max"
type="xs:integer" use="required"/>
                                    </xs:extension>
                              </xs:complexContent>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="intequals">
                        <xs:complexType>
                              <xs:complexContent>
                                    <xs:extension base="attType">
                                          <xs:attribute name="val"
type="xs:integer" use="required"/>
                                    </xs:extension>
                              </xs:complexContent>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="intrange">
                        <xs:complexType>
```

```
                              <xs:complexContent>
                                    <xs:extension base="attType">
                                          <xs:attribute name="min"
type="xs:integer" use="required"/>
                                          <xs:attribute name="max"
type="xs:integer" use="required"/>
                                    </xs:extension>
                              </xs:complexContent>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="catom" type="atomType"/>
                  <xs:element ref="top"/>
                  <xs:element ref="bottom"/>
                  <xs:element name="and">
                        <xs:complexType>
                              <xs:group ref="conceptSequenceGroup"/>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="or">
                        <xs:complexType>
                              <xs:group ref="conceptSequenceGroup"/>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="not">
                        <xs:complexType>
                              <xs:group ref="conceptGroup"/>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="some">
                        <xs:complexType>
                              <xs:group ref="roleConceptGroup"/>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="all">
                        <xs:complexType>
                              <xs:group ref="roleConceptGroup"/>

                        </xs:complexType>
                  </xs:element>
                  <xs:element name="atmost">
                        <xs:complexType>
                              <xs:group ref="roleConceptGroup"/>
                              <xs:attribute name="num" type="xs:integer"
use="required"/>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="atleast">
                        <xs:complexType>
                              <xs:group ref="roleConceptGroup"/>
                              <xs:attribute name="num" type="xs:integer"
use="required"/>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="iset">
                        <xs:complexType>
                              <xs:sequence maxOccurs="unbounded">
                                    <xs:element name="individual"
type="atomType"/>
                              </xs:sequence>
                        </xs:complexType>
```

```
                  </xs:element>
            </xs:choice>
      </xs:group>
      <xs:group name="roleGroup">
            <xs:annotation>
                  <xs:documentation>The various possible role
description</xs:documentation>
            </xs:annotation>
            <xs:choice>
                  <xs:element name="ratom" type="atomType"/>
                  <xs:element name="feature" type="atomType"/>
                  <xs:element name="inverse">
                        <xs:complexType>
                              <xs:group ref="roleGroup"/>
                        </xs:complexType>
                  </xs:element>
            </xs:choice>
      </xs:group>
      <xs:group name="conceptPairGroup">
            <xs:annotation>
                  <xs:documentation>A pair of concept
descriptions</xs:documentation>
            </xs:annotation>
            <xs:sequence minOccurs="2" maxOccurs="2">
                  <xs:group ref="conceptGroup"/>
            </xs:sequence>
      </xs:group>
      <xs:group name="rolePairGroup">
            <xs:annotation>
                  <xs:documentation>A pair of role
descriptions</xs:documentation>
            </xs:annotation>
            <xs:sequence minOccurs="2" maxOccurs="2">
                  <xs:group ref="roleGroup"/>
            </xs:sequence>
      </xs:group>
      <xs:group name="conceptSequenceGroup">
            <xs:annotation>
                  <xs:documentation>A sequence of concept
descriptions</xs:documentation>
            </xs:annotation>
            <xs:sequence maxOccurs="unbounded">
                  <xs:group ref="conceptGroup"/>
            </xs:sequence>
      </xs:group>
      <xs:group name="roleConceptGroup">
            <xs:annotation>
                  <xs:documentation>A pair of a role and
concept</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                  <xs:group ref="roleGroup"/>
                  <xs:group ref="conceptGroup"/>
            </xs:sequence>
      </xs:group>
      <xs:complexType name="atomType">
            <xs:annotation>
                  <xs:documentation>Basic type with a single name
attribute</xs:documentation>
            </xs:annotation>
```

```
            <xs:attribute name="name" type="xs:string" use="required"/>
      </xs:complexType>
      <xs:attributeGroup name="ids">
            <xs:attribute name="id" type="xs:ID" use="required"/>
      </xs:attributeGroup>
      <!--
*************************************************************************
***** -->
      <xs:annotation>
            <xs:documentation>Identification of server</xs:documentation>
      </xs:annotation>
      <xs:element name="identifier">
            <xs:annotation>
                  <xs:documentation>Identification of a server. The server
must provide its name, version, and a description of the language that it
supports.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="supports" type="supportsType"/>
                  </xs:sequence>
                  <xs:attribute name="name" type="xs:string"
use="required"/>
                  <xs:attribute name="version" use="required">
                        <xs:annotation>
                              <xs:documentation>Version should be a sequence
of integers, separated by the "." character.</xs:documentation>
                        </xs:annotation>
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:pattern value="[0-9](\.?[0-9]+)*"/>
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:attribute>
                  <xs:attribute name="message" type="xs:string"
use="optional"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="getIdentifier">
            <xs:annotation>
                  <xs:documentation>Request Server Identification.
</xs:documentation>
            </xs:annotation>
      </xs:element>
      <!--
*************************************************************************
***** -->
      <xs:annotation>
            <xs:documentation>Tells</xs:documentation>
      </xs:annotation>
      <xs:complexType name="defType">
            <xs:annotation>
                  <xs:documentation>Definition of an atomic
object</xs:documentation>
            </xs:annotation>
            <xs:attribute name="name" type="xs:string" use="required"/>
      </xs:complexType>
      <xs:element name="tells">
            <xs:annotation>
```

