Ativação de componentes de software com a utilização de uma ontologia de componentes

Augusto Carbol Lorza

Ativação de componentes de software com a utilização de uma ontologia de componentes

Augusto Carbol Lorza

Orientador: Prof. Dr. Dilvan de Abreu Moreira

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. .

USP – São Carlos
Junho/2007

Agradecimentos

Agradeço ao meu orientador Dilvan Moreira por ter me aceito como aluno de mestrado.

Faço um agradecimento muito especial à professora Renata Pontin, que durante a ausência de meu orientador, me deu todo o apoio necessário e acompanhou meu trabalho por um grande tempo, me ajudando a produzir este documento.

Também devo agradecer a Flávia Linhalis, que além de sempre ter me ajudado, ofereceu a oportunidade de eu desenvolver este tema que faz parte de uma idéia que ela desenvolveu em seu doutorado.

E não posso me esquecer de agradecer à minha namorada Juliana, por sempre ter me apoiado, durante toda minha vida acadêmica e por todo amor, carinho, amizade e confiança.

Agradeço também aos meus pais e familiares, que sempre me apoiaram. E aos meus tios Flávio e Maristela que me ajudaram muito durante momentos difíceis.

Aos meus amigos Marcel e Adriana, que sempre estiveram ao meu lado.

E a Capes, pelo suporte financeiro a este trabalho.

Resumo

Atualmente, existem muitos estudos para agregar mais valor às informações disponíveis na Web visando melhorar os resultados da interação dos usuários com a Web; uma das linhas de estudo é a Web Semântica, que propõe a adição de informação semântica à Web atual por meio de ontologias. A organização internacional que define os padrões para a Web (W3C) já propôs vários padrões para tornar a Web Semântica viável, porém, além de padrões, também é preciso criar ou adaptar ferramentas que explorem as suas potencialidades. Uma ferramenta que dá um suporte significativo para a Web atual e que pode ser adaptada para trabalhar com a Web Semântica é o Servidor de Aplicações. Com adição de informações semânticas, na forma de ontologias, tem-se um Servidor de Aplicações Baseado em Ontologias (OBAS). Neste trabalho foi desenvolvido um sistema protótipo para oferecer as características mínimas de um OBAS, e desta forma, foram investigadas as tecnologias para a Web Semântica que viabilizassem uma solução de acordo com os padrões recomendados pela W3C. Os componentes de software de um OBAS têm suas propriedades e comportamentos relacionados de forma semântica usando-se ontologias. Como uma ontologia é um modelo conceitual explícito, suas descrições dos componentes podem ser consultadas e inferidas, melhorando o desempenho do servidor através da combinação dos componentes mais apropriados a uma tarefa, da simplificação da programação, pois não é mais necessário saber todos os detalhes de um componente para ativá-lo.

Abstract

Many studies have been carried out to add more value to the available information in the Web with a view to improving the results of the users' interaction with the Web. Semantic Web is one line of research with focus on this issue and proposes the insertion of semantic information to the current Web through ontologies. Several patterns have been proposed by W3C, the international organization that defines patterns to the Web as an attempt to make the Semantic Web viable. However, besides patterns, it is also necessary to create or adapt tools to explore their potentialities. Application Server is a tool which gives significant support to the current Web and could be adapted to work with the Semantic Web. By adding semantic information, in the ontology form, we have an Ontology-Based Application Server (OBAS). This study develops a protoptype system which aims to offer the minimum characteristics of an OBAS. We have therefore investigated the semantic web which could provide a solution according to the patterns recommended by W3C. Properties and behaviors of software components of OBAS are semantically related by means of ontologies. Given that ontology is an explicit conceptual model, its component descriptions can be consulted and inferred, and hence improve the performance of the server. This is done by applying the most appropriate components to a given task and simplifying programming since components can be activated with no need to know all their details.

Sumário

Lista de Figuras

1 Introdução

Atualmente, informações são recursos essenciais para as pessoas realizarem suas atividades na sociedade. No contexto de Tecnologias de Informação, a Web está cada vez mais integrada às tarefas do cotidiano das pessoas, e com isso, está fazendo com que a quantidade de informações nela contida seja cada vez maior. Da forma como a Web foi concebida no seu início, não se previa quais seriam sua escala de uso e a velocidade de crescimento das informações nela contida. Por esse motivo, também não foi previsto um plano para criação de ferramentas, softwares e mecanismos em geral, para lidar com essa grande quantidade de informações e a rápida velocidade de crescimento das informações na Web que vivenciamos hoje. Como conseqüência disso, está cada vez mais difícil encontrar as informações procuradas, dentre as contidas na Web. Muitos pesquisadores têm sugerido a criação de diversos meios para resolver esses problemas. Um dos meios mais discutidos tem sido a criação de uma extensão da Web, chamada de Web Semântica (Berners-Lee et al., 2001).

Para que a Web Semântica seja uma solução viável, mecanismos devem possibilitar a adição de semântica ao conteúdo da Web, possibilitando mais facilidades para relacionar as informações que se encontram dispersas na Web. Para adicionar semântica ao conteúdo da Web, vêm sendo pesquisados e propostos vários padrões, arquiteturas e linguagens. Nessa linha de pesquisas, as ontologias se apresentam como de fundamental importância para adicionar semântica à Web (Berners-Lee et al., 2001). Pode-se dizer, de maneira simples, que uma ontologia é uma maneira de relacionar semanticamente, informações sobre um determinado escopo.

No entanto, apenas o relacionamento semântico entre as informações nem sempre é suficiente para resolver todos os problemas encontrados na Web. Além das ontologias, em alguns casos, também é preciso adicionar um conjunto de regras sobre os dados contidos nelas. Com a adição de regras sobre as ontologias, é possível extrair informações que não estavam descritas de forma explícita nas ontologias. Para adicionar essas regras, linguagens foram criadas com o propósito exclusivo de descrever regras para a Web semântica, além de máquinas de inferência capazes de processar essas regras.

Considerando os conceitos de ontologias e Web Semântica, a Web como foi concebida, requer que sejam criadas novas ferramentas ou feitas adaptações às ferramentas já existentes, para trabalharem com essas novas tecnologias criadas para a Web Semântica. Uma ferramenta que dá um suporte relevante à Web atual e que pode ser adaptada para trabalhar com essas novas tecnologias é o Servidor de Aplicações. Uma forma encontrada para agrupar as descrições semânticas (ontologias) e as regras, dentro de um Servidor de Aplicações, foi proposta por Oberle et al. (2005), que propôs um Servidor de Aplicações Baseado em ontologias (OBAS - sigla do inglês – *Ontology Based Application Server*).

Segundo Linhalis & Moreira (2006): "Em um OBAS, uma ontologia captura as propriedades e comportamentos de componentes de software e o relacionamento entre eles. Como uma ontologia é um modelo conceitual explícito, com semântica formal baseada em lógica, suas descrições dos componentes podem ser consultadas e inferidas e assim melhorar a composição dinâmica de aplicativos através da combinação dos componentes apropriados".

1.1 Objetivo

No contexto dos suportes adequados para que a Web Semântica viabilize cada vez mais um auxílio efetivo para os seus usuários, com base no conteúdo de informações disponibilizado que cresce rapidamente, foram estudados os conceitos de ontologias, Web semântica e Servidores de Aplicação. O objetivo principal deste trabalho foi desenvolver um sistema protótipo que ofereça as características mínimas de um OBAS. Como parte deste trabalho, foi reutilizada e adaptada uma ontologia que descreve componentes de software; essa ontologia foi instanciada com um conjunto de componentes disponíveis livremente pela web. Como resultado, o sistema protótipo foi construído para utilizar uma máquina de inferência, que faz inferências sobre a ontologia instanciada, e ativa um componente adequado a partir de uma requisição. Dessa forma, foi possível a comprovação de que a utilização de ontologias para encontrar e ativar componentes é viável.

1.2 Estrutura

Primeiramente, no Capítulo 2, é feita uma apresentação sobre a Web Semântica e sobre as Ontologias, na Seção 2.1, é apresentada a Arquitetura da Web Semântica e pelas ontologias serem de particular importância para este trabalho, elas serão discutidas no capítulo 2, onde é apresentado como construí-las (Seção 2.3), quais são as linguagens utilizadas para descrevê-las (Seção 2.4) e algumas ferramentas para o gerenciamento de ontologias (Seção 2.5).

Na Seção 2.6 é feita uma breve apresentação sobre as linguagens de regras para a Web Semântica (Seção 2.6.1); um enfoque especial será dado para a linguagem SWRL (Seção 2.6.1.1), que tem se destacado entre as outras propostas de linguagens. Para finalizar, será abordado o assunto máquinas de inferência para as linguagens da Web Semântica (Seção 2.6.2).

O Capítulo 3 descreve um Servidor de Aplicações (Seção 3.2) e apresenta o termo Middleware (Seção 3.1), pois um Servidor de Aplicações é um Middleware com algumas características especiais, como por exemplo, o de estar disponível na Web. Neste capítulo também é discutido um pouco sobre Serviços Web (Seção 3.3), pois eles adicionam algumas características aos Servidores de Aplicações. Finalmente, o conceito de um Servidor de Aplicações Baseado em Ontologias (OBAS – sigla do inglês, *Ontology Based Aplication Server*) (Seção 3.4) e de Componentes de Software são apresentados (Seção 3.5), com exemplos de arquiteturas baseadas em componentes JEE (Seção 3.5.1) e componentes .NET (Seção 3.5.2).

No Capítulo 4 é apresento o protótipo desenvolvido neste trabalho, primeiramente é discutida a arquitetura do sistema (Seção 0) e nas seções subseqüentes são apresentadas as partes do sistema descritas na arquitetura, sendo elas, a Ontologia de Componentes (Seção 4.2), o Mapeador e API Bossam (Seção 4.3), o Ativador de Serviços Web (Seção 4.4) e as Interfaces do protótipo (Seção 4.5). Para finalizar o capítulo, é feita uma breve discussão sobre a importância das regras (Seção 4.6).

Para finalizar as idéias, no Capítulo 5 são discutidas as contribuições alcançadas com este trabalho e futuras pesquisas a serem continuadas.

2 Web Semântica e Ontologias

Na Web atual ainda observa-se que a grande maioria dos documentos está no formato HTML (*Hypertext Markup Language*). A linguagem HTML, de marcação de hipertexto, é um padrão W3C (Raggett et al., 1999). Os documentos HTML consistem em seu próprio conteúdo, juntamente com informações de apresentação, pouco suporte a metadados e praticamente nenhuma informação semântica.

Essa carência de metadados e de informação semântica dificulta a recuperação de informações na Web, resultando em falta de precisão do que é apresentado e em um elevado número de documentos recuperados (quando se faz uma busca). Assim, os resultados obtidos demandam, por sua vez, que o próprio usuário realize uma filtragem manual do que é realmente de interesse.

Recentemente, pesquisadores vêm explorando o potencial de se associar significados explícitos aos conteúdos dos documentos presentes na Web sob uma representação que habilite aplicações a processarem e interpretarem automaticamente essas informações. A essa pesquisa, cujo produto se caracteriza por ser uma extensão da Web dá-se o nome de Web Semântica (Berners-Lee et al., 2001). A principal meta das pesquisas em Web Semântica é desenvolver padrões, arquiteturas de metadados e linguagens de ontologias que auxiliem os computadores na tarefa de fornecer e processar o significado de informações na Web e, conseqüentemente, promover a expansão dos mecanismos de consulta e integração de informações e de automatização de tarefas. A Web Semântica tem como base, portanto, metadados e ontologias para o processamento e a integração de informações semânticas agregadas.

Neste capítulo são apresentados os conceitos que foram estudados, relacionados com Web Semântica. Assim, são descritos: Arquitetura da Web Semântica (Seção 2.1), Ontologias (Seção 2.2), Engenharia de Ontologias (Seção 2.3), Linguagens para Descrição de Ontologias (Seção 2.4), onde serão apresentados os conceitos sobre RDF (Seção 2.4.1) e OWL (Seção 2.4.2). Ainda são apresentadas Ferramentas para Manipulação de Ontologias (Seção 2.5) e conceitos sobre Linguagens de Regras e Máquinas de Inferência (Seção 2.6). Assim, é também apresentada a linguagem de

regras SWRL (Seção 2.6.1.1) e algumas Máquinas de Inferências para SWRL (Seções 2.6.2.1 e 2.6.2.3). Por fim são feitas as Considerações Finais (Seção 2.7).

2.1 Arquitetura da Web Semântica

A Web Semântica tem sua arquitetura divida em camadas, que estão ilustradas na Figura 1.

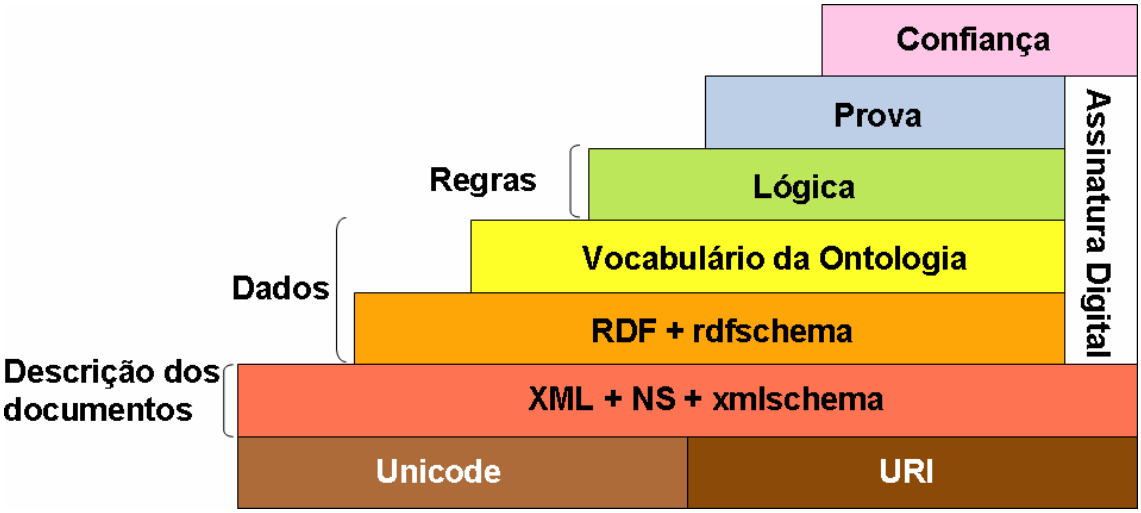

Figura 1 – Arquitetura da Web Semântica. Traduzida de (Berners-Lee, 2000)

A primeira camada, **Camada Unicode,** garante o uso padronizado do mesmo conjunto de caracteres (Unicode) e uma forma unívoca para identificação e localização de documentos (URI - *Uniform Resource Identifier*). Dessa forma, todos os documentos atualmente disponibilizados na Web podem ser considerados como conteúdo para essa camada da Web Semântica.

A **Camada XML + NS + XML Schema** adicional à primeira camada, tem o objetivo de descrever a estrutura dos documentos, deixando para as camadas que estão acima dela a definição do conteúdo desses documentos. A gramática XML (*Extensible Markup Language*) (McGrath, 1998) foi criada pelo W3C (*World Wide Web Consortium*) com o objetivo de permitir que fossem construídos documentos cuja informação fosse estruturada, isto é, documentos que contivessem não só conteúdo, mas também indicações a respeito desse conteúdo (metadados).

Por meio de XML, tem-se que os metadados são especificados utilizando-se rótulos (*tags*). Dessa forma, XML provê facilidades para a definição destes rótulos e das

relações estruturais entre elas. XML não especifica um conjunto pré-definido de rótulos, diferente de HTML. Essa liberdade de criação de elementos e atributos pode criar problemas de vocabulário, como colisões e dificuldade de reconhecimento. Isto pode ocorrer se documentos, de um mesmo domínio de aplicação, utilizarem termos diferentes ou se documentos de domínios diferentes utilizarem os mesmos termos.

Para amenizar esse problema foram criados os espaços de nomes (NS - *namespaces*). Um espaço de nomes XML é uma coleção de nomes, identificada por um URI, para ser utilizada em documentos XML como tipos de elementos e nomes de atributos. Dessa maneira, documentos semelhantes podem utilizar nomes universais.

A estrutura de um documento XML pode ser especificada por definições de tipos de documentos (DTDs – *Document Type Definitions*) ou por esquemas XML (XML Schema ou XMLS) (Fallside, 2001). Um DTD define o alinhamento léxico de um documento, as classes (ou etiquetas) e seus atributos, e a ordem dos dados no documento XML. Os esquemas têm a mesma função, mas são mais completos, pois se pode, por exemplo, definir tipo e formato exato dos atributos e número exato de instâncias de um aninhamento. De fato, definições em esquema XML são elas próprias documentos XML, o que possibilita que o esquema também se aproveite do mecanismo de espaços de nomes.

Os estudos sobre a **Camada RDF + RDF Schema** estão detalhados na Seção 2.4.1, pois foram aprofundados para o desenvolvimento do sistema protótipo realizado neste mestrado. O *framework* RDF (Resource Description Framework) (Lassila & Swick, 1999) adiciona mais semântica a um documento, com a vantagem de não precisar referir-se a sua estrutura.

Apesar da expressividade de RDF e RDF Schema, esses dois padrões ainda deixam a desejar quanto à modelagem de ontologias, na qual é possível se especificar as relações entre os conceitos de um vocabulário (de Freitas, 2003). A **Camada Vocabulário da Ontologia** é importante ressaltar que, apesar da expressividade de RDF e RDFS, esses dois padrões ainda deixam a desejar quanto à modelagem de ontologias (de Freitas, 2003). Permite-se que novas linguagens definam os relacionamentos do vocabulário de uma ontologia. Dessa forma, surgiram OIL, DAML+OIL e, mais recentemente, OWL. A OWL será detalhada na Seção 2.4.2.

As três camadas superiores, **Camadas de Lógica**, **Prova** e **Confiança** ainda não foram completamente especificadas pela W3C. A **camada de lógica** permite a especificação de regras que atuam sobre instâncias e recursos, enquanto a **camada de prova** as executa, e a **camada de confiança** avalia se a prova está correta ou não (Koivunen & Miller, 2001). Para que essas camadas entrem em operação, as camadas inferiores devem estar bem sedimentadas, o que ainda está acontecendo.

A **Camada de Lógica** já dispõe de uma proposta de linguagens, o SWRL (*Semantic Web Rule Language*) que utiliza a sintaxe OWL. A SWRL é utilizada para descrição genérica de regras e será descrita detalhadamente na Seção 2.6.1.

Neste trabalho, enfoque maior é dado nas camadas de ontologia e lógica, que compreendem, respectivamente, os dados e as regras. Elas foram utilizadas no sistema protótipo desenvolvido neste trabalho. Mais detalhes sobre ontologias e regras são apresentados a seguir.

2.2 Ontologias

No contexto da Web Semântica, sistemas de software são capazes de interpretar associações entre recursos que não apresentam quaisquer relacionamentos diretos. Para que isso seja possível, tais sistemas de software precisam ter acesso a coleções estruturadas de informações, bem como a um conjunto de regras de inferência que auxiliem no processo automatizado de obtenção dos resultados. Na Web Semântica, essas informações estruturadas são representadas por meio de ontologias.

