Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Proposta de Integração de Sistemas Computacionais Utilizando Ontologias

Eluzaí Souza dos Santos

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Profª. Drª. Célia Ghedini Ralha

Brasília
2006

Universidade de Brasília – UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenadora: Profª. Drª. Alba Cristina Magalhães de Mello

Banca examinadora composta por:

Profª. Drª. Célia Ghedini Ralha (Orientadora) – CIC/UnB
Profa. Dra. Marisa Bräscher – CID/UnB
Profa. Dra. Fernanda Lima – UCB

CIP – Catalogação Internacional na Publicação

Souza dos Santos, Eluzaí.

Uma Proposta de Integração de Sistemas Computacionais Utilizando Ontologias / Eluzaí Souza dos Santos. Brasília : UnB, 2006.
106 p. : il. ; 29,5 cm.

Tese (Mestre) – Universidade de Brasília, Brasília, 2006.

1. Web Semântica, 2. Ontologia, 3. Engenharia de Software,
4. Engenharia Ontológica, 5. Arquitetura Dirigida a Modelo,
6. Arquitetura Dirigida a Ontologia, 7. OWL, 8. XSLT.

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro – Asa Norte
CEP 70910–900
Brasília – DF – Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Proposta de Integração de Sistemas Computacionais Utilizando Ontologias

Eluzaí Souza dos Santos

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Profª. Drª. Célia Ghedini Ralha (Orientadora)
CIC/UnB

Profa. Dra. Marisa Bräscher
CID/UnB

Profa. Dra. Fernanda Lima
UCB

Profª. Drª. Alba Cristina Magalhães de Mello
Coordenadora do Mestrado em Informática

Brasília, 14 de Dezembro de 2006

Dedicatória

Dedico este trabalho aos meus pais, exemplos de persistência, que sempre me incentivaram a trilhar a estrada do conhecimento.

Agradecimentos

À Deus, razão do meu viver, por tudo que Ele é, e têm sido pra mim. Por fazer valer em minha vida o que diz em Josué 1:9 - "Esforça-te, e tem bom ânimo; não temas, nem te espantes; porque o Senhor teu Deus é contigo, por onde quer que andares". À Prof.ª Dr.ª Célia Ghedini Ralha por sua dedicação e incentivo, por ser uma orientadora presente, mesmo em momentos de cansaço que a vida de mestre proporciona. Agradeço também ao Prof. Dr. Hervaldo Sampaio Carvalho e ao mestrando Alexandre Bellezi José pelos valiosos esclarecimentos prestados sobre o funcionamento do coração e do Projeto ECO. Ao Prof. Dr. José Carlos Ralha, por seu auxílio na descoberta do mundo LaTeX. Às minhas irmãs, pela torcida organizada. Aos amigos do Centro de Informática (CPD/UnB), por todo o apoio proporcionado durante esta jornada.

Resumo

Apresentamos neste documento uma proposta de aplicação de ontologias para integração de sistemas computacionais, trabalhando com a definição de Espaços Tecnológicos, um conceito que permite averiguar como trabalhar de forma eficiente utilizando-se as melhores possibilidades de diferentes tecnologias. Utilizamos as tecnologias da Engenharia Ontológica e da Engenharia de Software, tecnologias que estão sendo desenvolvidas paralelamente por duas comunidades diferentes, mas que por possuírem pontos comuns, podem ser tratadas em conjunto: as linguagens padrão MDA, como a UML, e as linguagens ontológicas, como a OWL, padrão W3C. O foco do estudo é a transformação dos modelos definidos em MDA através da abordagem da Engenharia Ontológica, obtendo uma ontologia que pode ser aprimorada e utilizada para implementação prática em sistemas computacionais relacionados, eliminando ambiguidade e proporcionando integração entre os sistemas de software atuais e os novos, que foram derivados a partir da ontologia.

Palavras-chave: Web Semântica, Ontologia, Engenharia de Software, Engenharia Ontológica, Arquitetura Dirigida a Modelo, Arquitetura Dirigida a Ontologia, OWL, XSLT.

Abstract

In this document we present an ontological approach to integrate computational systems. This proposal adopts the use of technological spaces, a concept that allows the investigation of efficient ways to put together different technologies. We used two technological spaces: Ontology Engineering and Software Engineering. Although these technologies are being developed parallelly by two different communities, they have common points that can be treated together, such as: the MDA pattern languages, UML and the ontological languages, like the W3C pattern OWL. This work focus the transformation of the MDA defined models through an ontological approach, which allows to obtain an ontology that can be validated, extended and used for real computational systems development. Using the proposed approach one can eliminate ambiguity and provide integration among current computational systems and prospective new ones, the last derived from the obtained ontology.

Keywords: Semantic Web, Ontology, Software Engineering, Ontology Engineering, Model Driven Architecture, Ontology Driven Architecture, OWL, XSLT.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

DL

Description Logics 21

MDA

Model-Driven Architecture 52

MOF

Meta Object Facility 53

OUP

Ontology UML Profile 59

OWL

Web Ontology Language 20

RDF

Resource Description Framework 17

UML

Unified Modeling Language 14

URI

Uniform Resource Identifier 19

W3C

World Wide Web Consortium 16

XMI

XML Metadata Interchange 26

XML

eXtensible Markup Language 17

XSLT

Extensible Stylesheet Language Transformation 21

Capítulo 1

Introdução

As técnicas utilizadas pela Engenharia de Software para construção de sistemas provam sua eficácia através da ampla consolidação no mercado. Entre os diversos recursos disponibilizados pela Engenharia de Software, os modelos de domínio desempenham papel central durante todo o desenvolvimento do software, definindo a estrutura básica do sistema. Contudo, para proporcionar interoperabilidade e integração de sistemas faz-se necessário trabalhar em conjunto com uma tecnologia que está sendo reconhecida como um auxílio à construção de sistemas de informação interoperáveis e integrados: as ontologias.

Uma das principais motivações na construção de uma ontologia é a possibilidade de compartilhamento do conhecimento. Ao definirmos um domínio particular, há uma grande possibilidade que uma parte do conhecimento do domínio seja a mesma para uma variedade de aplicações relacionadas. Técnicas de Inteligência Artificial são utilizadas para criação de ontologias, mas segundo (GASEVIC et al., 2004a) estas técnicas estão mais relacionadas à laboratórios de pesquisa, de forma que são desconhecidas pela maior parte da população de Engenharia de Software.

Em nossa proposta, utilizamos as vantagens decorrentes das áreas de Engenharia Ontológica e Engenharia de Software empregando o conceito de Espaços Tecnológicos ou *Technological Spaces*. Este trata-se de um conceito que permite averiguar como trabalhar de forma eficiente utilizando-se as melhores possibilidades de diferentes tecnologias. Segundo (KURTEV; BéZIVIN; AKSIT, 2002), um espaço tecnológico é um contexto de trabalho com um conjunto de conceitos associados, corpo de conhecimento, ferramentas, habilidades necessárias e possibilidades.

Utilizando os Espaços Tecnológicos das duas áreas citadas, apresentamos uma proposta de aplicação de ontologias para integração de sistemas computacionais, com recursos relacionados à estes espaços. Através de estudos, verificamos que é possível utilizar os modelos de domínio especificados através da *Unified Modeling Language* (UML) como base para uma abordagem ontológica, os quais, aliados às facilidades disponibilizadas pelas ferramentas de software e em conjunto com a Arquitetura Dirigida a Modelos apóiam a transformação de conceitos necessários

para criação de ontologias.

Partimos de um diagrama de classes UML de um sistema com o qual pretendemos integrar um novo sistema. Este diagrama é adaptado de forma a englobar as definições do *Ontology UML Profile* (OUP) necessárias à transformação em ontologia, ou seja, aplicamos o OUP utilizando o Poseidon for UML. O OUP é exportado para um arquivo XMI, que é utilizado como entrada na transformação XSLT executada pelo processador Xalan. Nesta transformação além do arquivo de entrada são definidos também o arquivo xslt, que contém as regras de transformação, e o nome do arquivo OWL de saída. A ontologia OWL gerada é então editada no Protégé, onde os axiomas podem ser ajustados e definições necessárias podem passar a ser necessárias e suficientes. A partir do Protégé, e após ter inicializado o raciocinador, são utilizados os serviços de inferências, tais como verificação da consistência da ontologia e da classificação.

Desta forma, o capítulo 2 apresenta uma fundamentação teórica dos Espaços Tecnológicos em questão, descrevendo os principais recursos utilizados em cada área, bem como os modelos ontológicos e os modelos integrados. O capítulo 3 apresenta nossa proposta de trabalho, com a descrição do objetivo e a contribuição. O capítulo 4 descreve a abordagem utilizada neste trabalho, bem como os resultados obtidos da aplicação da proposta em um sistema utilizado como estudo de caso. Finalmente, no capítulo 5 descrevemos nossas conclusões e propostas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Tecnologias Semânticas

Nesta seção faremos uma breve revisão das tecnologias semânticas, incluindo a Web Semântica com a evolução da Web 2.0 e algumas principais linguagens definidas no contexto das tecnologias semânticas.

2.1.1 Web Semântica

A Web atual é interoperável apenas sintaticamente, ou seja, ela é projetada para processamento humano dos significados dos dados. Dessa forma, a Web Semântica é considerada uma evolução da Web atual, no sentido de prover uma infraestrutura aberta, com tecnologia de *Web Services*, para que haja compartilhamento e tratamento da informação e do conhecimento, que será processado automaticamente por máquinas. Esta infra-estrutura é baseada em um modelo de domínio formal denominado ontologia.

O *World Wide Web Consortium* (W3C) é responsável pela padronização da Web Semântica. Como órgão internacional formado por várias organizações e dirigido por Berners-Lee (o idealizador da Web), trabalha para desenvolver padrões da Web. De acordo com o consórcio, para a Web alcançar seu potencial completo é fundamental que as tecnologias sejam compatíveis umas com as outras de forma a permitir que qualquer *hardware* ou *software* possa acessar a Web (JACOBS, 2005).

As aplicações da Web Semântica são estruturadas em duas camadas separadas e ao mesmo tempo, interligadas: a camada da Web Semântica torna as ontologias e as interfaces disponíveis para o público, enquanto a camada interna consiste dos mecanismos de controle e raciocínio. Ou seja, os últimos componentes podem residir dentro de uma "caixa preta", enquanto os artefatos na camada da Web Semântica são compartilhados com outras aplicações, e, portanto devem encontrar padrões da mais alta qualidade do que os de componentes internos. Os modelos na camada da Web Semântica são usados para controlar o compor-

tamento interno, em particular o resultado dos algoritmos de raciocínio. Conseqüentemente, o código dentro dos componentes internos pode ser de tamanho relativamente pequeno e encontrar padrões de mais baixa qualidade do que os módulos visíveis externamente (KNUBLAUCH, 2004).

Atualmente, boa parte dos esforços de pesquisa tem sido dirigida para ontologias de sistemas. Modernas ferramentas de desenvolvimento de ontologias, como o Protégé (com o Plugin OWL), permitem aos usuários encontrar erros e detectar inconsistências, semelhante a um depurador em um ambiente de programação. Além disto, o Protégé gera código a partir de uma ontologia em OWL, criando classes correspondentes em Java.

O desenvolvimento de software dirigido a ontologia (*Ontology-Driven Software Development*) emprega modelos de domínios que não são usados apenas para geração de código, mas também são usados como artefatos executáveis em tempo de execução. Este seria um dos principais benefícios que a Web Semântica pode trazer para a Engenharia de Software. Os modelos de domínios também podem ser compartilhados entre as aplicações em um ambiente aberto proporcionando reuso e interoperabilidade, tal como está previsto na Web Semântica.

Berners-Lee apud (DJURIC; GASEVIC; DEVEDZIC, 2005) define três níveis distintos na arquitetura da Web Semântica que introduzem primitivas expressivas: *metadata layer*, *schema layer* e *logical layer*. A figura 2.1, de (DJURIC; GASEVIC; DEVEDZIC, 2005), ilustra esta arquitetura. A linguagem *eXtensible Markup Language* (XML)/XML *Schema* proporciona uma sintaxe comum, bem definida e de fácil processamento, porém a definição sobre os tipos de dados que elas manipulam é insuficiente. A semântica dos dados é abordada a partir da camada de metadados (*metadata layer*), através do *Resource Description Framework* (RDF), e da camada de esquema (*schema layer*), com a linguagem RDF *Schema* (RDFS). Conforme abordado na seção 2.1.3, o modelo RDF define os relacionamentos entre recursos, mas para permitir raciocínio na Web Semântica seria necessário outra camada. A camada lógica (*logical layer*) introduz linguagens ontológicas, tais como OIL, DAML e OWL, que são baseadas na arquitetura de metamodelo da camada mais baixa.

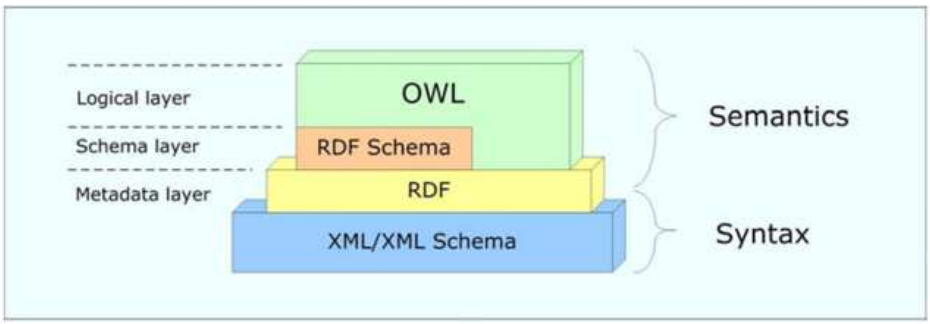

Figura 2.1: Linguagens na arquitetura da Web Semântica (DJURIC; GASEVIC; DEVEDZIC, 2005)

2.1.2 Web 2.0

A Web 2.0 apresenta-se como uma nova idéia cujo foco principal é o conteúdo e a geração de informação de forma interativa. Ela faz uso de recursos com tecnologia não necessariamente nova, como é o caso do *Asynchronous Javascript and XML* (AJAX), que utiliza *Cascating StyleSheet* (CSS) e XML, entre outras. Com isso, aplicações tornam-se acessíveis de qualquer computador, bastando apenas ter um navegador instalado e acesso à Internet.

Partindo deste enfoque, novas funcionalidades e características podem ser constantemente disponibilizadas e sua utilidade e aceitação verificadas diariamente, a partir do comportamento dos usuários que as acessem, que deixam de ser meros consumidores para assumir posição de destaque, interagindo através de uma postura colaborativa.

Reilly (2002) apresenta alguns padrões de projeto da Web 2.0, os quais serão resumidos a seguir:

- O tamanho - pequenos sites, em conjunto, formam o conteúdo da Internet.
- O foco está nos dados - as aplicações estão sendo dirigidas a dados. Por vantagens competitivas, cada empresa procura ser a única dona dos dados, cujas fontes são difíceis de serem recriadas.
- Usuários adicionam valor - a chave para aplicações da Internet com vantagens competitivas está no fato de permitir que os usuários adicionem seus próprios dados àqueles já providos pelas aplicações.
- Alguns direitos reservados - a proteção de propriedade intelectual limita o reuso e evita o experimento. Surge então a proposta de adoção de barreiras pequenas e uso de licenças com poucas restrições, permitindo a mistura de dados e a reutilização por quem quer que seja.
- Perpétua versão Beta - quando dispositivos e programas são conectados pela Internet, aplicações não são artefatos de software: são serviços contínuos. Novas características são adicionadas como parte da experiência dos usuários, que ocupam a posição de testadores em tempo real.
- Cooperação, não controle - aplicações da Web 2.0 são construídas sobre uma rede de serviços de dados cooperados, permitindo sistemas fracamente acoplados.
- Software acima do nível de um único dispositivo - o computador pessoal não é o único dispositivo de acesso às aplicações da Internet, pois as aplicações limitadas por um único dispositivo se tornam menos valiosas. É desejável que os serviços integrados estejam disponíveis também através de dispositivos móveis, servidores de Internet e computadores pessoais.

A Web 2.0 pode ser considerada uma evolução da Web, uma vez que utiliza abordagem colaborativa e evolutiva de conteúdos, facilitando a interoperabilidade e integração de recursos heterogêneos.

2.1.3 Recursos e Linguagens

Encontramos na literatura várias linguagens para a representação de ontologias. Algumas são consideradas base, tal como a linguagem XML, que provê uma sintaxe para documentos estruturados e permite a definição de marcações específicas de uma área, mas não impõe restrições semânticas sobre o significado.

Grande parte das linguagens ontológicas foram definidas pela comunidade da Web Semântica, a saber: RDFS, DAML, DAML+OIL, OWL. Apesar destas linguagens serem baseadas em XML elas obedecem fundamentos de paradigmas de Inteligência Artificial (IA), tais como Lógica Descritiva, redes semânticas, frames, entre outros, permitindo que muitas ontologias estejam sendo desenvolvidas em laboratórios de IA (GóMEZ-PéREZ; CORCHO, 2002). O objetivo da linguagem é especificar como um sistema pode tornar explícito a hierarquia lógica de estruturas de informação, possibilitando seu intercâmbio (HOLMAN, 2000).

Passaremos a exemplificar o uso de XML através de um trecho de código de uma transação comercial onde temos definido a identificação de compra, o código do cliente e do produto no banco de dados, e o valor do produto.

```
01  <?xml version="1.0"?>
02  <purchase id="p001">
03    <customer db="cust123"/>
04    <product db="prod345">
05      <amount>23.45</amount>
06    </product>
07  </purchase>
```

O RDF proporciona meios para tornar os dados mais ricos e mais flexíveis, e portanto, capazes de existir em ambientes fora daqueles explicitamente definidos pelos programadores do sistema e modeladores de dados. Ele descreve modelos conceituais para objetos e as relações entre eles, codificando a informação em conjuntos de triplas, onde cada tripla é comparada a sujeito, verbo e objeto de uma sentença elementar e descrita em uma sintaxe XML (NIEMANN et al., 2005). Um documento ou elemento RDF faz declarações sobre recursos. Uma declaração é composta de três partes: recurso, propriedade e valores. Seu objetivo é dizer que um certo recurso, que pode ser qualquer objeto identificável unicamente por um *Uniform Resource Identifier* (URI), tem uma ou mais propriedades. Cada propriedade tem um tipo, ou seja, um nome, e um valor. O valor de uma propriedade pode ser um literal, tal como um *string*, um número ou o valor pode ser outro recurso (CHAVES, 2001).

RDF *Schema* define um modelo orientado a objeto para RDF e também como classes, subclasses, relacionamentos e propriedades são representados. Dessa forma, RDF *Schema* provê uma infraestrutura para ligar metadados distribuídos (KNUBLAUCH et al., 2005).

XML *Schema* possui um vocabulário para descrever propriedades e classes de um recurso RDF, com uma semântica para generalização e hierarquia das propriedades e classes. Permite definição de elementos, atributos, elementos-filhos e tipos definidos pelo usuário. É também uma linguagem recomendada pelo W3C (W3C, 2005).

DARPA Agent Markup Language (DAML) é uma linguagem que foi desenvolvida como uma extensão para XML e RDF. Provê uma infra-estrutura básica que permite às máquinas fazerem a mesma classificação de inferências simples que os seres humanos fazem. Um sistema em DAML pode inferir conclusões que não estão explicitamente declaradas em DAML (W3C, 2005).

Ontology Inference Layer (OIL) é uma camada de inferência e representação baseada na Web, que combina a utilização de modelagem de primitivas provenientes das linguagens baseadas em frames com a semântica formal. A sintaxe de OIL é baseada em RDF Schema e provê definição de classes e slots (relações), e um conjunto limitado de axiomas (SU; ILEBREKKE, 2002).

A linguagem DAML+OIL foi construída com base na linguagem DAML em um esforço para combinar vários componentes da linguagem OIL. É construída sobre padrões W3C tais como RDF and RDF Schema, e estende estas linguagens com primitivas de modelagem mais ricas, além de possuir semântica clara e bem definida (W3C, 2001).

A *Web Ontology Language* (OWL) é uma linguagem de marcação semântica para publicação e compartilhamento de ontologias na Web. OWL foi desenvolvida como uma extensão do vocabulário RDF e derivada da linguagem ontológica para Web DAML+OIL (Miller apud (DJURIC; GASEVIC; DEVEDZIC, 2005)). A OWL representa o significado de termos em vocabulários e relacionamentos entre eles, facilitando a interpretação do conteúdo Web por máquinas, já que oferece um vocabulário adicional com uma semântica formal. Ex: relações entre classes, cardinalidade, características de propriedades. A OWL é uma extensão do RDF. Enquanto o RDF promove a associação e integração de dados distribuídos, a OWL possibilita o raciocínio sobre esses dados (NIEMANN et al., 2005). OWL é dividida em três sub-linguagens, de acordo com o nível de formalidade exigido e a liberdade dada ao usuário para a definição de ontologias: OWL *Lite*, OWL DL e OWL *Full*.

Sintaticamente, OWL *Lite* é a sub-linguagem mais simples. É indicada para usuários que precisam de uma hierarquia de classificação e restrições simples. Pode ser útil em migrações de taxonomias existentes para OWL, e possui complexidade formal mais baixa do que OWL DL.

OWL DL proporciona expressividade máxima e garantia de decidibilidade e completude computacional. Esta linguagem inclui todos os construtores OWL

mas adiciona algumas restrições. As restrições mais significativas são: uma classe não pode ser um indivíduo ou propriedade e uma propriedade não pode ser um indivíduo ou classe. OWL DL possui esse nome devido à sua correspondência com *Description Logics* (DL), um campo de pesquisa da lógica formal sobre o qual se baseia a linguagem OWL (MCGUINNESS; HARMELEN, 2004).

A OWL *Full* provê expressividade máxima, além da sintaxe livre do RDF, mas não provê qualquer garantia computacional. A principal característica da OWL *Full* em comparação com a OWL DL e OWL Lite é que uma classe, por definição uma coleção de indivíduos, pode ser um indivíduo de outra classe, como em RDF *Schema*. Segundo (DJURIC; GASEVIC; DEVEDZIC, 2005), OWL *Full* é uma extensão da OWL DL, que é uma extensão da OWL *Lite*; desta forma, toda ontologia OWL *Lite* é uma ontologia OWL DL e OWL *Full*, e toda ontologia OWL DL é uma ontologia OWL *Full*.

A *Extensible Stylesheet Language Transformation* (XSLT) é uma recomendação W3C que permite interoperabilidade de informações. É uma linguagem projetada inicialmente para transformar vocabulários XML para vocabulário de formatação *Extensible Stylesheet Language* (XSL), o que não impede sua utilização para outras transformações. A natureza declarativa da marcação das folhas de estilo torna a XSLT muito mais acessível a usuários não especialistas em computação. Em XSLT a informação é vista como uma árvore de nodos abstratos. XSLT inclui construtores que podemos usar para identificar e interagir sobre as estruturas encontradas em nossa fonte de informação. A informação que está sendo transformada pode ser percorrida em qualquer ordem, quantas vezes forem necessárias, para produzir o resultado desejado (HOLMAN, 2000).

O vocabulário hierárquico incluso na linguagem XSL captura o formato semântico para interpretação de informação gráfica e textual em diferentes mídias. Um agente de interpretação tem a atribuição de interpretar uma instância do vocabulário para uma dada mídia. De fato, o que ocorre é a transformação de um documento XML em um documento estruturado com outra sintaxe, através da transformação de instâncias do vocabulário XML em instâncias de um vocabulário de interpretação particular.

XSLT é um componente que integra a linguagem XSL. Quando utiliza um formato de vocabulário semântico como uma linguagem de interpretação, o objetivo de um escritor de folhas de estilo (*stylesheet*) é converter uma instância XML de algum vocabulário XML arbitrário em uma instância de um formato de vocabulário semântico. Holman (2000) ressalta que o resultado da transformação não pode conter qualquer construtor do vocabulário definido pelo usuário (e.g., um identificador de cliente) porque o agente de construção pode não saber o que fazer com os construtores nomeados com identificadores desconhecidos.