```
                  <xs:documentation>Wrapper round multiple
tells</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:choice>
                              <xs:element name="defconcept" type="defType">
                                    <xs:annotation>
                                          <xs:documentation>Introduces a new
concept</xs:documentation>
                                    </xs:annotation>
                              </xs:element>
                              <xs:element name="defrole" type="defType">
                                    <xs:annotation>
                                          <xs:documentation>Introduces a new
role</xs:documentation>
                                    </xs:annotation>
                              </xs:element>
                              <xs:element name="deffeature" type="defType">
                                    <xs:annotation>
                                          <xs:documentation>Introduces a new
feature</xs:documentation>
                                    </xs:annotation>
                              </xs:element>
                              <xs:element name="defattribute"
type="defType">
                                    <xs:annotation>
                                          <xs:documentation>Introduces an
attribute </xs:documentation>
                                    </xs:annotation>
                              </xs:element>
                              <xs:element name="defindividual"
type="defType">
                                    <xs:annotation>
                                          <xs:documentation>Introduces a new
individual</xs:documentation>
                                    </xs:annotation>
                              </xs:element>
                              <xs:element name="impliesc">
                                    <xs:complexType>
                                          <xs:group ref="conceptPairGroup"/>
                                    </xs:complexType>
                              </xs:element>
                              <xs:element name="impliesr">
                                    <xs:complexType>
                                          <xs:group ref="rolePairGroup"/>
                                    </xs:complexType>
                              </xs:element>
                              <xs:element name="equalc">
                                    <xs:complexType>
                                          <xs:group ref="conceptPairGroup"/>
                                    </xs:complexType>
                              </xs:element>
                              <xs:element name="equalr">
                                    <xs:complexType>
                                          <xs:group ref="rolePairGroup"/>
                                    </xs:complexType>
                              </xs:element>
                              <xs:element name="domain">
                                    <xs:complexType>
```

```
                                    <xs:group ref="roleConceptGroup"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="range">
                              <xs:complexType>
                                    <xs:group ref="roleConceptGroup"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="rangeint">
                              <xs:annotation>
                                    <xs:documentation>Assertion that
an attribute has the integers as its range</xs:documentation>
                              </xs:annotation>
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element name="attribute"
type="atomType"/>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="rangestring">
                              <xs:annotation>
                                    <xs:documentation>Assertion that
an attribute has strings as its range</xs:documentation>
                              </xs:annotation>
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element name="attribute"
type="atomType"/>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="transitive">
                              <xs:complexType>
                                    <xs:group ref="roleGroup"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="functional">
                              <xs:complexType>
                                    <xs:group ref="roleGroup"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="disjoint">
                              <xs:complexType>
                                    <xs:group ref="conceptPairGroup"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element ref="clearKB"/>
                  </xs:choice>
                  <xs:choice>
                        <xs:element name="instanceof">
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element
name="individual" type="atomType"/>
                                          <xs:group
ref="conceptGroup"/>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
```

```
                              <xs:element name="related">
                                    <xs:complexType>
                                          <xs:sequence>
                                                <xs:element
name="individual" type="atomType"/>
                                                <xs:group ref="roleGroup"/>
                                                <xs:element
name="individual" type="atomType"/>
                                          </xs:sequence>
                                    </xs:complexType>
                              </xs:element>
                              <xs:element name="value">
                                    <xs:complexType>
                                          <xs:sequence>
                                                <xs:element
name="individual" type="atomType"/>
                                                <xs:choice>
                                                      <xs:element
name="attribute" type="atomType"/>
                                                      <xs:element
name="chain" type="chainType"/>
                                                </xs:choice>
                                                <xs:choice>
                                                      <xs:element
name="sval" type="xs:string"/>
                                                      <xs:element
name="ival" type="xs:integer"/>
                                                </xs:choice>
                                          </xs:sequence>
                                    </xs:complexType>
                              </xs:element>
                        </xs:choice>
                  </xs:choice>
            </xs:complexType>
      </xs:element>
      <xs:element name="clearKB">
            <xs:annotation>
                  <xs:documentation>Clear the knowledge
base</xs:documentation>
            </xs:annotation>
      </xs:element>
      <xs:element name="response">

            <xs:complexType>
                  <xs:choice>
                        <xs:element name="ok">
                              <xs:annotation>
                                    <xs:documentation>Signifies that the
reasoner received the message.</xs:documentation>
                              </xs:annotation>
                              <xs:complexType>
                                    <xs:sequence minOccurs="0"
maxOccurs="unbounded">
                                          <xs:element name="warning">
                                                <xs:annotation>
                                                      <xs:documentation>A
warning from the reasoner.</xs:documentation>
                                                </xs:annotation>
                                                <xs:complexType>
                                                      <xs:simpleContent>
```