Uma definição clássica de ontologia é a proposta por Gruber (1993), em que "*ontologia é uma especificação explícita de uma conceituação; uma descrição formal dos conceitos e relacionamentos de uma área de conhecimento*". Ontologias podem ser utilizadas por aplicações que necessitam compartilhar informação de um determinado domínio de estudo. Para isso, utiliza-se um vocabulário específico e um conjunto de axiomas lógicos que não apenas fornecem a semântica pretendida aos termos desse vocabulário, mas que também restringem a interpretação e utilização desses termos.

Muitos dos benefícios obtidos com a utilização de ontologias advêm da abordagem declarativa de construção de sistemas de software. A motivação principal do paradigma declarativo é modelar sistemas em alto nível e mais próximos do conhecimento sobre o

domínio a ser modelado. O modelo declarativo consiste em uma forma mais flexível de descrever um domínio sem qualquer compromisso com sua implementação (Freitas, 2003).

2.2.1 Propriedades de Ontologias

Existem ontologias com informações de naturezas diversas modeladas sob diferentes níveis de especificação. Segundo Hefiln (2004), de forma geral, ontologias devem apresentar as seguintes propriedades comuns:

Interoperabilidade - diferentes ontologias podem modelar os mesmos conceitos sob diferentes perspectivas. Para permitir que sistemas de software sejam capazes de integrar informação a partir de ontologias diferentes, é necessário um mecanismo de mapeamento de equivalência entre conceitos dessas ontologias.

Compartilhamento - ontologias devem ser publicamente disponíveis e fontes de dados diferentes devem ser capazes de concordar com definições e regras de uma mesma ontologia.

Extensibilidade e reuso - ontologias devem ser capazes de usar e estender outras ontologias no sentido de fornecer definições adicionais.

Evolução - assim como informações na Web, ontologias podem mudar durante seu ciclo de vida. Portanto, uma fonte de dados deve especificar a versão da ontologia com a qual ela concorda.

Balanço de expressividade e escalabilidade - ontologias devem não apenas ser capazes de expressar uma grande quantidade de conhecimento, mas também de fornecer meios eficientes para interpretá-la. Estas duas propriedades são bastante dependentes da linguagem de ontologia utilizada.

Facilidade de uso - ontologias devem apresentar conceitos e significados de forma clara, independentemente da sintaxe da linguagem de ontologia utilizada.

Detecção de inconsistências - diferentes ontologias ou fontes de dados podem ser contraditórias. Ontologias devem, portanto, fornecer um mecanismo que auxiliem as ferramentas de detecção dessas inconsistências.

Compatibilidade com padrões - para promover sua disseminação, ontologias devem utilizar padrões Web e da indústria.

Essas propriedades devem ser consideradas principalmente por autores de novas ontologias, ou mesmo, os potenciais autores de extensões a ontologias existentes. Dessa forma, tais autores podem garantir que a ontologia seja consistente e possa ser reutilizada.

2.2.2 Tipos de Informações em Ontologias

Uma ontologia compreende o conhecimento que seu projetista tem de um domínio particular, e o descreve sob a forma de conceitos e relações (Bézvin, 1998). Esse conhecimento expresso em ontologias é formalizado usando três tipos básicos de informação:

Terminológica - conjunto básico de conceitos e relações da ontologia, denominada de camada de definição da ontologia. (Exemplo: animal, macho, fêmea, etc.)

Assertiva - denominada de camada de axiomas da ontologia, é o conjunto de assertivas que se aplicam aos conceitos e relações. Por exemplo: O ser humano pode ser homem ou mulher, o homem é ser humano e macho, a mulher é ser humano e fêmea e o pai é ancestral do homem e da mulher e é homem.

Pragmática - pode-se classificar nessa categoria a forma de apresentação dos conceitos e relações embutidos na ontologia, utilizando, por exemplo, ferramentas que manipulem essas informações, tais como editores, validadores, máquinas de inferência, etc.

Vale ressaltar que uma ontologia geralmente possui esses três tipos informações.

2.2.3 Classificação de Ontologias

Segundo Guarino (1998), as ontologias podem ser classificadas em Ontologia de Alto Nível (ou *upper ontology*), Ontologia de Domínio, Ontologia de Tarefa e Ontologia de Aplicação. Na Figura 2 é ilustrado o relacionamento entre esses tipos de ontologias.

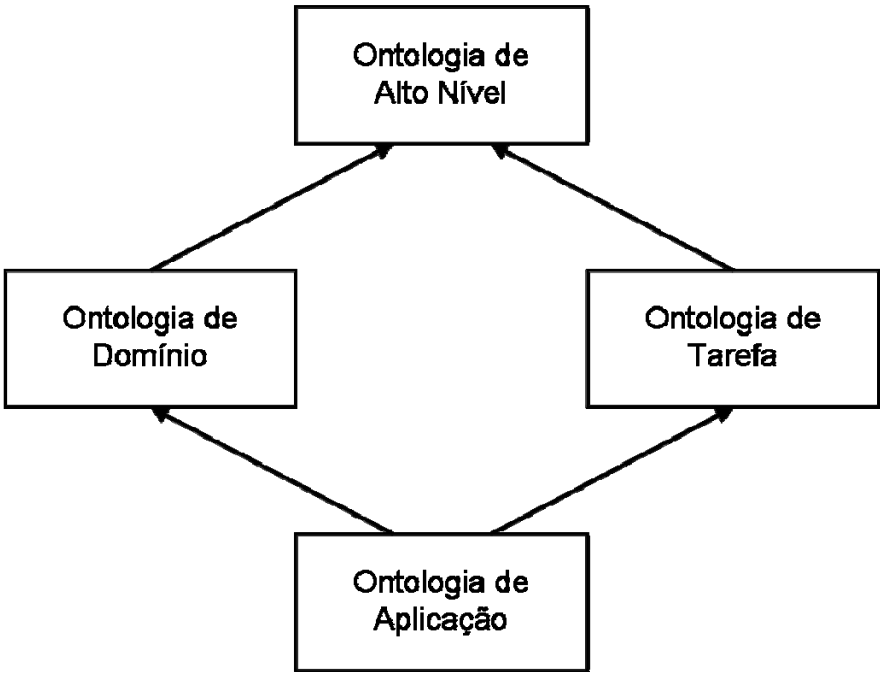

Figura 2 - Classificação de Ontologia segundo Guarino (1998)

A **Ontologia de Alto Nível:** descreve conceitos gerais como espaço, tempo, assunto, objeto, evento, ação, etc., os quais são independentes de um problema ou domínio específico.

A **Ontologia de Domínio/Tarefa:** descreve o vocabulário relacionado ao domínio genérico (exemplo: medicina, automóvel), ou uma tarefa/atividade genérica (exemplo: diagnóstico, venda), especializando os termos introduzidos na ontologia de alto nível.

A **Ontologia de Aplicação:** descreve conceitos dependendo de um domínio e tarefa específicos, os quais são freqüentemente especializações das ontologias relacionadas. Esses conceitos correspondem aos papéis das entidades do domínio, enquanto desempenham certa atividade como unidade substituível ou componente dispensável.

O relacionamento entre essas classes de ontologias pode ser representado de forma hierárquica e neste trabalho uma Ontologia de Aplicação foi desenvolvida.

2.3 Engenharia de Ontologias

A engenharia de ontologias deve ser conduzida segundo princípios de um projeto de software no sentido de serem tomadas decisões que visam sua qualidade segundo critérios como eficiência, legibilidade, portabilidade, extensibilidade, interoperabilidade e reuso. São listados, a seguir, princípios para a construção de ontologias segundo Gruber (1993):

Legibilidade: ontologias devem conter um vocabulário compartilhado, normalmente o jargão e a terminologia utilizados por especialistas do domínio.

Coerência: inferências derivadas de uma ontologia devem ser corretas e consistentes do ponto de vista formal e informal com as definições da ontologia.

Extensibilidade: ontologias devem permitir extensões e especializações com coerência, sem a necessidade de revisão de teoria em busca de contradições.

Modularidade: deve-se minimizar o acoplamento entre conceitos de uma ontologia.

Mínima codificação: devem ser especificados conceitos genéricos independentemente de padrões estabelecidos para notação e codificação, garantindo a extensibilidade da ontologia.

Mínimo compromisso ontológico: para maximizar o reuso, apenas o conhecimento essencial de cada conceito deve ser incluído de forma a permitir a criação de conceitos novos, conceitos mais especializados ou estendidos.

Embora não exista uma metodologia adotada como padrão para o desenvolvimento de ontologias, algumas iniciativas recentes consideram que a concepção de ontologias requer passos similares aos da engenharia de software, como a especificação, a conceituação e a implementação de ontologias. Esses passos devem seguir um processo iterativo, ou seja, com revisões constantes (Gómez-Pérez, 1999).

Mesmo não tendo sido construída uma ontologia a partir do ponto inicial, neste trabalho, os princípios de Engenharia de Ontologias foram observados para manter a qualidade da ontologia estendida.

2.4 Linguagem para Construção de Ontologias

A linguagem para construção de ontologias OWL (*Web Ontology Language*) fornece um conjunto de elementos e atributos para descrever classes e relacionamentos entre classes correspondentes a informações e aplicações na Web (Bechhofer et al., 2004). A linguagem OWL é um padrão recomendado pela W3C desde 10 de fevereiro de 2004 como linguagem padrão para ontologias na Web (McGuinness & van Harmelen, 2004).

Antes de apresentar a OWL é necessário fazer uma breve apresentação do modelo de dados e da sintaxe do padrão RDF (*Resourse Description Framework*) (Manola et al., 2004), pois a sintaxe da OWL é a mesma que a sintaxe RDF/XML.

2.4.1 RDF

A solução proposta pela W3C para tornar as informações da Web compreensíveis por computadores e facilitar a automatização de tarefas na Web, é o fornecimento de um mecanismo que permite a descrição de metadados relacionados a qualquer recurso Web: o RDF (*Resourse Description Framework*) (Klyne et al., 2004).

O RDF é composto por três elementos principais: o modelo de dados; a sintaxe para intercâmbio de metadados; e o esquema. Esses componentes são apresentados a seguir.

2.4.1.1 Modelo de Dados RDF

O modelo de dados RDF fornece uma estrutura conceitual e abstrata para definir metadados. O modelo de dados consiste em três tipos de objetos:

Recursos - são sempre denominados por URIs mais identificadores opcionais de âncora. Geralmente os recursos são páginas Web, mas podem ser uma parte de uma página Web (por exemplo, elemento HTML ou XML), uma coleção inteira de páginas (por exemplo, um Web site), ou até mesmo objetos que não são diretamente acessíveis via Web (por exemplo, livro impresso).

Propriedades - são aspectos, características, atributos, ou relações específicas usadas para descrever um recurso (por exemplo, uma propriedade do recurso

http://www.w3.org/XML/Schema#dev é a data de revisão, que pode ser visualizada na Figura 4). Cada propriedade possui um significado específico, definem seus valores permitidos, os tipos de recursos que descreve e seus relacionamentos com outras propriedades.

Sentenças - são triplas formadas por recursos, propriedades e valores das propriedades. O valor de uma propriedade pode ser um outro recurso ou um literal. Um literal geralmente é uma cadeia de caracteres (string).

Um exemplo de uma sentença (*statement*) é mostrado a seguir:

"A data de revisão de http://www.w3.org/XML/Schema#dev possui valor 2007/01/23"

Essa sentença é composta das partes recurso, propriedade e valor, conforme apresentadas na Figura 3.

Recurso	http://www.w3.org/XML/Schema#dev
Propriedade	data de revisão
Valor	2007/01/23

Figura 3 - As partes de uma sentença(statement) RDF: recurso, propriedade e valor

Uma sentença também pode ser representada por um grafo dirigido rotulado, também chamado de "diagrama de nós e arcos", como apresentado na Figura 4. O recurso é representado por um nó oval, o valor da propriedade é representado por um nó retangular, e a propriedade é representada por um arco que conecta o recurso ao valor da propriedade. Essa notação é sugerida por Manola et al. (2004).

Figura 4 - Exemplo de uma sentença RDF representado por um diagrama de nós e arcos

Essas formas de representação das sentenças RDF são de fácil compreensão para seres humanos, porém para sistemas computadorizados não o são. A seguir é apresentada a sintaxe RDF, que é adequada para processamento a partir de um software.

2.4.1.2 Sintaxe RDF

Existem duas sintaxes XML para codificar uma instância do modelo de dados RDF, a sintaxe de serialização e a sintaxe abreviada. A sintaxe de serialização mostra claramente a estrutura de um modelo RDF e a sintaxe abreviada fornece uma forma mais compacta de representar o modelo de dados. Neste trabalho, apenas a sintaxe de serialização será descrita a seguir, pois apresenta os elementos de forma mais detalhada.

Sintaxe de Serialização RDF

O conteúdo entre os elementos <rdf:RDF> e </rdf:RDF> em um documento XML representa uma instância do modelo de dados RDF. O elemento Description é utilizado para agrupar múltiplas sentenças em um mesmo recurso, ou seja, fornece uma maneira de atribuir um nome ao recurso apenas uma vez para várias sentenças. O nome utilizado para representar o recurso é inserido no atributo about caso o recurso exista, ou inserido no atributo ID caso contrário.

O exemplo apresentado na Figura 4 é representado no formato RDF/XML na Figura 5.

```
(1)  <rdf:RDF>
(2)    <rdf:Description about="http://www.w3.org/XML/Schema#dev">
(3)      <s:data_revisao>2007/01/23</s:data_revisao>
(4)    </rdf:Description>
(5)  </rdf:RDF>
```

Figura 5 – Representação do RDF no formato RDF/XML

E o documento XML completo é apresentado na Figura 6.

```
(1)  <?xml version="1.0"?>
(2)  <rdf:RDF>
(3)    <xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
(4)    <xmlns:s="http://description.org/schema/">
(5)    <rdf:Description about="http://www.w3.org/XML/Schema#dev">
(6)      <s:data_revisao>2007/01/23</s:data_revisao>
(7)    </rdf:Description>
(8)  </rdf:RDF>
```

Figura 6 – Documento XML completo para um documento RDF

Como se pode observar na Figura 5 e na Figura 6, elas possuem apenas a diferença entre as linguagesn RDF e RDF/XML. Assim, na Figura 6, as linhas para definição do espaço de nomes e indicação de inicio de documento XML estão presentes.

Verificando o exemplo acima, percebe-se que, quando expressões são muito condensadas e escritas à mão, a utilização da sintaxe de serialização é desencorajadora e aumenta a possibilidade da sintaxe estar errada.

2.4.1.3 Esquema RDF

O modelo de dados define um modelo simples para descrever inter-relacionamentos entre recursos em termos de propriedades e valores. Entretanto, não fornece mecanismos para definir: **(a)** as propriedades em si (por exemplo, título, autor, tamanho, cor, etc.), **(b)** os relacionamentos entre as propriedades e recursos, e **(c)** os tipos de recursos sendo descritos (por exemplo, livros, páginas Web, pessoas, etc.). Esse é o papel do Esquema RDF (Brickley et al., 2004).

Esquemas RDF podem ser comparados a Definições de Tipo de Documento XML (DTDs) e Esquemas XML (Manola et al., 2004). Diferente de um DTD ou Esquema XML, que fornecem restrições sobre a estrutura de um documento XML, um Esquema RDF fornece informação sobre a interpretação das sentenças em um modelo de dados RDF.

Para ilustrar as funcionalidades oferecidas pelo Esquema RDF, bem como a respectiva sintaxe em XML, um exemplo é apresentado na Figura 7.

```
(01)  <rdf:RDF xml:lang="en"
(02)    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
(03)    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
(04)  <rdfs:Class rdf:ID="Person">
(05)    <rdfs:comment>The class of people.</rdfs:comment>
(06)    <rdfs:subClassOfrdf:resource="http://www.w3.org/2000/03/example/classes #Animal"/>
(07)  </rdfs:Class>
(08)  <rdf:Property ID="maritalStatus">
(09)    <rdfs:range rdf:resource="#MaritalStatus"/>
(10)    <rdfs:domain rdf:resource="#Person"/>
(11)  </rdf:Property>
(12)  <rdf:Property ID="ssn">
(13)    <rdfs:comment>Social Security Number</rdfs:comment>
(14)    <rdfs:range rdf:resource="http://www.w3.org/2000/03/example/classes #Integer"/>
(15)    <rdfs:domain rdf:resource="#Person"/>
(16)  </rdf:Property>
(17)  <rdf:Property ID="age">
(18)    <rdfs:range rdf:resource="http://www.w3.org/2000/03/example/classes #Integer"/>
(19)    <rdfs:domain rdf:resource="#Person"/>
(20)  </rdf:Property>
(21)  <rdfs:Class rdf:ID="MaritalStatus"/>
(22)  <MaritalStatus rdf:ID="Married"/>
(23)  <MaritalStatus rdf:ID="Divorced"/>
(24)  <MaritalStatus rdf:ID="Single"/>
(25)  <MaritalStatus rdf:ID="Widowed"/>
(26) </rdf:RDF>
```

Figura 7 - Exemplo de esquema RDF

O núcleo dos vocabulários do RDF e do Esquema RDF são definidos, respectivamente, nos espaços de nomes XML rdf (linha 02) e rdfs (linha 03). O idioma da documentação é indicado pela utilização do elemento xml:lang. No exemplo, os recursos definidos no Esquema RDF estão no idioma inglês, segundo a notação xml:lang='en' (linha 01).

Person (linha 04) é uma classe, denotada através de rdfs:Class, com a seguinte descrição facilmente compreensível por humanos: "A classe de pessoas". Essa descrição é denotada através de comment (linha 05), que é uma propriedade utilizada para documentação.

Todas as pessoas são animais. Assim, Person é subclasse de Animal (linha 06). Essa relação é denotada por rdfs:subClassOf (linha 06). A classe Animal é definida em outro esquema. A propriedade rdfs:domain (linhas 15 e 19) é utilizada para indicar as classes em que as propriedades podem ser usadas. Por exemplo, as propriedades MaritalStatus (linha 21), ssn (linha 12) e age (linha 17) só podem ser usadas na classe Person.

Uma pessoa pode possuir as propriedades age e ssn, cujos valores são inteiros. Essa relação é denotada pela restrição rdfs:range. A classe Integer é definida em um outro esquema. O estado civil de uma pessoa pode ser: Single, Married, Divorced e Widowed. Essa relação também é obtida através do uso da restrição rdfs:range.

2.4.2 OWL

A partir de que RDF e do RDFS representam relacionamentos semânticos, a linguagem OWL (*Web Ontology Language*) utiliza as sintaxes de RDF e RDFS como base.

A linguagem OWL fornece um conjunto de elementos e atributos para descrever classes e relacionamentos entre classes correspondentes a informações e aplicações na Web (Bechhofer et al., 2004). A linguagem OWL (*Web Ontology Language*) (McGuinness & van Harmelen, 2004) é uma recomendação W3C. Ela é uma revisão de DAML+OIL, incorporando o que se aprendeu no desenvolvimento e aplicação de DAML+OIL.