2.2 Engenharia de Software e Sistemas

Os projetistas de software procuram construir software evitando problemas decorrentes de esforços mal-sucedidos. No entanto, para se obter sucesso, faz-se necessário disciplina e técnica. A Engenharia de Software proporciona a técnica e disciplinas necessárias, inerentes a todo projeto de software. Podemos verificar esses elementos essenciais através das várias definições encontradas na literatura. Segundo (PRESSMAN, 2002) são definições de Engenharia de software:

- A criação e utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe eficientemente em máquinas reais.
- Aplicação de uma abordagem sistemática, disciplinada e quantificável, para o desenvolvimento, operação e manutenção do software; isto é, a aplicação de engenharia ao software.

As próximas subseções abordarão dois fatores essenciais para a Engenharia de Software: o processo de desenvolvimento de sistemas de software e a linguagem de modelagem.

2.2.1 Metodologias

2.2.1.1 UP

O *Unified Process* (UP) é um método de desenvolvimento de sistemas de software com um *framework* genérico para processos de desenvolvimento, totalmente integrado à linguagem *Unified Modeling Language* (UML). O método foi proposto por três dos principais nomes da indústria de software: Ivar Jacobson, James Rumbaugh e Grady Booch, sendo o resultado de mais de 30 anos de experiência acumulada.

O UP também é conhecido como *Rational Unified Process* (RUP) pois após ser vendido para a *Rational Software*, teve o nome da empresa incorporado ao seu. Normalmente as organizações que desejam utilizar a terminologia e o estilo global do RUP, sem usar os produtos licenciados da *Rational*, se referem ao processo como UP.

Este processo de desenvolvimento explora integralmente as capacidades do padrão UML e baseia-se nas práticas comuns aos projetos de software do mercado. O UP é basicamente formado por: Processo + Métodos + Linguagem (UML) (JACOBSON; BOOCH; RUMBAUGH, 1999). Ele proporciona uma abordagem disciplinada para a atribuição de tarefas e de responsabilidades dentro de uma organização de desenvolvimento. Sua meta é garantir a produção de software de alta qualidade que atenda às necessidades dos usuários, dentro de uma programação e orçamento previsíveis.

O desenvolvimento de sistemas seguindo o UP é um processo: dirigido por casos de uso (Use Cases), centrado na arquitetura, iterativo e incremental.

- Processo dirigido por casos de uso (Use Cases)

O caso de uso é um modelo que define o fluxo de trabalho do sistema da perspectiva dos usuários e subsistemas. O conjunto de casos de uso do sistema é definido no Diagrama de Casos de Uso, que compõe a especificação funcional do sistema, ou seja, define os requisitos do sistema. Ator é algo que interage com o sistema a ser desenvolvido. Pode ser usuário ou mesmo um outro sistema. Os casos de uso dirigem várias atividades de desenvolvimento, tais como: criação e validação da arquitetura do sistema; criação de casos de teste; planejamento das iterações; criação da documentação do usuário e implantação do sistema (JACOBSON; BOOCH; RUMBAUGH, 1999).

- Processo centrado na arquitetura

O processo fornece melhor compreensão do sistema e organização do desenvolvimento, bem como uma base sólida para a construção do software. A descrição da arquitetura envolve elementos mais importantes, como a coleção de visões dos modelos do sistema, além de prescrever o refinamento sucessivo da arquitetura. A arquitetura representa a forma, enquanto que os casos de uso representam a funcionalidade (JACOBSON; BOOCH; RUMBAUGH, 1999).

- Processo iterativo e incremental

Através dessa abordagem a identificação de riscos é adiantada e o sistema encontra-se melhor preparado para aceitar mudanças de requisitos. A integração dos módulos do sistema é contínua, facilitando os testes e aprendizado. O desenvolvimento é efetuado em mini-projetos (iterações) que geram módulos adicionados na medida em que são concluídos.

Uma das características da UML é que esta é uma linguagem extensível, o que permite sua aplicação em ontologias (abordadas na seção 2.3).

2.2.1.2 Métodos estruturados

Na abordagem estruturada o sistema é dividido em conjuntos de funções e dados. A ênfase está nos procedimentos, que são implementados em blocos estruturados, conforme descrito no Diagrama de Fluxo de Dados (DFD). O DFD executa papel central nesta abordagem, pois ele define os modelos de processo descrevendo o comportamento e o contexto de cada processo. O comportamento do sistema também é definido através de máquinas de estado e a estrutura é definida através do Modelo Entidade-Relacionamento (MER), que posteriormente é traduzido para o Diagrama Entidade-Relacionamento (DER) do banco de dados.

Os métodos estruturados não fornecem apoio para a modelagem de requisitos não funcionais. Geralmente produzem muita documentação detalhada, o que pode ocultar a essência dos requisitos do sistema e dificultar a compreensão.

2.2.2 Linguagem

2.2.2.1 UML

A UML é uma linguagem visual para especificação, construção e documentação de artefatos de sistemas. É uma linguagem de modelagem de propósito geral que pode ser usada com todos os métodos componentes e que pode ser aplicada em todos os domínios de aplicação (e.g, saúde, finanças) e plataformas de implementação (e.g, J2EE e .NET) (OMG, 2004). A definição feita por (BOOCH; RUMBAUGH; JACOBSON, 2005) apresenta a UML como uma linguagem-padrão para elaboração da estrutura de projetos de software.

Uma linguagem de modelagem é a linguagem cujo vocabulário e regras têm seu foco voltado para a representação conceitual e física de um sistema, com abrangência nas diversas fases de desenvolvimento. A visualização proporcionada por essa linguagem permite que o significado de uma hierarquia de classes possa ser inferido, o que não seria percebido diretamente, examinando-se o código fonte de um sistema à procura de todas as classes possíveis na hierarquia (BOOCH; RUMBAUGH; JACOBSON, 2005).

Bastante utilizada em Engenharia de Software para modelar sistemas orientados a objeto, a UML facilita a comunicação entre os analistas e desenvolvedores porque especifica a semântica dos símbolos que utiliza, permitindo assim uma melhor compreensão do sistema. Especificar significa construir modelos precisos, sem ambiguidades e completos. Dessa forma, facilita a tomada de decisões no desenvolvimento de sistemas de software, em termos de análise, projeto e implementação dos mesmos. A UML não é uma linguagem visual de programação, mas seus modelos podem ser diretamente conectados a várias linguagens de programação com orientação a objetos (BOOCH; RUMBAUGH; JACOBSON, 2005).

A UML foi adotada pelo Grupo de Gerenciamento de Objetos, ou *Object Management Group* (OMG) (http://www.omg.org), como linguagem de especificação padrão para sistemas de software com orientação a objetos e atualmente é uma das especificações mais utilizadas. Entre os diversos diagramas definidos pela UML, destacamos o diagrama de classes, comumente utilizado para representar modelos de domínio. Os modelos de domínio possuem um papel central durante todo o ciclo de desenvolvimento do software. Pelo fato de ser uma linguagem independente, a UML pode ser utilizada com qualquer processo de desenvolvimento do software, mas vem sendo bastante utilizada em conjunto com o Processo Unificado.

Muitos autores já sugeriram o uso de UML com os conceitos de Web Semântica (KOGUT et al., 2002). A UML é baseada no paradigma de orientação a objetos; por essa razão, apresenta restrições para o desenvolvimento de ontologias. Se-

gundo (DUDDY, 2002), as propriedades nas linguagens ontológicas obedecem a conceitos de lógica de primeira ordem, enquanto na UML as propriedades (atributos e associações) são definidas dentro do escopo da classe a qual pertencem. Conforme exposto, as extensões de UML (UML 2.0), os *profiles* UML e os padrões da OMG (MDA e ODM) vê minimizar as restrições existentes.

O modelo da UML foi inicialmente projetado para documentar e comunicar idéias de projeto de alto nível. Com o passar do tempo, um crescente número de arquitetos de software queriam que seus modelos UML fossem especificações precisas que pudessem servir como um esquema formal e fiel à implementação correspondente do software (SELIC, 2006). Isto gerou uma pressão para definir a semântica da UML muito mais precisa, ou seja, definir precisamente o relacionamento formal entre os gráficos da UML e o código, e também a semântica dos diagramas UML, de forma a reduzir as ambiguidades do padrão original (SELIC, 2006). Este foi o primeiro motivo para a criação da UML 2.0.

Outro fator importante foi a necessidade de modelar novas tecnologias importantes que surgiram desde a primeira versão, como as aplicações baseadas na Web e arquiteturas orientadas a serviço, que ganharam meios mais diretos de modelagem.

Selic (2006) apresenta detalhadamente as principais novidades da UML 2.0, agrupadas em cinco categorias, que citamos abaixo:

1. Um nível significativamente mais alto de precisão na definição da linguagem - A precisão é necessária para a automação da geração de código, tendo em vista a eliminação de ambiguidade e imprecisão dos modelos.

2. A organização da linguagem foi melhorada - Isto é caracterizado pela modularidade, que torna a linguagem mais acessível a novos usuários. Desta forma, os usuários podem aprender apenas as partes da linguagem que lhes interessam; além disso, interação de ferramentas é facilitada permitindo interoperabilidade entre elas, mesmo que sejam de vendedores diferentes.

3. Melhorias significativas na habilidade para modelar softwares de grande escala - Algumas aplicações de software modernas representam integração de aplicações *stand-alone* existentes dentro de sistemas complexos. Para oferecer suporte a esta realidade, novas capacidades hierárquicas foram adicionadas à linguagem para prover modelagem de software em níveis arbitrários de complexidade.

4. Suporte melhorado para especialização baseada em domínio - Mecanismos de extensão foram consolidados e refinados para permitir refinamentos mais simples e precisos da linguagem base, incluindo a modelagem de processos de negócio e engenharia de sistemas. Os mecanismos de extensão permitem aos modeladores adicionar suas próprias metaclasses, tornando mais fácil definir novos *profiles* UML e estender a modelagem para novas áreas de aplicação.

5. Consolidação global, racionalização e melhoria de vários conceitos de modelagem resultando em uma linguagem simplificada e mais consistente - Isto envolveu consolidação de conceitos e exclusão de conceitos redundantes.

A versão da UML 2.0 foi revista durante um ano por uma *Finalization Task Force* (FTF) e adotada pelo OMG no início de 2005 (BOOCH; RUMBAUGH; JACOBSON, 2005). O resultado geral é uma linguagem com potencial para ajudar os desenvolvedores a construir mais rapidamente sistemas sofisticados e mais confiáveis.

2.2.3 Ferramentas

As ferramentas *Computer Aided Software Engineering* (CASE) são ferramentas utilizadas para modelagem de software. Entre as opções disponíveis no mercado, encontramos algumas que oferecem versões de distribuição livre. Passaremos a descrever duas delas, identificando suas principais vantagens.

2.2.3.1 Java and UML Developers' Environment (JUDE)

JUDE é uma ferramenta de modelagem que suporta projeto de software orientado a objeto em Java combinado com Mapas Mentais (*Mind Map*). A ferramenta é disponibilizada em dois tipos de versões: JUDE/Community e JUDE/Professional. A versão JUDE/Community pode ser obtida livremente (em http://jude.change-vision.com/jude-web). O JUDE/Community apresenta características de modelagem simples e amigáveis, podendo exibir a interface gráfica em várias linguagens, tais como Espanhol e Chinês (JUDE, 2006). Todos os principais diagramas da UML são suportados, utilizando a versão da UML 1.4; além disso, permite a importação de arquivos de código Java e geração automática de diagramas de classe com base em um modelo de informação (JUDE, 2006). Já o JUDE/Professional, o qual é pago, agrega todas as características da outra versão além de disponibilizar os diagramas na versão UML 2.0, permitir a criação e exportação de documentos, junção de projetos, e entrada e saída de arquivos no formato *XML Metadata Interchange* (XMI), que será abordado na seção 2.5.1.

2.2.3.2 Poseidon for UML

O Poseidon for UML apresenta uma interface intuitiva, levando em consideração a usabilidade e a produtividade. O Poseidon for UML é uma ferramenta derivada do ArgoUML, que foi desenvolvido por um projeto de código aberto da Universidade de Hamburg. A ferramenta é disponibilizada em seis tipos de edições, com características e níveis de suporte diferentes.

Entre as edições destacamos a Community Edition, distribuída livremente (em http://www.gentleware.com). Ela é totalmente implementada em Java e permite

a modelagem UML completa, pois contém todos os diagramas UML e elementos dos diagramas implementados, podendo ser utilizada para propósito não comercial e principalmente para fins educacionais. Dentre os recursos disponíveis nesta edição citamos a geração de código Java e a exportação dos diagramas para vários formatos, incluindo XMI, além da utilização da UML 2.0. Os recursos não disponíveis são engenharia reversa e carregamento de plugins, disponibilizados nas versões comerciais (GENTLEWARE, 2006). Utilizaremos esta ferramenta para implementação da proposta deste trabalho, tendo em vista as vantagens apresentadas em sua versão livre.

2.3 Engenharia Ontológica

Faremos uma revisão da área de Engenharia Ontológica desde a sua origem, incluindo metodologias de desenvolvimento de ontologias e editores ontológicos.

2.3.1 Visão Geral

O termo Ontologia tem sua origem na Filosofia mas há algum tempo vem sendo utilizado pela comunidade de Ciência da Computação. Antes de apresentar as definições para este termo vale ressaltar a distinção existente entre o significado da palavra Ontologia (com "O"maiúsculo) e ontologia (com "o"minúsculo). Ontologia, conforme (GUARINO, 1998), se refere a um sistema particular de categorias, de acordo com uma certa visão do mundo. Abaixo citamos algumas definições para ontologia:

- Segundo (ALMEIDA, 2003), uma ontologia é a representação de um vocabulário comum de um domínio, no qual são definidos o significado dos termos e as relações entre eles.
- Gruber (2005) define ontologia como sendo a especificação de uma conceitualização, ou seja, ontologia é uma descrição (como uma especificação formal de um programa) dos conceitos e relacionamentos que podem existir para um agente ou comunidade de agentes.
- De acordo com (GUARINO, 1998), uma ontologia refere-se a um artefato de engenharia, constituído por um vocabulário específico usado para descrever uma certa realidade.

Levando em consideração a área abordada por este trabalho, será adotado o conceito de ontologia utilizado pela Ciência da Computação, conforme as definições especificadas por (GUARINO, 1998) e (GRUBER, 2005).

Conforme (KNUBLAUCH, 2004) os componentes básicos de uma ontologia são:

- Classes - organizadas em uma taxonomia,
- Relações - que representam o tipo de interação entre os conceitos de um domínio,
- Axiomas - usados para modelar sentenças sempre verdadeiras e
- Instâncias - utilizadas para representar elementos específicos, ou seja, os próprios dados.

A Engenharia Ontológica é definida por (CANTELE et al., 2004) como a área que estuda aspectos relacionados à construção de ontologias, bem como desenvolvimento de sistemas que utilizem ontologias em sua estrutura. Devedzic (2002) define Engenharia Ontológica como o conjunto de atividades conduzidas não só durante a construção como também durante o projeto e implementação da ontologia. Ele ressalta também que, como ontologias são formadas por entidades e relacionamentos, metodologias da Engenharia de Software tradicional podem ser utilizadas para representá-las, sendo que o resultado da análise orientada a objeto é um esboço da ontologia do domínio relevante para a aplicação.

Ontologias podem ser desenvolvidas por pessoas especializadas em um domínio específico, onde se tem o controle direto sobre o comportamento e execução do sistema e alguns aspectos da implementação. Uma boa prática para garantir implementações eficientes é o trabalho em dupla entre os programadores e os especialistas no domínio. Outro aspecto importante é o teste sistemático, tanto durante o projeto como em tempo de execução (KNUBLAUCH, 2004).

2.3.2 Metodologias

Passaremos a expor alguns métodos e metodologias encontrados na literatura de engenharia ontológica.

2.3.2.1 Ontoclean

A metodologia OntoClean é baseada em noções formais, que são gerais o bastante para serem usadas em qualquer esforço de ontologia, independentemente de um domínio particular. Ela utiliza estas noções para definir um conjunto de meta-propriedades que, por sua vez, são usadas para caracterizar aspectos relevantes do significado pretendido das propriedades, classes e relações que formam uma ontologia (GUARINO; WELTY, 2002). Em adição, as metapropriedades impõem várias restrições na estrutura taxonômica de uma ontologia, que ajudam a avaliar as escolhas feitas.

As noções básicas do OntoClean são: essência, rigidez, identidade e unidade. Uma propriedade não é essencial se ela ocorre como verdadeira acidentalmente. Tomemos como exemplo a propriedade *ser dura*, aplicada a esponjas. Algumas esponjas podem ser duras mas isto não torna *ser dura* uma propriedade essencial

de esponja. Uma forma especial de essência é rigidez. Uma propriedade é rígida se ela é essencial para todas as suas instâncias; uma instância de uma propriedade rígida não pode deixar de ser uma instância desta propriedade em um mundo diferente (GUARINO; WELTY, 2002).

Há propriedades que não são essenciais para suas instâncias. Propriedades que são essenciais para algumas entidades e não são essenciais para outras são chamadas semi-rígidas e as que nunca são essenciais são chamadas anti-rígidas (GUARINO; WELTY, 2002). *Ser dura* é semi-rígida, pois há instâncias que devem ser duras (e.g. martelo) e instâncias que devem ser duras mas podem não ser (por exemplo, esponjas).

As metapropriedades impõem restrições sobre a relação de subordinação. Uma destas restrições é que propriedades anti-rígidas não podem ter propriedades rígidas como suas subordinadas (e.g. a classe pessoa não pode ser subordinada à classe estudante pois esta é anti-rígida e aquela, rígida). Toda instância de estudante pode deixar de ser um estudante enquanto que uma pessoa não pode deixar de ser uma pessoa.

Duas noções filosóficas são usadas nesta metodologia: identidade e unidade (GUARINO; WELTY, 2002). Identidade refere-se ao problema ser capaz de reconhecer entidades individuais no mundo como sendo as mesmas ou diferentes. Unidade se refere a ser capaz de reconhecer todas as partes que formam uma entidade individual. Os critérios de identidade são condições usadas para determinar igualdade. Unidade refere-se ao problema de descrever a forma como as partes de um objeto estão ligadas, de forma a saber o que é parte do objeto e o que não é, e sob quais condições o objeto é um todo. Para algumas classes, todas as suas instâncias são todo, para outras nenhuma das suas instâncias são todo. Por exemplo, a classe água não representa objetos todo. Já oceano pode ser uma classe que representa objetos todo, porque que uma instância dessa classe é uma entidade única.

O OntoClean anexa para cada propriedade (classe) em uma ontologia, metapropriedades adequadas que descrevem seu comportamento com respeito às noções ontológicas anteriormente descritas. Segundo (GUARINO; WELTY, 2002) seus benefícios incluem:

1. Identificação de um *backbone* taxonômico. Este consiste de todas as propriedades rígidas na ontologia, destacando as propriedades mais importantes.

2. Descoberta de inconsistência de hierarquia.

2.3.2.2 Uschold e King

A metodologia de Uschold e King foi desenvolvida com base na experiência adquirida na construção da *Enterprise Ontology* e, segundo (GóMEZ-PéREZ, 1999) propõe os seguintes passos:

1. Identificação do propósito - justificativa da construção da ontologia e seus objetivos

2. Construção da ontologia - dividida em três passos:

   (a) Capturar a ontologia:

      i. Identificação dos conceitos chave e os relacionamentos no domínio de interesse, ou seja, o escopo.

      ii. Produção de definições de texto não ambíguas para os conceitos e relacionamentos

      iii. Identificação dos termos que se referem aos conceitos e relacionamentos.

   (b) Codificar a ontologia - representar o conhecimento adquirido no passo anterior em uma linguagem formal.

   (c) Integrar ontologias existentes - verificar a necessidade de utilização de ontologias existentes e como.

3. Avaliação - os autores propõem um julgamento técnico das ontologias, o ambiente de software associado, e a documentação, que pode ser especificações de requisitos e questões de competência.

4. Documentação - recomenda que diretrizes sejam estabelecidas para documentar ontologias, determinando o tipo e o propósito da ontologia.

2.3.2.3 TOronto Virtual Enterprise (TOVE)

Esta metodologia, desenvolvida por Gruninger e Fox, é baseada na experiência do desenvolvimento da ontologia do projeto TOVE, da Universidade de Toronto, dentro do domínio de processos de negócios e modelagem de atividades. Segundo (LóPEZ, 1999), TOVE constrói um modelo lógico de conhecimento que será especificado pelo significado da ontologia . Esse modelo não é construído diretamente. Primeiro, é feita uma descrição informal das especificações para serem conhecidas pela ontologia e então, essa descrição é formalizada. São propostos os seguintes passos:

1. Captura dos cenários de motivação. De acordo com os autores da metodologia, o desenvolvimento das ontologias é motivado pelos cenários que surgem na aplicação. Os cenários são problemas armazenados ou exemplos que não são adequadamente focalizados pelas ontologias existentes. Um cenário de motivação também proporciona um conjunto de soluções intuitivamente possíveis para os problemas do cenário.

2. Formulação das questões de competência. São baseadas nos cenários obtidos no passo anterior e podem ser consideradas como exigências na forma de questões. Uma ontologia deve ser capaz de representar estas questões usando sua terminologia e caracterizar as respostas para as questões usando

axiomas e definições. Estas questões são questões de competência informais, já que não são expressas em uma linguagem formal de ontologia. As questões servem como restrições sobre o que a ontologia pode ser, e são usadas para avaliar se os propósitos ontológicos estão de acordo com os requisitos.

3. Especificação da terminologia da ontologia dentro de uma linguagem formal.

   (a) Obter a terminologia informal. O conjunto de termos usado pode ser obtido das questões de competência informais, já definidas. Estes termos servem de base para a especificação da ontologia em uma linguagem formal.

   (b) Especificação da terminologia formal. A terminologia da ontologia é especificada usando formalismo. Estes termos permitirão que definições e restrições sejam posteriormente expressas por meio de axiomas.

4. Formulação das questões de competência formais usando a terminologia da ontologia. Após a terminologia da ontologia ser definida, as questões de competência são definidas formalmente.

5. Especificação dos axiomas e definições para os termos na ontologia dentro de uma linguagem formal. Os axiomas em uma ontologia especificam as definições dos termos na ontologia e restrições sobre sua interpretação; eles são definidos como sentenças de primeira ordem. O desenvolvimento de axiomas para a ontologia, verificando as questões de competência formais é um processo iterativo.

6. Estabelecimento de condições para caracterizar a completude da ontologia. Deve-se definir as condições sobre as quais as soluções para as questões de competência são completas.

2.3.2.4 METHONTOLOGY

O *framework* METHONTOLOGY permite a construção de ontologias no nível de conhecimento e inclui a identificação do processo de desenvolvimento da ontologia, um ciclo de vida baseado na evolução dos protótipos e técnicas particulares para execução de cada atividade, cujos detalhes serão apresentados nos próximos parágrafos. Este *framework* é apoiado por um ambiente chamado *Ontology Design Environment*(ODE) (LóPEZ, 1999).

O processo de desenvolvimento da ontologia define as atividades executadas durante a construção de ontologias. Segundo (LóPEZ, 1999), existem três categorias de atividades:

1. Atividades de Gerenciamento de Projeto - incluem planejamento, controle e garantia de qualidade. O planejamento identifica quais tarefas serão executadas, como elas serão organizadas, quanto tempo e quais os recursos são necessários para sua conclusão. O controle garante que as tarefas planejadas

serão concluídas da forma que elas foram projetadas para serem desempenhadas. A garantia da qualidade garante que a qualidade de cada produto resultante (ontologia, software e documentação) seja satisfatória.