```
                                                                  <xs:extension
base="xs:string">

      <xs:attribute name="message" type="xs:string"/>

      <xs:attribute name="code" type="xs:integer"/>
                                                                  </xs:extension>
                                                            </xs:simpleContent>
                                                      </xs:complexType>
                                                </xs:element>

                                          </xs:sequence>
                                    </xs:complexType>
                              </xs:element>
                              <xs:element name="error">
                                    <xs:complexType>
                                          <xs:simpleContent>
                                                <xs:extension base="xs:string">
                                                      <xs:attribute name="message"
type="xs:string" use="optional"/>
                                                      <xs:attribute name="code"
type="xs:integer" use="optional"/>
                                                </xs:extension>
                                          </xs:simpleContent>
                                    </xs:complexType>
                              </xs:element>
                        </xs:choice>
                  </xs:complexType>
      </xs:element>
      <!--
*************************************************************************
***** -->
      <xs:annotation>
            <xs:documentation>Asks</xs:documentation>
      </xs:annotation>
      <xs:element name="asks">
            <xs:annotation>
                  <xs:documentation>Wrapper around multiple
queries</xs:documentation>

            </xs:annotation>
            <xs:complexType>
                  <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element name="allConceptNames">
                              <xs:complexType>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="allRoleNames">
                              <xs:complexType>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="allIndividuals">
                              <xs:complexType>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="satisfiable">
                              <xs:complexType>
```

```
            <xs:group ref="conceptGroup"/>
            <xs:attributeGroup ref="ids"/>
      </xs:complexType>
</xs:element>
<xs:element name="subsumes">
      <xs:complexType>
            <xs:group ref="conceptPairGroup"/>
            <xs:attributeGroup ref="ids"/>
      </xs:complexType>
</xs:element>
<xs:element name="disjoint">
      <xs:complexType>
            <xs:group ref="conceptPairGroup"/>
            <xs:attributeGroup ref="ids"/>
      </xs:complexType>
</xs:element>
<xs:element name="parents">
      <xs:complexType>
            <xs:group ref="conceptGroup"/>
            <xs:attributeGroup ref="ids"/>
      </xs:complexType>
</xs:element>
<xs:element name="children">
      <xs:complexType>
            <xs:group ref="conceptGroup"/>
            <xs:attributeGroup ref="ids"/>
      </xs:complexType>
</xs:element>
<xs:element name="ancestors">
      <xs:complexType>
            <xs:group ref="conceptGroup"/>
            <xs:attributeGroup ref="ids"/>
      </xs:complexType>
</xs:element>
<xs:element name="descendants">
      <xs:complexType>
            <xs:group ref="conceptGroup"/>
            <xs:attributeGroup ref="ids"/>
      </xs:complexType>
</xs:element>
<xs:element name="equivalents">
      <xs:complexType>
            <xs:group ref="conceptGroup"/>
            <xs:attributeGroup ref="ids"/>
      </xs:complexType>
</xs:element>
<xs:element name="rparents">
      <xs:complexType>
            <xs:group ref="roleGroup"/>
            <xs:attributeGroup ref="ids"/>
      </xs:complexType>
</xs:element>
<xs:element name="rchildren">
      <xs:complexType>
            <xs:group ref="roleGroup"/>
            <xs:attributeGroup ref="ids"/>
      </xs:complexType>
</xs:element>
<xs:element name="rancestors">
      <xs:complexType>
```

```
                                    <xs:group ref="roleGroup"/>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="rdescendants">
                              <xs:complexType>
                                    <xs:group ref="roleGroup"/>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="instances">
                              <xs:complexType>
                                    <xs:group ref="conceptGroup"/>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="types">
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element name="individual"
type="atomType"/>
                                    </xs:sequence>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="roleFillers">
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element name="individual"
type="atomType"/>
                                          <xs:group ref="roleGroup"/>
                                    </xs:sequence>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="toldValues">
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element name="individual"
type="atomType"/>
                                          <xs:choice>
                                                <xs:element name="attribute"
type="atomType"/>
                                                <xs:element name="chain"
type="chainType"/>
                                          </xs:choice>
                                    </xs:sequence>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="relatedIndividuals">
                              <xs:complexType>
                                    <xs:group ref="roleGroup"/>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                  </xs:choice>
            </xs:complexType>
      </xs:element>
      <xs:annotation>
```