A *Ontology Inference Layer* (OIL) (Horrocks et al., 2000) é uma linguagem criada para representar semântica de uma maneira acessível por máquinas, modelando domínios de conhecimento na forma de ontologias. Desenvolvida para ser compatível com padrões do W3C, incluindo XML e RDF, OIL explora as primitivas de modelagem de Esquema RDF. Desta maneira, aplicações que suportam apenas RDF podem entender pelo menos parcialmente um documento OIL.

A linguagem DARPA *Agent Markup Language* (DAML) (Pagels, 2006), desenvolvida pelo DARPA *Agent Markup Language Program*, foi criada devido a incapacidade do XML de expressar as relações existentes entre os conceitos representados nos seus documentos. Assim, DAML foi desenvolvida como uma extensão de XML e de RDF, a fim de aumentar o poder de expressão e modelagem do conhecimento em documentos Web.

A linguagem de ontologia DAML+OIL fornece meios para modelar domínios de conhecimento utilizando ontologias. Pois incorpora aspectos tanto da linguagem DAML quanto da linguagem OIL, e pode ser vista como um subdialeto dessas. Existem várias diferenças entre as linguagens OIL e DAML+OIL (McGuinness et al., 2002). Elas

existem principalmente devido ao fato de que DAML+OIL foi baseada em RDF. Assim, algumas construções em RDF são possíveis em DAML+OIL, mas não em OIL.

2.4.2.1 Módulos da OWL

Devido à influência da linguagem de ontologia OIL, a linguagem OWL também seguiu uma estrutura modular, dividindo-se em três sub-linguagens de acordo com sua capacidade de expressão (McGuinness & van Harmelen, 2004). A seguir essas sub-linguagens são descritas:

OWL Lite: herda mecanismos de definição de classes e propriedades, de hierarquização de classes e documentação da especificação do esquema RDF. Além disso, o módulo OWL Lite fornece a semântica necessária para definir versionamento de ontologias, características de propriedades (transitividade, simetria, relação inversa), igualdade e desigualdade entre classes, propriedades e instâncias, tipos de dados das propriedades segundo a especificação do esquema XML e restrições na cardinalidade máxima e mínima de propriedades (0 ou 1). O objetivo de OWL Lite é prover uma linguagem que seja vista por desenvolvedores de ferramentas como suficientemente fácil e útil de utilizar. OWL Lite fornece várias das características comumente utilizadas em OWL e DAML+OIL. Com o OWL Lite espera-se facilitar a adoção de OWL na Web.

OWL DL: DL significa descrição lógica (*description logics*) e oferece suporte a propriedades que permitem definir relações entre instâncias de classes, bem como relacionar instâncias de classes a literais do RDF ou a tipos de dados, segundo a especificação do esquema XML. No módulo OWL/DL, classes podem ser construídas por união, intersecção e complemento, ou pela enumeração de instâncias, além de poderem ter disjunções.

OWL Full: utiliza o mesmo vocabulário da OWL DL, porém com suporte completo à cardinalidade máxima e mínima de propriedades (números inteiros positivos arbitrários, por exemplo). Enquanto os outros módulos da OWL restringem sua sintaxe de classes às classes definidas por esquemas RDF e pequenas extensões, por exemplo, owl:equivalentClass, o módulo OWL Full permite a criação e manipulação de metaclasses via enumerações, restrições de propriedades e

combinações booleanas. No entanto, não há garantia de computabilidade desse vocabulário.

As linguagens menos expressivas, OWL Lite e DL, estão contidas, respectivamente, nas mais expressivas, OWL DL e Full, de maneira que uma ontologia definida em uma linguagem menos expressiva é aceita por uma linguagem mais expressiva. No entanto, a recíproca não é verdadeira.

2.4.2.2 Sintaxe da OWL

Para definir uma ontologia em OWL, é necessário indicar que vocabulários específicos são utilizados por meio de um conjunto de espaços de nomes XML declarado no início da definição da ontologia (Smith et al., 2004). A Figura 8 ilustra o trecho inicial de uma ontologia.

```
(1)  <rdf:RDF
(2)     xmlns ="http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#"
(3)     xmlns:vin ="http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#"
(4)     xmlns:food="http://www.w3.org/TR/2003/CR-owl-guide-20030818/food#"
(5)     xmlns:owl ="http://www.w3.org/2002/07/owl#"
(6)     xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
(7)     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
(8)     xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#">
```

Figura 8 - Declaração dos espaços de nomes de uma ontologia OWL

As duas primeiras declarações (linhas 2 e 3) identificam o espaço de nomes XML associados à ontologia. A primeira (linha 2) declara que quaisquer nomes sem referência a espaços de nomes XML pertencem à ontologia corrente. A segunda declaração (linha 3) associa o espaço de nomes XML da ontologia com o prefixo vin: para referenciar definições de seu vocabulário. A terceira declaração (linha 4) identifica e associa o espaço de nomes XML da ontologia de comidas ao prefixo food:. As outras declarações de espaços de nomes XML indicam que os elementos com os prefixos owl: (linha 5), rdf: (linha 6), rdfs: (linha 7) e xsd: (linha 8) devem ser entendidos como definições pertencentes aos vocabulários das linguagens OWL, RDF, esquema RDF e esquema XML, respectivamente.

A maior parte dos elementos contidos em uma ontologia OWL trata de classes, propriedades, instâncias de classes e relacionamentos entre instâncias. Os conceitos

mais básicos de um domínio de conhecimento devem originar classes como raízes de árvores taxonômicas. Tudo declarado em OWL é membro da classe owl:Thing. Assim, cada classe de uma ontologia é subclasse de owl:Thing. Os construtores clássicos para definição e hierarquização de classes são owl:Class e rdfs:subClassOf, respectivamente. Portanto, se X é subclasse de Y, cada instância de X é instância de Y. O exemplo ilustrado na Figura 9 declara que a classe de vinhos (linha 1) é subclasse da classe de líquidos potáveis (linha 2).

```
(1) <owl:Class rdf:ID="Wine">
(2)    <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
(3)    ...
(4) </owl:Class>
```

Figura 9 - Exemplo de Classes e Sub-Classe em OWL

Essa definição de vinhos indica que vinhos são "líquidos potáveis", portanto sem informação suficiente para criar e interpretar indivíduos dessa classe. Para descrever membros de uma classe, utiliza-se o nome da classe a qual pertence, seguido do construtor rdf:ID para identificá-lo, como ilustrado na Figura 10: uvas para o vinho *Cabernet Sauvignon* (linha 4) pertencem à classe de uvas de vinho (linha 1) que, por sua vez, são da classe de uvas (linha 2) declarada no espaço de nomes XML de comidas (food).

```
(1) <owl:Class rdf:ID="WineGrape">
(2)    <rdfs:subClassOf rdf:resource="&food;Grape" />
(3) </owl:Class>
(4) <WineGrape rdf:ID="CabernetSauvignonGrape" />
```

Figura 10 – Exemplo de membros de uma Classe em OWL

Este exemplo, ilustrado na Figura 10, de ontologia com classes de vinhos e suas instâncias pode ser enriquecido com a associação de propriedades a classes. Propriedades permitem declarar fatos genéricos sobre membros de classes, bem como fatos específicos sobre instâncias. O exemplo apresentado na Figura 11 declara que vinhos (linha 2) possuem uma propriedade que indica que estes são feitos de uva (linha 1). Essa propriedade (madeFromGrape) assume valores que pertencem à classe de uvas de vinho (linha 3). O construtor owl:ObjectProperty relaciona instâncias de duas classes, neste caso, instâncias de vinhos e de uvas de vinhos.

```
(1) <owl:ObjectProperty rdf:ID="madeFromGrape">
(2)    <rdfs:domain rdf:resource="#Wine"/>
(3)    <rdfs:range rdf:resource="#WineGrape"/>
(4) </owl:ObjectProperty>
```

Figura 11 - Exemplo de propriedades em OWL

Para enriquecer o exemplo apresentado na Figura 11, podem-se expressar restrições sobre a cardinalidade de propriedades. O próximo exemplo (Figura 12) declara que vinhos (linha 1) são feitos (linha 5) a partir de, no mínimo, um tipo de uva (linha 6). Deve-se ressaltar a utilização de tipos de dados segundo esquemas XML (linha 6) na restrição da propriedade em questão.

```
(1)<owl:Class rdf:ID="Wine">
(2)  <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
(3)  <rdfs:subClassOf>
(4)    <owl:Restriction>
(5)     <owl:onProperty rdf:resource="#madeFromGrape"/>
(6)     <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
(7)    </owl:Restriction>
(8)   </rdfs:subClassOf>
(9) </owl:Class>
```

Figura 12 - Exemplo de cardinalidade das propriedades em OWL

A linguagem OWL fornece construtores para a definição de propriedades transitivas, simétricas, inversas, funcionais e funcionais inversas. Os trechos de ontologias em OWL ilustrados na Figura 13 de forma respectiva essas definições. A transitividade indica que se uma região de produção de vinho X está localizada em uma região Y e Y está localizada em uma região Z, então X está localizada na região Z (linhas 01 e 02). Simetria indica que a ordem das premissas não altera o resultado, ou seja, se uma região X produtora de vinho está próxima a uma região Y, então Y está próxima a X (linhas 06 e 07). Inversão indica que existe uma propriedade cuja semântica é oposta à de outra propriedade. Neste caso, se uma entidade X produz o vinho Y (linha 14), há uma propriedade inversa que indica que o vinho Y é produzido pela entidade X (linha 16). As propriedades funcionais são aquelas cujo valor é único para cada instância. Por exemplo, a propriedade que se refere à entidade produtora de vinho (linhas 11 e 12), já que um vinho não pode ter mais de uma entidade produtora. As propriedades funcionais inversas são aquelas cujo valor se aplica a uma única instância. Neste caso, o vinho Y é produzido por uma única entidade que possui identificação também única (linha 15).

```
(01) <owl:ObjectProperty rdf:ID="locatedIn">
(02)    <rdf:type rdf:resource="&owl;TransitiveProperty" />
(03)    <rdfs:domain rdf:resource="&owl;Thing" />
(04)    <rdfs:range rdf:resource="#Region" />
(05) </owl:ObjectProperty>
(06) <owl:ObjectProperty rdf:ID="adjacentRegion">
(07)    <rdf:type rdf:resource="&owl;SymmetricProperty" />
(08)    <rdfs:domain rdf:resource="#Region" />
(09)    <rdfs:range rdf:resource="#Region" />
(10) </owl:ObjectProperty>
(11) <owl:ObjectProperty rdf:ID="hasMaker" />
(12)    <rdf:type rdf:resource="&owl;FunctionalProperty" />
(13) </owl:ObjectProperty>
(14) <owl:ObjectProperty rdf:ID="producesWine">
(15)    <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
(16)    <owl:inverseOf rdf:resource="#hasMaker" />
(17) </owl:ObjectProperty>
```

Figura 13 - Exemplo de propriedades transitivas, simétricas, inversas, funcionais e funcionais inversas em OWL

Na Figura 14 é apresentada uma declaração em OWL (em conjunto com as declarações anteriores) que, se submetida a uma máquina de inferência para a linguagem OWL, é inferido que o elemento declarado (linha 1) é um líquido potável classificado como vinho, pois possui uma propriedade (linha 2) que é do domínio da classe de vinhos. Além disso, esse vinho chamado LindemansBin65Chardonnay tem propriedades que indicam sua entidade produtora, bem como a região na qual esta se encontra e regiões adjacentes.

```
(1) <owl:Thing rdf:ID="LindemansBin65Chardonnay">
(2)    <madeFromGrape rdf:resource="#ChardonnayGrape" />
(3) </owl:Thing>
```

Figura 14 - Exemplo de declaração OWL

Os exemplos apresentados desta seção foram baseados nos exemplos disponíveis no endereço eletrônico (Smith et al., 2004) e demonstram apenas alguns dos construtores utilizados no módulo OWL Lite, módulo este que já apresenta um número considerável de ferramentas disponíveis, como editores, validadores e máquinas de inferência.

2.5 Ferramentas para Manipulação de Ontologias

Existem muitas ferramentas para manipulação de ontologias, porém essas ferramentas foram utilizadas apenas para auxiliar o desenvolvimento do trabalho. Assim, elas são apresentadas sem entrar em muitos detalhes técnicos, ressaltando apenas as principais características relevantes ao projeto. Nesta seção é apresentado um resumo das ferramentas Protégé (Knublauch et al., 2004) (seção 2.5.1) e do KAON (Studer, 2001) (seção 2.5.2).

2.5.1 Protégé

O desenvolvimento do Protégé (Knublauch et al., 2004) teve início nos anos 80 pelo Departamento de Informática Médica da Universidade de Stanford, face às necessidades de ontologias médicas na época.

Em seu projeto original, o Protégé era uma ferramenta de aquisição de conhecimento limitada a um sistema especialista para oncologia, o Oncocin (Gennari et al., 2003). Ele foi se modernizando para, gradativamente, acompanhar a evolução de tecnologia dos sistemas baseados em conhecimento, e deve: servir para aquisição de conhecimento diretamente de especialistas de domínios, permitir diversos formalismos e estratégias de inferência, integrar tarefas (aquisição de ontologias e instâncias, ambiente de teste com inferência) num mesmo ambiente. Neste trabalho, ele foi utilizado como uma ferramenta gráfica para criação e manipulação de ontologias.

A equipe responsável pelo Protégé, ao perceber o potencial de desenvolvimento de seus usuários, optou por abrir seu código. Assim, surgiu uma arquitetura integrável a diversas aplicações, via componentes que podem ser conectados ao sistema (*plugins*). Como conseqüência desta decisão e de sua difusão, componentes de vários matizes, elaborados por grupos de pesquisa de usuários, puderam ser adicionados ao sistema, sem necessitar o redesenvolvimento. Dois exemplos de *plugins* que se integram ao Protégé são o Jess-Tab e o SWRL-Tab, que estão descritos nas Seções 2.6.2.1 e 2.6.2.2 respectivamente.

2.5.2 KAON

O grupo de ontologias da cidade de Karlsruhe, Alemanha, que congrega hoje o Instituto de Métodos Formais do Departamento de Economia, o Centro de Pesquisas em Informática (Forschungszentrum Informatik - FZI) e uma empresa, a Ontoprise, é um dos mais prolíficos em produção científica e prática na área de ontologias, tendo inclusive criado o primeiro protótipo do que viria a ser a Web semântica, o Ontobroker (Benjamins et al., 1998).

Este grupo desenvolveu um pacote de ferramentas gratuitas e de código fonte aberto para ontologia e Web semântica chamado KAON (*the KArlsruhe ONtology and semantic web tool suite*) (Studer R., 2001), e ferramentas comerciais para a aplicação de ontologias em comércio eletrônico, gestão de conhecimento em empresas e Web semântica. Entre elas estão um editor de ontologias chamado de OntoEdit, um servidor de ontologias baseado em bancos de dados, ferramentas para criação de ontologias a partir de texto chamado de Text-to-Onto (Maedche & Staab 2001), para busca sobre bases de texto baseada em ontologias chamada de SemanticMiner (Ontoprise, 2003), para anotação semi-automática de referências a ontologias em páginas para a Web, e para agrupamento de textos baseados em ontologias.

Apesar do KAON ser uma boa ferramenta para edição de ontologias ele não oferece de forma integrada outras ferramentas necessárias para o desenvolvimento deste trabalho, como a SWRL-Tab do Protégé.

2.6 Linguagens de Regras e Máquinas de Inferência

Regras são elementos fundamentais no contexto da Web Semântica, pois podem permitir a integração, derivação e transformação de dados dentro de múltiplas fontes distribuídas, de maneira transparente e escalável. Regras, por si só, podem ser tratadas como dados, publicadas na web, e quando URIs são usados como símbolos-constantes numa linguagem de regras, elas podem formar *links* úteis entre bases de conhecimento. Em um ambiente de Serviços Web, regras oferecem a oportunidade para ativar a automação da execução e composição de políticas que governam a entrega de informação, o acesso para serviços, ou a execução de processos.

As regras possuem vantagens como flexibilidade e maneabilidade. Além disso, a natureza declarativa das regras lhes cede uma atração especial a dispositivos de representação de conhecimento em ambientes distribuídos, baseados na Web, onde elas podem ser armazenadas, especificadas e administradas a partir de um único lugar, e aplicadas em muitos outros lugares. Porém, isto requer uma forma padrão para representar regras de forma não ambígua para publicação e intercâmbio de propósitos.

As principais tentativas de linguagens de regras para a Web Semântica são a RuleML (*Rule Markup Language*) (Boley et al., 2001), que utiliza os padrões RDF e XML, e a SWRL (*Semantic Web Rule Language*) (Horrocks et al., 2004), que utiliza a sintaxe de OWL. Porém, a SWRL tem se destacado por ser mais completa e apresentar integração com o OWL, apesar de ainda ser apenas uma proposta de linguagem de regras para a Web Semântica. O W3C ainda não recomenda nenhuma linguagem de regras. Na Seção 2.6.1 é apresentada a linguagem SWRL e, a seguir, na Seção 2.6.2 é discutido brevemente sobre máquinas de inferência para as linguagens de regras.

2.6.1 Linguagem de Regras para Web Semântica (SWRL)

A Linguagem de Regras para Web Semântica (SWRL - *Semantic Web Rule Language*) (Horrocks et al., 2004) é baseada na combinação entre OWL DL e OWL Lite, sub-linguagens de OWL, e *Unary/Binary Datalog RuleML*, sub-linguagens de RuleML (RuleML, 2000). A SWRL estende um conjunto de axiomas da OWL. Uma sintaxe abstrata de alto-nível é provida para estender a sintaxe abstrata da OWL descrita em OWL *Semantics and Abstract Syntax Document* (Peter et al., 2004).

2.6.1.1 Regras SWRL

A sintaxe abstrata de uma ontologia OWL contém uma sucessão de axiomas e fatos. Axiomas podem ser de vários tipos, por exemplo, axiomas subClass e axiomas de equivalentClass. A SWRL propõe estender isto com axiomas de regra.

Um axioma de regra consiste em um antecedente (corpo) e um conseqüente (cabeça), que por sua vez consistem em um conjunto de átomos (este conjunto pode ser

vazio). Informalmente, uma regra pode ser definida como: quando o antecedente for verdadeiro, então o conseqüente deverá ser "executado".

A sintaxe abstrata da OWL não é facilmente lida por humanos. Para uma melhor compreensão, será utilizada aqui, uma sintaxe relativamente compreensível por humanos, que pode ser representada com a notação ilustrada na Figura 15.

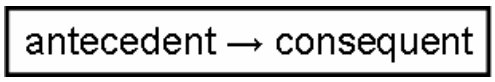

Figura 15 - Exemplo de notação para SWRL

Em comum com muitas outras linguagens de regra, as regras representadas na SWRL, são escritas na forma de pares antecedente-conseqüente. Na terminologia da SWRL, o antecedente é referido como o corpo da regra e o conseqüente é referido como a cabeça da regra. A cabeça e o corpo consistem na conjunção de um ou mais átomos. No momento, a SWRL não suporta combinações lógicas mais complexas do que átomos.