2. Atividades Orientadas a Desenvolvimento - incluem especificação, conceitualização, formalização e implementação. Especificação define o objetivo da ontologia, em quais situações pretende-se utilizá-la e identifica o conjunto de termos a serem representados, juntamente com suas características. A fase de conceitualização organiza e converte a visão do domínio percebida informalmente em uma especificação semi-formal, usando um conjunto de representações intermediárias que o especialista no domínio e o ontologista (desenvolvedor da ontologia) podem entender. A formalização transforma o modelo conceitual em um modelo formal. Implementação constrói modelos computáveis em uma linguagem computacional. A manutenção atualiza e corrige a ontologia.

3. Atividades de Suporte - incluem uma série de atividades, desempenhadas ao mesmo tempo que as atividades orientadas ao desenvolvimento, sem as quais a ontologia não pode ser construída. Elas incluem aquisição do conhecimento, avaliação, integração, documentação e gerenciamento de configuração. Aquisição do conhecimento adquire o conhecimento através de um dado domínio. Avaliação faz um julgamento técnico das ontologias, seus ambientes de software e documentação associados com respeito a cada fase e entre as fases dos seus ciclos de vida. Integração de ontologias é necessária quando a nova ontologia que está sendo construída reusa outras ontologias que já estão disponíveis. A documentação detalha, claramente e exaustivamente, cada uma das fases concluídas e os produtos gerados. Gerenciamento de configuração registra todas as versões da documentação, software e código de ontologia para controle de mudanças.

O ciclo de vida da ontologia identifica o conjunto de estágios através do qual a ontologia segue durante seu tempo de vida, descreve quais atividades serão desempenhadas em cada estágio e como os estágios estão relacionados (LóPEZ, 1999).

2.3.2.5 Método 101

Este método utiliza um enfoque iterativo para o desenvolvimento de ontologia, ou seja, ele parte de uma versão inicial da ontologia, que será revisada e refinada aos poucos. Noy e McGuinness (2001) enfatizam algumas regras consideradas fundamentais no projeto de ontologia:

- Não há um modo correto de modelar um domínio - há alternativas viáveis.
- O desenvolvimento de ontologia é necessariamente um processo iterativo.

- Conceitos em ontologia devem estar ligados a objetos e relacionamentos em seu domínio de interesse.

Levando em consideração estas regras, o método é baseado em sete passos, a saber:

1. Determinar o domínio e escopo da ontologia. Para começar definindo o escopo e domínio da ontologia são apresentadas algumas das questões básicas:

   - Qual o domínio que a ontologia cobrirá?
   - Qual será o uso da ontologia?
   - Para quais tipos de perguntas a informação na ontologia deve proporcionar respostas?
   - Quem usará e manterá a ontologia?

   As respostas para estas perguntas podem mudar durante o processo de projeto da ontologia, mas em qualquer momento pode ajudar a definir o escopo do modelo. Uma das formas de determinar o escopo da ontologia é esboçar uma lista de questões que uma base de conhecimento baseada na ontologia deve ser capaz de responder: as questões de competência (GRÜNINGER; FOX, 1995). Estas questões servirão para testar a ontologia posteriormente e podem ser apenas um esboço, não precisam ser exaustivas.

2. Considerar o reuso de ontologias existentes. O reuso de ontologias pode ser um requisito se o novo sistema interage com outras aplicações que já tem ontologias ou vocabulários controlados. Podemos encontrar bibliotecas de ontologias reusáveis na Web, tais como Ontolingua. Através do reuso de ontologias é possível refinar e extender as fontes existentes para o nosso domínio particular.

3. Enumerar os termos importantes da ontologia. Os termos sobre os quais podem ser feitas declarações devem ser listados, junto com suas propriedades. Por exemplo, em uma ontologia sobre vinhos, termos importantes relacionados a vinhos podem ser: localização, cor e uva. Inicialmente é importante ter uma lista de termos, sem se preocupar com sobreposição entre os conceitos que eles representam, relações entre os termos e as propriedades que eles podem ter. Os próximos dois passos - definição da hierarquia de classes e das propriedades dos conceitos - estão intimamente interligados. Geralmente, nós criamos algumas definições de conceitos na hierarquia e descrevemos propriedades desses conceitos; depois definimos mais conceitos e assim por diante.

4. Definir classes e a hierarquia de classes. Há vários enfoques no desenvolvimento de hierarquia:

   - Um processo de desenvolvimento *top-down* começa com a definição dos conceitos mais gerais no domínio e posteriormente, é feita a especialização dos conceitos.

- Um processo de desenvolvimento *bottom-up* começa com a definição das classes mais específicas, com subseqüente agrupamento destas classes em conceitos mais gerais.
- Um processo de desenvolvimento de combinação é formado pela combinação dos dois enfoques anteriores. Nós definimos os conceitos que mais se sobressaem primeiro e depois generalizamos e especializamos estes conceitos adequadamente.

Qualquer que seja o enfoque escolhido, geralmente começamos definindo classes. Da lista formada no passo 3, selecionamos os termos que descrevem os objetos que possuem uma existência independente. Esses termos serão classes na ontologia. Organizamos então as classes em uma taxonomia hierárquica, verificando que: se a classe A é uma superclasse de B, então toda instância de B é também uma instância de A. Neste caso, a classe B representa um conceito que é "um tipo de"A.

5. Definir as propriedades das classes. Após definirmos algumas classes, devemos descrever a estrutura interna dos conceitos. Os termos que sobraram da lista de termos do passo 3 provavelmente são propriedades. Para cada propriedade na lista, nós devemos determinar quais as classes que ela descreve. Todas as subclasses de uma classe herdam as propriedades desta classe.

6. Definir os valores das propriedades. Propriedades podem descrever o tipo de valor, valores permitidos, o número de valores (cardinalidade). Uma propriedade pode ter como tipo um valor que é uma instância, cujo escopo é uma classe específica. Em algumas linguagens, como OWL, é permitido utilizar tipos de dados no preenchimento de valores de propriedades. Segundo (BREITMAN, 1999), os tipos de dados mais comuns são: cadeia de caracteres, números, booleanos e listas enumeradas de elementos.

7. Criar instâncias. O último passo é criar instâncias individuais das classes na hierarquia. Para isto é necessário: escolher uma classe, criar uma instância dessa classe e preencher os valores dos *slots*.

Neste trabalho utilizamos este método por apresentar passos bem definidos e explicados, de forma que é possível seguí-los com facilidade, conforme será apresentado no capítulo 4.

2.3.3 Editores

Faremos uma revisão dos editores ontológicos mais utilizados, apresentando suas principais funcionalidades.

2.3.3.1 Protégé

O Protégé é um ambiente integrado usado por desenvolvedores de sistemas e especialistas em um domínio específico para desenvolver e manter uma ontologia. Através dele é possível descrever os conceitos pertencentes à ontologia, juntamente com seus atributos e relacionamentos. O Protégé possui uma arquitetura de metaclasses, documentos de formato padrão usados para definir novas classes em uma ontologia, que o tornam facilmente extensível e permite o seu uso em conjunto com outros modelos de conhecimento. Os objetivos do Protégé são: apresentar interoperabilidade com outros sistemas de representação do conhecimento e facilidade de uso e configuração.

Esta ferramenta foi desenvolvida na Universidade de Stanford e está disponível para utilização gratuita (http://protege.stanford.edu). Fornece uma interface gráfica amigável para o desenvolvimento de ontologias, proporcionando o acesso rápido às informações importantes, podendo-se manipular diretamente a ferramenta para navegação e gerenciamento da ontologia. Ao contrário de outras ferramentas, o Protégé utiliza a instalação local ao invés de utilizar uma arquitetura cliente-servidor através da Internet. Além disso, gera saídas em algumas linguagens ontológicas e possibilita aos usuários modificá-lo para ser editor de uma linguagem específica (MIZOGUCHI, 2005).

Para utilizar serviços de raciocínio sobre ontologias, o Protégé oferece uma opção para definição do raciocinador a ser utilizado e para conectar-se ao raciocinador, utiliza a interface DIG provida pelo *Description Logic Implementers Group* (DIG). O acesso aos principais serviços de raciocínio, tais como verificação da consistência, classificação da taxonomia e de indivíduos, é facilitado por estar disponível tanto através do menu, como através de botões.

Utilizaremos esta ferramenta para implementação da proposta deste trabalho, tendo em vista as vantagens apresentadas, objetivos e a disponibilidade da ferramenta sem custo.

2.3.3.2 OntoEdit

OntoEdit é um ambiente gráfico de desenvolvimento para projeto e manutenção de ontologias. Possui embutida uma máquina de inferência chamada FaCT. O processo de desenvolvimento de ontologia no OntoEdit é baseado em uma metodologia própria denominada On-To-Knowledge. Dois plugins, OntoKick e Mind2Onto, são utilizados para apoiar a fase de captura de ontologia. OntoKick é projetado para engenheiros da computação que estão familiarizados com o processo de desenvolvimento de software visto que seus objetivos são a criação da especificação de requisitos e a construção de estruturas relevantes para descrição de uma ontologia semi-formal (MIZOGUCHI, 2005).

Mind2Onto é um plugin que oferece suporte para *brainstorming* e discussão sobre estruturas da ontologia. Métodos de *brainstorming* são comumente utilizados para captura de conhecimento relevante. Mind2Onto integra uma ferramenta

de mapas mentais chamada MindManager. Mapas mentais facilitam as discussões e proporcionam aprendizado eficiente.

Atualmente, o OntoEdit foi sucedido pelo OntoStudio, que é baseado no ambiente de desenvolvimento Eclipse, da IBM, e está disponível nas versões profissional e livre, esta última para uso não comercial. A versão livre possui o módulo de memória principal e a versão profissional apresenta também integração com Banco de Dados. Segundo (ONTOPRISE, 2006), as linguagens suportadas são: OWL e RDF, além de F-logic e OXML. Estas últimas são linguagens otimizadas para processamento de ontologias baseado em lógica. O OntoStudio inclui também um avaliador, chamado OntoStudio Evaluator, usado na implementação de regras durante a modelagem (ONTOPRISE, 2006).

2.3.3.3 DOE - The Differential Ontology Editor

É um editor simples de ontologia que utiliza a metodologia de Bruno Bachimont para criação de ontologias. Segundo (ISAAC; TRONCY, 2006), o processo é dividido em três passos :

1. São construídas as taxonomias dos conceitos e relações, justificando explicitamente a posição de cada item na hierarquia. Feito isto, o usuário pode então adicionar sinônimos e definições.

2. São consideradas duas taxonomias do ponto de vista semântico extensional. O usuário poderá aumentá-las ou adicionar restrições às relações.

3. A ontologia pode ser traduzida em uma linguagem de representação do conhecimento. As linguagens disponibilizadas são: RDFS, OWL, DAML+OIL, OIL, CGXML (linguagem para especificação de gráficos conceituais).

O DOE não é um ambiente de desenvolvimento completo de ontologia. Ele oferece técnicas inspiradas em lingüística, que atribuem uma definição léxica aos conceitos e relações usados, e justifica a hierarquia do ponto de vista teórico, que pode ser entendido por pessoas.

2.3.3.4 OilEd

O OilEd é um editor cujo objetivo inicial era demonstrar o uso e estimular o interesse na linguagem OIl. Ele utiliza o raciocinador FaCT, para verificar a consistência das ontologias, junto com a interface DIG, que permite o uso de outros raciocinadores para classificação das ontologias. O OilEd atualmente se encontra na versão 3.5.7 e permite exportação para OWL, além de DAML+OIL e RDFS (MANCHESTER, 2006).

2.3.3.5 Chimaera

Chimaera é um sistema de software que permite aos usuários criar e manter ontologias distribuídas na Web. As duas maiores funções que ele suporta são merge de múltiplas ontologias e diagnóstico de várias ontologias ou individualmente. Também permite o tratamento de bases de conhecimento em diferentes formatos, reorganização de taxonomias, resolução de conflitos de nome, edição de termos, entre outras opções. O Chimaera está disponível para uso na Internet, solicitando registro para acesso completo à sua versão funcional. Ele pode carregar e exportar arquivos no formato OWL e DAML (STANFORD, 2006).

2.4 Modelos Ontológicos

Nesta seção apresentamos modelos de ontologias através das principais linguagens.

2.4.1 Lógica de Descrição

Lógica de Descrição ou *Description Logic* (DL) faz parte de uma família de formalismos para representação do conhecimento baseada em lógica. Trata-se de uma forma restrita da lógica de primeira ordem, que proporciona raciocínio automatizado e descreve o domínio com base em três noções fundamentais:

1. Indivíduos: representam objetos no domínio, e.g. José, Maria, Itália.
2. Conceitos: descrevem conjuntos de indivíduos (classes), e.g. Pessoa, País.
3. Papéis: são relações binárias entre indivíduos, e.g. temCriança.

Há dois tipos de conceito: primitivo e definido. O conceito primitivo é aquele cujas instâncias devem necessariamente preencher certas condições para ser membro do conceito, ao passo que o conceito definido estabelece condições necessárias e suficientes que, se atendidas, automaticamente classifica algo como sendo membro deste conceito.

Segundo (KEPLER et al., 2006), uma DL pode ser caracterizada pelos construtores que ela provê, que são utilizados para formar conceitos e papéis. A lógica *Architecture Language Committee* ($\mathcal{ALC}$) é a DL que engloba os construtores $\sqcup$, $\sqcap$, $\neg$, $\forall$ (quantificador universal) e $\exists$ (quantificador existencial). A *Attributive Language* ($\mathcal{AL}$) é a menor DL de interesse prático; outras Lógicas de Descrição desta família são extensões da $\mathcal{AL}$. A tabela 2.1, encontrada em (KEPLER et al., 2006), ilustra os principais construtores de DLs.

No exemplo abaixo, obtido de (HORROCKS, 2006), representamos em DL todas as pessoas cujos filhos são Doutores ou tem um filho que é Doutor:

$$\text{Pessoa} \sqcap \forall \text{temFilho}.(\text{Doutor} \sqcup \exists \text{temFilho}.\text{Doutor})$$

Símbolo		Nome
$\sqcup$	$\mathcal{U}$	União
$\sqcap$	$\mathcal{AL}$	Interseção
$\neg$	$\mathcal{C}$	Negação de conceito (complemento)
	$\mathcal{A}$	Negação atômica
$\forall$	$\mathcal{AL}$	Restrição de valor
$\exists$	$\mathcal{E}$	Quantificação existencial
$\geq, \leq, =$	$\mathcal{Q}$	Restrição qualificada de número
	$\mathcal{F}$	Restrição funcional de número
I	$\mathcal{O}$	Nominais (enumeração de instâncias)
	$\mathcal{I}$	Papéis inversos
	$\mathcal{H}$	Hierarquia de papéis
	$\mathcal{D}$	Restrição de tipo de dado

Tabela 2.1: Principais construtores de DLs

A principal inferência que a DL desempenha com relação aos conceitos é a hierarquização (*subsumption*), ou seja, a definição de que um conceito está subordinado a outro. Outras deduções desempenhadas pela DL são:

1. Incoerência: Um conceito A é incoerente se é impossível criar um indivíduo que é uma instância de A. Por exemplo, se A for definido como tendo pelo menos 4 portas e no máximo 2 portas.

2. Disjunção: Dois conceitos, A e B, são disjuntos se é impossível criar um indivíduo que é uma instância de ambos.

3. Equivalência: Dois conceitos A e B são equivalentes se todo indivíduo que é uma instância de A, deve ser uma instância de B e todo indivíduo que é uma instância de B deve ser uma instância de A.

4. Classificação: Dada uma base de conhecimento de declarações de conceitos, eles são organizados em uma hierarquia onde para cada conceito A nós identificamos os conceitos mais específicos (*subsumers*) e os conceitos mais gerais (*subsumees*).

A seguir, será apresentada uma breve revisão sobre DL com base em (NARDI; BRACHMAN, 2002). O enfoque funcional para a representação do conhecimento é derivado de esforços históricos utilizados em redes semânticas e *frames*. A descrição funcional é especificada através da interface *Tell&Ask*, que especifica operações que permitem a construção da base de conhecimento (operações *Tell*) e operações que permitem obter informações da base de conhecimento (operações *Ask*).

Dentro de uma base de conhecimento é possível distinguir o conhecimento intensional, ou conhecimento geral sobre o domínio do problema, e o conhecimento

extensional, que é específico de um problema particular. Uma base de conhecimento da DL é tipicamente composta por dois componentes: uma *TBox* e uma *ABox*.

A *TBox* contém o conhecimento intensional na forma de uma terminologia e é construída através de declarações que descrevem propriedades gerais de conceitos. A forma básica de declaração em uma *TBox* é a definição de conceito, ou seja, a definição de um novo conceito com base em outros previamente definidos. Em bases de conhecimento DL uma terminologia é constituída por um conjunto de definições de conceito. Conceitos atômicos e papéis atômicos são descrições elementares; entretanto, há algumas suposições importantes para terminologias DL, a saber:

- É permitida somente uma definição para um conceito;
- Definições são acíclicas, no sentido de que os conceitos não são definidos com base neles mesmos, nem com base em outros conceitos que indiretamente se referem a eles.

Uma *ABox* contém o conhecimento extensional, conhecimento que é específico dos indivíduos do domínio. Uma tarefa básica de raciocínio em *ABox* é a checagem de instância, que verifica se um dado indivíduo é uma instância de um conceito específico. Os papéis da *TBox* e *ABox* foram motivados pela necessidade de distinguir o conhecimento geral sobre o domínio de interesse, do conhecimento específico sobre os indivíduos que caracterizam uma situação específica em consideração.

Uma linguagem de modelagem é o veículo para a expressão de noções de modelagem que são fornecidas aos projetistas. A modelagem em DL requer que o projetista especifique os conceitos do domínio de discurso e caracterize seus relacionamentos para outros conceitos e para indivíduos específicos.

A DL pode ser considerada uma linguagem de modelagem centrada em objetos, já que permite especificar indivíduos (objetos) e definir explicitamente suas propriedades, bem como os relacionamentos entre eles. A capacidade de explorar a descrição do modelo para inferir conclusões sobre o problema em questão é uma vantagem particular da modelagem usando DL.

De acordo com o exposto, podemos perceber que DL pode ser utilizada como base para linguagens cujo objetivo é formalizar um domínio, bem como permitir raciocínio automatizado sobre este domínio.

2.4.2 OWL

A linguagem OWL pode ser considerada como um componente do esforço desenvolvido pela Web Semântica. Como a Web Semântica é distribuída a OWL deve permitir que informações distribuídas sejam tratadas de forma conjunta. Além

disto, a OWL tem como premissa a hipótese de mundo aberto (*open world assumption*), ou seja, as descrições de recursos não ficam confinadas a um escopo único ou arquivo.

Esta seção será escrita com base em (SMITH; WELTY; MCGUINNESS, 2004), de forma pragmática. Uma classe C1 pode ser definida originalmente em uma ontologia O1 e ser extendida em outras ontologias. As conseqüências destas proposições adicionais sobre C1 são monotônicas, ou seja, uma nova informação não pode anular a informação anterior, embora possa ser contraditória e gerar um erro.

Antes de descrevermos os termos de uma ontologia OWL, vamos analizar um componente padrão de toda ontologia: *namespace*. Declarações de *XML namespace* são incluídas em uma tag iniciada com `rdf:RDF` e proporcionam um significado não ambíguo para os identificadores. Uma típica ontologia OWL inicia com uma declaração de *namespace* semelhante ao exemplo abaixo, onde as URIs são personalizadas para cada ontologia:

```
<rdf:RDF
    xmlns ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
    xmlns:vin ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/
    wine#"
    xml:base ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/
     wine#"
    xmlns:food="http://www.w3.org/TR/2004/REC-owl-guide-20040210/
     food#"
    xmlns:owl ="http://www.w3.org/2002/07/owl#"
    xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
```

As duas primeiras declarações identificam o *namespace* associado a esta ontologia. A primeira define o *namespace* padrão, especificando que os nomes utilizados sem prefixo se referem à ontologia atual. A segunda identifica o *namespace* da ontologia atual com o prefixo `vin`. A terceira identifica a URI base para este documento. A quarta declaração identifica o *namespace* de suporte da ontologia *food*, através do prefixo `food`. A quinta declaração é uma declaração OWL convencional, usada para introduzir o vocabulário OWL.

A maioria dos elementos de uma ontologia diz respeito a classes, subclasses, indivíduos, propriedades e relacionamentos entre indivíduos. Passaremos a expor os elementos básicos de uma ontologia OWL:

1. Classe - classes são conceitos básicos em um domínio, que correspondem à raiz de várias árvores taxonômicas. Todo indivíduo em OWL é um membro da classe `owl:Thing`, então todas as classes raiz definidas pelo usuário são subclasses desta classe. OWL também define uma classe vazia: `owl:Nothing`. O código a seguir cria uma classe chamada Winery:

`<owl:Class rdf:ID="Winery"/>`. Uma definição de classe tem duas partes: o nome ou referência e uma lista de restrições. As instâncias da classe pertencem à interseção das restrições.

2. Subclasse - um construtor fundamental para classes é o `rdfs:subClassOf`. Ele relaciona uma classe mais específica a uma classe mais geral. O código do exemplo abaixo cria a classe `EdibleThing` como subclasse da classe `Pasta`:

```
<owl:Class rdf:ID="Pasta">
<rdfs:subClassOf rdf:resource="#EdibleThing"/>
</owl:Class>
```

3. Indivíduo - um indivíduo é introduzido ao declará-lo membro de uma classe. No código abaixo, podemos observar que o indivíduo `CentralCoastRegion` é declarado como membro da classe `Region`.

```
<Region rdf:ID="CentralCoastRegion"/>
```

4. Propriedade - as propriedades nos permitem declarar fatos gerais sobre os membros de classes e fatos específicos sobre indivíduos. OWL possui dois tipos principais de propriedades:

   - propriedade de tipo de dado (*datatype property*) - define relações entre instâncias de classes e tipos de dados de XML *Schema*, tais como `xsd:string` e `xsd:integer`, e literais RDF;
   - propriedade de objeto (*object property*) - define relações entre instâncias de duas classes. Uma propriedade também pode ser uma especialização de uma propriedade existente. Quando definimos uma propriedade há vários meios de restringir a relação. Podemos especificar o domínio (`domain`) e o escopo (`range`) da propriedade. Eles são considerados mecanismos globais, pois se aplicam a todas as instâncias da propriedade, e não apenas a uma propriedade quando ela é associada a uma classe particular. Vejamos um exemplo:

     ```
     <owl:ObjectProperty rdf:ID="madeFromGrape">
        <rdfs:domain rdf:resource="#Wine"/>
        <rdfs:range rdf:resource="#WineGrape"/>
     </owl:ObjectProperty>
     ```

     Neste exemplo, a propriedade `madeFromGrape` tem `Wine` como domínio e `WineGrape` como escopo. A propriedade relaciona instâncias da classe `Wine` à instâncias da classe `WineGrape`.

     Em OWL, `domain` e `range` podem ser usados como axiomas em raciocínio. Por exemplo, se a propriedade `temDefeito` possui `Carro` como seu domínio, e nós aplicarmos esta propriedade para a classe `Navio`, pode-se inferir que a classe `Navio` é uma subclasse de `Carro`; mas se as classes forem disjuntas então será gerado um erro. Da mesma

forma, podemos definir que a propriedade `temFilho` possui a classe `Mamífero` como seu `domain`. A partir disto, um raciocinador pode deduzir que se Frank `temFilho` Ana, então Frank deve ser um `Mamífero`. Se vários domínios forem especificados para uma propriedade então o domínio da propriedade é a interseção das classes.

O `range` de uma propriedade limita os indivíduos que a propriedade pode ter como seu valor. Se a propriedade relaciona um indivíduo a outro indivíduo, e a propriedade tem uma classe como seu `range`, então o outro indivíduo deve pertencer à classe definida no `range`.