```
            <xs:documentation>Responses</xs:documentation>
      </xs:annotation>
      <xs:complexType name="conceptSetType">
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
                  <xs:element name="synonyms">
                        <xs:complexType>
                              <xs:choice maxOccurs="unbounded">
                                    <xs:element name="catom"
type="atomType"/>
                                    <xs:element ref="top"/>
                                    <xs:element ref="bottom"/>
                              </xs:choice>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="individualSetType">
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
                  <xs:element name="individual" type="atomType"/>
            </xs:sequence>
      </xs:complexType>
      <xs:element name="responses">
            <xs:complexType>
                  <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element name="conceptSet">
                              <xs:complexType>
                                    <xs:complexContent>
                                          <xs:extension
base="conceptSetType">
                                                <xs:attributeGroup
ref="ids"/>
                                          </xs:extension>
                                    </xs:complexContent>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="roleSet">
                              <xs:complexType>
                                    <xs:complexContent>
                                          <xs:extension base="roleSetType">
                                                <xs:attributeGroup
ref="ids"/>
                                          </xs:extension>
                                    </xs:complexContent>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="individualSet">
                              <xs:complexType>
                                    <xs:complexContent>
                                          <xs:extension
base="individualSetType">
                                                <xs:attributeGroup
ref="ids"/>
                                          </xs:extension>
                                    </xs:complexContent>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="individualPairSet">
                              <xs:complexType>
                                    <xs:complexContent>
```

```
                                          <xs:extension
base="individualPairSetType">
                                                <xs:attributeGroup
ref="ids"/>
                                          </xs:extension>
                                    </xs:complexContent>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="sval">
                              <xs:complexType>
                                    <xs:simpleContent>
                                          <xs:extension base="xs:string">
                                                <xs:attributeGroup
ref="ids"/>
                                          </xs:extension>
                                    </xs:simpleContent>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="ival">
                              <xs:complexType>
                                    <xs:simpleContent>
                                          <xs:extension base="xs:integer">
                                                <xs:attributeGroup
ref="ids"/>
                                          </xs:extension>
                                    </xs:simpleContent>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="true">
                              <xs:complexType>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="false">
                              <xs:complexType>
                                    <xs:attributeGroup ref="ids"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="error">
                              <xs:complexType>
                                    <xs:complexContent>
                                          <xs:extension base="errorType">
                                                <xs:attributeGroup
ref="ids"/>
                                          </xs:extension>
                                    </xs:complexContent>
                              </xs:complexType>
                        </xs:element>
                  </xs:choice>
            </xs:complexType>
      </xs:element>
      <xs:complexType name="roleSetType">
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
                  <xs:element name="synonyms">
                        <xs:complexType>
                              <xs:choice maxOccurs="unbounded">
                                    <xs:element name="ratom"
type="atomType"/>
                                    <xs:element name="feature"
type="atomType"/>
```

```
                              </xs:choice>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="errorType">
            <xs:attribute name="message" type="xs:string" use="optional"/>
            <xs:attribute name="code" type="xs:integer" use="required"/>
      </xs:complexType>
      <xs:complexType name="individualPairSetType">
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
                  <xs:element name="individualPair">
                        <xs:complexType>
                              <xs:sequence minOccurs="2" maxOccurs="2">
                                    <xs:element name="individual"
type="atomType"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
      <!--
*************************************************************************
***** -->
      <xs:annotation>
            <xs:documentation>Description of a reasoner's capabilities.
</xs:documentation>
      </xs:annotation>
      <xs:complexType name="supportsType">
            <xs:sequence>
                  <xs:element name="language">
                        <xs:annotation>
                              <xs:documentation>Describes the concept
language that the reasoner knows about.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:choice>
                                          <xs:element name="and">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes conjunction.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="or">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes disjunction.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="not">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes negation.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="some">
```

```
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes existential quantification.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="all">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes universal quantification.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="atmost">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes atmost restrictions.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="atleast">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes atleast restricitions</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="iset">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes the one-of construct. This in turn implies that
individuals are supported.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="concrete">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes concrete domains. In the current specification, this
amounts to supporting strings and integers, along with min, max, equals and
range predicates.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="individual">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes individuals (i.e. some A-box
functionality).</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="feature">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes features, i.e. functional roles.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="attribute">
```

```
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes attributes (i.e. relationships whose domain is a concrete
domain). This in turn implies that concrete domains are
supported.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="chain">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes feature chains, i.e. chains of functional roles followed
by an attribute.  This in turn implies that concrete domains are
supported</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="inverse">
                                                <xs:annotation>

      <xs:documentation>Indicates that the language that the reasoner
supports includes inverse roles. </xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                    </xs:choice>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="tell">
                        <xs:annotation>
                              <xs:documentation>Describes the tell language
that the reasoner supports.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:choice>
                                          <xs:element name="impliesc">
                                                <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes concept implications.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="impliesr">
                                                <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes role implications.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="equalc">
                                                <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes concept equaility.</xs:documentation>
                                                </xs:annotation>
                                          </xs:element>
                                          <xs:element name="equalr">
                                                <xs:annotation>
```

```
      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes role equality.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="domain">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes domain restrictions.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="range">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes range restrictions.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="rangeint">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes integer range restrictions.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="rangestring">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes string range restrictions.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="transitive">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes assertions about transitive
roles.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="functional">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes assertions about functional
roles.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="disjoint">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes disjointness axioms.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="instanceof">
                                                      <xs:annotation>
```