As regras descritas com a SWRL argumentam sobre indivíduos da OWL, principalmente em termos de classes OWL e propriedades. Por exemplo, uma regra representada na SWRL que expressa que uma pessoa (*person*), que é filha do mesmo pai de outra pessoa (*sibling*) do sexo masculino (*male*), tem um irmão (*brother*). Para este exemplo, os conceitos de 'pessoa' e 'macho' são capturados de uma classe da OWL chamada Person com uma subclasse chamada Man; as relações 'filho do mesmo pai' e 'irmão' são expressas utilizando as propriedades hasSibling e hasBrother da OWL, que são anexadas a Person. Esta regra representada em SWRL seria como ilustrado na Figura 16.

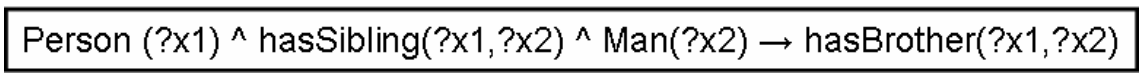

Figura 16 - Exemplo de regra SWRL

A execução desta regra teria o efeito de fixar a propriedade hasBrother a x2 no indivíduo que satisfaz a regra, nomeado x1.

As regras SWRL também podem se referir explicitamente a indivíduos da OWL. Por exemplo, de forma variante ao anterior, é possível deduzir que um indivíduo em particular, chamado Fred, tem um irmão, como na Figura 17.

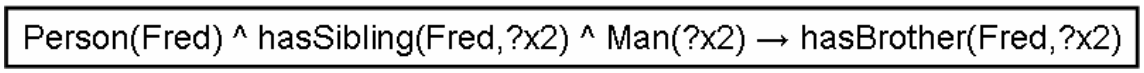

Figura 17 - Exemplo de regra SWRL

A SWRL também suporta dados literais. Por exemplo, assumindo um indivíduo que possui a propriedade hasAge, é possível perguntar se Fred tem um irmão com 40 anos de idade, como pode ser visto na Figura 18.

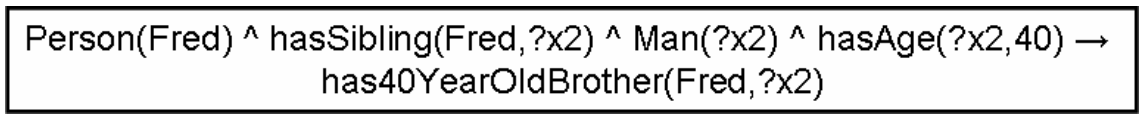

Figura 18 - Exemplo de regra SWRL

Também são aceitas pela SWRL seqüências literais entre aspas simples, bem como os conceitos mesmo-que (*same-as*) e diferente-de (*different-from*). Por exemplo, o átomo SWRL sameAs pode determinar se dois indivíduos da OWL, Fred e Frederick, são o mesmo indivíduo, ilustrado na Figura 19.

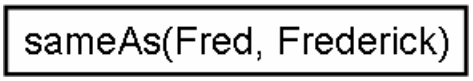

Figura 19 - Exemplo do átomo sameAs em SWRL

De forma similar, o átomo SWRL differentFrom pode ser usado para expressar se dois indivíduos da OWL não são o mesmo.

A SWRL também tem um átomo para determinar se um indivíduo, propriedade ou variável, é de algum tipo em particular. No exemplo ilustrado na Figura 20 é determinado se a variável x é do tipo unsigned integer.

Figura 20 - Exemplo de definição de tipos em SWRL

Este átomo SWRL invoca um intervalo de dados (*data range*), mas quando precedido do espaço de nomes xsd:, o tipo especificado dever ser um tipo de dados do Esquema XML. Outra forma de representar um intervalo de dados com um átomo SWRL que expressa uma relação, pode ser visto na Figura 21, onde o átomo SWRL indica que a variável x pode assumir os valores 3, 4 ou 5.

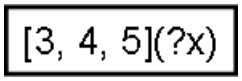

Figura 21 - Exemplo de como indicar os valores possíveis de uma variável SWRL

A SWRL também aceita uma gama de predicados embutidos (*Built-in*) que ampliam seu poder expressivo. Os predicados embutidos da SWRL aceitam vários argumentos. Eles são descritos em mais detalhes em SWRL *Built-in Specification* (2004). O predicado embutido mais simples é a operação de comparação. Por exemplo, o predicado embutido greaterThan determina se um indivíduo tem um irmão mais velho, como pode ser visto na Figura 22.

```
hasBrother(?x1,?x2) ^ hasAge(?x1,?age1) ^ hasAge(?x2,?age2) ^
  swrlb:greaterThan(?age2,?age1) → hasOlderBrother(?x1,?x2)
```

Figura 22 - Exemplo de predicados embutidos em regras SWRL

Todo predicado embutido da SWRL deve ser precedido pelo espaço de nomes 'swrlb:.'. A SWRL também suporta predicados embutidos matemáticos, como no exemplo a seguir, onde a regra determina se um indivíduo tem exatamente 10 anos de idade a mais do que seu irmão, como na Figura 23.

```
hasBrother(?x1,?x2) ^ hasAge(?x1,?age1) ^ hasAge(?x2,?age2) ^
swrlb:subtract(10,?age2,?age1) → hasDecadeOlderBrother(?x1,?x2)
```

Figura 23 - Exemplo de predicados embutidos matemáticos em regras SWRL

Além dos predicados matemáticos existem predicados embutidos para cadeias, datas e listas. Futuramente, poderão ser adicionados mais predicados embutidos na descrição da SWRL (Horrocks et al., 2004).

2.6.2 Máquinas de Inferência sobre Regras SWRL

De acordo com o dicionário Houaiss[1], inferência é uma "operação intelectual por meio da qual se afirma a verdade de uma proposição em decorrência de sua ligação com outras já reconhecidas como verdadeiras". Por meio da inferência, é possível tornar explícita uma informação que ainda não estava escrita claramente, mas que pode ser derivada a partir de outras já conhecidas.

Atualmente existem poucas opções para realizar inferência sobre regras SWRL. O Projeto SweetRules (Grosof et al., 2005) consiste em um conjunto integrado de ferramentas para trabalhar com ontologias e regras na Web Semântica, o que inclui uma

[1] http://houaiss.uol.com.br/

máquina de inferência capaz de realizar inferência de forma integrada e consistente em regras SWRL. Apesar do SweetRules ser um projeto de código aberto, algumas ferramentas utilizadas para realizar inferência em SWRL são proprietárias e de uso não gratuito, o que dificulta sua utilização neste projeto.

Outra opção para fazer inferência sobre regras SWRL é o Editor SWRL (SWRL Tab) do Protégé (Knublauch et al., 2004). Segundo O'Connor et al. (2005), o editor possui APIs que permitem a integração com diversas máquinas de inferência, mecanismo que é suportado por um subsistema denominado SWRL *Factory*. O editor SWRL foi integrado com a máquina de inferência do Jess (Seção 2.6.2.1) (O'Connor et al., 2005), com isso, as regras SWRL são traduzidas para regras Jess e a máquina da inferência do Jess é utilizada para realizar a inferência.

Além do SweetRules e do Editor SWRL do Protege, que trabalha em conjunto com a máquina de inferência Jess (SWRL Tab), também foi feito um estudo detalhado sobre a Bossam (Jang and Sohn., 2004), uma máquina de inferência capaz de manipular tanto OWL quanto SWRL.

Nas Subseções a seguir serão apresentados de forma mais detalhada a máquina de inferências Jess, da SWRL Tab do Protégé e a máquina de Inferência para Web Semântica Bossam (Jang & Sohn, 2004).

2.6.2.1 Jess

O Jess (Java *Expert System Shell*) é um mecanismo de regras e ambiente de interpretação de *scripts* escrito na linguagem JAVA por Ernest Friedman-Hill, do Sandia National Laboratories, em Livermore, na Califórnia (Friedman-Hill, 1997). Apesar do Jess não ter seu código fonte aberto, há uma licença acadêmica gratuita por dois anos.

O Jess originalmente inspirado pelo CLIPS *Expert System Shell* e pela linguagem CLIPS (C *Language Integrated Production System*) (NASA, 1985) é uma ferramenta para desenvolvimento produtivo de sistemas especialistas. Ele provê um ambiente completo para a construção de regras e sistemas especialistas baseados em objetos. Como o CLIPS, o Jess também usa o algoritmo *Rete* ("rede" em latim) (Forgy, 1982) para processar regras, cuja idéia principal é melhorar a velocidade de sistemas de regra

de encadeamento para frente (*forward-chained rule systems*), limitando o esforço requerido para recomputar o conjunto de conflitos depois que uma regra seja disparada. Porém, o Jess também acrescenta muitas características ao CLIPS, incluindo encadeando para trás (*backwards chaining*), consultas em memória, e habilidade para manipular e argumentar diretamente sobre objetos Java. Em resumo, o Jess cresceu e se tornou um ambiente completo, distinto, dinâmico, com capacidades poderosas de interfaceamento com Java.

Em Jess, uma regra consiste em *Left-Hand Side* (LHS) e em *Right-Hand Side* (RHS), nesta ordem. O LHS é composto por zero ou mais elementos condicionais, enquanto o RHS consiste em zero ou mais chamadas de funções. Um elemento condicional é qualquer padrão, ou um agrupamento, construído com “and”, “or”, ou “not”. Os elementos condicionais são comparados na memória de trabalho do Jess, quando todos eles se emparelham, o código na RHS da regra será executado. Essencialmente, uma função Jess é um método Java que é chamado como um procedimento.

Neste trabalho o Jess foi utilizado para realizar testes com regras SWRL criadas no Protege. Mais detalhes de como isso foi possível, está descrito na Seção 2.6.2.2.

2.6.2.2 SWRL Tab

A SWRL Tab é um ambiente de desenvolvimento para trabalhar com regras SWRL no Protégé-OWL. Ela suporta editar e executar regras SWRL. Ela também prove mecanismos que permitem a interoperabilidade entre diversas máquinas de inferência sobre regras e incorpora as bibliotecas definidas pelo usuário que podem ser utilizadas nas regras. Diversas bibliotecas são disponibilizadas. Essas bibliotecas incluem métodos para manipulação de *strings* e funções matemáticas.

Vários componentes estão disponíveis, como:

- **SWRL *Editor*:** dá suporte a edição e pode salvar regras SWRL em ontologias OWL;
- **SWRL *Factory*:** provê uma API Java para manipulação de regras SWRL em ontologias OWL;

- **SWRL *Bridge*:** provê a infra-estrutura necessária para incorporar mecanismos de regras dentro do Protégé-OWL para executar regras SWRL;
- **SWRL Jess *Bridge*:** incorpora a máquina de Inferências Jess ao Protégé-OWL, ela é chama de SWRLJessTAB;
- **SWRL *Built-in Bridge*:** são predicados que aceitam um ou mais argumentos. Estes predicados podem ser utilizados nas regras SWRL;
- **SWRL *Built-in Libraries*:** incluem as *bult-in* (predicados embutidos) da definição do SWRL e do OWL;
- **SWRL *Query* Tab:** provê uma interface gráfica para visualizar o resultado das regras SWRL;
- **SWRL *Query* API:** provê uma API Java para realizar as consultas.

Neste trabalho foram utilizados dois componentes da SWRL Tab. O Editor SWRL, que foi utilizado para criar as regras SWRL dentro da Ontologia, e a SWRLJessTab que foi utilizado para testar as regras.

2.6.2.3 Bossam

Bossam (Jang & Sohn, 2004) é um mecanismo de inferência para Web Semântica com suporte nativo a OWL, SWRL e RuleML. Ele é um sistema de regras RETE-Based (Algoritmo eficiente para produção de regras) que utiliza o método *forward chaining* para realizar inferências.

Com a API Java disponível é possível utilizar o Bossam de maneira simples possibilitando uma fácil integração com outras aplicações. Além disso, existe uma interface de interação via linha de comando. Para realizar inferência é necessário utilizar uma linguagem própria chama Buchingae (Jang, 2005). A Buchingae possibilita que as consultas à Ontologia sejam realizadas de foram muito simples. Esta linguagem também possibilita a produção de regras, porém neste trabalho as regras foram produzidas em SWRL.

Com esta máquina de inferências sobre SWRL e OWL foi possível inferir todas as respostas necessárias com simplicidade e agilidade, com total integração à aplicação Java construída.

2.7 Considerações Finais

Neste capítulo foi feita uma apresentação sobre as tecnologias necessárias para viabilizar a Web Semântica e que foram relevantes ao desenvolvimento deste trabalho. Todas as informações contidas neste capítulo foram muito importantes durante a extensão da Ontologia que contém as informações sobre os Componentes presentes na aplicação final desenvolvida.

A partir dos estudos descritos neste capítulo, foi possível perceber que todas as tecnologias relacionadas à Web Semântica ainda são muito recentes e podem sofrer mudanças até que sejam utilizadas em larga escala. Porém, como o sistema desenvolvido não visa manipular de forma direta as linguagens de descrição de ontologias, futuramente ele poderá ser facilmente adaptado às novas tecnologias, trocando-se apenas a máquina de inferência.

3 Servidores de Aplicações e Componentes de Software

A reutilização de componentes de software é uma tendência atual e que tem evoluído significativamente ao longo da história da computação. O desenvolvimento e uso de componentes em níveis de abstração cada vez mais elevados têm resultado em grandes melhorias na produtividade e qualidade do desenvolvimento de software (Tetlow et al., 2005; Ousterhout, 1998).

Em um Servidor de Aplicações Baseado em Ontologias, que será discutido na Seção 3.4, os componentes de software têm suas propriedades e comportamentos relacionados de forma semântica dentro de ontologias. Como uma ontologia é um modelo conceitual explícito, com semântica formal baseada em lógica, suas descrições dos componentes podem ser consultadas e inferidas, melhorando sua performance através da combinação dos componentes apropriados e diminuindo dependências dos componentes, pois fica mais fácil substituir os componentes sem a necessidade de reprogramação do sistema.

Neste capítulo serão apresentados os conceitos sobre Servidores de Aplicações (Seção 3.2), mas para entender melhor o que é um Servidor de Aplicações primeiro será apresentado o conceito de *Middleware* (Seção 3.1). Também será apresentado o conceito de Serviços Web (Seção 3.3) e sua ligação com um Servidor de Aplicações. Logo a seguir, na Seção 3.4, será apresentado o conceito de Servidores de Aplicações Baseados em Ontologias (OBAS). Para finalizar o capítulo será feita uma breve discussão sobre Componentes de Software (Seção 3.5) e serão apresentados os componentes J2EE (Seção 3.5.1) e .NET (Seção 3.5.2).

3.1 Middleware

A portabilidade e a interoperabilidade entre os sistemas computacionais são problemas crucias para as empresas poderem desenvolver seus negócios. Atualmente, na maior parte das empresas, existe uma grande variedade de sistemas de informação

heterogêneos, tanto no lado dos servidores como no dos clientes. Geralmente esses sistemas possuem interfaces e sistema operacionais diferentes e até funcionam em redes distintas. Esses fatores causam maior dificuldade na integração entre os sistemas das empresas.

Uma solução utilizada para resolver este tipo de problema é conhecida pelo nome de *middleware*. Este consiste em um sistema distribuído de serviços, com interfaces e protocolos padronizados, que busca solucionar problemas de heterogeneidade e interoperabilidade entre produtos de diferentes fabricantes. O *middleware* é genérico, suporta necessidades de uma grande variedade de aplicações e indústrias e é executado em múltiplas plataformas (Bernstein, 1996).

Um *middleware* pode ser definido de uma forma simples como um software de interoperabilidade que permite a interação entre aplicações através de uma rede de computadores. Segundo Mahmoud (2004): "*Middleware* é uma camada de software distribuída que se situa sobre o sistema operacional de rede, abaixo da camada de aplicação, abstraindo a heterogeneidade do ambiente que está por baixo. Isto provê um ambiente distribuído integrado cujo objetivo é simplificar a tarefa de programar e administrar aplicações distribuídas e também prover serviços de valor agregado como nomear e negociar para ativar desenvolvimento de aplicação distribuído. Isto é, sobre a integração e interoperabilidade de aplicações e serviços que executam em computadores heterogêneos e dispositivos de comunicações."

A história do *middleware* atravessa mais de duas décadas. Diferentes tipos de *middleware* foram desenvolvidos para tentar resolver a crescente complexidade dos sistemas de informação distribuídos. Por exemplo, *Remote Procedure Calls*, *Transaction Monitors*, *Object Brokers*, *Object Monitors* e *Message Oriented Middleware* (Alonso et al., 2003). Neste trabalho, vamos nos concentrar apenas nos *middleware* voltados para o desenvolvimento de Aplicações Web.

3.2 Servidores de Aplicações

O uso crescente da Web como um canal para acessar sistemas de informação forçou as plataformas *middleware* a prover apoio ao acesso a Web. Este apoio é provido tipicamente na forma de Servidores de Aplicação.

Servidores de aplicações compreendem as funcionalidades de plataformas de *middleware* convencionais e, além disso, eles incorporam a Web como um canal de acesso fundamental para as funcionalidades implementadas usando o *middleware*. Incorporar a Web como um canal de acesso tem várias implicações importantes, a mais significante é que a lógica de apresentação da aplicação adquire um papel muito mais pertinente do que em um *middleware* convencional. Esta é uma conseqüência direta de como o protocolo HTTP e a Web trabalham, onde todas as formas de troca de informação ocorrem através de documentos. Preparar, gerar dinamicamente, e administrar esses documentos, constituem as principais exigências para ser reconhecido como um Servidor de Aplicação que, além disso, precisam dar uma resposta (Alonso et al., 2003).

As principais funcionalidades de um Servidor de Aplicações podem ser descritas examinando-se dois *frameworks* concorrentes, o J2EE da Sun e o .NET da Microsoft. Ambos são semelhantes em termos de funcionalidades e se classificam como um *middleware* convencional, mas oferecem APIs para processar XML, geração dinâmica de páginas Web e suporte a Serviços Web. Serviços foram adicionados para permitir balanceamento de carga, agrupamento e *caching* como também administração e aspectos de segurança. Eles também promovem a engenharia de software baseada em componentes, como o *Enterprise JavaBeans* (EJB) no J2EE e o *Component Object Model* (COM) no .NET. O Servidor de Aplicação de código fonte aberto JBoss (Fleury & Reverbel, 2003) é um exemplo que implementa a estrutura do J2EE. Uma lista exaustiva de comparação de Servidores de Aplicação pode ser encontrada em Application Server Matrix (2005).

3.3 Serviços Web (*Web Services*)

O aparecimento do paradigma *Service Oriented Architecture* (SOA) (Artus, 2006) foi provocado pelo desejo de terceirizar o desenvolvimento de TI e serviços, como também pelo barateamento da largura de banda da rede. Os sistemas distribuídos baseados em SOA dividem as funcionalidades entre uma relação flexível entre dois sistemas computacionais (*loosely-coupled*) e serviços independentes. O benefício de um sistema *loosely-coupled* encontra-se em sua agilidade e em sua habilidade de sobreviver a

mudanças evolucionárias na estrutura e na implementação de cada serviço que compõe a aplicação inteira.