OWL também possui um terceiro tipo de propriedade: propriedade de anotação (*annotation property*). Esta propriedade pode ser usada para adicionar informação às classes, indivíduos, propriedades de objeto e de tipos de dados, como metadados.

5. Restrição - como ocorre com as propriedades, definições de classe possuem várias partes. Uma restrição pode fazer parte da definição de uma classe. No exemplo abaixo, a restrição define uma classe sem nome que representa o conjunto de coisas com pelo menos uma propriedade `madeFromGrape`.

```
<owl:Restriction>
   <owl:onProperty rdf:resource="#madeFromGrape"/>
   <owl:minCardinality
    rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:minCardinality>
</owl:Restriction>
```

Já apresentamos os elementos básicos de uma ontologia, e agora passaremos a expor algumas características das propriedades de objetos (*object property*) que proporcionam mecanismos poderosos de raciocínio:

1. Propriedade Transitiva - se uma propriedade P é transitiva, então para qualquer **x**, **y**, e **z**, temos que:

   $$P(x,y) \text{ e } P(y,z) \text{ implica } P(x,z)$$

   Se uma propriedade P é transitiva e relaciona o indivíduo **x** ao indivíduo **y**, e o indivíduo **y** ao indivíduo **z**, então podemos inferir que o indivíduo **x** está relacionado com **z** através da propriedade P.

2. Propriedade Simétrica - se uma propriedade P é simétrica, então para qualquer **x** e **y**, temos que:

   $$P(x,y) \text{ se e somente se } P(y,x)$$

   Se uma propriedade P é simétrica e relaciona o indivíduo **x** ao indivíduo **y** então o indivíduo **y** está relacionado ao indivíduo **x** através da propriedade P.

3. Propriedade Funcional - se uma propriedade P é funcional, então para todo **x**, **y**, e **z**, temos que:

   $$P(x,y) \text{ e } P(x,z) \text{ implica } y = z$$

   Isto significa que uma propriedade funcional só pode ser associada a um único valor.

4. Inverso Funcional (*InverseFunctional*) - se uma propriedade P é definida como *InverseFunctional* então para todo **x**, **y**, e **z**, temos que:

   $$P(y,x) \text{ e } P(z,x) \text{ implica } y = z$$

   O inverso de uma propriedade funcional deve ser *inverse functional*. OWL DL não permite que *InverseFunctional* seja aplicada a uma propriedade de tipo de dado (*datatype property*). Desta forma, se definirmos que `Maria` é mãe de `Carlos`, através da propriedade `éMãeDe` - propriedade inversa de `temComoMãe` (funcional), e também definirmos que `Joana` é mãe de `Carlos`, é possível inferir que `Maria` e `Joana` são o mesmo indivíduo (HORRIDGE, 2004).

5. Inverso de (*inverseOf*) - se uma propriedade P1 é definida como `owl:inverseOf` P2, então para todo **x** e **y**, temos que:

   $$P1(x,y) \text{ se e somente se } P2(y,x)$$

   Em outras palavras, P1 se e somente se P2 significa que P1 implica P2 e P2 implica P1, pois a sintaxe requer um nome de propriedade como argumento.

Além de especificar as características das propriedades, é possível restringir o escopo (`range`) de uma propriedade em contextos específicos. Isto é feito através das restrições de propriedades. Descrevemos a seguir as várias formas de restrição, que só podem ser usadas dentro do contexto de uma `owl:Restriction`, citada anteriormente. Por isso, estas restrições são consideradas mecanismos de restrição local, a saber:

1. allValuesFrom - A restrição `owl:allValuesFrom` define que em todas as intâncias da propriedade especificada, os valores da propriedade são todos membros da classe indicada pela cláusula `owl:allValuesFrom`. O exemplo abaixo especifica que todos os relacionamentos `hasMaker` da classe `Wine` devem ser para um indivíduo da classe `Winery`. O elemento `owl:onProperty` indica a propriedade restringida.

```
<owl:Class rdf:ID="Wine">
  ...
  <rdfs:subClassOf>
```

```
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasMaker" />
        <owl:allValuesFrom rdf:resource="#Winery" />
      </owl:Restriction>
    </rdfs:subClassOf>
    ...
  </owl:Class>
```

2. someValuesFrom - é similar à `allValuesFrom`. Com base no exemplo acima, se substituirmos owl:allValuesFrom por `owl:someValuesFrom`, significa que pelo menos uma das propriedades `hasMaker` da classe `Wine` deve estar relacionada a um indivíduo da classe `Winery`. A diferença entre as duas formulações é a diferença entre a quantificação universal e existencial. Utilizando `allValuesFrom`, significa que se houver um relacionamento `hasMaker`, ele é do tipo `Winery`. Já someValuesFrom define que existe pelo menos um relacionamento *hasMaker*, que é do tipo `Winery`.

3. Cardinalidade - a restrição de cardinalidade permite a especificação do número exato de elementos na relação. A OWL Lite só permite expressões de cardinalidade com valores limitados entre 0 e 1. Em OWL DL são permitidos outros valores inteiros positivos. A cardinalidade é definida com a tag `owl:cardinality`, mas também pode ser usada `owl:maxCardinality` para especificar um valor máximo, e `owl:minCardinality` para especificar um valor mínimo. Uma combinação pode ser utilizada para definir um intervalo numérico.

4. hasValue - `hasValue` permite-nos especificar classes baseadas na existência de valores particulares da propriedade. Desta forma, um indivíduo será membro de tal classe sempre que pelo menos um dos seus valores de propriedade for igual ao valor determinado em `hasValue`.

Quando se trabalha com um conjunto de componentes de ontologias distintas como parte de uma terceira ontologia, pode ser útil indicar que uma classe ou propriedade em uma ontologia é equivalente a uma classe ou propriedade em uma segunda ontologia. Analisemos o código abaixo:

```
<owl:Class rdf:ID="Wine">
   <owl:equivalentClass rdf:resource="&vin;Wine"/>
</owl:Class>
```

Neste exemplo, a classe `Wine` em uma dada ontologia é equivalente à classe `Wine` na ontologia indicada pelo *namespace* `vin`. Neste caso, `owl:equivalentClass` é usada para indicar que as duas classes possuem as mesmas instâncias. `owl:equivalentClass` também pode ser usada para proporcionar um poderosa capacidade de definição baseada na satisfação de uma propriedade.

De acordo com o exemplo abaixo, a classe `TexasThings` possui exatamente coisas que estão localizadas na região do Texas. A diferença entre usar `equivalentClass` ou `subClassOf`, é que com esta última tag, coisas localizadas no Texas não são necessariamente `TexasThings`. Mas usando `owl:equivalentClass`, se alguma coisa está localizada no Texas, então deve estar na classe `TexasThing`.

```
<owl:Class rdf:ID="TexasThings">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#locatedIn" />
      <owl:someValuesFrom rdf:resource="#TexasRegion" />
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

Apresentamos uma breve revisão da linguagem OWL como modelo ontológico formal com seus elementos básicos, características das propriedades de objetos, restrição de escopo e equivalência entre classes. Agora passaremos a descrever alguns construtores adicionais para manipulação de classes, caracterização de classes enumeradas e classes disjuntas.

Operadores básicos

A OWL fornece construtores adicionais com os quais podemos formar classes. Estes construtores podem ser usados para criar expressões de classe (*class expressions*). OWL suporta os operadores básicos, como a união, interseção e complemento. Eles são especificados como: `owl:unionOf`, `owl:intersectionOf`, e `owl:complementOf`.

Expressões de classe podem ser aninhadas sem precisar da criação de nomes para toda classe intermediária. Isto permite o uso de um conjunto de operações para construir classes complexas de classes anônimas (*anonymous classes*) ou classes com restrições de valor.

Extensões de classe são conjuntos formados por indivíduos que são membros da classe. A OWL fornece meios para manipular extensões de classe usando o conjunto de operadores básicos, que descrevemos a seguir:

1. Interseção - para realizar uma interseção é utilizado o construtor `intersectionOf`. Exemplificamos abaixo a sua utilização:

   ```
   <owl:Class rdf:ID="VinhoBranco">
     <owl:intersectionOf rdf:parseType="Collection">
       <owl:Class rdf:about="#Vinho" />
       <owl:Restriction>
   ```

```
        <owl:onProperty rdf:resource="#temCor" />
        <owl:hasValue rdf:resource="#Branco" />
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>
```

Em classes construídas usando um conjunto de operações, os membros da classe são completamente especificados por este conjunto. O exemplo acima define que a classe `VinhoBranco` é uma interseção da classe `Vinho` e o conjunto de coisas que possuem `Branco` como cor (`temCor`). Este é um mecanismo importante para categorizar indivíduos.

2. União - O construtor `unionOf` é usado da mesma forma que o construtor `intersectionOf`, mas por sua vez especifica a união de classes.

3. Complemento - o construtor `complementOf` seleciona todos os indivíduos do domínio de discurso que não pertencem a uma certa classe.

```
<owl:Class rdf:ID="CoisaConsumível" />
<owl:Class rdf:ID="CoisaNãoConsumível">
    <owl:complementOf rdf:resource="#CoisaConsumível" />
</owl:Class>
```

A classe `CoisaNãoConsumível` inclui como seus membros todos os indivíduos que não pertencem à `CoisaConsumível`. É a diferença entre `owl:Thing` e `CoisaConsumível`.

Classes enumeradas

Com OWL é possível especificar uma classe através de uma enumeração direta dos seus membros. Isto é feito usando o construtor `oneOf`. Esta definição especifica uma extensão de classe, de forma que outros indivíduos não podem ser declarados como pertencentes à classe. O exemplo abaixo define a classe `WineColor` cujos membros são indivíduos `White`, `Rose` e `Red`.

```
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor"/>
  <owl:oneOf rdf:parseType="Collection">
    <WineColor rdf:about="#White"/>
    <WineColor rdf:about="#Rose"/>
    <WineColor rdf:about="#Red"/>
  </owl:oneOf>
</owl:Class>
```

Classes disjuntas

A disjunção de um conjunto de classes pode ser expressa usando o construtor `owl:disjointWith`. Isto assegura que um indivíduo que é membro de uma classe não pode simultaneamente ser uma instância de uma outra classe. Classes OWL partem da suposição de que se sobrepõem; portanto, não podemos presumir que um indivíduo não é membro de uma determinada classe simplesmente porque ele não foi definido como sendo membro da classe (HORRIDGE, 2004).