```
      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes instanceof assertions.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="related">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes relationships between
individuals.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="value">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the assertion language that the
reasoner supports includes association of concrete domain values with
individuals. </xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                          </xs:choice>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="ask">
                              <xs:annotation>
                                    <xs:documentation>Describes the query language
that the  reasoner supports.</xs:documentation>
                              </xs:annotation>
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:choice>
                                                <xs:element
name="allConceptNames">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes asking for all concepts.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="allRoleNames">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes asking for all roles.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="allIndividuals">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes asking for all individuals.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="satisfiable">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes satisfiability testing.</xs:documentation>
```

```
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="subsumes">
                                        <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes concept subsumption testing.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="disjoint">
                                        <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes disjointness testing.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="parents">
                                        <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for parents.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="children">
                                        <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for children</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="ancestors">
                                        <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for ancestors.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="descendants">
                                        <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for descendants.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="equivalents">
                                        <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for equivalent concepts.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="rparents">
                                        <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for role parents.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="rchildren">
                                        <xs:annotation>
```

```
      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for role children.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="rancestors">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for role ancestors.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="rdescendants">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for role descendants.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="instances">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for instances.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="types">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for known types of an
individual.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="roleFillers">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for known role fillers of an individual.
</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element name="values">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for known concrete values associated with an
individual.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                                <xs:element
name="relatedIndividuals">
                                                      <xs:annotation>

      <xs:documentation>Indicates that the query language that the reasoner
supports includes querying for all pairs of related individuals.
</xs:documentation>
                                                      </xs:annotation>
                                                </xs:element>
                                          </xs:choice>
```