O *middleware* baseado na Web para tais métodos é chamado de Serviços Web que, usando um conjunto de protocolos e linguagem XML para descrição de interface, fazem descoberta e composição de serviços. De fato, padrões já estão disponíveis para a descrição de interface (*Web Service Description Language* - WSDL) (Christensen et al., 2001), de chamada (*invocation*) (*Simple Object-based Access Protocol* - SOAP) (Gudgin et al., 2003) e descoberta (*Universal Description Discovery & Integration* - UDDI) (Bellwood, 2004). Existem vários esforços que competem para esta composição. Também existem outros aspectos como, por exemplo, *WS-Security*, *WS-Policy*, *WS-Trust* e muitos outros, que formam um conjunto rotulado por WS*. Os WS*, são especificações adicionais para estender as capacidades dos Serviços Web, podendo existir mais de uma extensão com o mesmo propósito, porém implementadas de maneira diferente. Esses aspectos não serão discutidos porque fogem ao escopo deste trabalho.

Os Serviços Web poderiam complementar componentes que residem em Servidores de Aplicação distribuídos em locais físicos diferentes, organizações e plataformas heterogêneas. Como os Servidores de Aplicações provêem a infra-estrutura básica (servidor Web, XML *parsers*, etc.), eles são um alvo para incorporar estas funcionalidades. Por exemplo, no JBoss, o desenvolvedor só precisa marcar o código fonte com uma *meta-tag* que o JBoss gera a descrição WSDL automaticamente e controla as mensagens SOAP.

Além disso, os Serviços Web podem ser compostos por unidades maiores, incluindo outros Serviços Web. Este suporte é provido pelos *Workflow Execution Engines*, normalmente implementados em cima de Servidores de Aplicações, por exemplo, o JBoss oferece o jBpm (*Java Business Process Management*). Foram propostas várias linguagens independentes de plataforma para definir um *WorkFlow* de modo declarativo, como por exemplo, o *Business Process Execution Language for Web Services* (BPEL) (Andrews et al., 2003).

Apesar deste relacionamento próximo, projetar e desenvolver sistemas distribuídos baseados em SOA, e portar as aplicações já existentes na Web, não é um assunto muito simples. Vários aspectos como transações e segurança, tipicamente providos por

Servidores de Aplicações em ambientes distribuídos fechados são desafiados pelos ambientes de natureza descentralizada e anônima da Web.

Nas Seções 3.3.1, 3.3.2 e 3.3.3 é feita uma breve apresentação dos protocolos padrões para Serviços Web.

3.3.1 SOAP

O protocolo SOAP foi inicialmente sigla para *Simple Object Access Protocol*, posteriormente foi alterado para *Service Oriented Architecture Protocol* e atualmente é simplesmente chamado de SOAP.

SOAP é um protocolo para troca de mensagens baseadas em XML sobre uma rede de computadores, normalmente utilizado sobre os protocolos HTTP e HTTPS. SOAP é uma camada fundamental para os Serviços Web. Originalmente o protocolo foi criado pela Microsoft, porem atualmente sua especificação é mantida pelo W3C.

A estrutura de um envelope SOAP pode ser visto na Figura 24.

```
<?xml version='1.0' ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/Soap-envelope">
  <env:Header>
  ...
  </env:Header>
  <env:Body>
  ...
  </env:Body>
 </env:Envelope>
```

Figura 24 - Envelope SOAP

O cabeçalho (*Header*) é opcional e geralmente utilizado para passar informações de controle, que não são diretamente ligadas à requisição. O corpo da mensagem (*Body*) é obrigatório e contém as informações da mensagem. E o envelope encapsula o cabeçalho e o corpo.

Um exemplo real de requisição SOAP pode ser vista na Figura 25.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns1="http://www.webserviceX.NET/">
 <SOAP-ENV:Body>
  <ns1:ConversionRate>
   <ns1:FromCurrency>USD</ns1:FromCurrency>
   <ns1:ToCurrency>BRL</ns1:ToCurrency>
  </ns1:ConversionRate>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 25 - Exemplo de requisição SOAP

Nesta requisição SOAP, vê-se que o cabeçalho não foi utilizado e a mensagem contida no corpo do envelope utiliza a função ConversionRate com os parâmetros FromCurrency e ToCurrency com os valores USD e BRL respectivamente.

Na Figura 26 é apresentada a mensagem SOAP de resposta à requisição ilustrada na Figura 25.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <soap:Body>
  <ConversionRateResponse xmlns="http://www.webserviceX.NET/">
   <ConversionRateResult>1.9618</ConversionRateResult>
  </ConversionRateResponse>
 </soap:Body>
</soap:Envelope>
```

Figura 26 - Resposta a uma requisição SOAP

Nesta requisição pode-se ver que ConversionRateResponse é a resposta para o método ConversionRate e ConversionRateResult é o valor esperado.

3.3.2 WSDL

A linguagem WSDL (*Web Services Description Language*), é uma linguagem para descrição de Serviços Web. A WSDL descreve a interface pública dos serviços, e é uma linguagem baseada em XML, sendo uma combinação entre SOAP e XML Schema (utilizado para definir os tipos de dados).

Em WSDL os serviços são descritos como uma coleção de *ports*. Uma *port* é definida como uma coleção de endereços de rede com ligações reusáveis.

O WSDL foi criado pela Microsoft, mas a partir da versão 2.0, versão atual, o WSDL passou a ser mantido pelo W3C.

A estrutura da WSDL é dividida em quatro partes:

- <portType>: Operações executais pelos serviços;
- <message>: Mensagem utilizada pelo Serviço Web;
- <types>: Tipo de dados utilizado pelo Serviço Web;
- <binding>: Protocolo de comunicação utilizado pelo Serviço Web.

O documento que descreve o Serviço Web utilizado na Seção 3.3.1 encontra-se disponível em http://www.webservicex.net/CurrencyConvertor.asmx?WSDL.

Para facilitar a compreensão do WSDL, na Figura 27 é apresentado um exemplo simplificado.

```
<message name="getTermRequest">
 <part name="term" type"xs:string"/>
</message>

<message name="getTermResponse">
 <part name="value" type"xs:string"/>
</message>