```
<owl:Class rdf:ID="Pasta">
  <rdfs:subClassOf rdf:resource="#EdibleThing"/>
  <owl:disjointWith rdf:resource="#Meat"/>
  <owl:disjointWith rdf:resource="#Fowl"/>
  <owl:disjointWith rdf:resource="#Seafood"/>
  <owl:disjointWith rdf:resource="#Dessert"/>
  <owl:disjointWith rdf:resource="#Fruit"/>
</owl:Class>
```

O código acima especifica que a classe `Pasta`, subclasse de `EdibleThing`, é disjunta de todas as demais classes, porém não define que as outras classes são disjuntas entre si. Para definir que um conjunto de classes é mutuamente disjuntas, deve haver uma definição `owl:disjointWith` para cada par.

Do exposto, vemos que a OWL é uma linguagem para definição de ontologias, utilizada para descrever classes e as relações entre elas. Além dos elementos básicos, OWL possibilita o uso de propriedades com características específicas que, aliadas às várias formas de restrições e aos operadores básicos, proporcionam mecanismos poderosos de raciocínio. Desta forma, a semântica formal da OWL especifica como derivar suas conseqüências lógicas, ou seja, os fatos que não são literalmente apresentados na ontologia, mas são derivados da semântica.

2.4.3 OWL DL

A maior parte das primitivas oferecidas pela OWL podem ser encontradas na DL. A decisão de basear a OWL sobre DL foi motivada pela exigência de que problemas de inferência sejam decidíveis, e que deve ser possível prover serviços de raciocínio para auxiliar o projeto e desenvolvimento de ontologias.

Segundo (BROCKMANS et al., 2004), OWL DL corresponde à Lógica de Descrição $\mathcal{SHOIN}$(D) e todas as características da linguagem podem ser reduzidas às primitivas da lógica $\mathcal{SHOIN}$(D). Isto significa que uma ontologia OWL DL é expressa em um formalismo com semântica bem definida, sobre o qual pode ser realizado raciocínio automatizado.

Em OWL DL é possível expressar, por exemplo, que cada instância de uma classe A está relacionada a pelo menos uma instância da classe B, através de uma propriedade P. OWL DL permite restrições existencial, universal e de cardinalidade; além disto, é possível também combinar estas restrições com construtores

booleanos, tais como AND, OR e NOT (STEVENS et al., IN PRESS). Uma restrição descreve uma classe anônima (*anonymous class*), ou seja, uma classe sem nome. A classe anônima contém todos os indivíduos que satisfazem a restrição. Quando restrições são usadas para descrever classes, na verdade especificam superclasses anônimas da classe que está sendo descrita (HORRIDGE, 2004).

Para (HORRIDGE, 2004), usando restrições a OWL DL fornece duas formas diferentes de descrever uma classe: expressando condições necessárias ou condições necessárias e suficientes para um indivíduo ser uma instância da classe. Condições necessárias podem ser lidas como: "Se alguma coisa é um membro desta classe então é necessário satisfazer estas condições". Contudo, somente com as condições necessárias não podemos dizer que se alguma coisa satisfaz estas condições então deve ser um membro desta classe. Uma classe que possui somente condições necessárias é conhecida como classe primitiva e também como classe parcial. Condições necessárias e suficientes são condições não somente necessárias para os membros da classe, mas também são suficientes para determinar que qualquer indivíduo que as satisfaz deve ser um membro da classe. Uma classe que possui pelo menos um conjunto de condições necessárias e suficientes é conhecida como classe definida e também como classe completa.

Alguns sistemas, tais como bancos de dados, utilizam a hipótese de mundo fechado (*closed world*). Em modelos do mundo fechado, se alguma coisa não é explicitamente declarada então assume-se que não existe ou que qualquer inferência sobre ela não é verdadeira. Ao contrário disto, seguindo a hipótese de mundo aberto (*Open World Assumption* ou *Open World Reasoning*), tal como em OWL DL, se alguma coisa não é explicitamente declarada, então ela pode ou não assumir o comportamento que se espera (STEVENS et al., IN PRESS). Em outras palavras, significa que nós não podemos assumir que alguma coisa não existe até que seja explicitamente declarado que ela não existe (HORRIDGE, 2004).

Como exemplo, suponhamos que criamos `MinhaPizza` como um indivíduo da classe `Pizza` e definimos que ele tem cobertura de queijo através da propriedade `temCobertura`. Suponhamos também que a classe `PizzaDeQueijo`, subclasse de Pizza, possui um axioma de condição necessária e suficiente que define que um indivíduo desta classe deve ter pelo menos uma cobertura de queijo. Desta forma, um raciocinador poderá inferir que o indivíduo `MinhaPizza` é uma instância também da classe `PizzaDeQueijo`, pois satisfaz as condições necessárias e suficientes para fazer parte desta classe, embora tenha sido definido como uma instância apenas de `Pizza`.

OWL DL pode ser submetida a um raciocinador de lógica de descrição, tal como RacerPro. O raciocinador realiza os seguintes serviços (STEVENS et al., IN PRESS):

- analiza cada classe para verificar a consistência com a ontologia. A consistência pode ser verificada pela definição de classes disjuntas. Se definimos duas classes A e B como sendo disjuntas e colocamos a classe C como subclasse de A e de B, o raciocinador determinará uma inconsistência.

- verifica a classificação da hierarquia, levando em conta todas as restrições dadas na ontologia. Isto pode revelar novas subordinações entre as classes bem como erros de modelagem, que precisam ser corrigidos. Além disso, em alguns raciocinadores (e.g. RacerPro), dada a descrição de um indivíduo, é possível computar as classes das quais esse indivíduo é uma instância. O raciocinador somente poderá classificar classes automaticamente, sob *classes definidas*.

2.4.4 Axiomas OWL

Como já citamos, os axiomas são sentenças sempre verdadeiras, utilizadas em raciocínio. De acordo com (VOLZ, 2004), descrevemos na tabela 2.2 os axiomas OWL. Além destes, alguns autores, tais como (HORRIDGE, 2004), consideram também *range* e *domain* como axiomas que podem ser utilizados em raciocínio.

Axioma	**Sintaxe DL**	**Exemplo**
subClassOf	$C_1 \vee C_2$	Human $\vee$ Animal u Biped
equivalentClass	$C_1 \acute{} C_2$	Man $\acute{}$ Human u Male
subPropertyOf	$P_1 \vee P_1$	hasDaugther $\vee$ hasChild
equivalentProperty	$P_1 \acute{} P_2$	cost $\acute{}$ price
sameAs	$\{x_1\} \acute{} \{x_2\}$	{PresidentBush}
disjointWith	$C_1 \vee : C_2$	Male $\vee$: Female
differentIndividualAs	$\{x_1\} \vee : \{x_2\}$	{marc} $\vee$: {hans}
inverseOf	$P_1 \acute{} P_2^-$	hasChild $\acute{}$ hasParent$^-$
TransitiveProperty	$P^+ \vee P$	ancestor$^+ \vee$ ancestor
FunctionalProperty	$> \vee 6\ 1\ P$	$> \vee 6\ 1$ hasMother
InverseFunctionalProperty	$> \vee 6\ 1\ P^-$	$> \vee 6\ 1$ isMotherOf$^-$
SymmetricProperty	$P \acute{} P^-$	connectedTo $\acute{}$ connectedTo$^-$

Tabela 2.2: Axiomas OWL

2.4.4.1 Axioma de fechamento

Um axioma de fechamento ou *Closure Axiom* sobre uma propriedade consiste de uma restrição universal que atua sobre a propriedade para explicitar que ela só pode ser preenchida por preenchedores (*fillers*), ou tipos de valores, especificados (HORRIDGE, 2004). A restrição tem um preenchedor que é a união dos preenchedores que aparecem nas restrições existenciais da propriedade. Vejamos um exemplo extraído de (HORRIDGE, 2004), onde definimos a classe Margueritа-Pizza, seguindo a sintaxe do Protégé 3.3.1:

$\forall$ `temCobertura(CoberturaMussarela` $\sqcup$ `CoberturaTomate)`
$\exists$ `temCobertura CoberturaTomate`
$\exists$ `temCobertura CoberturaMussarela`

O exemplo acima define que se um indivíduo é membro da classe MargueritaPizza então ele deve ter pelo menos uma cobertura que é do tipo CoberturaMussarela e deve ter pelo menos uma cobertura que é membro da classe CoberturaTomate e a cobertura deve ter somente tipos de CoberturaMussarela e CoberturaTomate.

2.4.4.2 Restrições de Cardinalidade

Em OWL podemos descrever uma classe de indivíduos com cardinalidade mínima, máxima ou com um número específico de relacionamentos com outros indivíduos ou tipos de dados (HORRIDGE, 2004). Estas restrições são chamadas restrições de cardinalidade. A restrição de cardinalidade mínima especifica o número mínimo de relacionamentos em que um indivíduo pode participar. A restrição de cardinalidade máxima é análoga à mínima.

2.4.4.3 Padrões de Projeto de Ontologia

Padrões de Projeto de Ontologia ou *Ontology Design Patterns* (ODPs) são baseados nos mesmos princípios que os Padrões de Projeto na Programação Orientada a Objeto. ODPs são abstrações de soluções aplicadas à problemas comuns na modelagem de um domínio de conhecimento (STEVENS et al., IN PRESS). Padrões de modelagem capturam as melhores práticas no desenvolvimento da ontologia, tais como partições de valor.

Partições de Valor

Partições de Valor ou *Value Partitions* são criadas para refinar as descrições de classes. Partições de valor restringem o escopo (*range*) de valores possíveis a uma lista exaustiva. De acordo com (HORRIDGE, 2004), criar uma *value partition* em OWL consiste de vários passos, a saber:

1. Criar uma classe para representar a *value partition*. Por exemplo, para representar o tipo de tempero referente à cobertura de pizzas, podemos criar a classe `TemperoValuePartition`;
2. Criar subclasses da *value partition* para representar as possíveis opções. Neste caso, podemos criar as classes `Leve`, `Moderado` e `Forte` como subclasses de `TemperoValuePartition`;
3. Tornar as subclasses de *value partition* classes disjuntas;
4. Definir um axioma de cobertura (*covering axiom*) para tornar a lista de tipos de valores exaustiva;
5. Criar uma propriedade de objeto para a *value partition*;

6. Definir a propriedade como funcional, por exemplo, `temTempero`;

7. Definir o escopo da propriedade como sendo a classe *value partition*; e.g., a propriedade `temTempero` terá como escopo a classe `TemperoValuePartition`.

Axioma de cobertura - O Axioma de Cobertura ou *Covering Axiom* faz parte do padrão *Value Partition*. Um axioma de cobertura consiste de duas partes: a classe que está sendo coberta e as classes que forma a cobertura (HORRIDGE, 2004). Vejamos um exemplo onde temos três classes: A, B e C, com B e C sendo subclasses de A. Definir um axioma de cobertura para A, significa dizer que um membro da classe A deve ser um membro da classe B ou C (uma ou outra classe, se as classes forem disjuntas).

No Protégé-OWL um axioma de cobertura é definido como uma classe que é a união das classes que formam a cobertura, união esta que forma uma superclasse da classe que está sendo coberta (HORRIDGE, 2004). No caso do exemplo, a classe A deve ter uma superclasse B $\sqcup$ C. Sem o axioma de cobertura um indivíduo pode ser membro de A e não ser membro de B ou C.

Utilizamos a linguagem OWL DL neste trabalho por disponibilizar uma representação formal, que ao mesmo tempo em que pode ser processada por máquina, também permite o entendimento humano.

2.4.5 Karlsruhe

Existem modelos que definem formalmente, através de uma abordagem unificadora, as várias caracterizações de ontologias encontradas na literatura. O modelo de Karlsruhe pertence a este grupo e será apresentado no apêndice A.

O Modelo Ontológico de Karlsruhe (*Karlsruhe Ontology Model*) é um modelo formal para ontologias, definido a partir de discussões no seminário do Grupo de Gerenciamento do Conhecimento do Instituto AIFB da Universidade de Karlsruhe (http://www.aifb.uni-karlsruhe.de). Como a maior parte das definições de ontologia existente na literatura possui um alto nível de generalização, o grupo sentiu a necessidade de criar um *framework* de representação de ontologia específico, com definição precisa e detalhada, para estudar os aspectos estruturais (EHRIG et al., 2003). O objetivo é alcançar um entendimento comum dentro do grupo sobre noções essenciais, como conceito e relações.

Neste trabalho não utilizamos o modelo de Karlsruhe; entendemos que não era necessário visto que a OWL DL permite a axiomatização completa da ontologia. Além do mais, o foco do nosso trabalho não é a formalização de ontologias.

2.5 Modelos Integrados

Podemos observar que a maior parte das tecnologias existentes atualmente na área da computação trata a conexão e a disponibilização de informações de diversas formas, mas ainda são reduzidos os esforços dirigidos para a conexão de sistemas de informação heterogêneos e distribuídos. O desafio atual é fazer com que um sistema seja transparente ao outro, de forma que a informação seja independente de vinculação a sistemas. Os dados e informações hoje residentes dentro de sistemas específicos devem estar acessíveis para processamento em qualquer máquina interconectada, facilitando a utilização e o tratamento do conhecimento. A informação na Web está crescentemente sendo fragmentada e variada. Vários governos têm se preocupado em tornar informações disponíveis para o público online e o estado atual da Web tem colocado fatores limitantes na acessibilidade e aplicabilidade da informação (NIEMANN et al., 2005).

Em 2004, duas novas estruturas lógicas e de dados foram aprovadas pelo W3C com o objetivo de tornar as informações mais autônomas, acessíveis e adaptáveis: a linguagem RDF e OWL, que tornam extensível o uso dos princípios de representação do conhecimento para adicionar funcionalidade e compatibilidade às linguagens de marcação existentes (MILLER et al., 2005). Estes padrões - juntamente com as novas ferramentas e componentes de estrutura construídos para apoiá-los, tal como o Protégé, estão dirigindo o desenvolvimento de uma computação adaptativa, como também o crescimento da próxima geração da Web, chamada de Web Semântica (NIEMANN et al., 2005).

A proposta da Web Semântica é estender a plataforma da Web atual permitindo enriquecimento da informação transmitida e acessada através da Internet, com significados semânticos definidos. Desta forma, o processamento do conhecimento é alcançado e não meramente o tratamento de informações desassociadas de um contexto. Os trabalhos desenvolvidos no campo de Engenharia de Software e Sistemas têm-se mesclado com a área de representação formal da informação na Web através do uso da proposta da Web Semântica. O desenvolvimento de sistemas tem combinado tecnologias da Web Semântica e técnicas com formalismos já estabelecidos, como a UML (TETLOW et al., 2005).

2.5.1 MDA

A área de pesquisa de Engenharia de Software e Sistemas ainda apresenta muitos desafios, apesar dos avanços alcançados na representação e composição de sistemas. Por isto, a modelagem do domínio de conhecimento através de notações formais ou semi-formais configura-se como uma prática essencial. Os modelos propostos não são usados somente para projetos específicos mas, associados a ferramentas e técnicas, podem ser utilizados para gerar artefatos executáveis para uso posterior no ciclo de vida do software (TETLOW et al., 2005). A Arquitetura Dirigida a Modelos ou *Model-Driven Architecture* (MDA) é uma iniciativa do O*bject Management Group*. A abordagem utilizada pelo grupo permite especi-

ficar sistemas de forma independente da plataforma em que serão desenvolvidos, possibilitando também a geração de artefatos de software e especificações para um sistema específico, em uma plataforma de desenvolvimento particular (OMG, 2005).

A abordagem de orientação a objetos através do uso da linguagem UML forma a base da proposta do MDA. Um modelo UML pode ser expresso através de uma plataforma independente ou específica, de acordo com a finalidade de uso, sendo que ambos são utilizados como processo de desenvolvimento da MDA. Todo padrão MDA ou aplicação é baseado em um Modelo Independente de Plataforma ou *Platform-Independent Model* (PIM), o qual representa a funcionalidade do negócio e o comportamento de forma precisa, sem incluir aspectos técnicos de transações. A partir do PIM, as ferramentas de desenvolvimento que seguem o padrão MDA do OMG fazem um mapeamento para produzir um ou mais Modelos Específicos de Plataforma ou *Platform-Specific Models* (PSM), também em UML, um para cada plataforma alvo que o desenvolvedor optar (OMG, 2005). O PSM contém a mesma informação do PIM na forma de um modelo UML ao invés de código executável. No próximo passo, a ferramenta gerará o código executável do PSM, junto com outros arquivos executáveis.

Em suma, o MDA é considerado um *framework* conceitual e provê uma metodologia com os requisitos básicos para ferramentas que prescrevem os tipos de modelos que podem ser utilizados em cada fase do projeto. O objetivo é descrever os requisitos básicos que devem ser desenvolvidos para maximizar o reuso, portabilidade e interoperabilidade, bem como explicitar o relacionamento entre eles para oferecer suporte à geração de código.

De acordo com (BROWN, 2004), o MDA possui quatro princípios básicos:

1. Modelos expressos em uma notação bem definida são a base para o entendimento de sistemas.

2. A construção de sistemas pode ser organizada ao redor de um conjunto de modelos através de uma série de transformações entre modelos, organizadas dentro de um *framework* arquitetural de camadas e transformações.

3. Uma descrição formal para descrever modelos através do uso de um conjunto de metamodelos, o qual facilita a integração e transformação entre modelos, sendo a base para a automação através de ferramentas.

4. A aceitação e ampla adoção do enfoque baseado em modelo necessita de padrões da indústria para prover abertura aos consumidores e promover competição entre os vendedores.

A figura 2.2 ilustra a arquitetura MDA em quatro camadas, conforme citado por (DJURIC; GASEVIC; DEVEDZIC, 2003). No topo desta arquitetura está o *Meta Object Facility* (MOF), na camada de Meta-metamodelo (M3). O MOF define uma linguagem abstrata e um *framework* para especificar, construir e gerenciar tecnologia independente de metamodelos. Todos os metamodelos e

padrões, definidos pelo MOF estão localizados na camada de Metamodelo (M2), que engloba a UML. A camada de Modelo (M1) contém os modelos do mundo real, representados por conceitos definidos em um metamodelo correspondente na camada M2 (e.g. metamodelo UML). Finalmente, na camada de Instância (M0) estão objetos do mundo real. Um exemplo pode ser: uma classe MOF (na camada M3) é usada para definir uma classe UML (na camada M2), que é usada para definir o modelo: Pessoa (classe UML) e Tom, Dick e Harry (objetos UML, na camada M1), que representam a realidade (camada M0).

Conforme ilustrado na figura 2.2, podemos observar que o objetivo principal de apresentar metamodelos nas camadas M1 e M2 é proporcionar extensibilidade, integração e criação de modelos genéricos, além de metamodelos gerenciais do cliente. Desta forma, no nível M1 de modelos o cliente pode definir um metamodelo gerencial. O alinhamento do metamodelo da UML com o metamodelo do MOF simplificará o intercâmbio de modelos através do XMI, que será abordado ainda nesta seção, proporcionando interoperabilidade entre ferramentas (BPTRENDS, 2003).

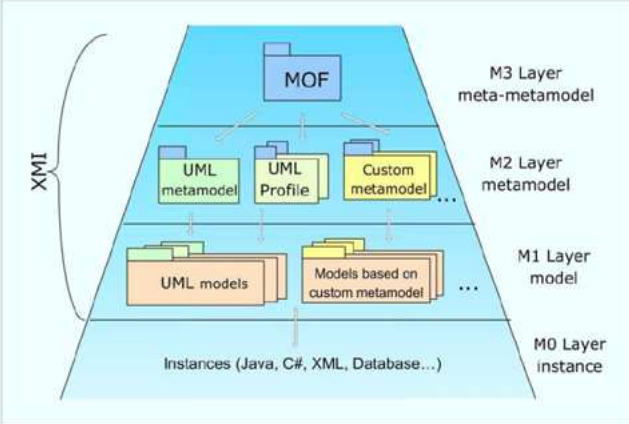

Figura 2.2: Arquitetura MDA em quatro camadas (DJURIC; GASEVIC; DEVEDZIC, 2003)

Os padrões propostos pelo MDA do OMG possibilitam o gerenciamento de metadados e integração. São eles: *Common Warehouse Metamodel* (CWM), UML, MOF e XMI (POOLE, 2001). A seguir, abordamos as principais características de cada um, exceto da UML, já abordada anteriormente.

- CWM

O CWM define um metamodelo para representação de metadados técnicos e de negócio que são freqüentemente encontrados em *data warehousing* e domínio de análise de negócio. Este padrão representa um enfoque baseado em modelo para troca de metadados entre sistemas de software. Metadados compartilhados entre produtos são formulados de acordo com os modelos de dados que são consistentes com um ou mais metamodelos CWM (POOLE, 2001). Modelos CWM pretendem

ser altamente genéricos. São considerados representações externas de metadados compartilhados.

- MOF

Conforme citado anteriormente, o *Meta Object Facility* define uma linguagem comum e abstrata para a especificação de metamodelos. Ele próprio é um exemplo de um modelo de metamodelo e por vezes chega a ser chamado de ontologia (POOLE, 2001). Em outras palavras, é um modelo orientado a objeto que define elementos essenciais, sintaxe e estrutura de metamodelos que são usados para construir modelos orientados a objeto de sistemas. Portanto, permite interoperabilidade entre metamodelos baseados no MOF.

- XMI

O XMI é um padrão que mapeia o MOF para XML. Dessa forma, são definidas quais a tags XML são usadas para representar os modelos MOF em XML. De acordo com (POOLE, 2001), o processo de transformação é feito da seguinte forma: os metamodelos baseados no MOF são transformados para *Document Type Definitions* (DTDs) e os modelos são traduzidos dentro de documentos XML que são consistentes com seus respectivos DTDs.

2.5.2 ODA

Enquanto MDA proporciona um poderoso *framework* para a Engenharia de Software e Sistemas, as tecnologias de Web Semântica provêem uma extensão natural deste framework. Modelos semânticos, comumente chamados de ontologias, atribuem o paradigma semântico ao modelo, o que originou a denominação de *Ontology Driven Architecture* (ODA).

Nos últimos cinco anos, houve várias tentativas para reunir linguagens e ferramentas, tais como a UML, desenvolvidas para Engenharia de Software, com linguagens da Web Semântica, como RDF e OWL. Um fator marcante foi a iniciativa do OMG, que começou a desenvolver o *Ontology Definition Metamodel* (seção 2.5.3). Este trabalho inclui a utilização da linguagem UML para a criação de ontologias, tal como utilizadas na Web Semântica, conforme citado em (TETLOW et al., 2005).

É comum as aplicações serem desenvolvidas e gerenciadas com a ajuda de ferramentas de administração que utilizam arquivos de configuração em XML. A desvantagem nisso é que o modelo conceitual sob as diferentes configurações é implícito. Embora o MDA permita separar o enfoque conceitual do enfoque específico da implementação, atualmente não tem sido aplicado para características relevantes de gerenciamento de componentes em tempo de execução, como por exemplo, qual versão de uma interface de aplicação requer qual versão de bibliotecas.

A proposta da ODA surge como um complemento ao MDA. Na ODA, uma ontologia captura propriedades de relacionamentos e comportamentos de componentes que são necessários para o propósito de desenvolvimento e administração. Como uma ontologia é um modelo conceitual explícito, com semânticas baseadas em lógica formal, suas descrições de componentes podem ser consultadas, componentes que são necessários indiretamente podem ser pré-carregados ou podem ser verificados para evitar inconsistência nas configurações do sistema, tanto durante o desenvolvimento quanto em tempo de execução (TETLOW et al., 2005). Sendo assim, a ODA mantém a flexibilidade original na configuração e execução do servidor de aplicação, mas adiciona novas capacidades para o desenvolvedor e usuário do sistema.

2.5.3 ODM

O *Ontology Definition Metamodel* (ODM) é um metamodelo que está sendo definido por uma *Request for Proposal* (RFP) dentro do OMG (OMG, 2003). Passaremos a descrever as propostas contidas neste documento.

As linguagens de ontologia, no contexto da RFP, são aquelas consideradas fragmentos da lógica de predicado. Esta estrutura tem conduzido a uma sintaxe de linguagem que não é familiar aos especialistas do domínio e, freqüentemente, para as pessoas familiarizadas com linguagens de modelagem de informação. A RFP leva em consideração tal situação. A familiaridade dos usuários com a UML, a disponibilidade de ferramentas UML, a existência de vários modelos de domínio em UML e a similaridade destes modelos com ontologias sugerem que a UML pode ser um meio mais rápido de desenvolvimento de ontologias. Desta forma, a RFP solicitou especificações normativas conforme figura 2.3, de (OMG, 2003):

- um metamodelo, de acordo com o padrão MOF2, para definição de ontologia (ODM),
- um *profile* UML2 para permitir o reuso da notação UML para definição de ontologia,