```
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="uniqueNameAssumption">
                        <xs:annotation>
                              <xs:documentation>Indicates that the reasoner
employs the UNA for individuals.</xs:documentation>
                        </xs:annotation>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
</xs:schema>
```

BIBLIOGRAFIA

(Abitebul et al., 2000) Abitebul, S.; Buneman, P.; Suciu, D. Data on the Web: from Relations to Semistructured Data and XML. San Francisco: Morgan Kaufmann. 2000.

(Albertin, 2001) A. L. ALBERTIN. Comércio Eletrônico: Modelos, Aspectos e Contribuições de suas aplicações. Atlas, São Paulo. 2001.

(Alexaki et al., 2001) V. Alexaki; et al. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases, Proceedings of the Second International Workshop on the Semantic Web. SemWeb'2001. May 2001.

(Anupriya, 2002) Anupriya A. et al. DAML-S: Semantic Markup for Web Services. DAML services Coalition. 2002.

(Arruda, 2002) Arruda, Ladjane.MEDIWEB: Um Integrador Semântico de Dados na Web Baseado em Mediador. Dissertação de Mestrado. UFPB, Campina Grande. 2002.

(Austin, 1962) Austin, J. L.; 1962.How to Do Things with Words. Clarendon Press,Oxford. 1962.

(Bartolini, 2001) Bartolini, C., and Preist, C. A Framework for Automated Negotiation. HP Labs Technical Report. 2001.

(Bechhofer et al., 2003) Bechhofer, S. et al. OWL Web Ontology Language Reference. Draft Report, 2003. Acessado em Agosto de 2003. Disponível na Internet por www em: http://www.w3.org/TR/owl-ref.

(Bechhofer, 2001) Bechhofer S., Horrocks I., Goble C., Stevens R. OilEd: a Reason-able Ontology Editor for the Semantic Web. Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.

(Berners – Lee et al., 2000) BERNERS – LEE, Tim et al. Semantic Web Development Proposal. Acessado em Janeiro de 2002. Disponível na Internet por www em: http://www.w3c.org/2001/sw/. 2000.

(Berners-Lee et al., 2001) T. Berners-Lee, J. Handler, and O. Lassila: The Semantic Web, Scientific American. May 2001.

(Bonifácio, 2002) Bonifácio, A. Ontologias e Consultas Semânticas: Uma Aplicação ao Caso Lattes. Dissertação de Mestrado. UFRGS, Porto Alegre. 2002.

(Broekstra et al., 2000) J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks: Enabling knowledge representation on the Web by Extending RDF Schema. In Proceedings of the Tenth International World Wide Web Conference (WWW10), Hong Kong. May 2001.

(Burbeck, 2000) Burbeck, S. The Tao of e-vusiness services – The evolution of Web applications into service-oriented components with Web Services. IBM Software Group. 2000.

(Cameron, 1997) CAMERON, Debra.– Eletronic commercer: the new business plataform of the Internet. Charleston: Compurter Tecnology Research Corp., 1997.

(Corradi, 1999) Corradi, A. et al. Mobile Agent Integrity for Eletronic Commerce Applications. Information Systems. Vol. 24. N. 6. 1999.

(Dasgupta et al., 1999) Dasgupta, P.; Narasimhan, N.; Moser, L.E.; Melliar-Smith, P.M., MAgNET: mobile agents for networked electronic trading, IEEE Transactions on Knowledge and Data Engineering, Volume: 11 Issue: 4, July-Aug. 1999, pp. 509-525.

(Date, 1999) Chris Date, "An Introduction to Database Systems", 7th Edition (31 October, 1999) Addison Wesley; ISBN: 0201684195. 1999.

(Daum, 2002) Daum, Bertold. Arquitetura de sistems com XML: conteúdo, processo e apresentação. Rio de Janeiro. Ed. Campus. 2002.

(Erdmann apud Arruda, 2000) Erdmann, M.; Decker, S. Ontology-aware XML-Queries. Submission for WebDB. 2000.

(Eudenia, 2001) Xavier, Eudênia. Integração de Agentes de Informação. Jornada de Atualização de Inteligência Artificial IME-USP. Acessado em Agosto de 2002. Disponível na Internet por www em: http://www.ime.usp.br/~eudenia/jaia/. 2001

(Fensel et al., 2001) D. Fensel, I. Horrocks, F. van Harmelen, D. McGuinness, and P. F. Patel-Schneider: OIL: Ontology Infrastructure to Enable the Semantic Web, IEEE Intelligent Systems, 16(2). 2001.

(Fensel, 2001) D. Fensel: Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, Springer-Verlag, Berlin. 2001.

(Fonseca et al., 2003) Luis C. Fonseca, Sofiane Labidi, Othon B. Filho and Edson Nascimento, ICS- An Agent Mediated E-Commerce System. To apperar in the proceedings of 5th International Conference On Enterprise Information Systems, Angers – France. 23-26 April, 2003.

(Flanagan, 1997) Flanagan, D. Java in a Nutshell. 2nd Edition. CA: O'Reilly, 1997.

(Freitas et al., 2002) Freitas, F. ; Bittencourt, G. Comunicação entre Agentes em Ambientes Distribuídos Abertos: o Modelo "peer-to-peer".Revista Eletrônica de Iniciação Científica. Ano II. Vol II. Número II. Junho de 2002. ISSN 1519-8219. 2002

(Freitas, 2002) Freitas, F. Sistemas Multiagentes Cognitivos para a Recuperação, Classificação e Extração Integradas de Inforamção da Web. Tese de Doutorado, UFSC, Florianópolis. 2002.

(Gruber, 1992) Gruber, T. A Translation Approach to Portable Ontology Specifications. Knowledge Systems Laboratory - Stanford University, Stanford, CA, Technical Report KSL 92-71. 1992.

(Gruber, 1993) Gruber, T. Towards principles for the design of ontologies used for knowledgesharing. Starford university, Knowledge Systems Laboratory Technical Report KSL93-04. 1993.