<portType name="glossaryTerms">
 <operation name="getTerm">
  <input message="getTermRequest">
  <output message="getTermResponse">
 </operation>
</portType>
```

Figura 27 - Exemplo de documento WSDL

O item "glossaryTerms" é nome de um Port, "getTerm" é o nome da operação, esta operação recebe a mensagens chamada "getTermRequest" e responde a mensagem chamada "getTermResponse". O elemento <message> define as partes de cada mensagem e os tipos associados. Comparando com o método tradicional de programação "glossaryTerms" é uma coleção de funções, onde "getTerm" é uma função que tem "getTermRequest" como parâmetro de entrada e "getTermResponse" como parâmetro de saída (retorno).

3.3.3 UDDI

O UDDI (*Universal Description, Discovery and Integration*) é um registro utilizado para publicar, listar e encontrar Serviços Web, baseado em XML e independente de plataforma. Não é um padrão da W3C, porém é um padrão para Serviços Web criado pela indústria.

UDDI é divido em 3 componentes:

- ***White Pages***: endereço, contato e identificadores conhecidos;
- ***Yellow Pages***: categorização industrial baseada taxonomias padrões;
- ***Green Pages***: informações técnicas sobre os serviços expostos pelo negócio.

Os tipos de dados encontrados no UDDI são:

- **businessEntity**: estrutura de nível mais alto, descreve um negócio ou outra entidade registrada;
- **businessService**: descreve um conjunto de serviços que podem conter um ou mais bindingTemplate;
- **bindingTemplate**: informações necessárias para invocar um serviço especifico, que podem abranger mais de um protocolo (ex.: HTTP, SMTP, ...);
- **tModel**: "impressão digital" para identificar outras entidades ou até outros tMdels, funciona como espaço de nomes.

Os servidores que dão suporte a UDDI são chamados de *nodes* e um conjunto de *nodes* é chamado de *Registry*. Apesar do UDDI não ser um padrão W3C existe grande quantidade de clientes e servidores para diversas plataformas.

3.4 Servidores de Aplicação Baseados em Ontologia

Em um Servidor de Aplicações Baseado em Ontologias (OBAS - *Ontology Based Application Server*), uma ontologia armazena as propriedades e comportamentos de componentes e os relacionamentos entre eles, utilizando para isso uma linguagem formal baseada em lógica. As características dos componentes podem sofrer inferência e podem ser consultadas e permitir, por exemplo, que determinadas ações sejam previstas, que seja possível verificar inconsistências em tempo de execução, e assim, melhorar e facilitar a composição de aplicações (Oberle et al., 2005).

Oberle et al. (2005) discute as vantagens de um OBAS para o gerenciamento e reutilização de componentes. Os autores discutem sobre como poderia ser projetada a arquitetura de um OBAS. Na Figura 28 é ilustrada a arquitetura proposta por eles.

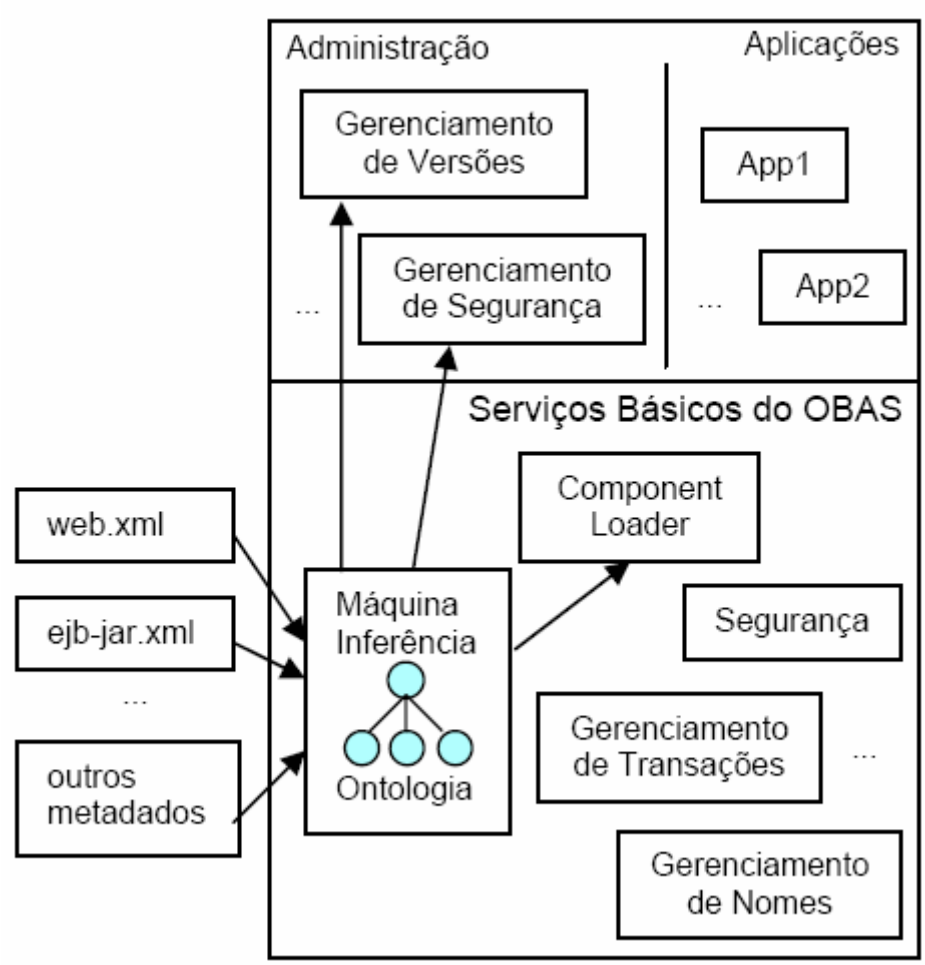

Figura 28 - Arquitetura para OBAS proposta por Oberle et al (2005)

De acordo com a Figura 28, metadados semânticos e uma ontologia são carregados em uma máquina de inferência. Os serviços e as ferramentas podem tirar proveito da capacidade de inferência embutida no Servidor de Aplicação.

No lado esquerdo da Figura 28 estão exemplificadas potenciais fontes de dados, as quais podem incluir arquivos contendo metadados, arquivos de configuração do Servidor de Aplicações, etc. Essas informações passam por um *parser* e são convertidas em metadados semânticos, isto é, em metadados que estejam de acordo com uma ontologia pré-definida. Dessa maneira, esses dados estão agora disponíveis e em conformidade com um modelo conceitual, o que possibilita integrar aspectos que antes eram separados, tais como segurança, dependência entre componentes e informações de desenvolvimento. Os metadados semânticos e a ontologia servem de entrada para uma máquina de inferência, a qual está embutida no próprio Servidor de Aplicação. A capacidade de inferência pode ser usada por um conjunto de ferramentas e serviços em tempo de desenvolvimento e de execução.

Ainda na Figura 28, pode-se observar que a arquitetura proposta por Oberle et al. (2005) é também apropriada para aplicações corporativas, pois inclui serviços como gerenciamento de transações e segurança. Mas, é importante salientar que, o que caracteriza um OBAS, é a presença de uma ontologia com o objetivo de integrar os serviços em um Servidor de Aplicações.

Oberle et al. (2005) implementou parte da arquitetura proposta por ele, com exemplos de como uma aplicação multimídia envolvendo componentes de software pode se beneficiar do uso de uma ontologia no Servidor de Aplicação.

3.5 Componentes de Software

Não há uma definição consensual sobre o conceito de componentes de software. Lind (2001) usa a seguinte definição para o termo: “Um pedaço de software auto-contido, com interfaces bem definidas, que pode ser instalado e entregue de maneira independente, pode ser facilmente combinado com outros componentes, o que é fundamental para sua utilidade”.

A definição utilizada por Krutisch et al. (2003), é a seguinte: “Um componente é um pacote coerente de software que: pode ser desenvolvido e entregue de maneira independente; possui a especificação explícita de interfaces para os serviços que fornece; possui a especificação explícita de interfaces para os serviços que espera de outros; e pode ser composto de outros componentes, talvez customizando algumas de suas propriedades, sem modificar os componentes em si.”

As definições do termo componente de software concordam que um componente deve possibilitar grande reutilização de código, o que, conseqüentemente, reduz os custos de implementação. Esse fato tem levado os componentes a ganharem notoriedade. Atualmente, existem desenvolvedores que se dedicam exclusivamente à criação de componentes, que posteriormente podem ser comprados e utilizados pelos mais diversos usuários, de acordo com as necessidades individuais de cada um (Deitel & Deitel, 2000).

Um componente oferece serviços aos usuários por meio da sua interface. A interface de um componente é composta por um conjunto de operações que determinam quais são os serviços que um determinado componente pode oferecer. Pode-se dizer que

os componentes possibilitaram a criação de um novo estilo de programação, pois eles normalmente são usados em conjunto com as linguagens script.

Neste trabalho, como o principal objetivo foi a ativação de componentes, não foi adotada uma definição específica para componentes. A idéia é de que o ativador, desenvolvido por meio do sistema protótipo descrito neste trabalho, possa ativar qualquer elemento de software que possa ser descrito e contenha uma interface especificada.

3.5.1 Componentes J2EE

A atualmente o termo J2EE esta começando a cair em desuso. A própria Sun, que propôs a especificação J2EE, sugeriu esta mudança. Antes o nome completo era J2EE versão 1.4. Agora a próxima versão que seria a versão 1.5 passa a se chamar JEE 5. Porém, por ser mais difundida, neste trabalho é utilizada a sigla J2EE.

O ambiente de execução dos programas Java (máquina virtual) é portável para diferentes plataformas. Assim, programas escritos em uma plataforma podem ser executados em qualquer outra que forneça o ambiente necessário. No modelo de componentes de Java, conhecido como J2EE (Armstrong, 2003), os componentes são todos desenvolvidos para serem executados pela máquina virtual Java e ficam localizados em um *container*, que fornece acesso aos serviços do ambiente do servidor J2EE. Cada tipo de componente tem seu *container*, o que poupa o desenvolvedor de escrever código para lidar com transações, gerenciamento de estado, *multithreading*, agrupamento (*pooling*) de recursos, etc. Pode-se dizer que os *containers* são a interface entre um componente e a funcionalidade de baixo nível específica da plataforma que o suporta.

Aplicações J2EE são construídas com a utilização de diferentes componentes. É importante ressaltar que os Java Beans, apesar de serem componentes Java, não são considerados componentes do modelo J2EE. A seguir são apresentados os tipos de componentes da especificação J2EE:

- Aplicações cliente e *Applets* são componentes cliente.
- Java *Servlets* e *Java Server Pages* (JSP) são componentes Web, estes são executados no servidor.

- *Enterprise Java Beans* (EJB) são componentes de negócios, o EJB também é executado no servidor.

Na Figura 29 são ilustradas as camadas e os componentes da arquitetura J2EE. Logo a seguir são explicados com mais detalhes os componentes J2EE, de acordo com (Pawlan, 2001), ilustrados na Figura 29.

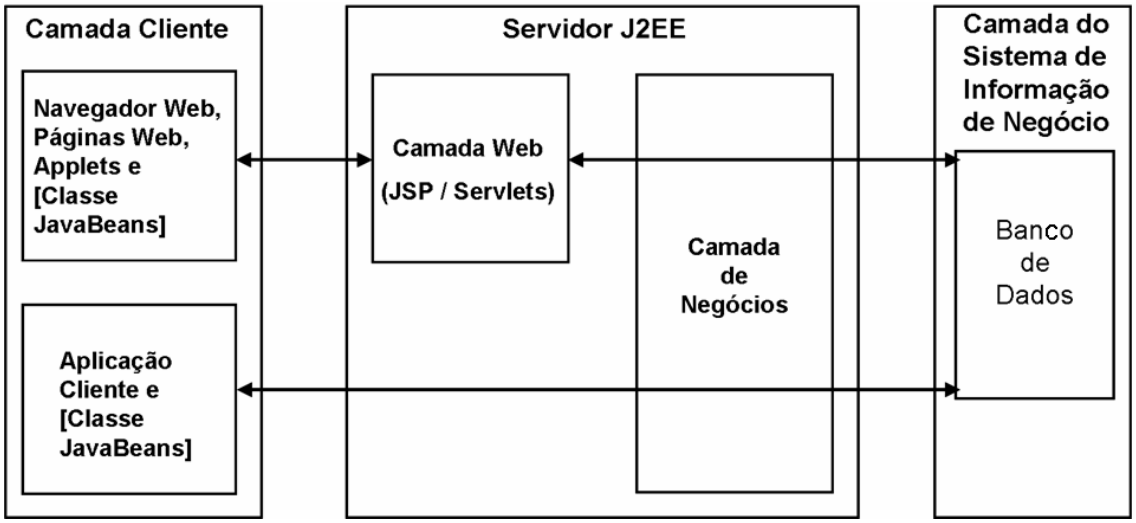

Figura 29 - Camadas e Componentes da Arquitetura J2EE. Adaptada de Pawlan (2001)

Componentes Cliente: Uma aplicação J2EE pode ou não ser baseada na Web. Uma aplicação cliente executa na máquina cliente para uma aplicação J2EE não baseada na Web. Para uma aplicação J2EE baseada na Web, um navegador carrega páginas Web e *Applets* para a máquina cliente.

Aplicações Cliente: Uma aplicação cliente executa em uma máquina cliente e provê um modo para usuários executarem tarefas num sistema J2EE. Normalmente possui uma interface gráfica de usuário usando as bibliotecas *Swing* ou *Abstract Window Toolkit* (AWT), mas certamente é possível ter uma interface baseada em texto. Aplicações cliente acessam diretamente o *Enterprise Bean* na camada de negócio.

Navegadores Web: O navegador Web do usuário carrega páginas Web estáticas ou dinâmicas nos formatos HTML (*Hypertext Markup Language*), WML (*Wireless Markup Language*) ou XML (*Extensible Markup Language*) da camada Web. Páginas Web dinâmicas são geradas por *Servlets* ou por páginas JSP que são executadas na camada Web.

***Applets*:** Uma página Web carregada a partir da camada Web pode incluir um *Applet* embutido. Um *Applet* é uma aplicação cliente pequena escrita em Java que é executada pela máquina virtual Java instalada no navegador Web. Porém o sistema cliente precisa ter um *plug-in* Java e um arquivo com políticas de segurança para que o *Applet* possa ser executado com sucesso no navegador Web.

Páginas JSP: Páginas JSP são a API preferida para criar um programa cliente baseado na Web, pois não é preciso nenhum tipo de *plug-in* ou arquivo de segurança no sistema do cliente. Elas também facilitam um projeto limpo e aplicações modulares, porque provêem um modo para separar a programação da aplicação do projeto das páginas Web. Isso significa que os desenvolvedores de páginas Web não precisam entender a linguagem de programação Java para realizar seu trabalho.

Arquitetura do Componente JavaBean: Na camada cliente também pode ser incluso um componente baseado na arquitetura de componentes JavaBean (Componente JavaBean) para administrar o fluxo de dados entre uma aplicação cliente ou um *Applet* e componentes que executam no servidor J2EE.

Componentes JavaBeans escritos para a plataforma de J2EE têm variáveis de instância e adquirem e ativam métodos para acessar os métodos nas variáveis de instância. Componentes JavaBeans usados deste modo são tipicamente simples em projeto e implementação, mas deveriam estar de acordo com o nome e convenções de projeto esboçadas na arquitetura de componente JavaBeans.

Componentes Web: Os componentes Web J2EE podem ser páginas JSP ou *Servlets*. *Servlets* são classes Java que processam requisições dinamicamente e constroem respostas. Páginas JSP são documentos baseados em textos que contém conteúdo estático e pedaços de código Java para gerar conteúdo dinamicamente.

Componentes de Negócio: A codificação para negócios, que logicamente resolve ou satisfaz um domínio de negócio específico é controlado por um EJB (*Enterprise Java Bean*) executando na Camada de Negócios. Um EJB recebe dados de um programa cliente, o processa, se necessário, e manda esta informação para a Camada de Sistema de Informação de Negócio para armazenamento. Um EJB também pode recuperar um dado armazenado, processá-lo, se necessário, e enviá-lo de volta para o programa cliente.

Existem 3 tipos de EJBs: *Session Beans*, *Entity Beans* e *Message-Driven Beans*. Um *Session Bean* representa uma conexão temporária com um cliente, quando o cliente termina de executar, o *Session Bean* e os seus dados são descartados. Em contraste, um *Entity Bean* representa o armazenamento persistente dos dados em um banco de dados, se o cliente terminar ou se o servidor for desligado, os serviços subjacentes garantem que os dados do *Entity Bean* foram salvos. O *Message-Driven Bean* combina as características de um *Session Bean* com JMS (*Java Message Service*), permitindo a troca de mensagens assíncronas entre componentes.

3.5.2 Componentes .NET

O modelo de componentes da Microsoft, chamado .Net (Platt, 2001), possui a independência de plataforma e, assim como J2EE, utiliza uma máquina virtual chamada CLR (*Common Language Runtime*). Assim, o código a ser executado na máquina virtual pode ser escrito em muitas linguagens diferentes, desde que essas forneçam o compilador adequado. O modelo de componentes .Net é baseado nas chamadas montagens (*assemblies*). Uma montagem é uma coleção de itens, tais como bibliotecas dinâmicas (DLLs – *Dynamic Link Libraries*), programas executáveis ou recursos (como imagens). Cada montagem contém um arquivo *manifest* que descreve quais itens pertencem a ela e como o usuário de um componente pode usar a montagem. As montagens podem ser usadas para construir aplicações locais ou remotas. No primeiro caso, várias montagens são colocadas juntas no mesmo diretório e a CLR gerencia a integração de tais montagens. No segundo caso, as montagens são colocadas no *Internet Information Server* (IIS), o servidor de Web da Microsoft, o qual as torna disponíveis para usuários remotos com o nome de Serviços Web. A infra-estrutura dos Serviços Web consiste principalmente dos três elementos a seguir:

- Formato para Ligação de Serviços Web: Os Serviços Web são montagens que são acessados pela Internet usando o protocolo SOAP (*Simple Object Access Protocol*) (Gudgin et al., 2003), que é baseado em XML. SOAP define um protocolo leve para a troca de informações em ambientes descentralizados e distribuídos. Esse protocolo consiste de três partes: (1) a especificação de um "envelope", que define um *framework* para descrever o conteúdo da mensagem;

(2) uma especificação para codificação de conteúdo, tais como descrições de tipos de dados; e (3) define as convenções para representar chamadas a procedimentos remotos e respostas. SOAP é um protocolo assíncrono que pode usar padrões de protocolos de transporte diferentes, tais como HTTP e SSL (*Secure Socket Layer*).

- Descrição de Serviços Web: Para ser capaz de interagir com um Serviço Web particular, o cliente deve compreender qual é a função do serviço. Por isso, a descrição de um Serviço Web define quais interações um Serviço Web fornece e quais tipos de dados são necessários como argumentos ou resultado. A linguagem de descrição de Serviços Web (WSDL - *Web Service Description Language*) (Christensen et al., 2001) define um padrão que deve ser usado para descrever a interface de um Serviço Web de maneira que ele possa ser usado por clientes.

- Descoberta de Serviços Web: Antes de usar um serviço em particular, o cliente deve saber quais serviços estão disponíveis para uso. Para avisar sobre a existência de um Serviço Web, o .Net define o protocolo para descoberta de Serviços Web (UDDI). O UDDI permite que clientes saibam quais Serviços Web existem dentro do ambiente de desenvolvimento local e em diretórios de Serviços Web na Internet.

3.6 Considerações Finais

A disponibilização de componentes de software é fundamental para o funcionamento adequado dos Servidores de Aplicações. Esses servidores funcionam de forma dinâmica, podendo ser modificados em tempo de execução efetuando-se apenas a troca de seus componentes. Com a adição de uma ontologia, dentro de um Servidor de Aplicações, fica mais fácil escolher adequadamente quais componentes serão utilizados.

Pelo fato da arquitetura J2EE ser uma arquitetura aberta, a criação de um mecanismo para adicionar o uso de ontologias dentro de Servidores de Aplicações é facilitada. É importante ressaltar que, mesmo que o mecanismo de ontologias esteja dentro de um Servidor de Aplicações J2EE, é possível ativar componentes e serviços de outras plataformas, como a .NET.

O fato dos Serviços Web serem disponíveis em grande quantidade possibilita também a utilização desses serviços como massa de testes.

4 Descrição do Protótipo

O objetivo deste trabalho foi criar um sistema protótipo capaz de ativar componentes ou serviços a partir de informações semânticas dos componentes ou serviços. Para isso, foi adicionada mais uma camada entre os componentes ou serviços e o usuário/aplicação.

Desta forma, o funcionamento do protótipo ocorre da seguinte forma: o usuário faz uma requisição ao sistema, contendo as informações semânticas do componente e os parâmetros necessários para que o componente seja ativado. Com essas informações, o sistema faz uma busca na ontologia de componentes e descobre qual componente pode ser utilizado para aquela requisição. Após descobrir qual componente deve ser utilizado, o sistema deve ativar este componente; caso o componente dê uma resposta, esta deve ser repassada ao usuário que fez a requisição. Assim, o conceito de OBAS, conforme descrito no Capítulo 3, foi adotado.

Nas seções a seguir são apresentadas as principais características deste sistema protótipo desenvolvido.

4.1 Arquitetura do Sistema

Na Figura 30 é apresentada a arquitetura do sistema protótipo. Em seguida é feita a descrição de cada elemento da arquitetura.

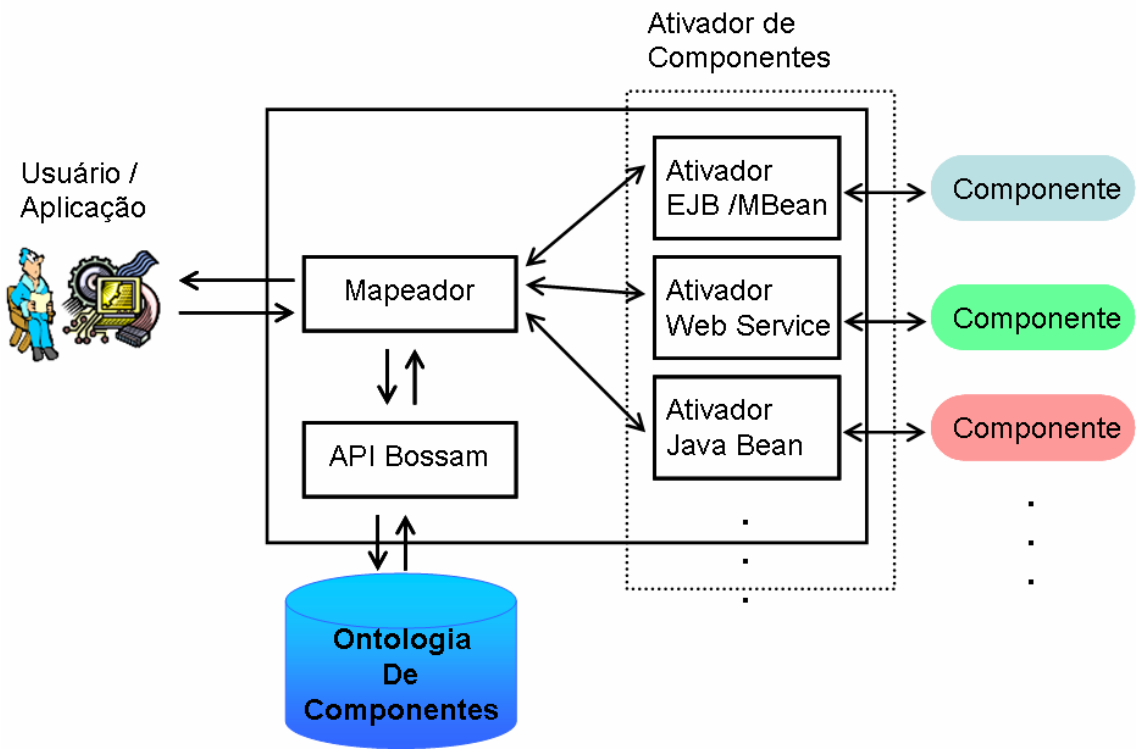

Figura 30 - Arquitetura do Sistema

O elemento da arquitetura proposta, denomiando **"Usuário / Aplicação"** indica que o sistema pode ser utilizado por usuários ou por aplicações. Além de existir uma interface para o sistema ser utilizado por aplicações, também existe uma interface onde o usuário pode interagir diretamente com o sistema protótipo. O usuário pode interagir com o sistema realizando buscas por componentes a partir das informações semânticas que ele contém. Essa interface de consulta será detalhada na Seção 4.5.

O módulo **Mapeador** é responsável por processar as requisições dos usuários ou aplicações, realizar as consulta à Ontologia de Componentes por meio da API Bossam, e após saber qual componente e qual ativador de componentes devem ser utilizados, o ativador executa a requisição recebida.

O componente **API Bossam** é responsável por fazer as inferências e executar as regras na ontologia de componentes.

O módulo **Ativador de Componentes** é responsável por executar os componentes, ele é composto por vários ativadores, cada um deles é capaz de ativar o tipo de componente ao qual foi programado. O termo Ativador de Componentes é apenas uma representação lógica, pois na prática existem vários ativadores específicos, um para cada tipo de componente, e todos eles agrupados, recebem o nome Ativador de Componentes. No sistema protótipo desenvolvido neste trabalho foi implementado

apenas um dos ativadores de componentes, este é capaz de ativar apenas Serviços Web. Na Seção 4.4 o Ativador de Componentes para Serviços Web será detalhado.

O elemento **Componente** na arquitetura, à direita da Figura 30, representa o componente em si, sendo este compatível com o Ativador de Componentes associado.

E, finalmente, a **Ontologia de Componentes** na Figura 30 representa a Ontologia de componentes, onde estão todos os relacionamentos semânticos existentes entre os componentes, os métodos e outras informações referentes ao componente. A Ontologia de Componentes será detalhada na Seção 4.2.

4.2 Ontologia de Componentes

Uma Ontologia de Componentes contém relacionamentos semânticos entre componentes, métodos e outras informações relevantes ao componente. A Ontologia de Componentes utilizada neste trabalho foi originalmente concebida por Linhalis (2007), onde foi proposta a arquitetura ONTOMAP, que contém uma Ontologia de Componentes. Porém, nessa Ontologia de Componentes, foram feitas algumas alterações necessárias para o funcionamento do protótipo proposto no presente trabalho.

Na Figura 31, a Ontologia de Componentes, proposta por Linhalis (2007) é representada na forma de diagrama de classes.

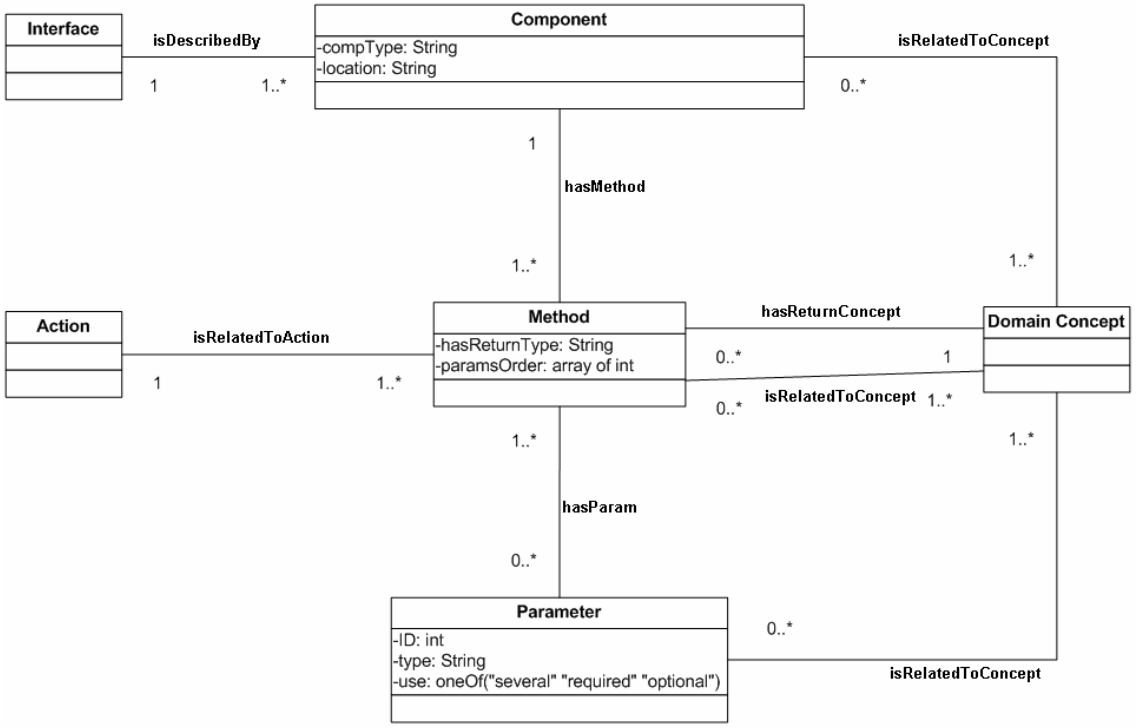

Figura 31 – Ontologia de Componentes proposta por (Linhalis, 2007)

A classe Component representa o componente em si e tem as propriedades compType que indica o qual o tipo de componente representado e location que indica o local onde o componente esta disponível. Esta classe está diretamente relacionada com as classes Interface, que representa a interface do componente, Domain Concept, que representa os conceitos de um determinado domínio e Method, que representa os métodos do componente e tem as propriedades hasReturnType, que indica o tipo de dados esperado para retorno e paramsOrder que contém a ordem que os parâmetros devem ser passados para o método. A classe Method está relacionada com as classes Component, Domain Comcept e Action, que indica a ação a qual o método é relacionado e Parameter, onde são descritos os parâmetros dos métodos. Esses parâmetros são representados com as propriedades ID, que identifica o parâmetro, type, que indica o tipo de dados do parâmetro e use, que indica o tipo de uso, podendo ser opcional, requerido ou *several* (coleção de parâmetros do mesmo tipo).

A semântica existente nesta Ontologia de Componentes pode ser observada nos relacionamentos existentes entre as classes. A relação isDescribedBy, indica que um componente é descrito por uma interface e que uma interface de componentes pode descrever um ou mais componentes. Já a relação isRelatedToConcept indica o relacionamento existente entre o componente, métodos e parâmetros com conceitos de um domínio particular. A relação hasMethod indica que componentes possuem ao

menos um método, hasParam indica que métodos podem ter ou não parâmetros, isRelatedToAction indica que uma ação pode ser relacionada a vários métodos e hasReturnComcept indica a qual conceito de domínio o valor de retorno do método é relacionado.

Na Figura 32 encontra-se a ontologia de componentes utilizada no protótipo desenvolvido neste trabalho. Nela podem-se ver as alterações feitas na ontologia original.

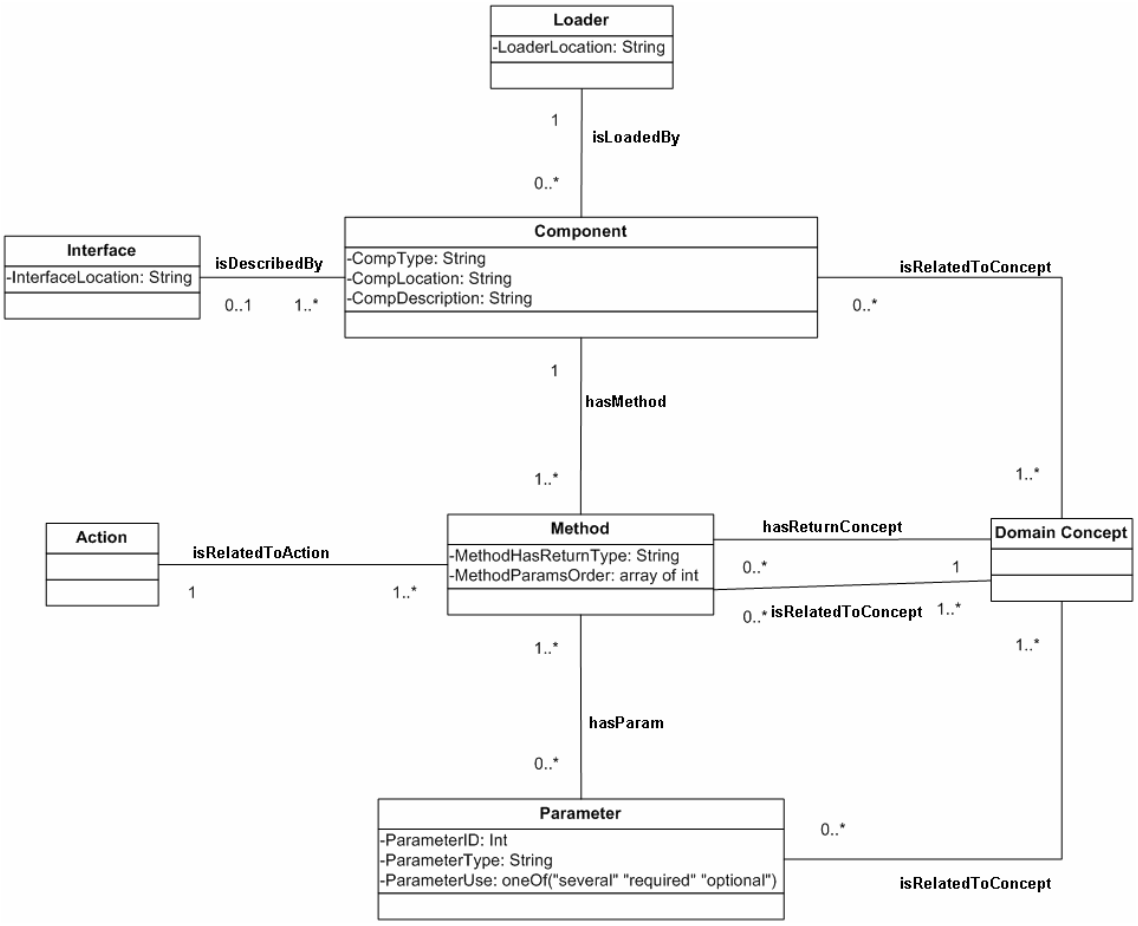

Figura 32 - Ontologia de Componentes atualizada

Como pode ser visto na Figura 32, os rótulos das propriedades foram alterados para evitar ambigüidades. Além disso, na classe Interface foi adicionada a propriedade InterfaceLocation e a cardinalidade do relacionamento isDescribedBy foi alterada. Desta forma, um componente não precisa obrigatoriamente ter uma interface, pois no caso de serviços web não é obrigatório se ter uma interface associada.

A adição da classe Loader foi necessária para relacionar, de forma semântica, um componente a um ativador de componentes adequado. Esse relacionamento é rotulado por isLoadedBy e indica que um componente pode ser ativado por apenas um ativador

de componentes e que um ativador de componentes pode ativar nenhum ou vários componentes.

A propriedade LoaderLocation indica o local onde o ativador se encontra disponível. Também foi adicionada a propriedade CompDescription na classe Component, que deve conter uma descrição do componente.

Por esta ontologia ter sido testada apenas com um pequeno conjunto de componentes, existem casos que não foram previstos e algumas alterações podem ser necessárias para que alguns componentes possam ser enquadrados nesta Ontologia de Componentes. Por exemplo, para que os serviços web disponíveis pela Yahoo para acessar a plataforma Flickr (álbum de fotografias on-line) fossem utilizados, seria necessário acrescentar três novas propriedades na classe Componente, que seriam authkey, secretKey e urn, que representam respectivamente código de usuário, senha de acesso e espaço de nomes do serviço web. Com apenas as duas primeiras propriedades adicionadas, além dos serviços web do Yahoo, também seria possível utilizar os serviços da Amazon e qualquer outro conjunto de serviços web que necessitam de autenticação de usuário. O espaço de nomes (urn) só é necessário para serviços web que não tem sua interface WSDL disponível.

Os serviços web que foram utilizados para alimentar a Ontologia de Componentes, podem ser encontrados na url http://www.webservicex.net/WS/wscatlist.aspx. Nesse site são listados mais de cem serviços disponíveis livremente, para diversos fins, como previsão do tempo, conversão entre moedas, conversão de unidades, envio de mensagens sms, validação de cartões de credito, entre outros.

4.3 Mapeador e API Bossam

O Mapeador em conjunto com a API Bossam são responsáveis por descobrir quais componentes são adequados à requisição do usuário. Quando é feita uma requisição ao Mapeador, ela é processada e suas partes são separadas, para poder ser feita uma inferência na ontologia, que é realizada utilizando a API Bossam.

Para o processamento das requisições serem simplificados, foi fixado um padrão para as requisições. As requisições devem seguir o seguinte formato:

Ação + Conceito + Parâmetros

Desta forma, é possível se separar as partes da requisição e posteriormente realizar a inferência na Ontologia de Componentes. Para facilitar a utilização do sistema, existe uma ferramenta onde o usuário pode realizar consultas à ontologia sem necessitar da construção de uma aplicação. Essa ferramenta está detalhada na Seção 4.5.

Com a semântica existente nesta requisição, é possível descobrir quais componentes e quais métodos dos componentes estão associados à **Ação** e ao **Conceito**. Olhando-se apenas essas duas palavras, em alguns casos, pode ser difícil compreender a semântica, porém o relacionamento semântico existente é entre os componentes e não entre essas duas palavras. Para entender melhor está requisição vamos analisar um exemplo real.

get weather brazil, são Paulo

Nesta requisição é possível ver que a **Ação** é **get**, o **Conceito** é **weather** e os parâmetros são **brazil** e **são paulo**. A ação **get** esta associada a diversos componentes da ontologia de componentes, porém associados a esta ação e ao conceito **weather**, existe apenas um componente, chamado "WeatherService" que disponibiliza dois métodos associados à ação **get**. Esses métodos são "GetCitiesByCountry" que necessita apenas um parâmetro rotulado "CountryName" e "GetWeather" que pede dois parâmetros rotulados "CountryName" e "CityName". Analisando a requisição, percebemos que existe apenas um componente e um método que satisfaz a requisição.

O modulo Mapeador não foi preparado para decidir qual é o melhor componente quando existirem mais de um que satisfaça a requisição, para fazer esse tipo de escolha seria necessário fazer uma busca mais detalhada contendo, por exemplo, qual foi o ultimo componente utilizado ou realizar inferências sobre os requisitos não funcionais relacionados ao componente. Na versão da Ontologia de Componentes, presente neste trabalho, não existem os parâmetros de requisitos não funcionais.

Para realizar a inferência na Ontologia de Componentes, por meio da API Bossam, são utilizados apenas os parâmetros **Ação** e **Conceito**. Com esses parâmetros é possível descobrir quais são os componentes e os métodos relacionados. Outros critérios como quantidade de parâmetros e tipo de dados podem ser utilizados, pelo mapeador, como forma de desempate entre componentes, porém neste protótipo apenas a quantidade de parâmetros foi levada em conta, pois em Serviços Web, geralmente,

apenas informações textuais são passadas; a comparação de tipos de dados é muito útil quando os componentes em questão são do tipo EJB.

Na Figura 33, pode-se ver como é feita uma inferência utilizando a API Bossam.