- um mapeamento do ODM para o *profile*,
- uma linguagem de mapeamento do ODM para a OWL DL.

De acordo com a especificação, o objetivo do ODM é definir uma meta linguagem padrão para modelagem da Web Semântica, possibilitando o emprego da Engenharia de Software baseada no MDA em conjunto com as tecnologias da Web Semântica, que oferecem interoperabilidade e aplicação da semântica de negócios. Além disso, o ODM define também as especificações para mapeamentos entre os artefatos da UML e sua transformação correspondente em recursos empregados na Web Semântica. Portanto o ODM deve ser projetado para compreensão comum dos conceitos de ontologia. Um bom ponto de partida é a OWL, já que

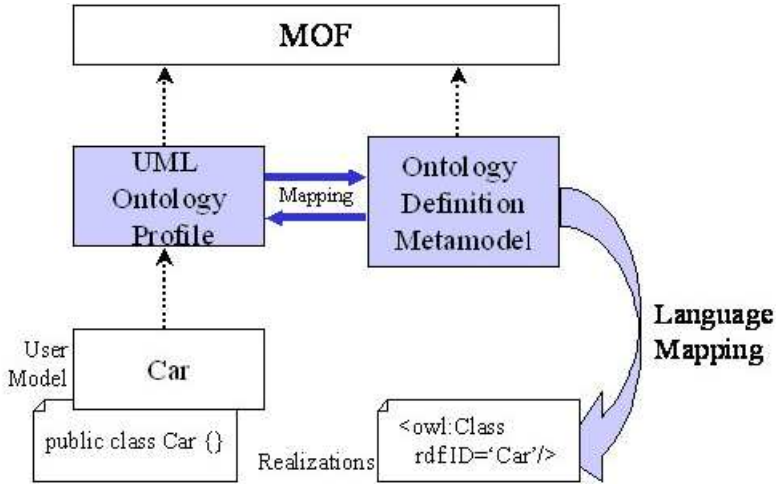

Figura 2.3: Arquitetura da proposta do ODM (OMG, 2003)

é o resultado da evolução das linguagens de representação de ontologias e uma recomendação do W3C.

Em resposta à requisição do OMG foram submetidas quatro propostas: (IBM; SOFTWARE, 2003), (DSTC, 2003), (GENTLEWARE, 2003), (SOFTWARE; LABORATORY, 2003). Posteriormente, as quatro propostas foram unidas em uma única proposta, que tem passado por diversas revisões. A proposta de (IBM; SOFTWARE, 2003) será relatada na seção 3.2.

Apesar do diagrama de classes ser um sistema de representação muito rico, uma ontologia não pode ser suficientemente representada em UML, sendo necessária uma linguagem de modelagem visual de ontologia. Segundo (BROCKMANS et al., 2006), as duas representações compartilham um conjunto de funcionalidades mas apesar disto, há muitas características que só são expressas em OWL e outras, só em UML. Por exemplo, propriedades transitivas em OWL e métodos em UML. Para utilizar as capacidades de modelagem gráficas da UML o ODM deve ter um *profile* UML correspondente, tratando então de um mecanismo de extensão da UML para aplicação em áreas específicas, que proporciona a especialização de construtores UML existentes, através de estereótipos, ou seja, novos tipos de elementos de modelagem podem ser introduzidos.

Segundo (DJURIC; GASEVIC; DEVEDZIC, 2005), tendo em vista que a OWL é construída sobre o RDF e RDF *Schema*, e como o RDF *Schema* e o MOF possuem várias diferenças, os conceitos OWL não podem ser diretamente copiados para conceitos ODM: faz-se necessário uma adaptação.

Conforme apresentado nesta seção os Modelos Integrados estão evoluindo para um metamodelo que integre Engenharia Ontológica à Engenharia de Software através do uso de linguagens de modelagem e modelos de domínio adequados. Nesta direção, este trabalho apresenta uma proposta de integração destes Espaços Tecnológicos que se mostra adequada, conforme apresentada por (DJURIC; GASEVIC; DEVEDZIC, 2005).

Capítulo 3

Proposta de trabalho

Apesar da Engenharia de Software utilizar técnicas derivadas de vários anos de pesquisa, as quais já se encontram consolidadas no mercado, quando se trata da interoperabilidade de sistemas faz-se necessário o uso também de uma tecnologia que está sendo reconhecida como um grande auxílio na construção de sistemas de informação interoperáveis: as ontologias. À medida em que as agregações de software ocorrem é necessário interoperar com semânticas mais precisas, o que geralmente ocorre em ambientes muito heterogêneos e com informações distribuídas. O compartilhamento de informações através da Web reforça a necessidade de comunicação não ambígua de conceitos detalhados, para que sistemas de um mesmo domínio de conhecimento ou não possam se comunicar. O uso de ontologia é uma solução que proporciona interoperabilidade e conseqüente integração entre os sistemas computacionais.

3.1 Descrição e objetivo

Para definição deste trabalho foram estudados modelos e técnicas que auxiliam na integração de sistemas, através da Engenharia Ontológica. Utilizamos uma das diversas abordagens propostas por pesquisadores e divulgadas em artigos, aplicando-a em um estudo de caso e identificando as vantagens e problemas. Fazemos uso do conceito de Espaços Tecnológicos o qual permite averiguar como trabalhar de forma eficiente utilizando-se as melhores possibilidades de diferentes tecnologias.

Esta proposta tem por objetivo a utilização dos Espaços Tecnológicos da Engenharia Ontológica e Engenharia de Software para facilitar a integração de sistemas computacionais. Estes espaços incluem tecnologias que estão sendo desenvolvidas paralelamente por duas comunidades diferentes, mas que, por possuírem pontos comuns, podem ser tratadas em conjunto, e.g. as linguagens do padrão MDA do OMG, com as linguagens ontológicas, como a OWL do W3C.

O trabalho inclui o estudo de identificação dos pontos comuns dos Espaços Tecnológicos em questão, como por exemplo a arquitetura MDA em quatro ca-

madas (figura 2.2), e a arquitetura de linguagens ontológicas como OWL, em três camadas (figura 2.1) conforme citada em (DECKER et al., 2000).

Modelos de domínio encerram em si informações suficientes para ser a base de uma ontologia, sendo um artefato com papel central durante todo o desenvolvimento de software. Portanto, o foco do estudo é a transformação de linguagens MDA para OWL, partindo de um modelo de domínio, de forma que a ontologia resultante possa ser utilizada para implementação prática em sistemas computacionais, não só no âmbito acadêmico mas também fora dele.

Para concretização deste trabalho fizemos uso das facilidades disponibilizadas pela ferramenta CASE Poseidon for UML, que em conjunto com a MDA, apóia a transformação dos conceitos necessários para a criação de ontologias. Utilizamos a abordagem de (GASEVIC; DJURIC; DEVEDZIC, 2005), baseada em XSLT para transformação de *profiles* UML em ontologias OWL. A figura 3.1 ilustra o *workflow* da proposta de trabalho.

Partimos de um diagrama de classes UML de um sistema com o qual pretendemos integrar um novo sistema. Este diagrama é adaptado de forma a englobar as definições do *Ontology UML Profile* (OUP) necessárias à transformação em ontologia, ou seja, aplicamos o OUP utilizando o Poseidon for UML. O OUP é exportado para um arquivo XMI, que é utilizado como entrada na transformação XSLT executada pelo processador Xalan. Nesta transformação além do arquivo de entrada são definidos também o arquivo xslt, que contém as regras de transformação, e o nome do arquivo OWL de saída. A ontologia OWL gerada é então editada no Protégé, onde os axiomas podem ser ajustados e definições necessárias podem passar a ser necessárias e suficientes. A partir do Protégé, e após ter inicializado o raciocinador RacerPro, são utilizados os serviços de inferências, tais como verificação da consistência da ontologia e da classificação.

Segundo (BRUGGEMANN; PORTO, 2006), existem diversas ferramentas para geração de classes Java a partir de uma ontologia, tais como Jastor, Jsave, Kazuki, Jena e beangenerator, entre outras. A maioria dessas ferramentas gera classes Java a partir de uma ontologia mas o objetivo é o desenvolvimento de agentes inteligentes ou a manipulação direta de ontologias através de código Java. Desta forma, a mais indicada no contexto do nosso trabalho é o beangenerator, que apesar de também ser voltado para o desenvolvimento de agentes inteligentes, gera classes Java para as entidades na ontologia, com os atributos definidos e cria os métodos de acesso (métodos *get* e *set*).

Beangenerator é um *plugin* do Protégé, que gera arquivos Java representando uma ontologia definida no Protégé. Apesar de sua fácil utilização, o *plugin* apresenta problemas na instalação pelo fato do projeto ter sido descontinuado. Em conseqüência, as versões mais novas do Protégé não são compatíveis com o beangenerator. Bruggemann e Porto (2006) explicam que após uma pesquisa extensa e inúmeros testes foram encontradas duas versões compatíveis: a versão 1.8 do Protégé e o beangenerator for Jade 2.5.

A contribuição desta proposta está em verificar a possibilidade de integrar sistemas em produção com novos sistemas, através da abordagem ontológica, re-

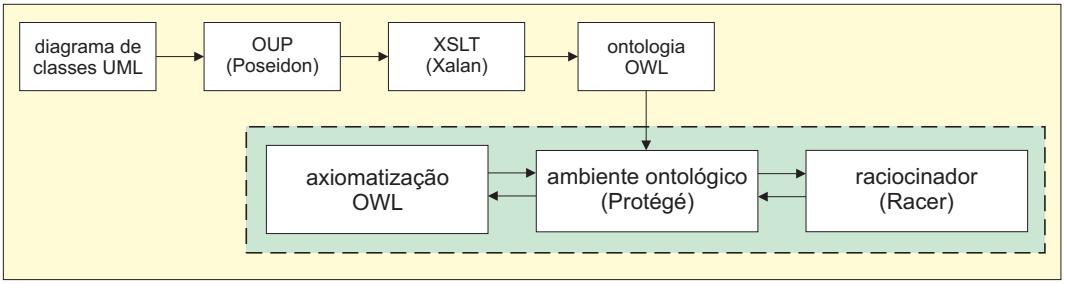

Figura 3.1: *Workflow* da proposta

aproveitando artefatos produzidos com tecnologia amplamente conhecida, como é o caso da UML, e utilizando padrões disponibilizados na maior parte das ferramentas de modelagem atuais, como o XMI. A abordagem ontológica é facilitada através da manipulação da ontologia com ferramentas disponibilizadas livremente, tal como o Protégé.

O estudo de caso descrito no capítulo 4, aplica a abordagem de Espaços Tecnológicos para transformação de *profiles* UML, que utilizam padrão MDA, em ontologias OWL. Utilizamos uma solução eficiente para unir as vantagens das ferramentas UML com a utilização de ontologias, através do mapeamento de linguagens MDA para OWL. A abordagem em uso não é normativa, já que não existe nenhuma proposta normativa tendo em vista a RFP do ODM.

Um ponto importante da nossa proposta está na produção de uma ontologia que incorpora os principais axiomas da linguagem OWL, quer sejam definidos no *profile* ou no ambiente ontológico, possibilitando melhor definição do domínio abordado e raciocínio sobre a ontologia.

3.2 Trabalhos Correlatos

Recentemente, vários esforços têm sido empenhados no sentido de unir as vantagens da modelagem visual da UML ao desenvolvimento de ontologias, bem como na utilização de ontologias para integração de sistemas. Descrevemos a seguir alguns trabalhos relacionados.

Cranefield (2001) foi um dos primeiros pesquisadores que propôs o uso da UML como linguagem de representação gráfica de ontologias. De acordo com sua proposta um especialista do domínio projeta uma ontologia graficamente, que é exportada pela ferramenta UML no formato XMI. O arquivo XMI passa por uma transformação XSLT, que pode resultar tanto em uma ontologia na linguagem RDFS, como em classes Java.

O trabalho de (CANTELE et al., 2004) propõe a utilização da engenharia reversa para a obtenção de uma ontologia. Como resultado da engenharia reversa sobre os sistemas legados obtém-se uma ontologia inicial, que é representada em UML estendida. A partir daí é feita uma engenharia progressiva, com repre-

sentação em RDF, aplicando-se uma metodologia para o desenvolvimento de uma ontologia completa. Não encontrarmos claramente definido o procedimento das transformações utilizadas na engenharia reversa.

Uma possibilidade citada por (GASEVIC; DJURIC; DEVEDZIC, 2005) é o uso dos *backends* do Protégé (UML e XMI). Os dois *backends* usam o *MetaData Repository* do NetBeans. O primeiro troca modelos UML (ou seja, classes e seus relacionamentos) usando o formato padrão XMI, enquanto o segundo usa o formato XMI, que é compatível com o metamodelo do Protégé definido com base no MOF. Com isto, é possível compartilhar ontologias através desta ferramenta (ou seja, importar uma ontologia no formato UML XMI e armazená-la no formato OWL). Contudo, o Protégé tem uma limitação em seu suporte a arquivos UML XMI: ele não mapeia as relações das classes (isto é, associações). Esta limitação já era esperada já que o Protégé importa modelos UML sem qualquer extensão, ou seja, um *profile* UML (GASEVIC; DJURIC; DEVEDZIC, 2005). Além disto, os *backends* só estão disponíveis para a versão 2.0 do Protégé, uma versão antiga.

Djuric, Gasevic e Devedzic (2003) além de apresentar um ODM com suporte para OWL DL, propõem, no contexto da RFP da OMG, a definição de outras especificações. São elas: *Ontology* UML *Profile* - melhor explicado no capítulo 4, e duas formas de mapeamento entre OWL e ODM, ODM e *Ontology* UML *Profile* e do *Ontology* UML *Profile* para outros *profiles*.

Neste ODM proposto, `rdfs:Resource`, um conceito raiz em RDF e OWL, é representado como uma instância de `Class` do MOF. Classificadores, que descrevem um conceito geral, e propriedades, que descrevem alguma característica genérica, são descendentes de `Resource`. O conceito OWL `owl:Class` é mais facilmente representado no ODM, já que o MOF também suporta agrupamento de instâncias de dados, embora ainda haja diferenças entre as classes dos dois modelos. No ODM, `rdf:Class` é representada como um `Classifier`, e suas subclasses `owl:Class` e `owl:Datatype`, como `AbstractClass` e `DataType`. Propriedade em OWL não é dependente de uma classe, como ocorre em UML; portanto, `Property` é definida como uma instância de `Class` do MOF que herda de `Resource`. Os conceitos OWL `rdf:Property`, `owl:ObjectProperty` e `owl:DatatypeProperty`, que herdam de `owl:Property`, são representados no ODM como `ObjectProperty` e `DatatypeProperty`. Propriedades OWL com características especiais, por exemplo propriedade simétrica, são modeladas como instâncias de `Class` do MOF.

Brockmans et al. (2004) propõem um ODM para OWL DL, cujo objetivo é alcançar uma notação intuitiva para os usuários da UML e OWL DL. O ODM proposto utiliza as características de modelagem centrais providas pelo MOF. Primeiramente são utilizados classes, atributos e associações. Adicionalmente, o metamodelo é aumentado com restrições feitas com *Object Constraint Language* (OCL), que especificam invariantes que devem ser cumpridas por todos os modelos que instanciam o ODM. Tais modelos são visualmente codificados usando o *profile* UML para ontologias definido pelos autores, que passaremos a descrever.

A proposta contém um conjunto de extensões e restrições para o metamodelo UML, utilizando estereótipos adaptados que geralmente levam o nome do

elemento de linguagem OWL correspondente. Neste *profile* um `Namespace` é representado por pacote, enquanto o estereótipo `<<owl::Ontology>>` indica uma ontologia. Classes são descritas de forma trivial, ou seja, como classes UML. A inclusão de classe é descrita usando o símbolo de generalização da UML. A equivalência de classe é expressa por duas inclusões de classe, ou seja, duas setas de generalização em sentidos opostos. Classes disjuntas são descritas como uma dependência estereotipada e bi-direcional. Classes enumeradas são conectadas aos indivíduos enumerados por dependências.

Em geral, uma restrição é descrita por uma classe com estereótipo correspondente. Se a propriedade que participa na restrição é uma propriedade de objeto (*object property*), então é descrita como uma associação para a classe participante. Se for uma propriedade de tipo de dado (*datatype property*), é descrita como um atributo. As cardinalidades são descritas utilizando a notação padrão da UML, ou seja, aparecem próximas à associação. A quantificação existencial pode, e a restrição de valor deve, ser indicada por um estereótipo dedicado, por exemplo, `<<owl::someValuesFrom>>`. Propriedades específicas, tais como propriedade simétrica e funcional, são descritas através de dependências e tipos de dados são representados através de classes estereotipadas.

Os autores desta proposta acreditam que a influência da UML para o desenvolvimento e manutenção de ontologias é uma enfoque muito promissor. É o primeiro passo para trazer a visão W3C da tecnologia da Web Semântica em conjunto com a visão OMG de uma Arquitetura Dirigida a Modelo.

Brockmans et al. (2006) apresentam um metamodelo para OWL DL e OWL Full, descrito com detalhes em (IBM; Sandpiper Software, 2006), e define um *profile* UML baseado no MOF para permitir o uso da notação UML e suas ferramentas para modelagem de ontologia. Segundo a proposta, dentro do *framework* do MOF os modelos UML são transformados em definições OWL e vice versa.

O metamodelo para OWL contém três pacotes e estende o metamodelo para RDFS, da mesma forma que a linguagem OWL estende a linguagem RDFS. O pacote OWLBase contém os construtores do metamodelo que são comuns tanto para OWL DL como para OWL Full. Dois subpacotes adicionais, o pacote OWLDL e o pacote OWLFull, consistem de restrições sobre o pacote OWLBase, bem como restrições necessárias para distinguir as duas sublinguagens. Todos os pacotes do metamodelo possuem restrições definidas através da OCL.

Passaremos a descrever algumas partes do pacote OWLBase. O metamodelo possui uma classe `OWLClass`, para definições de classe OWL simples, definida como um tipo especial de `RDFSClass`. Além deste, há outros tipos especiais de descrições de classes OWL: `ComplementClass`, `EnumeratedClass`, `UnionClass`, `OWLRestriction` e `IntersectionClass`. Uma classe enumerada é conectada aos indivíduos através de uma associação com papel `OWLoneOf`. As associações entre as classes definem as classes. Por exemplo, associações `EquivalentClass` e `DisjointClass` representam axiomas de classe OWL, ou seja, `EquivalentClass` conecta uma classe a outra que é definida como sendo equivalente. A classe `OWLRestriction` é definida como subclasse da classe `OWLClass` e possui como

subclasses todos os tipos de restrições de propriedade OWL. Uma restrição de classe deve ter exatamente uma propriedade `OWLonProperty` ligando a restrição a uma propriedade particular.

Como resultado da extensão do metamodelo RDFS, o metamodelo OWL refina a classe `RDFProperty` para permitir propriedades específicas da OWL. Como ambas as propriedades de objeto e de tipo de dados podem ser declaradas como funcional, os autores definiram a classe `FunctionalProperty` como subclasse de `Property`. Esta última é uma classe abstrata que simplifica a representação de equivalência de propriedades e restrições para OWL DL e OWL Full, e facilita o mapeamento para outros modelos. As características específicas de propriedade de objeto são definidas como subclasses da classe `OWLObjectProperty`, como por exemplo, a classe `SymmetricProperty`. Os axiomas de propriedades são providos através de associações `EquivalentProperty` e `InverseProperty`.

Indivíduos são representados pela classe `Individual`, subclasse de `RDFSResource`. O ODM apresenta também duas associações conectadas à classe `Individual`, `DifferentIndividual` e `SameIndividual`, para permitir que dois indivíduos sejam declarados diferentes ou iguais. O construtor `owl:AllDiferent` é representado pela classe `OWLAllDifferent`, subclasse de `OWLClass`. A propriedade `DistinctIndividuals` é definida para ligar uma instância de `OWLAllDifferent` a uma lista de `Individuals`.

O *profile* UML proposto foi projetado para proporcionar suporte aos modeladores no desenvolvimento de ontologias em OWL, através do reuso da notação UML utilizando ferramentas com suporte a mecanismos de extensão da UML 2. A notação padrão da UML 2 é utilizada quando os construtores possuem a mesma semântica intuitiva da OWL; quando isto não é possível, são utilizados construtores UML estereotipados que são consistentes e o mais próximo possível da semântica OWL. Passaremos a descrever o *profile* a seguir.

Primeiro, são representados os construtores para propriedades RDF já que, de acordo com o ODM que os autores propuseram, o pacote do *profile* OWL importa o pacote do *profile* RDF. Em RDF e OWL, propriedades são definidas globalmente. Para propriedades RDF que são definidas sem especificar um *domain* ou *range*, o *profile* utiliza a classe `Thing` com o estereótipo `<<rdfsClass>>`. A propriedade é representada como uma propriedade UML (ou atributo), cujo tipo é o seu *range*.

Da perspectiva da UML, propriedades são semanticamente equivalentes a associações binárias com navegação unidirecional. Desta forma, há uma representação alternativa para propriedades, e.g., pode haver uma navegação de uma instância da classe `Thing` para uma instância da classe Color através da propriedade `temCor` localizada no final da associação. Outra forma de representar propriedades é utilizando associações como classes. Uma classe de associação pode ter propriedades, associações e participar em generalização como qualquer outra classe. Neste caso, para distinguir a classe de associação da propriedade, a primeira letra do nome da classe de associação deve ser maiúscula. O estereótipo `rdfProperty` é introduzido para destacar a associação binária e unidirecional das

classes. A representação de subpropriedade (`rdfs:subPropertyOf`) depende de qual das três notações descritas acima é utilizada.

Para propriedades OWL específicas, são utilizados estereótipos como `objectProperty` ao invés de `rdfProperty`. Nestas propriedades, características adicionais, e.g. uma propriedade ser funcional ou simétrica, são representadas como propriedades UML. Para especificar uma relação `owl:equivalentProperty` ou `owl:inverseOf` entre duas propriedades utilizamos associações com navegação, onde a associação é nomeada com o tipo de relação desejada, e.g `<<inverseOf>>`.

Brockmans et al. (2006) apresentam também os mapeamentos possíveis para transformar modelos entre OWL e UML, baseados no metamodelo e *profile* que os autores descreveram. No desenvolvimento dos mapeamentos para as várias linguagens do ODM, os autores concluíram que os mapeamentos especificados não podem, na prática, ser normativos, conforme solicita a RFP do OMG.

Do exposto nesta seção, verificamos que vários esforços têm sido empenhados para concretização de trabalhos conjuntos entre as áreas da Engenharia de Software e Engenharia Ontológica. Dentre os trabalhos apresentados pudemos perceber algumas semelhanças com a nossa proposta. O trabalho de (CRANEFIELD, 2001) é o que mais se assemelha com a nossa abordagem por utilizar transformações XSLT a partir de uma ontologia definida através da UML e exportada no formato XMI. Cantele et al. (2004) apresentam uma abordagem que se assemelha em parte à nossa proposta de trabalho no sentido de efetuar transformações para obter uma ontologia inicial, que pode ser melhorada através de uma metodologia.

Capítulo 4

Estudo de caso

A RFP da OMG para definição do ODM propõe transformações entre ODM e OUP, bem como transformações entre ODM e OWL, conforme explicado na seção 2.5.3. Dessa forma, se quisermos transformar uma ontologia definida com um OUP em OWL, esta ontologia deve primeiro ser transformada em ODM e depois, de ODM para OWL (GASEVIC; DJURIC; DEVEDZIC, 2005). Enquanto o ODM não é finalizado, algumas propostas apresentam alternativas para implementar parte dos mapeamentos indicados pela RFP.

A figura 4.1 ilustra a proposta de (GASEVIC et al., 2004b), a qual define o OUP. Em (GASEVIC et al., 2004a) encontramos uma explicação melhor sobre OUP: um *profile* que permite a edição gráfica e desenvolvimento de ontologias utilizando diagramas UML, bem como outras facilidades decorrentes do uso tradicional de ferramentas CASE atuais para modelagem utilizando UML. Os modelos UML derivados do OUP são convertidos para o formato XMI e depois, transformados em uma ontologia na linguagem OWL, utilizando a tecnologia XSLT.

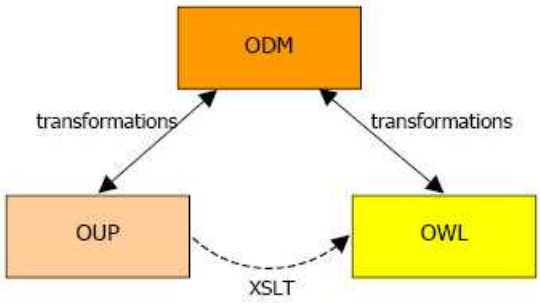

Figura 4.1: Relação entre a solução de (GASEVIC et al., 2004b) e as transformações recomendadas pela RFP (GASEVIC; DJURIC; DEVEDZIC, 2005)

Como base para concretização deste trabalho foram estudados modelos e técnicas que auxiliam na integração de sistemas utilizando ontologia através do enfoque da Engenharia Ontológica. Utilizamos como referência básica a abordagem de (GASEVIC et al., 2004b), que utiliza XSLT para transformação de profiles

UML em ontologias na linguagem OWL. De uma forma breve, em (GASEVIC et al., 2004b) é apresentada uma arquitetura baseada no ODM. O arquivo XMI exportado da ferramenta UML serve como entrada para um processador XSLT, que produz como saída um documento OWL, o qual pode ser importado em uma ferramenta especializada para desenvolvimento de ontologia, onde a ontologia pode ser refinada. Para testar a solução os autores da proposta utilizaram a ontologia do vinho (*Wine ontology*), bastante conhecida e distribuída como exemplo junto com o Protégé.

Através de estudos, observamos que os modelos de domínios representados através da UML possuem informação suficiente para serem utilizados como ponto de partida para a criação de ontologias. Para execução deste estudo de caso utilizamos como ferramenta CASE o Poseidon for UML Community Edition 4.2 e como ferramenta ontológica, o Protégé 3.1.1. As etapas seguidas em nosso estudo de caso são descritas abaixo:

1. Utilizar a documentação UML do sistema, mais especificamente o modelo de domínio, adequá-la ao OUP e exportar como um arquivo XMI.