(Guarino, 1996) Guarino, N. Formal Ontology and Information Systems. In: (Ed.) Formal Ontology in Information Systems. Pp. 3-15, IOS Press, Amsterdam, Nederlands. 1996.

(Guarino, 1998) Guarino, N. Formal Ontology and Information Systems. in: N. Guarino, (Ed.) Formal Ontology in Information Systems. pp. 3-15, IOS Press, Amsterdam, Netherlands. 1998.

(Hendler, 2002) Hendler, James, Berners-Lee, Tim and Miller, Eric. Integrating Applications on the Semantic Web. Journal of the Institute of Electrical Engineers of Japan. Vol. 122(10), p. 676-680. October, 2002.

(Horrocks et al., 2000) Horrocks, I. et al. OIL in a Nutshell, Proc. ECAI' 00 Workshop on Application of Ontologies and PSMs, Berlim, Germany, 2000, pp. 4.4-4.12. 2000.

(Horrocks, 2001) Horrocks, I; Harmelen, F. V. Reference Description of the DAML+OIL Ontology Markup Language. Draft Report, 2001. Acessado em Janeiro de 2002. Disponível na Internet por www em: http://www.daml.org/2000/12/reference.html. 2001.

(Horrocks, 2002) Horrocks, I. Tessaris, Sergio. Querying the Semantic Web: a Formal Approch. Presented at 1ª. International Semantic Web Conferece (ISWC-2002). 2002.

(Januzzi, 2001) JANUZZI, G.F. Sistema cliente para Comércio Eletrônico “Business-to-Consumer” baseado em agentes. Dissertação de Mestrado (Mestrado em Engenharia de Sistemas e Computação), Instituto Militar de Engenharia. 2001.

(Jennings et al., 1996) Jennings, N. R., Faratin, P., Johnson, M. J., O'Brien, P. O., and Wiegand, M. E. 1996. Using intelligent agents to manage business processes. In First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96) (April 1996), pp. 345–360. 1996.

(Jennings et al., 2000) Jennings, N., and Wooldridge, M. Agent-oriented software engineering. In Handbook of Agent Technology (ed. J. Bradshaw) AAAI/MIT Press. 2000.

(Jennings, 1998) Jennings, N.R. e Wooldrigde M. Applications of intelligent agents. 1998.

(Kalakota et al., 1997) W.A. Kalakota, R. Eletronic Commerce: a Managers’s Guide. Addison-Wesley, New York, 1997.

(Kreps, 1990) Kreps, D. M. (1990). A course in microeconomic theory, Princeton University Press. 1990.

(Labidi et al., 2000) Sofiane Labidi, Slimane Hammoudi and Lassaad Gannoun. Cooperation and Temporal Organization in Workflow Management. In the Proceedongs of the 2000 International Conference on Artificial Intelligence (IC-AI'2000). World Scientific Engineering Society. Las Vegas, USA. June 20-22, 2000.

(Labidi et al., 2003) Sofiane Labidi, Luis C. Fonseca, Othon B. Filho and Edson Nascimento, Intelligent B2B Commerce System. In the textbook Techno-Legal Aspects of Information Society and New Economy: an Overview, Formatex. Spain. 2003.

(Labrou, 1999) Y Labrou, T. Finin, and Y. Peng. The current landscape of agent communication languages. IEEE Intelligent systems. 14(2):45-52, 1999.

(Martins, 2002) MARTINS, A. T. C., PEQUENO, M., PEQUENO, T. H. C. A Multiple Worlds Semantics to a Paraconsistent Nonmonotonic Logic In: PARACONSISTENCY: THE LOGICAL WAY TO THE INCONSISTENCY.1 ed.New York : Marcel Dekker,Inc., 2002

(Meira, 2000) MEIRA JR, W., MURTA, C.D., RESENDE, R.S.F. Comércio eletrônico na WWW. Escola de Computação do Instituto de Matemática e Estatística – Universidade de São Paulo. 2000.

(Minsky, 1975) Minsky, M. Frame System Theory. In P.N. Johnson-Laird and P.C. Wason (eds.), Thinking: Readings in Cognitive Science. Cambridge: Cambridge University Press. 355-376. 1975.

(Morales, 1999) Morales, Eduardo. Representación de Conocimiento. Acessado em Março de 2002. Disponível na Internet por www em: http://www.mor.itesm.mx/~rdec/principal.html.

(Montesco, 2001) Montesco, Carlos Alberto. UCL – Uma Linguagem de Comunicação para Agentes de Software. Dissertação de Mestrado. USP, São Carlos. 2001.

(Mukherjee, 2000) Mukherjee R. et al. Analysis of domain specific ontologies for agent-oriented information retrieval. In the working notes of AAAI 2000 Workshop on Agent Oriented Information Systems, Austin, TX. 2000.

(Muller et al., 1985) Eithan Muller and Mark A Satterwaithe. Strategyproofness: the existense of dominant strategy mechanisms. In Leonid Hurwicz, David Schmeidler and Hugo Sonnenschein, editors, Social Goals andsocial organization: Essays in memory of Elishna Pazner, chapter 5, pages 131-172. Cambridge University Press. 1985.

(Myerson, 1985) Roger B Myerson. Bayesian equilibrium and incentive compatibility. In Leonid Hurwicz, David Schmeidler and Hugo Sonnenschein, editors, Social Goals andsocial organization: Essays in memory of Elishna Pazner, chapter 8, pages 229-260. Cambridge University Press, 1985.

(Nardi et al., 2002) D. Nardi, R. J. Brachman. An Introduction to Description Logics. In the Description Logic Handbook, edited by F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, Cambridge University Press, 2002, pages 5-44. 2002.

(Omenayenko, 2002) Ying Ding, Dieter Fensel, Michel Klein, and Borys Omelayenko. The Semantic Web: Yet Another Hip? To appear in Data and Knowledge Engineering, 2002, 6.10.01 1 Division of Mathmatics Computer Science, Vrije University.

(OMG, 2000) The Object Management Group, The mobile agent system interoperability facility, OMG TC Document orbos/Jan/2000. URL: http://www.omg.org. 2000.

(Paolucci, 2002) M. Paolucci, T. Kawmura, T. Payne and K. Sycara. Semantic Matching of Web Servies Capabilities. In Firts Int. Semantic Web Conf. 2002.