```
"query q is C:isRelatedToConcept(?comp,C:"+conceito+") and
C:isRelatedToAction(?meto,C:"+acao+") and C:hasParam(?meto, ?para) and
C:MethodParamsOrder(?meto, ?ordp) and C:ParameterType(?para,?tipo) and
C:ParameterID(?para,?idpa) and C:CompLocation(?comp,?loc) and
C:isLoadedBy(?comp, ?act_temp) and C:LoaderLocation(?act_temp, ?act) and
C:isDescribedTo(?comp, ?inter_temp) and C:InterfaceLocation(?inter_temp, ?inter);"
```

Figura 33 - Exemplo de inferência utilizando a API Bossam

Todos os elementos, iniciados pelo símbolo ?, são as variáveis envolvidas na consulta. O símbolo C: representa o espaço de nomes que está sendo utilizado, este espaço de nomes é definido anteriormente como um parâmetro da API. Após o símbolo C: estão as propriedades contidas na Ontologia de Componentes (no diagrama de classes ilustrado na Figura 32, as propriedades são representas como relacionamentos entre classes e como propriedades das classes). Os elementos "+acao+" e "+conceito+" são os parâmetros **Ação** e **Conceito,** passados na requisição que, neste exemplo, são variáveis Java.

Esta consulta funciona relacionando dois indivíduos, por exemplo, em C:isRelatedToConcept(?comp,C:"+conceito+"), a propriedade isRelatedToConcept, relaciona um conceito a um componente. Quando essa comparação é feita utilizando a variável conceito, resulta em uma lista de componentes relacionados a este conceito. Essa lista de componentes é utilizada como parâmetro de comparação para as outras propriedades e, desta forma, é possível extrair todas as informações desejadas da Ontologia de Componentes.

As propriedades, em geral, são auto-explicativas, contendo nomes significativos e de fácil compreensão. Porém, as variáveis só fazem sentido, quando são observados os indivíduos contidos na ontologia. A seguir, uma listagem com o significado de cada variável:

- **?comp**: recebe os componentes relacionados ao conceito;
- **?meto**: recebe os métodos relacionados à ação;
- **?para**: recebe os parâmetros relacionados aos métodos;
- **?ordp**: recebe a ordem em que os parâmetros devem ser posicionados;
- **?tipo**: recebe os tipos de dados de cada parâmetro;

- **?idpa**: recebe o id de cada parâmetro, para que seja possível ordena-los;
- **?loc**: recebe o local onde esta o componente;
- **?act_temp**: recebe qual vai ser o ativador do componente ;
- **?act**: recebe o local do ativador do componente ;
- **?inter_temp**: recebe a interface do componente;
- **?inter**: recebe o endereço da interface do componente.

Neste protótipo, apenas esses parâmetros foram considerados, pois eles são suficientes para ativar os componentes que fazem parte do conjunto de testes. Entretanto, existem outros parâmetros que podem ser necessários para ativar outros tipos ou conjuntos de componentes, por exemplo, as propriedades "AuthyKey" e "SecretyKey", são necessárias para ativar componentes que necessitam de autenticação de usuário.

Com o resultado desta inferência, o Mapeador chama o Ativador de Componentes correto e passa os parâmetros de forma adequada. Dependendo do ativador de componentes, o Mapeador vai receber uma resposta diferente, no caso de Serviços Web, ele recebe uma URL que é repassada ao usuário/aplicação e no caso de ser um EJB, ele recebe uma variável com o tipo de dados de retorno deste EJB, que é repassada ao usuário/aplicação, que deve estar preparado para recebê-la. Na Seção 4.4 é detalhado o funcionamento de uma requisição para Serviços Web.

4.4 Ativador de Serviços Web

No protótipo construído neste trabalho, foi implementando apenas um Ativador de Serviços Web, mas outros ativadores podem facilmente ser acoplados a este protótipo.

O Ativador de Serviços Web foi construído utilizando-se duas linguagens de programação distintas. Com isso, fica fácil ver, como novos ativadores podem ser implementados e acoplados a este protótipo. As linguagens utilizadas foram Java e PHP. Utilizando Java foi implementada a parte do ativador que recebe a requisição do Mapeador, processando essa requisição, montando a chamada de forma adequada e, posteriormente retornando uma resposta; essa parte é chamada de Compositor de URL. Foi codificado em PHP um cliente SOAP genérico, chamado de Ativador Universal.

Com ele, é possível ativar qualquer Serviço Web que contenha uma interface WSDL disponível.

O Ativador Universal foi construído de forma genérica suficiente para ativar diversos Serviços Web disponíveis livremente na web. Dessa forma, ele pode ser incrementado para ativar outros Serviços Web não previstos durante a implementação deste, como o envio de arquivos binários e certificados digitais. Este Ativador Universal é uma aplicação que pode estar disponível local ou remotamente, basta existir um servidor PHP com suporte a SOAP.

Para utilizar o cliente universal basta fazer uma requisição http utilizando o método GET. Nesta requisição, devem ser passados como parâmetros o endereço do WSDL que descreve as interfaces do Serviço Web desejado, o método que vai ser utilizado e logo em seguida os parâmetros necessários para utilizar o método. Na Figura 34 é possível ver um caso real de requisição:

```
http://localhost/cliente/soap_with_wsdl.php?wsdl=http://www.webservicex.net/globalweather.asmx?WSDL&metodo=GetWeather&CountryName=Brazil&CityName=Sao Paulo
```

Figura 34 - Requisição feita ao Ativador Universal de Serviços Web

O início da requisição (http://localhost/cliente/soap_with_wsdl.php) indica o local onde o cliente universal está disponível (na Ontologia de Componentes é representado pela propriedade LoaderLocation), o parâmetro wsdl indica que, a seguir, deve ser indicado o endereço onde está disponível o arquivo WSDL (na Ontologia de Componentes é representado pela propriedade InterfaceLocation) contendo a interface do Serviço Web, o parâmetro metodo (na Ontologia de Componentes é representado pela classe Method) indica qual o método do Serviço Web vai ser ativado e a seguir podem ser passados quantos parâmetros forem necessários (na Ontologia de Componentes é representado pela classe Parameter), neste caso são passados apenas dois, CountryName e CityName que devem conter respectivamente o nome de um país e o nome de uma cidade. Este Serviço Web oferece o serviço de previsão do tempo. A resposta para esta requisição é uma mensagem SOAP contendo diversas informações sobre o clima da cidade desejada.

O Compositor de URL, implementado em Java, recebe as informações provenientes do Mapeador em um vetor, monta a requisição de forma adequada e repassa a url montada para o Mapeador que a repassa para a aplicação, como o resultado

desta requisição sempre é um envelope SOAP, a aplicação deve sempre esperar uma resposta SOAP.

Na Figura 35 é esquematizada a forma com que esse processo de ativação de Serviços Web disponíveis acontece no sistema protótipo desenvolvido.

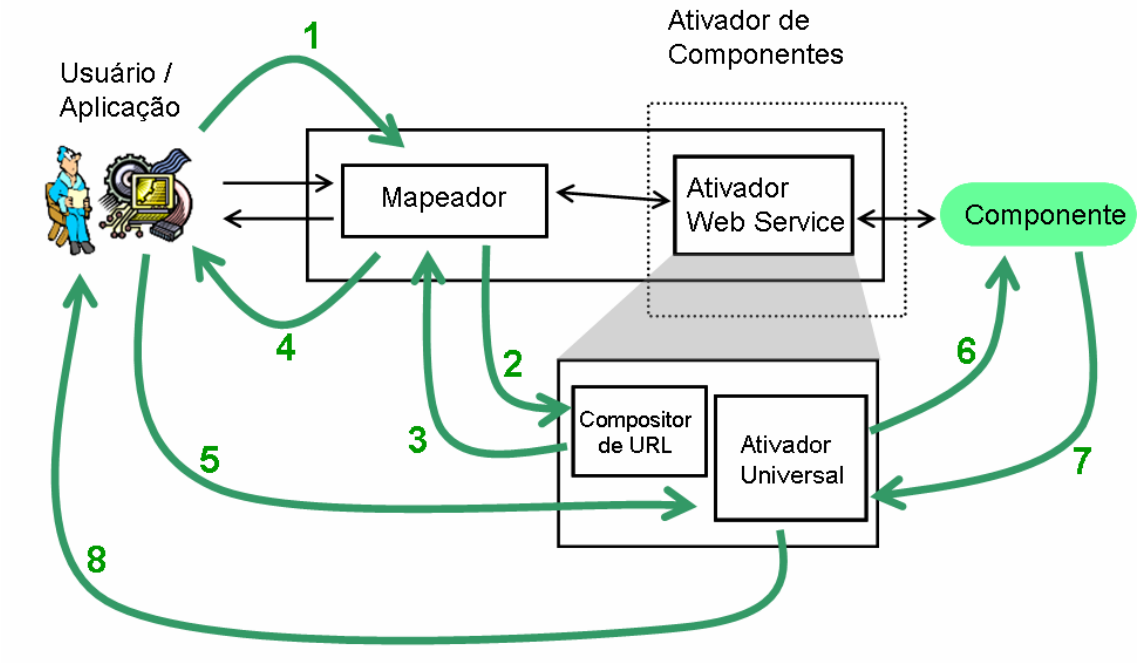

Figura 35 - Detalhamento do Ativador de Serviços Web

Na Figura 35 é ilustrado o funcionamento do sistema protótipo com foco no Ativador de Serviços Web.

Nesta Figura 35, foram removidos alguns itens da arquitetura original (Figura 30), para facilitar a visualização. As setas de números 1 e 2 representam o caminho da requisição do usuário, esta requisição é representa em dois passos, pois ela sofre alterações ao passar pelo Mapeador. Quando a requisição é passada ao Compositor de URL, os parâmetros recebidos são montados na forma de URL. Com as setas 3 e 4 é ilustrado que a URL gerada anteriormente é passada para o Mapeador, que a repassa, sem alterações, para o usuário/aplicação. Essa URL, ao ser acessada (seta 5), retorna (seta 8) a mensagem SOAP original do Serviço Web requisitado. As setas 6 e 7 representam o Ativador Universal acessando o Serviço Web através de uma mensagem SOAP e recebendo como resposta, outra mensagem SOAP que é repassada sem alterações para o usuário/aplicação, representado pela seta 8.

O Ativador de Serviços Web implementado neste trabalho, dá suporte somente aos Serviços Web com interfaces WSDL. Para construir um ativador com suporte a

Serviços Web sem interfaces WSDL, seria necessário fazer pequenas alterações no Ativador Universal, pois as plataformas de desenvolvimento para Serviços Web diferenciam a forma de chamada para estes casos.

4.5 Interfaces do Protótipo

Para facilitar a utilização do protótipo foi criada uma interface para o usuário poder consultar a Ontologia de Componentes sem a necessidade de construir uma aplicação.

Esta interface funciona diretamente na linha de comando, onde devem ser passados os parâmetros ação e conceito, nesta ordem. Assim tem-se como resposta quais são os componentes, métodos, parâmetros, ordem dos parâmetros e descrição dos componentes.

Na Figura 36 é ilustrado um exemplo de uso desta interface.

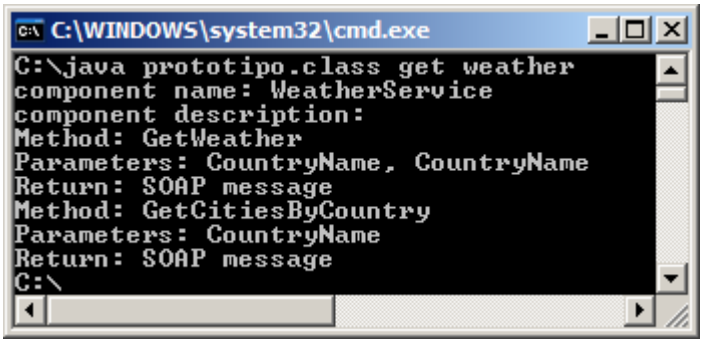

Figura 36 - Interface para usuário

No exemplo ilustrado na Figura 36, component name indica o nome do componente, que apesar de irrelevante para o uso do sistema, ele é apresentado apenas com caráter informativo. Em component description, é apresentado, quando disponível, uma descrição do componente. Em Method é apresentado o nome do método deste componente, da mesma forma que o nome do componente, esta informação também tem apenas caráter informativo. O rótulo Parameters indica os nomes das variáveis e a ordem na qual elas devem ser passadas na chama ao sistema. E finalmente em Return é indicado o tipo de resposta que o sistema dará.

Como poder ser visto na Figura 36, existem dois métodos disponíveis para o usuário, sendo que para ele utilizar um deles, basta passar o parâmetro correto. Apesar deste Serviço Web não ter disponível o campo component description, nele deveria

conter as informações necessários para o uso do componente em questão, indicando o que significa cada variável, pois, caso os nomes da variáveis não sejam significativos, a única forma de utilizar corretamente o serviço, é seguindo a documentação disponive no campo component description.

Com isso, fica mais fácil para o usuário descobrir quais são os componentes disponíveis sem a necessidade da construção de uma aplicação.

4.6 A importância das Regras

Apesar das regras não terem sido utilizadas intensamente neste trabalho, elas são muito importantes dentro de contexto de ontologias e web semântica. Com as regras é possível fazer integração entre diferentes ontologias que contêm o mesmo conteúdo, mas estão estruturadas de formas distintas.

Com as regras também é possível fazer alteração na ontologia em tempo de execução, por exemplo, se um ativador de componentes for alterado, criando apenas uma regra essa alteração pode se propagar automaticamente por toda a ontologia.

Neste trabalho foram criadas apenas regras de pouca complexidade, destinadas apenas para a indicação dos ativadores de componentes adequados para cada tipo de componente. Desta forma um ativador de componente pode ser trocado modificando apenas uma regra, não havendo necessidade de recompilar o protótipo.

Na Figura 37, pode-se ver um exemplo de regra utilizada neste trabalho.