(Parunak, 2001) H. Van Dyke Parunak and James Odell, “Representing Social Structures in UML”, Proc. of the Agent-Oriented Software Engineering Workshop, Agents 2001, Michael Wooldridge, Paolo Ciancarini, and Gerhard Weiss, eds., (Held at the Agents 2001 conference, Montreal, Canada), 2001.

(Peppers, 1994) PEPPERS, D., ROGERS, M. Marketing um a um. Editor Campus. Rio de Janeiro. 1994.

(Pitts et al., 2000) Pitts -Moultis, N. XML BLACK BOOK, São Paulo: MAKRON BOOKS. 2000.

(Postlewaite, 1985) Andrew Postlewaite. Implementation via Nash equilibria in economic enviroments. In Leonid Hurwicz, David Schmeidler and Hugo Sonnenschein, editors, Social Goals andsocial organization: Essays in memory of Elishna Pazner, chapter 7, pages 205-228. Cambridge University Press. 1985.

(Rasmusen, 1989) E. Rasmusen. Games and Information. Basil Blackwell, 1989.

(Rich, 1993) Rich, E.; Knight, K. Inyeligência Artificial. Segunda Edição. Ed. Makon Books, São Paulo.1993.

(Rosenschein et al., 1994) Jeffrey Rosenschein and Gilad Zlotkin. Rules of Encounter. MIT Press, Cambridge, Massachusetts. 1994.

(Russel et al., 1995) Russell, S. J., Norving P., Artificial intelligence a modern aproach, Prentice Hall. 1995.

(Sandholm, 1996) T. Sandholm. Negotiation among Self-Interested Computationally Limited Agents. PhD thesis, University of Massachusetts, Amherst, 1996.

(Sandholm, 1997) SANDHOLM, T. W. and Lesser, V. R. Issues in automated negotiations and electronic commerce: Extending the contract net framework. In proceedings of the First International Conference on Multi-Agent Systems (ICMAS), Pp. 228-235, 1997, San Francisco, CA, June 1995. Reprinted in Readings in Agents, Huhns and Singh, eds., pp.66-73. 1997.

(Sandholm, 1999) SANDHOLM, T. Distributed Rational Decision Making. In the textbook Multiagent Systems: A Modern Introduction to Distributed Artificial Intelligence, Weiß, G., ed., MIT Press. Pp. 201-258. 1999.

(Smith, 1998) Smith, B. An Introduction to Ontology. in: D. Peuquet, B. Smith, and B. Brogaard, (Eds.), The Ontology of Fields. pp. 10-14, NCGIA, Bar Harbor, ME. 1998.

(Sparck, 1997) Sparck-Jones, Karen; Willet, Peter. Readings in Information Retrieval. San Francisco. Morgan Kaufmann. 1997.

(Stanley, 1999) L., Stanley. Descoberta de Conhecimento em Textos. Exame de Qualificação CPGCC-UFRGS, Porto Alegre, 1999.

(Steven, 1998) Steven D. Kaehler. Fuzzy Logic Tutorial. The Boeing Company. Seatle, 1998. Disponível em: http://www.seattlerobotics.org/encoder/mar98/fuz/flindex.html. Acessado Julho de 2003.

(Studer, 1998) Studer, Rudi et al. Knowledge Engineering: principles and methods. Data & Knowledge Engineering, v.25, n.1/2, Março de 1998.

(Sycara, 2002) Katia Sycara, Seth Widoff, Matthias Klusch and Jianguo Lu, "LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace." Autonomous Agents and Multi-Agent Systems, 5, 173–203, 2002.

(Tomaz, 2003) Tomaz, Ricardo F., Labidi S. and W. Benardo. A Semantic Matching Method for Clustering Traders in B2B Systems. To appear in 1st Latin American Web Congress, Santiago, Chile. November 2003.

(Tsvetovatyy, 1997) M. Tsvetovatyy, M. Gini, B. Mobasher, Z. Wieckowski. MAGMA: An Agent-Based Virtual Market for Electronic Commerce. Journal of Applied Artificial Intelligence, special issue on Intelligent Agents, Vol. 11, Nº 6, 1997

(Wooldridge, 1998) N. R. Jennings, Katia Sycara, and Michael Wooldridge: “A Roadmap of AgentResearch and Development”, Autonomous Agents and Multi-Agent Systems, 1, Kluwer Academic Publishers, Boston, 1998, pp. 275-306. 1998.

(Wooldridge, 1999) M. Wooldridge. Intelligent Agents In G. Weiss, editor: Multiagent Systems, The MIT Press, April 1999. Pp. 27-77. 1999.

URLs

(URL 1) Página do W3C "*World Wide Web Consotioum*",. 2003. URL: http://www.w3.org/

(URL 2) Página do projeto DAM-S Semantic Matchmaker desenvolvido no CMU http://www.ri.cmu.edu/projects/project_480.html

(URL 3) Descrição da Arquitetura do DAML-S Semantic Matchmaker desenvolvido no CMU http://www-2.cs.cmu.edu/~softagents/daml_Mmaker/daml-s_matchmaker_instructions.htm#APD

(URL 4) Página do Projeto LARKS em desenvolvimento no CMU. http://www-2.cs.cmu.edu/~softagents/larks.html

(URL 5) Página Rule Markup Language RuleML. http://www.dfki.uni-kl.de/ruleml/

(URL 6) Página da Sociedade Brasilerira de Gestão do Conhecimento. http://www.sbgc.org.br/cgi/cgilua.exe/sys/start.htm?tpl=home.

(URL 7) Foudation For Intelligent Physical Agents FIPA Personal AssistantSpecification. http://www.fipa.org/specs/fipa00083/XC00083B.html.

(URL 8) Especificação 1.0 da Linguagem Xquery. XQuery 1.0: An XML Query LanguageW3C Working Draft 02 May 2003. http://www.w3.org/TR/xquery

(URL 9) Página do Description Logic Group. (DIG). http://dl.kr.org/dig/

(URL10) Página do Reasoner FACT: (http://www.cs.man.ac.uk/~horrocks/FaCT/.)

(URL11) Página SoftwareAG empresa fbricante do Tamino Server.Http://www.softwareag.com/corporat/default.htm.

(URL 12) Página do Editor de Ontologias OilEd. http://oiled.man.ac.uk/.

(URL13) Página da HP com descrição do JENA Toolkit. http://www.hpl.hp.com/semweb/jena-top.html

(URL 14) Página do RACER. http://wwwfh-weedel.de/~mo/racer/index.html.

(URL 15) Página com documentação sobre JESS. http://herzberg.ca.sandia.gov/jess/docs/index.shtml.