```
CompType(?x, "WebService")  ^ isLoadedBy(?x, ?y)
→ LoaderLocation(?y, "http://localhost/cliente/soap_with_wsdl.php")
```

Figura 37 - Exemplo de regra

Nesta regra, a máquina de inferência, verifica se CompType é igual a “WebService” e em isLoadedBy, com a variável ?y, lista todos os componentes que são do tipo “WebService” e isso implica que todas as propriedades LoaderLocation associadas a estes componentes listados em ?y, tem seu valor alterado para http://localhost/cliente/soap_with_wsdl.php.

Outro exemplo de regra que poderia existir caso fosse construído um Ativador de Serviços Web com suporte a serviços web sem interface WSDL é ilustrado na Figura 38.

```
CompType(?x, "WebService") ^ isLoadedBy(?x, ?z) ^
isDescribedTo(?x, ?y) ^ swrlb:empty(?y) ^ swrlb:empty(?z)
→ isDescribedTo(?x, ?z) ^
  LoaderLocation(?z, "http://localhost/cliente/soap_without_wsdl.php")
```

Figura 38 - Exemplo de regra

Nesta regra os dois primeiros termos são idênticos à regra anterior, depois é verificado em isDescribedTo, quais são os componentes que contém uma interface. Em seguida, com o uso de um predicado embutido do SWRL, é verificado se as variáveis ?y (lista de componentes que tem interface) e ?z (lista de componentes que um ativador de componentes associado) são vazias, Se um componente não tem interface e não tem um ativador de componentes associado, então é atribuído um ativador "http://localhost/cliente/soap_without_wsdl.php".

Desta forma, as regras podem substituir a lógica de programação facilitando a alteração da aplicação sem a necessidade de recompilar o código.

4.7 Considerações Finais

Foram descritas neste capítulo as principais características do sistema protótipo desenvolvido: a arquitetura e as tecnologias que foram adotadas. Pôde-se observar que o agrupamento de diferentes tecnologias possibilita um auxílio ao usuário, criando novas soluções que facilitam o uso das já existentes.

Com esse protótipo, foi possível adicionar semântica aos componentes, visando tornar mais fácil a descoberta de qual o componente mais adequado a ser utilizado, e que a adição dessa camada semântica facilita na codificação de novas aplicações que utilizarão os componentes descritos na ontologia.

Também pudemos perceber que a utilização de regras e ontologias facilita a composição de uma aplicação dinamicamente, pois os componentes podem ser substituídos de forma transparente, sem prejuízo ao sistema e sem a necessidade de recompilação.

5 Conclusões

A cada dia que passa, a Web Semântica vem se tornando mais madura e viável, devido ao trabalho de diversos pesquisadores e empresas que se empenham em adicionar um significado lógico à Web.

Quando a Web Semântica estiver plenamente implementada, será possível associar todas as informações desejadas de uma forma consistente. Dessa forma, poderíamos ter nossas contas bancárias e nossos álbuns de fotográfica associados ao nosso calendário. Com isso, podemos ver que as informações passarão a ter seus significados relacionados.

Para que a Web Semântica torne-se viável, as ferramentas que dão suporte à Web precisam ser adaptadas, adicionando-se suporte a essa semântica. Para isso, as novas ferramentas devem ser implementadas de acordo com as arquiteturas, padrões e linguagens recomendadas pela W3C (organização internacional que tem o propósito de definir padrões a Web).

Na Web atual, para se construir sistemas confiáveis e altamente disponíveis, são utilizados Servidores de Aplicações. Tais servidores são plataformas que provêem os recursos básicos para que diversas aplicações possam se comunicar, permitindo a interoperabilidade entre as aplicações via Web. Em um Servidor de Aplicações, um conjunto de componentes de software constitui uma aplicação, no entanto, um mesmo componente pode ser utilizado por mais de uma aplicação. Esses componentes também podem ser disponibilizados na Web na forma de Serviços Web.

Os componentes de um Servidor de Aplicações, atualmente, não são relacionados semanticamente, eles simplesmente são listados em um diretório, ficando a cargo de o desenvolvedor conhecer cada componente. Uma forma de prover relacionamento semântico, para os componentes de software em um Servidor de aplicações, é a adição de uma ontologia, onde os componentes são relacionados, de forma semântica, ao domínio de aplicação que se está trabalhando. A esta ontologia é dado o nome de Ontologia de Componentes.

Como apresentado no Capitulo 4, neste trabalho foi construído um protótipo que tornou possível ativar componentes de software a partir de suas descrições disponíveis

em uma Ontologia de Componentes. Para isso, uma Ontologia de Componentes pré-existente foi especializada, estabelecendo o relacionamento entre um componente e seu ativador. Neste protótipo, também foi construído um Ativador de Componentes capaz de ativar componentes a partir das informações contidas na Ontologia de Componentes.

A partir deste protótipo, foi possível observar a adição de semântica a uma ferramenta pré-existente da Web atual. A adição de semântica aos componentes tornou a forma de utilização desses, mais simples e intuitiva. Além disso, foi dado um passo para que no futuro esse protótipo evolua permitindo a geração dinâmica de aplicações, onde, com apenas uma requisição, mais de um componente pode ser ativado para resolver a requisição.

Como trabalhos futuros, que podem vir a ser desenvolvidos a partir dos estudos e desenvolvimento realizados neste trabalho, podemos citar:

- Estender o Ativador de Serviços Web, possibilitando a ativação de uma gama maior de Serviços Web (por exemplo, possibilitar ativação de serviços que requeiram autenticação);
- Implementar outros Ativadores de Componentes, como por exemplo, Ativadores para EJB;
- Implementar a busca por componentes em múltiplos repositórios (ontologias), permitindo maior integração entre diferentes ontologias.

6 Referências Bibliográficas

Todos os links listados neste documento estavam disponíveis na data 13 de Junho de 2007.

Alonso, G.; Casati, F.; Kuno, H.; Machiraju, V.. Web Services. Springer, Sep 2003.

Application Server Matrix (2005). TheServerSide.com. Disponível em: http://www.theserverside.com/reviews/matrix.tss

Armstrong, E. The J2EE 1.4 Tutorial, 2003. Disponível em: http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html.

Artus, D. J. N. SOA realization: Service design principles, 17 Feb 2006. Disponível em: http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design/.

Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; and Stein, L. A. OWL Web Ontology Language Reference - W3C Recommendation 10 February 2004. Disponivel em: http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

Beckett, D. Refactoring RDF/XML Syntax - W3C Working Draft 06 September 2001. Disponível em: http://www.w3.org/TR/2001/WD-rdf-syntax-grammar-20010906/.

Benjamins, V.R., Fensel, D., Gomez Peres, A.: Knowledge Management through Ontologies. In Reimer, U., ed.: Proceedings of the Second International Conference on Practical Aspects of Knowledge Management. (1998). Disponível em: http://www.aifb.uni-karlsruhe.de/WBS/Publ/pre1997/km.ps.

Berners-Lee, T. (2000). Semantic Web. Disponível em: http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html.

Berners-Lee, T.; Hendler, J.; Lassila, O. (2001). The Semantic Web. Scientific American, 284(5):35-43. Disponível em: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&pageNumber=1&catID=2.

Bernstein, P. Middleware: A Model for Distributed System Services. Communications of the ACM – Vol. 39, N 2, Fev/1996, pgs. 86 – 98.

Bézivin, J. (1998). Who is Afraid of Ontologies? In Proceedings of the OOPSLA '98 Workshop: Model Engineering, Methods and Tools Integration with CDIF. Disponível em: http://www.metamodel.com/oopsla98-cdif-workshop/bezivin1/.

Boley, H., Tabet, S., and Wagner, G. (2001). Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In Proceedings of the International Workshop on the Semantic Web.

Bray, T., Paoli, J., Sperberg McQueen, C. M., and Maler, E. (2006). Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation. Disponível em: http://www.w3.org/TR/REC-xml.

Brickley, D.; Guha, R. V.; McBride, B. RDF Vocabulary Description Language 1.0: RDF Schema - W3C Recommendation 10 February 2004. Disponível em: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

Andrews, T.;Curbera, F.;Dholakia, H.; Goland, Y.; Leymann, J. K. F.; Liu, K.; Roller, D.; Smith, D.; Thatte, S.; Trickovic, I.; Weerawarana, S. Business Process Execution Language for Web Services Version 1. Specification, May 2003. Disponível em: http://www.ibm.com/developerworks/library/ws-bpel/.

Pagels, M. The DARPA Agent Markup Language Homepage. Program will end in early 2006. Disponível em: http://www.daml.org/.

Deitel, H. M.; Deitel, P. J. Java - How to Program. New Jersey: Prentice Hall, 2000.

Fleury, M.; Reverbel, F. The JBoss Extensible Server. In M. Endler and D. C. Schmidt, editors, Middleware 2003, ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, June 16-20, 2003, Proceedings, volume 2672 of Lecture Notes in Computer Science, pages 344-373. Springer, 2003. Disponível em: http://www.jboss.org

Forgy, C. L. Rete: A Fast Algorithm for the Many Pattern Many Object Pattern Match Problem. Artificial Intelligence, 19(1):17–37, 1982.

de Freitas, F. L. G. (2003). Ontologias e a Web Semântica, volume 8, pages 1-52. Sociedade Brasileira de Computação (SBC). II Jornada de Mini-Cursos de Inteligência Artificial.

Friedman-Hill, E. Jess: The Java Expert System Shell, Sandia National Laboratories, 1997. Disponível em: http://herzberg.ca.sandia.gov/jess/.

Gennari, J.; Musen, M.; Fergerson, R.; Grosso, W.; Crubézy, M.; Eriksson, H.; Noy, N.; Tu, S. 2003. The evolution of Protégé: an environment for knowledge-based systems development. Disponível em: http://smi.stanford.edu/pubs/SMI_Abstracts/SMI-2002-0943.html.

Gómez-Pérez, A. (1999). Ontological Engineering: A State of the Art. Expert Update. British Computer Society. Autumn. Vol. 2. Nº 3. 1999.

Grosof, B.; Dean, M.; Ganjugunte, S.; Tabet, S.; Neogy, C. SweetRules: The first open source platform for semantic web business rules v2.1a (2005). Disponível em: http://sweetrules.projects.semwebcentral.org/.

Gruber, T. R. (1993). A Translation Approach to Portable Ontologies. Knowledge Acquisition, 5(2):199-220.

Guarino, N. Formal Ontology and Information Systems. Proceedings of FOIS'98, Trento, Itália, 6-8 junho 1998. Amsterdan, IOS Press, pp. 3-15.

Heflin, J. OWL Web Ontology Language Use Cases and Requirements - W3C Recommendation 10 February 2004. Disponível em: http://www.w3.org/TR/2004/REC-webont-req-20040210/.

Horrocks, I.; Fensel, D.; Broekstra, J.; Decker, S.; Erdmann, M.; Goble, C.; Harmelen, F.; Klein, M.; Staab, S.; Studer, R.; Motta, E. (2000). The Ontology Inference Layer - OIL. Disponível em: http://www.ontoknowledge.org/oil/TR/oil.long.html.

Horrocks I.;Patel-Schneider P. F.;Boley H;Tabet S.;Grosof B.;Dean M.. SWRL: A Semantic Web Rule Language Combining OWL and RuleML - W3C Member Submission 21 May 2004. Disponível em: http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

Jang, M.; Sohn, J. (2004). Bossam: an extended rule engine for the web, Proceedings of RuleML 2004 (LNCS Vol. 3323), Nov. 8, 2004. Disponível em: http://zebehn.tistory.com/attachment/bk11.pdf.

Jang, M. (2005). Buchingae - A Rule Language for The Web. Disponível em: http://www.geocities.com/jangminsu/buchingae-rule-language.html.

Klyne, G.; Carroll, J.J.; McBride, B. Resource Description Framework (RDF): Concepts and Abstract Syntax - W3C Recommendation 10 February 2004. Disponível em: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

Knublauch, H.; Fergerson, R. W.; Noy, N. F.; Musen, M. A. (2004) The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications in 3rd International Semantic Web Conference (ISWC2004). Disponível em: http://protege.stanford.edu/plugins/owl/publications/ISWC2004-protege-owl.pdf

Krutisch, R.; Meier, P.; Wirsing, M. The AgentComponent Approach, Combining Agents and Components. German Conference for Multiagent System Technologies. 1. Heidelberg: Springer-Verlag, 2003. (Lecture Notes on Artificial Intelligence 2831). p. 1-12. Disponível em: www.pst.informatik.uni-muenchen.de/publications/agentcomponent.pdf.

Lind, J. Relating Agent Technology and Component Models. AgentLab, 2001. Disponível em: www.agentlab.de/documents/Lind2001e.pdf.

Linhalis, F.; Moreitra, D. A. Ontology-Based Application Server to the Execution of Imperative Natural Language Requests. Berlin: Springer, 2006. International Conference on Flexible Query Answering Systems - FQAS'06 (7, Milano, 2006). In: LECTURE NOTES IN ARTIFICIAL INTELLIGENCE. BERLIN, v. 4027, p. 589-600, 2006.

Linhalis, F. Mapeamento semântico entre UNL e componentes de software para execução de requisições imperativas em linguagem natural. 2007. 216f. Tese (Doutorado) - Instituto de Ciências Matemáticas e de Computação (ICMC), Universidade de São Paulo, São Carlos.

Maedche, A. and Staab, S. 2001. Ontology learning for the semantic Web. IEEE Intelligent Systems Mar/Apr pp. 72-79. EUA.

Mahmoud, Q.H. (ed.).Middleware for Communications. Wiley, 2004. ISBN: 978-0-470-86206-3.

Manola, F.; Miller, E.; McBride, B. RDF Primer - W3C Recommendation 10 February 2004. Disponível em: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

McGuinness, D. L.; Fikes, R.; Hendler, J.; Stein, L. DAML+OIL: an Ontology Language for the Semantic Web. IEEE Intelligent Systems. v. 17, n. 5, 2002. p. 72-80.

McGuinness, D. L.; Van Harmelen, F. OWL Web Ontology Language Overview - W3C Recommendation 10 February 2004. Disponível em: http://www.w3.org/TR/2004/REC-owl-features-20040210/.

Mei,J.; Bontas, E.P. Reasoning Paradigms for SWRL-enabled Ontologies. FU Berlin Technical Report B 05-01 March 2005.

Oberle, D.; Staab, S.; Eberhart, A. “Towards Semantic Middleware for Web Application Development”. IEEE Distributed Systems Online, February, 2005. ISSN: 1541-4922. Disponível em: http://dsonline.computer.org/portal/site/dsonline/menuitem.9ed3d9924aeb0dcd82ccc6716bbe36ec/index.jsp?&pName=dso_level1&path=dsonline/topics/was/papers&file=oberle.xml&xsl=article.xsl&.

Ontoprise - Ontoprise produkts. 2003. Disponível em http://www.ontoprise.de/content/index_eng.html

Ousterhout, J. K. Scripting: Higher Level Programming for the 21st Century. IEEE Computer. v. 31, n. 3, p. 23-30, 1998. Disponível em: http://home.pacbell.net/ouster/scripting.html.

Pawlan, M. Java 2 Enterprise Edition Technology Center. March 23, 2001. Disponível em http://java.sun.com/developer/technicalArticles/J2EE/Intro/.

Platt, D. S. Introducing Microsoft .Net. Redmond: Microsoft Press, 2001.

Raggett, D.; Le Hors, A.; Jacobs, I. (1999). Hypertext Markup Language (HTML) 4.01, W3C Recommendation. Disponível em http://www.w3.org/TR/1999/REC-html401-19991224/.

Smith, M. K.; Welty, C.; McGuinness D. L.. OWL Web Ontology Language Guide (2004). Disponível http://www.w3.org/TR/2004/REC-owl-guide-20040210/.

Gudgin, M.; Hadley, M.; Mendelsohn, N.; Moreau, J.; Nielsen, H. F. Simple Object Access Protocol (SOAP) 1.2. June 2003. W3C Recommendation. Disponível em http://www.w3.org/TR/soap12-part1/.

Studer R., 2001. KArlsruhe ONtology and semantic web tool suite. Disponível em http://kaon.semanticweb.org/researchers

Tetlow, P.; Pan, J.; Oberle, D.; Wallace, E.; Uschold, M.; Kendall, E. "Ontology Driven Architectures and Potential Uses of the Semantic Web in Software Engineering", W3C Draft, June 2005. Disponível em: http://www.w3.org/2001/sw/BestPractices/SE/ODA/.

Bellwood, T.; Clement, L.; von Riegen. C. Universal Description Discovery & Integration (UDDI). UDDI Version 3.0.2 - UDDI Spec Technical Committee Draft, Dated 20041019. Disponível em: http://uddi.org/pubs/uddi_v3.htm.

Volz, R., Oberle, D., Staab, S., Motik, B. " KAON SERVER - A Semantic Web Management System" In Alternate Track Proceedings of the Twelfth International World Wide Web Conference, WWW2003, Budapest, Hungary, 20-24 May 2003. ACM, 2003. Disponível em: http://www.aifb.uni-karlsruhe.de/Publikationen/showPublikation_english?publ_id=209

Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S. Web Services Description Language (WSDL) 1.1 - W3C Note 15 March 2001. Disponível em: http://www.w3.org/TR/2001/NOTE-wsdl-20010315.