2. Utilizar o arquivo XMI para gerar uma ontologia OWL preliminar. O arquivo XMI servirá de entrada para um processador XSLT (Xalan), que produzirá como saída um arquivo OWL.
3. Importar o arquivo OWL em uma ferramenta de tratamento ontológico (Protégé), onde ela será analisada e extendida para criar uma ontologia final axiomatizada, que servirá para o desenvolvimento de diversos sistemas relacionados ao tratamento da informação no domínio escolhido.

Na figura 4.2, que apresenta os passos de transformação, temos como ponto de partida uma ilustração da tela do Poseidon, de onde o OUP é exportado como arquivo XMI, e após passar pelo processador XSLT, se transforma em uma ontologia OWL que é importada no ambiente do Protégé, ilustrado mais à direita na figura, através de sua tela principal. De acordo com a figura, os autores sugerem que a ontologia gerada já pode ser utilizada em aplicações baseadas em ontologias. Porém, não utilizamos esta abordagem por verificarmos que alguns ajustes são necessários para que a ontologia, e conseqüentemente os sistemas que a utilizarem, possam fazer uso das vantagens advindas do formalismo que a DL oferece e que estão disponíveis na OWL DL.

Utilizamos como estudo de caso dois módulos do GSWeb, subprojeto do GIMPA, projeto da área médica aplicado no Hospital Universitário de Brasília (HUB) e que será melhor descrito na próxima seção.

4.1 Projeto GIMPA

O Projeto de Gerenciamento de Informações Médicas do Paciente (GIMPA) é um sistema subdividido em diversos módulos que capturam informações médicas

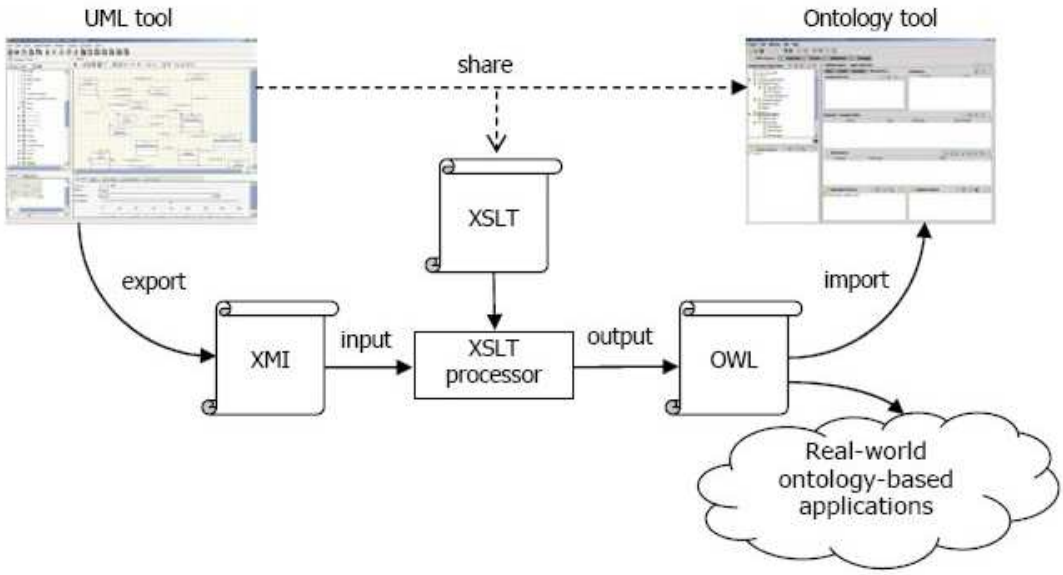

Figura 4.2: Passos de Transformação utilizando XSLT (GASEVIC; DJURIC; DEVEDZIC, 2005)

do paciente (CARVALHO; JR; HEINZELMAN, 2002). Os módulos são descritos abaixo e ilustrados na figura 4.3:

- Computador vestível - É um computador de dimensões reduzidas que interage com o ser humano, seja em situações estáticas ou dinâmicas, através do seu uso semelhante a uma roupa ou acessório. O computador vestível é utilizado para capturar sinais vitais, tais como pressão arterial não invasiva e oximetria do pulso, além de sintomas do paciente e outras informações.

- Prontuário Eletrônico do Paciente (PEP) - O PEP tem por missão capturar, processar, analisar, armazenar e disponibilizar informações médicas do paciente em um ambiente digital, tais como dados de exame físico, resultados de exames e descrição de diagnósticos.

- Módulo de Apoio à Decisão - Realiza diversos diagnósticos automáticos, tais como o de ECG, pressão arterial e avaliação Doppler do fluxo sanguíneo.

- Módulo de Processamento das Informações Médicas - integra os sistemas de apoio à decisão automáticos.

- Módulo de Recuperação de Informações baseada em Evidências - este módulo visa a recuperação de informações na Web ou em Bibliotecas digitais como Medline, utilizando critérios de medicina baseada em Evidências.

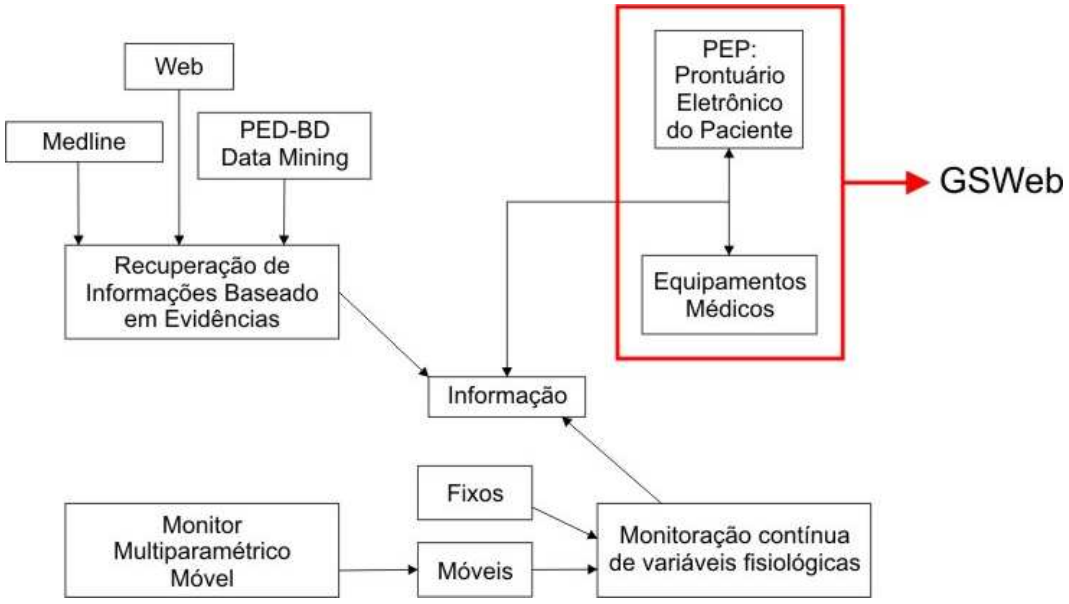

Figura 4.3: Estrutura Geral do GIMPA (CARVALHO; JR; HEINZELMAN, 2002)

4.1.1 GSWeb

O GSWeb é um sistema gerenciador de informações da saúde individual, que gerencia todas as informações referentes aos usuários do sistema de saúde do qual faz parte. Desenvolvido em plataforma aberta, é totalmente voltado para a Web, permitindo que em qualquer ponto do planeta, por meio de senhas apropriadas e conexões seguras, os diversos profissionais de saúde possam acessar o sistema e executar suas atividades rotineiras, integrando os relacionamentos existentes entre agendamento de consultas, atendimentos clínicos e prescrição de receitas.

É também um subprojeto do GIMPA que engloba o PEP e seus módulos relacionados, bem como os equipamentos médicos. O PEP foi implantado no HUB em junho de 2005. Utilizado através da intranet do HUB, até outubro de 2006 o PEP possuía 500.000 registros no banco de dados. No GSWeb estão inseridos os vários módulos para geração de laudos de exames que abordam aspectos do coração, válvulas e vasos. São eles: Cintilografia Cardíaca (CINT), Eletrocardiograma (ECG), Raio X do Tórax, Tomografia Computadorizada (CT) e Ressonância Magnética Nuclear (RMN). A figura 4.4 ilustra os módulos do GSWeb.

- Ecocardiograma (ECO) - Utiliza ultra-sons para examinar o coração, obtendo imagens que permitem avaliar o tamanho, espessura e movimento de diversas estruturas cardíacas.
- Cintilografia Cardíaca (CINT) - Verifica a anatomia e funcionalidade do coração por meio de marcadores rádio isotópicos.
- Eletrocardiograma (ECG) - Realiza a representação gráfica da atividade elétrica cardíaca, interpretando dados anatômicos e funcionais.

- Raio X do Tórax - Realiza a avaliação anatômica e indiretamente, a avaliação funcional do coração, vasos e outros órgãos.
- Tomografia Computadorizada (CT) - Verifica a anatomia e funcionalidade do coração com contraste e sem contraste.
- Ressonância Magnética Nuclear (RMN) - Por meio da distribuição de átomos, avalia a anatomia e fluxo sanguíneo.

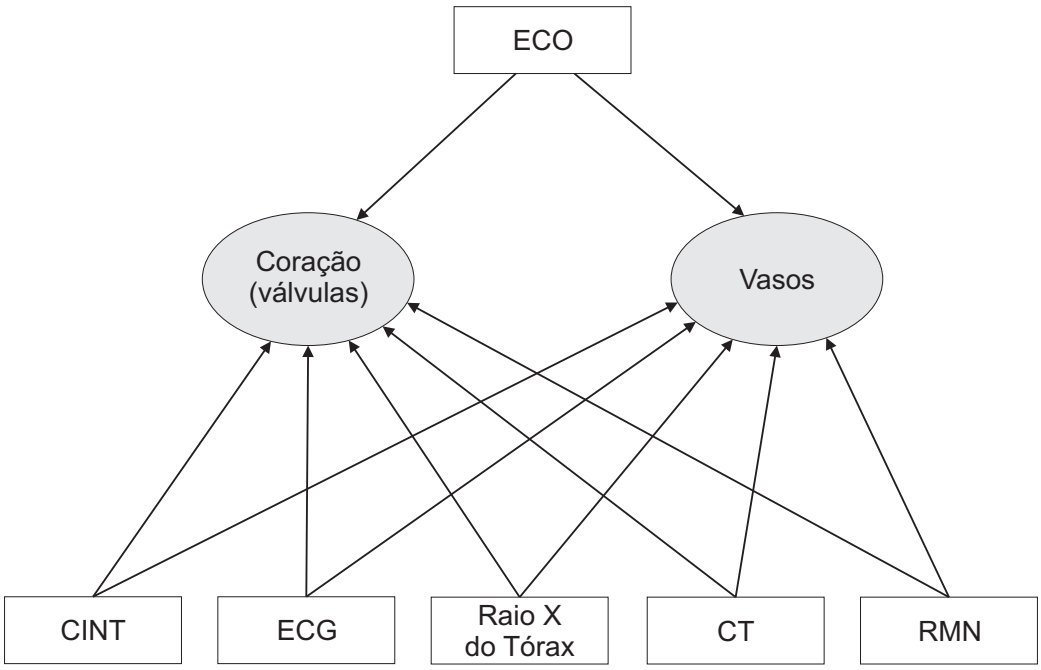

Figura 4.4: Módulos do GSWeb para geração de laudos

O Projeto ECO e o Projeto Raio foram utilizados como estudo de caso. O Projeto ECO é um sistema que registra informações referentes ao exame de Ecocardiograma (ECO) objetivando a geração do respectivo laudo. Ele foi implantado no HUB em dezembro de 2005 e até outubro de 2006 possuía 7.718 imagens armazenadas referentes a 971 pacientes. As informações utilizadas no Projeto ECO são provenientes do PEP e do equipamento que realiza o exame de Ecocardiograma. No equipamento é possível realizar medições sobre as imagens capturadas, cujos valores são registrados no sistema do Projeto ECO. As demais informações são obtidas pela análise das imagens armazenadas, realizada pelo médico, que registra no sistema as informações correspondentes e gera o laudo. A figura 4.5 ilustra o modelo de domínio do Projeto ECO, que serviu de base para aplicação do OUP proposto.

O Projeto Raio X objetiva a geração do laudo do exame de raio X, e está em fase de finalização da documentação necessária para sua implementação. Para permitir a integração do sistema existente (Projeto ECO) com o sistema que será implementado (Projeto Raio X), construímos uma ontologia que engloba

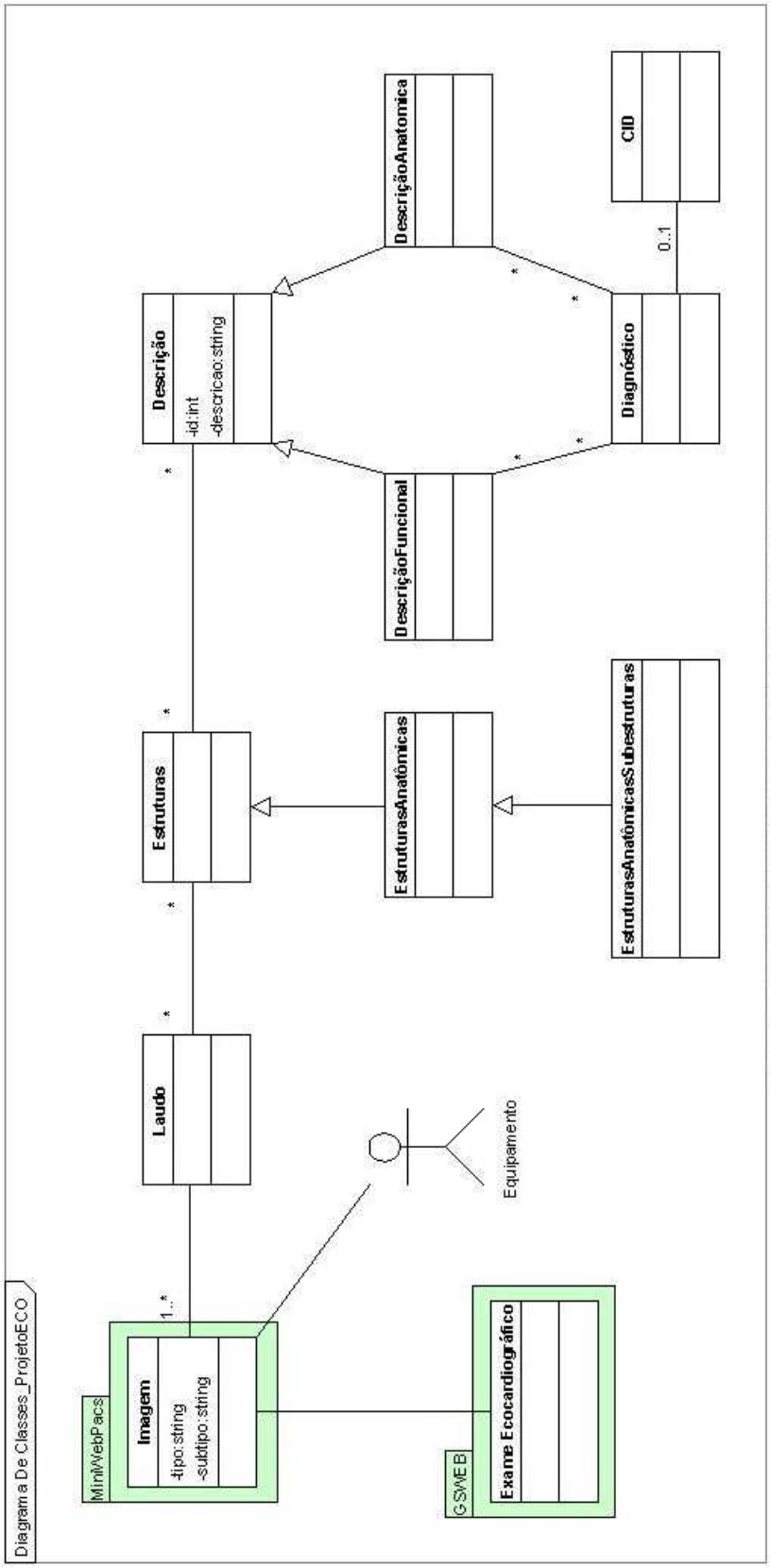

Figura 4.5: Modelo de Domínio - Projeto ECO

todos os conceitos e relacionamentos do Projeto ECO e os principais conceitos e relacionamentos do Projeto Raio X. A figura 4.6 ilustra o modelo de domínio do Projeto Raio X, utilizado como base para inserção das informações necessárias à integração dos dois sistemas através da ontologia.

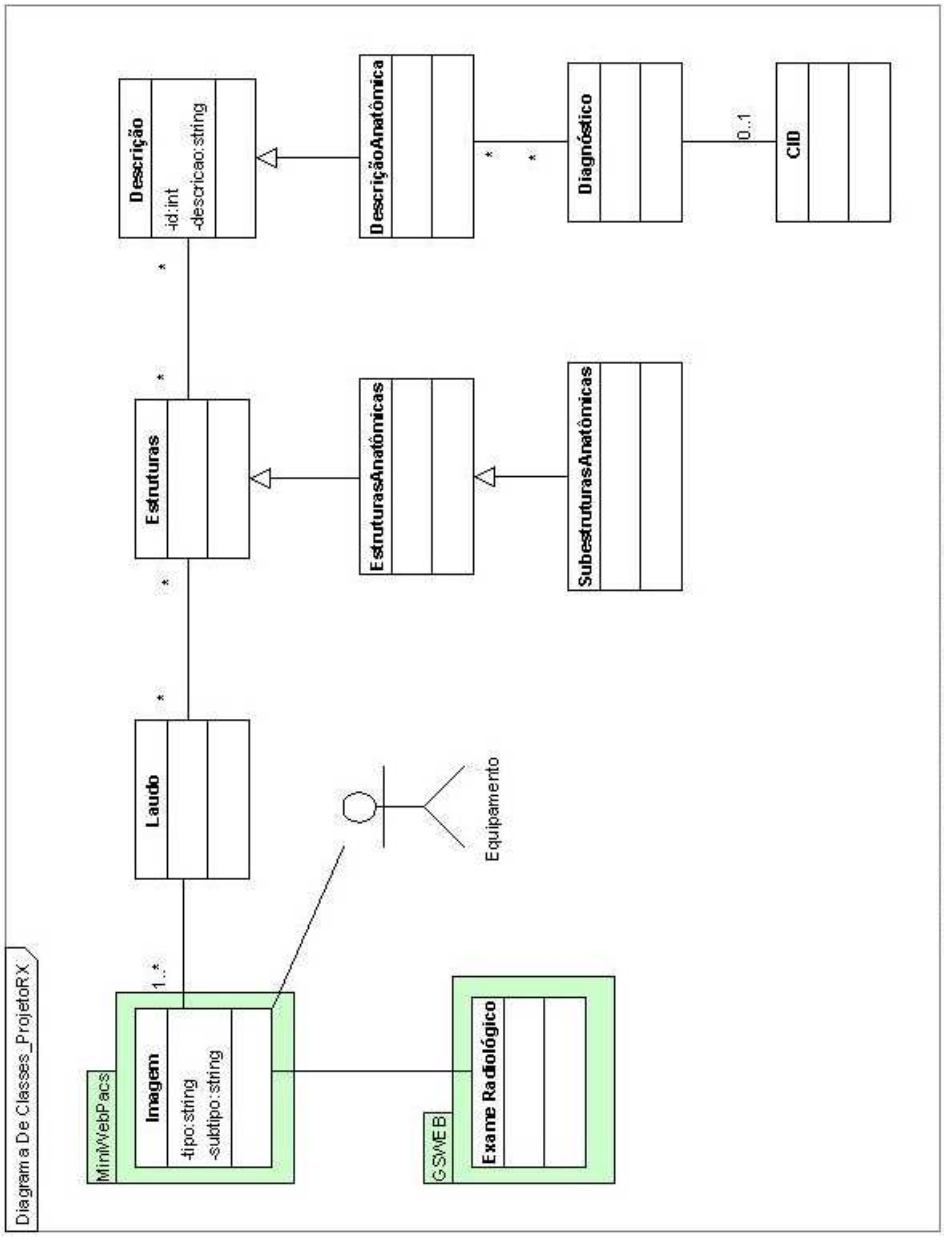

Figura 4.6: Modelo de Domínio - Projeto Raio X

Para melhor compreensão da ontologia desenvolvida, a próxima seção descreve o funcionamento básico do coração, abordando as principais funções exercidas por esse órgão através das suas estruturas e subestruturas.

4.1.2 Funcionamento do coração

O coração é um órgão constituído por paredes musculares que formam quatro cavidades: os átrios direito e esquerdo, e os ventrículos direito e esquerdo. Os átrios estão separados entre si pelo septo interatrial, e os ventrículos, pelo septo interventricular. Separando o átrio direito do ventrículo direito está a válvula tricúspide; entre o átrio esquerdo e o ventrículo esquerdo, encontra-se a válvula mitral.

No átrio esquerdo chegam quatro veias pulmonares, que conduzem sangue proveniente dos pulmões. Para o átrio direito chegam as veias cavas superior e inferior, que são os condutores terminais do sangue proveniente de todas as partes do organismo. Do ventrículo esquerdo sai a artéria aorta, que distribui sangue para todo o organismo. Na saída do ventrículo esquerdo situa-se a válvula aórtica, a qual separa este ventrículo da aorta. Do ventrículo direito emerge a artéria pulmonar, que é a condutora do sangue em direção aos pulmões; entre o ventrículo direito e o início da artéria pulmonar encontra-se a válvula pulmonar (JUNQUEIRA JR, 2006).

A atividade do coração consiste na alternância da contração, chamada de sístole, e do relaxamento, chamado de diástole, das paredes musculares dos átrios e dos ventrículos. Durante o período de relaxamento, o sangue flui das veias para os dois átrios, dilatando-as de forma gradual. Ao final deste período, suas paredes se contraem e impulsionam todo o seu conteúdo para os ventrículos. A contração e o relaxamento dos átrios e dos ventrículos não ocorre simultaneamente.

A contração do coração, tendo-se como referência os ventrículos, chama-se sístole cardíaca e o relaxamento chama-se diástole cardíaca. Durante a sístole ventricular, ou fase de esvaziamento do coração, os ventrículos se contraem, com conseqüente elevação da pressão no seu interior e abertura das válvulas aórtica e pulmonar em associação com o fechamento das válvulas mitral e tricúspide; assim, esvaziam seu conteúdo e diminuem de tamanho, sendo o volume de sangue ejetado chamado volume sistólico, o qual é da ordem de 60-70 ml (JUNQUEIRA JR, 2006).

Na diástole ventricular, ou fase de enchimento do coração, os ventrículos encontram-se relaxados, o que resulta em baixa pressão interna e na abertura das válvulas mitral e tricúspide em associação com o fechamento das válvulas aórtica e pulmonar. Em conseqüência, tornam-se repletos de sangue e aumentam seu tamanho, sendo o volume sangüíneo retido chamado volume diastólico (JUNQUEIRA JR, 2006).

O ciclo cardíaco, constituído do conjunto dos fenômenos que ocorrem nas fases de contração e relaxamento do coração, inclui alterações das dimensões e volumes atriais e ventriculares, bem como modificações das pressões no interior dos átrios e dos ventrículos, e os movimentos de fechamento e abertura das válvulas do coração. Os valores obtidos da análise do comportamento do coração durante o ciclo cardíaco são o alvo do Projeto ECO. A figura 4.7 (ROCHA, 2006) exibe as principais estruturas do coração.

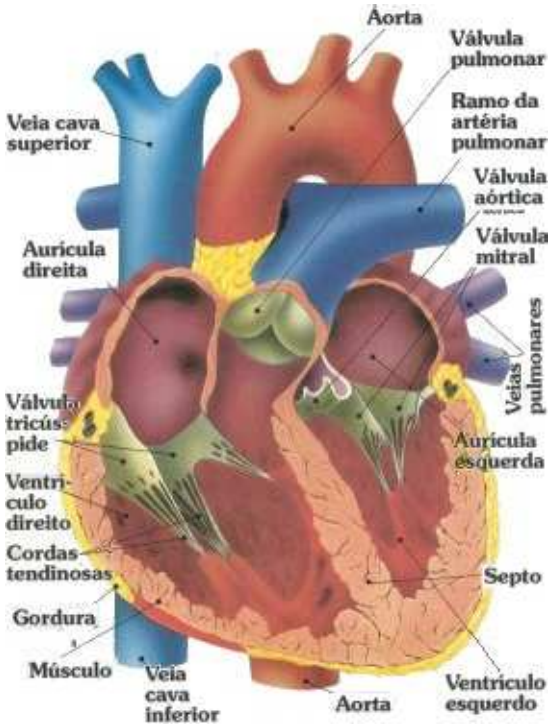

Figura 4.7: Estruturas do coração (ROCHA, 2006)

4.2 Aplicação do OUP

Passaremos a descrever os principais mecanismos de aplicação do OUP, conforme citado no *workflow* da figura 3.1, e utilizados neste estudo de caso. O OUP usa mecanismos de extensão da UML padrão definidos na especificação da UML: estereótipos, definições de tag (*tag definitions*), *tagged values* e restrições (*constraints*. Estereótipos permitem definir subclasses virtuais das metaclasses UML, atribuindo a elas semântica adicional. Há algumas diferenças entre classe UML tradicional e classe de ontologia, como é definida em OWL (owl:class). Por isto, são utilizadas classes UML estereotipadas com `<<OntClass>>` para modelar classes de ontologia no OUP. Os demais tipos de classes, tais como *Enumeration*, *Union* e *Restriction*, também são criadas com um novo estereótipo de classe para cada uma delas. Como o metamodelo UML encontra-se na camada M2 do MDA, as extensões também são feitas na camada M2.

Em UML, uma instância de uma classe é um objeto. Como um indivíduo de ontologia OWL e objeto UML possuem algumas diferenças, indivíduos OUP são modelados como objetos UML estereotipados, pois a especificação UML define que o estereótipo de um objeto deve coincidir com o estereótipo da sua classe. Desta forma, o estereótipo `<<OntClass>>` é anexado para as instâncias.

Em linguagens de ontologia como a OWL propriedade é um conceito autônomo, ao contrário dos atributos em UML, que são parte da classe UML à qual pertencem. Propriedade realiza em UML o papel dos relacionamentos (associações)

e atributos. De acordo com a semântica da OWL há dois tipos de propriedades: Propriedade de Objeto (*Object Property*) e Propriedade de tipo de dado (*Datatype Property*). *Object Property*, que pode ter somente indivíduos no seu escopo (*range*) e domínio, é representada no OUP com o estereótipo de classe `<<ObjectProperty>>`. *Datatype Property* é modelada com o estereótipo `<<DatatypeProperty>>` e relaciona indivíduos a valores de dados (figura 4.8).

Exemplos de diagramas de classes que descrevem propriedades de ontologia modeladas em UML são ilustrados nas figuras 4.8 e 4.9. A figura 4.8 mostra que as propriedades nome e sexo são do tipo String, porém os valores possíveis para a propriedade sexo são restringidos a feminino e masculino. No exemplo da figura 4.9 a classe Átrio é o domínio da propriedade temParedeAnormal, que é uma `<<ObjectProperty>>`, cujo escopo (*range*) é a classe Parede.

O estereótipo `<<Restriction>>` é utilizado para refinar as restrições de propriedades. Desta forma, a figura 4.9 também descreve uma restrição de classe sobre uma propriedade: a propriedade temParedeAnormal, da classe Átrio, tem uma restrição *allValuesFrom* para a `<<OntClass>>` Parede. Em outras palavras, cada instância da classe Átrio pode ter um relacionamento com uma instância da classe Parede, através da propriedade temParedeAnormal. Uma restrição adicional é a cardinalidade (ou seja, quantas instâncias de uma classe podem estar relacionadas com a propriedade) definida através da multiplicidade na associação entre a classe Átrio e a propriedade temParedeAnormal. O * indica que pode haver quantos relacionamentos forem necessários, ou seja, o número de relacionamentos não é restrito. Segundo Gasevic, a utilização de associação e dependência, ambas nomeadas como allValuesFrom, se deve à necessidade de distinguir visualmente, bem como enfatizar a relação entre a classe sobre a qual a propriedade é definida.

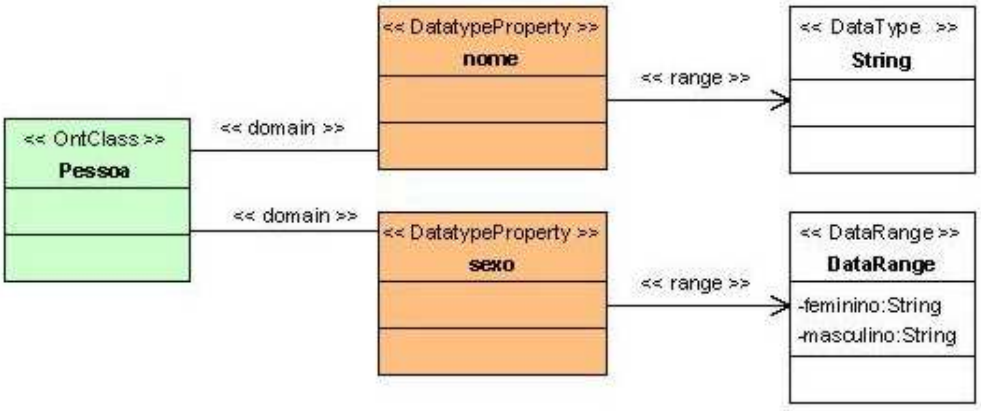

Figura 4.8: Propriedades *Datatype* nome e sexo da classe Pessoa

A hierarquia de classes da ontologia é definida no OUP utilizando o recurso de generalização proporcionado pela UML, conforme ilustra a figura 4.10. Tal como acontece nas principais linguagens orientada a objeto, a subclasse herda todas as definições de condições (tanto necessárias como necessárias e suficientes)

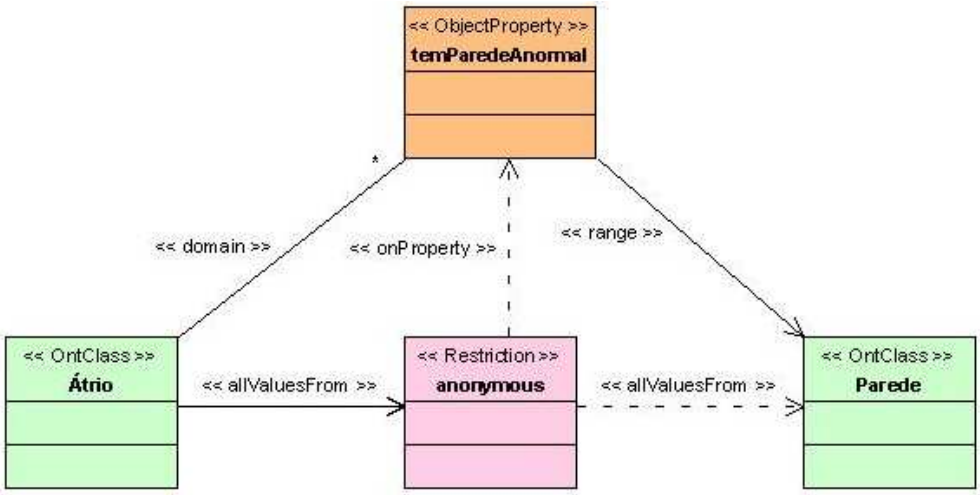

Figura 4.9: Propriedade temParedeAnormal da classe Átrio e suas restrições

da superclasse.

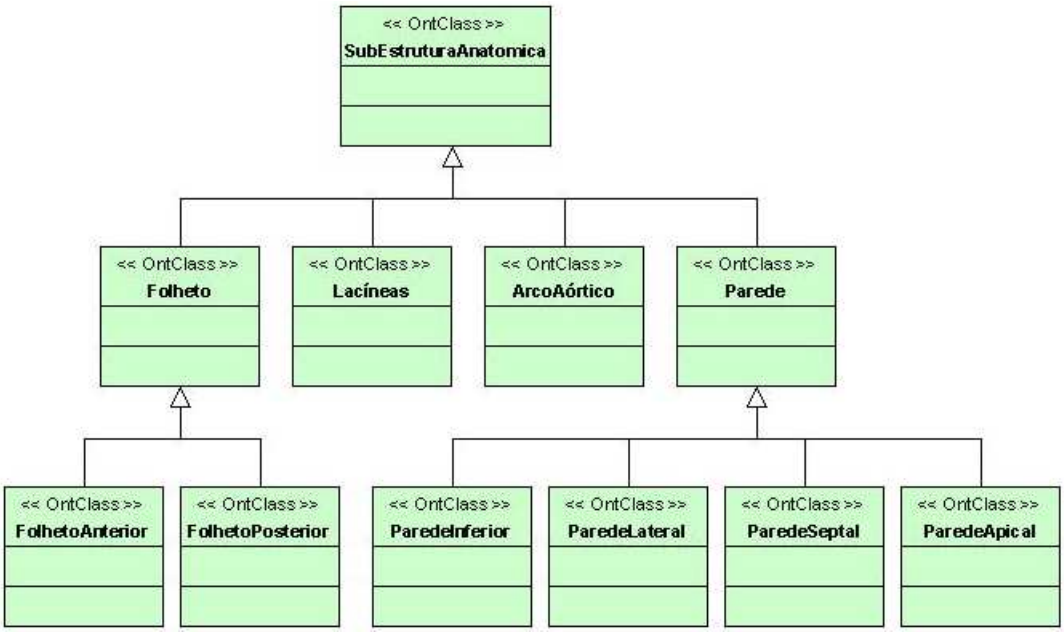

Figura 4.10: Subclasses da classe SubEstruturaAnatomica

Através das figuras, podemos ver que o modelo de domínio inicial é desmembrado em diversos diagramas de classes, que representam as classes da ontologia com suas respectivas propriedades. Se a propriedade apenas define o tipo de dado ao qual está relacionada, deve-se utilizar o estereótipo `<<DatatypeProperty>>`. *Domain* e *range* são definidos através de associações estereotipadas com `<<domain>>` ou `<<range>>`. No OUP, toda *DatatypeProperty* deve ter um *range* definido, mas não necessariamente um *domain*. Caso a propriedade relacione duas classes, é necessário utilizá-la como `<<ObjectProperty>>`, podendo estar

associada a uma `<<Restriction>>` com associações estereotipadas conforme a necessidade, por exemplo, `someValuesFrom`.

Após todas as classes e propriedades, com seus respectivos relacionamentos, serem modeladas em UML de acordo com o OUP, o arquivo foi exportado para o formato XMI. Este arquivo XMI foi utilizado como entrada para o processador XSLT Xalan que, utilizando o arquivo XSL proposto por (GASEVIC; DJURIC; DEVEDZIC, 2005), produziu como saída o arquivo OWL referente à ontologia modelada, que será descrita em detalhes na seção 4.3.

Durante a aplicação do OUP pudemos verificar que a utilização correta dos estereótipos é essencial nessa abordagem pois a transformação automática do OUP através de XSLT gera a ontologia com base nos estereótipos definidos na XSL. O OUP foi inicialmente projetado para utilização na versão 2.0 do Poseidon mas, segundo os autores, já está adequado para as versões 4.x do Poseidon, sendo utilizado em conjunto com uma nova versão do arquivo XSLT. Apesar disso, alguns recursos modelados na versão 4.2 do Poseidon não foram corretamente transformados para a ontologia. É o caso das classes enumeradas, definição de indivíduos e de características específicas de propriedade, e.g. propriedade funcional e simétrica, que foram implementadas diretamente no Protégé.

4.2.1 Metodologia

Utilizamos o Método 101 (NOY; MCGUINNESS, 2001) no desenvolvimento deste trabalho. A figura 4.11 encontrada em (BREITMAN; CASANOVA, 2005), ilustra os passos seguidos por este método, conforme descrito na seção 2.3.2.5. A seguir, descrevemos o que foi executado em cada passo, em resposta às principais perguntas sugeridas no método.

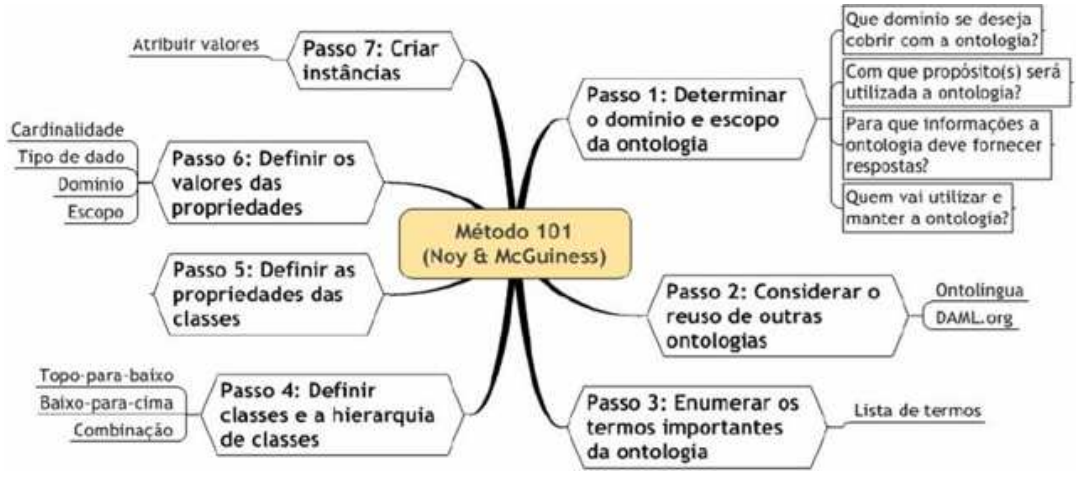

Figura 4.11: Ilustração do Método 101 (BREITMAN; CASANOVA, 2005)

1. A ontologia aborda conceitos relativos ao coração e vasos, com foco nos conceitos utilizados no laudo dos exames de Ecocardiograma e Raio X,

conforme descrito na seção 4.1.1. O propósito da ontologia é realizar o mapeamento dos conceitos do Projeto ECO com o sistema do Projeto Raio X, que será implementado, para auxiliar na integração dos sistemas. Desta forma, a ontologia deve definir de forma não ambígua todos os conceitos e relações deste domínio.

2. Neste trabalho não consideramos o reuso de ontologias pois o objetivo era criar uma ontologia a partir do Diagrama de Classes, conforme nossa proposta de transformação. Além disso, o estudo de caso trata de um domínio específico.

3. A lista de termos foi obtida com base na documentação dos sistemas, bem como entrevistas com o programador do Projeto ECO e com o médico responsável pelo projeto, Prof. Dr. Hervaldo Sampaio Carvalho, além de glossários de termos médicos e enciclopédias. O entendimento dos termos relativos ao funcionamento do coração foi facilitado pelo acompanhamento da realização do exame de Ecocardiograma.

4. Na definição da hierarquia de classes inicialmente utilizamos a abordagem *top-down* e em seguida, uma combinação das abordagens *top-down* e *bottom-up*. Por exemplo, primeiro foram definidas as subclasses da classe EstruturaAnatômica e da classe SubEstruturaAnatomica. A partir da definição das condições destas classes, foram sendo definidas as subclasses da classe Descrição. Durante este processo, algumas classes foram reagrupadas; é o caso das classes ÁtrioEsquerdo e ÁtrioDireito, subclasses da classe Átrio.

5. As propriedades das classes foram definidas com base nas opções disponibilizadas no Projeto ECO e requisitos do Projeto Raio X. Como exemplo citamos a propriedade ectasia, utilizada na definição da classe ArcoAórtico.

6. Os valores das propriedades foram definidos com base na documentação dos sistemas e no script do banco de dados do Projeto ECO. Por exemplo, a propriedade frequênciaCardíaca foi definida como sendo uma propriedade *Datatype*, do tipo *String*.

7. As instâncias foram definidas com base nas opções disponibilizadas no Projeto ECO e alguns requisitos do Projeto Raio X. Por exemplo, a classe Estado possui como instâncias os indivíduos normal e anormal.

4.3 Documentação da Ontologia

Nesta seção descrevemos as condições utilizadas para modelar a ontologia desenvolvida. A maior parte das condições foi definida no OUP, que após a transformação XSLT, gerou todas as condições como condições necessárias. Desta forma, após a transformação a ontologia foi editada no Protégé, onde algumas condições foram reajustadas para condições necessárias e suficientes, para permitir a execução de inferências. Além disto, também foram definidos tipos específicos de propriedades, classes enumeradas e indivíduos.

No Protégé, ao definirmos uma condição para uma classe que possui subclasses, as subclasses automaticamente herdam a definição da sua superclasse. Desta forma, as definições herdadas não são descritas aqui em cada subclasse, apenas na superclasse.

Para obtermos uma visão geral da ontologia, importamos o arquivo OWL na ferramenta Protégé e através do plugin OWL Viz, geramos a visualização gráfica das classes e subclasses da ontologia definida, conforme ilustra a figura 4.12. A figura 4.13 foi gerada utilizando o plugin Jambalaya e mostra a hierarquia das classes, destacando algumas subclasses da classe Estrutura. As principais classes da ontologia são listadas abaixo:

1. Descrição
2. Estrutura
3. Exame
4. Laudo
5. Pessoa
6. Órgão

Para melhor entendimento, apresentamos no Apêndice B a sintaxe OWL do Protégé utilizada na descrição das condições. A seguir descrevemos as seis classes principais, com suas respectivas subclasses, e as condições definidas para cada uma. Esclarecemos que as condições definidas apresentadas a seguir estão de acordo com as especificações de (HORRIDGE, 2004). A figura 4.14 exibe as classes da ontologia no Protégé, destacando as subclasses da classe Estrutura-Anatômica, onde se concentra a maior parte das classes definidas. Nesta figura também é possível observar as duas hierarquias: a) definida (*asserted*) e b)inferida (*inferred*).

1. **Classe Descrição** - é formada pelas subclasses DescriçãoAnatômica e DescriçãoFuncional, que são disjuntas. A disjunção das classes foi definida no Protégé.

   Classe DescriçãoAnatômica - é formada pela união de suas subclasses, definindo um axioma de cobertura.

   `Anormalidade` ⊔ `AumentoDeTamanho` ⊔ `Condição` ⊔ `DistribuiçãoArtériaPulmonar` ⊔ `DistribuiçãoVeiaPulmonar` ⊔ `Estado` ⊔ `Forma` ⊔ `Situação`

   Classe Anormalidade - é formada pela união de suas subclasses, definindo um axioma de cobertura.

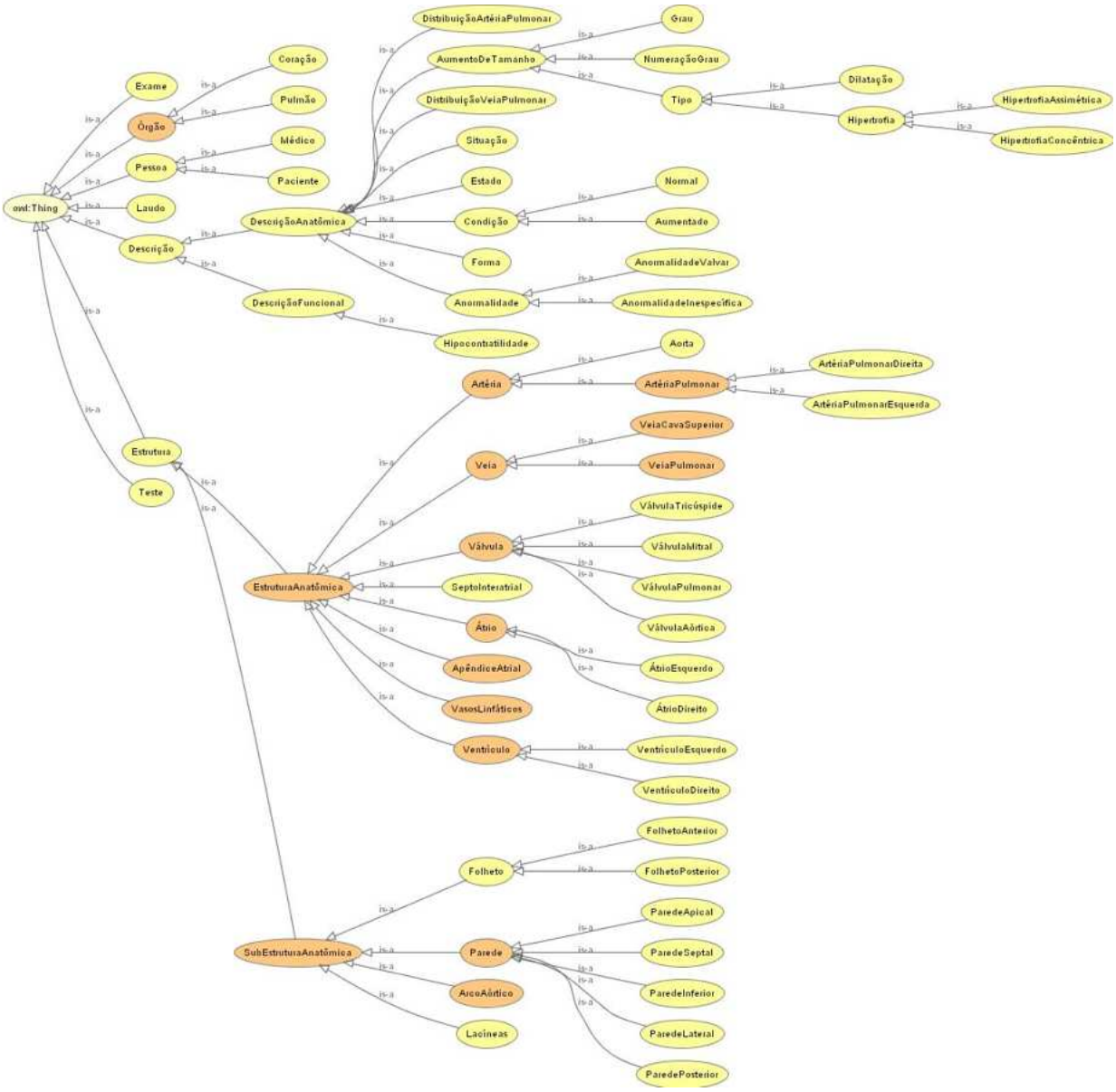

Figura 4.12: Classes da ontologia (*asserted hierarchy*)

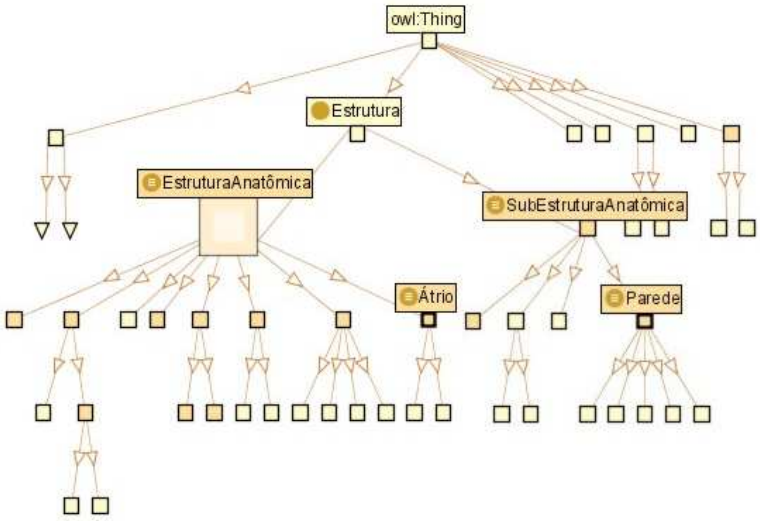

Figura 4.13: Hierarquia de Classes da Ontologia

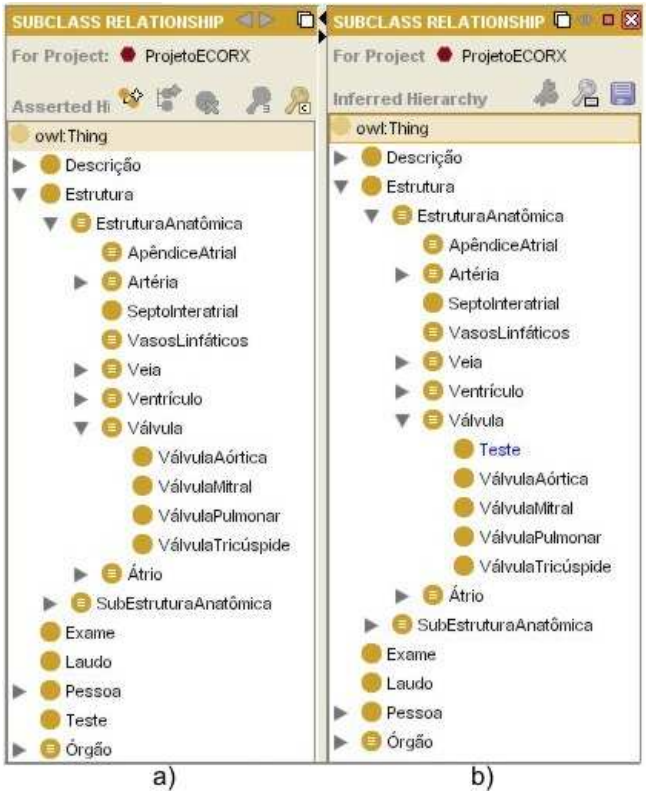

Figura 4.14: a)Hierarquia definida (*asserted*); b)Hierarquia inferida (*inferred*).

AnormalidadeInespecífica ⊔ AnormalidadeValvar

Classe AnormalidadeInespecífica - é definida como uma classe enumerada pelos seguintes indivíduos:

{aneurisma defeito trombo tumor}

Classe AnormalidadeValvar - é definida como uma classe enumerada pelos seguintes indivíduos:

{espessado calcificado vegetação perfurado}

Classe AumentoDeTamanho - é formada pela união de suas subclasses, definindo um axioma de cobertura.

Grau ⊔ NumeraçãoGrau ⊔ Tipo

Classe Grau - é definida como uma classe enumerada pelos seguintes indivíduos:

{leve moderado acentuado}

Classe NumeraçãoGrau - é definida como uma classe enumerada pelos seguintes indivíduos:

{Grau_I Grau_II Grau_III Grau_IV}

Classe Tipo - é formada pela união de suas subclasses, definindo um axioma de cobertura.

Dilatação ⊔ Hipertrofia

Classe Condição - é formada pela união de suas subclasses, definindo um axioma de cobertura.

Aumentado ⊔ Normal

Classe DistribuiçãoArtériaPulmonar - é definida como uma classe enumerada pelos seguintes indivíduos:

{calibre_e_distribuição_normal
aumento_de_calibre_de_artéria_pulmonar_direita
aumento_de_calibre_de_artéria_pulmonar_esquerda
hiperfluxo_arterial_pulmonar}

Classe DistribuiçãoVeiaPulmonar - é definida como uma classe enumerada pelos seguintes indivíduos:

{distribuição_normal
equalização_de_veias_para_o_ápice_e_base
inversão_da_trama_vascular_pulmonar}

Classe Estado - é definida como uma classe enumerada pelos seguintes indivíduos:

{normal anormal}

Classe Forma - é definida como uma classe enumerada pelos seguintes indivíduos:

{forma_normal retificada abaulada}

Classe Situação - é definida como uma classe enumerada pelos seguintes indivíduos:

{presente ausente}

Classe DescriçãoFuncional - possui como subclasse apenas a classe Hipocontratilidade

Classe Hipocontratilidade - é definida pelas seguintes condições necessárias:

{leve moderado acentuado}
∃ grauAumento Grau

2. **Classe Estrutura** - possui como subclasses as classes EstruturaAnatômica e SubEstruturaAnatômica.

   Classe EstruturaAnatômica - é formada pelas seguintes subclasses:

   ApêndiceAtrial, Artéria, SeptoInteratrial, VasosLinfáticos, Veia, Ventrículo, Válvula, Átrio e SubEstruturaAnatômica. Estas classes também fazem parte de um axioma de cobertura:

   ApêndiceAtrial ⊔ Artéria ⊔ SeptoInteratrial ⊔ VasosLinfáticos ⊔ Veia ⊔ Ventrículo ⊔ Válvula ⊔ Átrio

   Possui a seguinte condição necessária:

   ∃ éParteDe Órgão

A propriedade éParteDe é definida como propriedade transitiva e possui como propriedade inversa temComoParte.

Classe ApêndiceAtrial - possui as seguintes condições necessárias:

```
∃ estadoEstruturaApêndice (Estado ⊔ Grau ⊔
AnormalidadeInespecífica)
estadoEstruturaApêndice = 1
```

A propriedade estadoEstruturaApêndice é funcional.

Classe Artéria - possui como subclasses as classes Aorta e ArtériaPulmonar. É definida pela condição necessária e suficiente:

```
Aorta ⊔ ArtériaPulmonar
```

Classe ArtériaPulmonar - possui como subclasses as classes ArtériaPulmonarDireita e ArtériaPulmonarEsquerda, sendo definida pelas condições abaixo.

Condições necessárias e suficientes:

```
ArtériaPulmonarDireita ⊔ ArtériaPulmonarEsquerda
∃ temDistribuiçãoArtériaPulmonar DistribuiçãoArtériaPulmonar
∃ temForma Forma
```

Condições necessárias:

```
∃ dilataçãoAneurismática Situação
temDistribuiçãoArtériaPulmonar = 1
temForma = 1
```

Classe SeptoInteratrial - é definida através das condições abaixo.

Condição necessária e suficiente:

```
∀ temDefeito Grau
```

Condições necessárias:

```
∃ estadoDaEstrutura Estado
∀ temAnormalidade AnormalidadeInespecífica
temAnormalidade = 1
temDefeito = 1
```

Classe VasosLinfáticos - é definida pelas seguintes condições:

Condição necessária e suficiente:

∃ linhasBDeKerley Situação
Condição necessária: linhasBDeKerley = 1

Classe Veia - possui como subclasses as classes VeiaCavaSuperior e VeiaPulmonar.

Classe VeiaCavaSuperior - é definida pelas seguintes condições:

Condição necessária e suficiente:

∃ dilatacaoVeiaCavaSuperior Situação

Condição necessária:

dilatacaoVeiaCavaSuperior = 1

Classe VeiaPulmonar - é definida pelas seguintes condições:

Condição necessária e suficiente:

∃ temDistribuiçãoVeiaPulmonar DistribuiçãoVeiaPulmonar

Condição necessária:

temDistribuiçãoVeiaPulmonar = 1

Classe Ventrículo - possui como subclasses as classes VentrículoEsquerdo e VentrículoDireito, e como condições as seguintes:

Condição necessária e suficiente:

VentrículoDireito ⊔ VentrículoEsquerdo

Condições necessárias:

∃ condição Condição
∀ temAumento Grau

Classe VentrículoEsquerdo - é definida pelas seguintes condições necessárias:

```
∀ temAnormalidadeVE (Grau ⊔ AnormalidadeInespecífica ⊔
Hipertrofia)
∀ temTipo Tipo
∀ temParedeSeptal (AnormalidadeValvar ⊔ Grau ⊔
Hipocontratilidade ⊔ Hipertrofia)
∀ temParedeInferior (AnormalidadeValvar ⊔ Grau ⊔
Hipocontratilidade ⊔ Hipertrofia)
∀ temParedePosterior (AnormalidadeValvar ⊔ Grau ⊔
Hipocontratilidade ⊔ Hipertrofia)
∀ temParedeApical (AnormalidadeValvar ⊔ Grau ⊔
Hipocontratilidade ⊔ Hipertrofia)
∀ temParedeLateral (AnormalidadeValvar ⊔ Grau ⊔
Hipocontratilidade ⊔ Hipertrofia)
∃ estadoDaEstrutura Estado
temAnormalidadeVE = 1
temTipo = 1
```

Classe Válvula - é formada pelas subclasses VálvulaAórtica, VálvulaMitral, VálvulaPulmonar e VálvulaTricúspide, e definida pelas condições abaixo.

Condições necessárias e suficientes:

```
VálvulaAórtica ⊔ VálvulaMitral ⊔ VálvulaPulmonar ⊔
VálvulaTricúspide
∀ grauAberturaDiminuída Grau
∀ regurgitação Grau
```

Condição necessária:

```
∃ estadoDaEstrutura Estado
grauAberturaDiminuída = 1
regurgitação = 1
```

Classe VálvulaAórtica - é definida pela condição necessária:

```
∀ temLacíneas AnormalidadeValvar
```

Classe VálvulaPulmonar - é definida pela condição necessária:

```
∀ temLacíneas AnormalidadeValvar
```

Classe VálvulaMitral - é definida pela condição necessária:

```
∀ temFolhetoAnormal Folheto
```

Classe VálvulaTricúspide - é definida pela condição necessária:

∀ temFolhetoAnormal Folheto

Classe Átrio - é formada pelas subclasses ÁtrioDireito e ÁtrioEsquerdo, e definida pelas condições necessárias:

∃ dimensaoEstrutura (Estado ⊔ Grau)
dimensaoEstrutura = 1
∀ temAumento Grau
∃ temParedeAnormal Parede
∀ temAnormalidade AnormalidadeInespecífica

A propriedade dimensaoEstrutura é funcional.

Classe ÁtrioEsquerdo - é definida pelas condições:

Condições necessárias e suficientes:

∃ calcificaçõesAtriais Situação
∃ duploContorno Situação
∃ elevaçãoBrônquioFonteEsquerdo Situação
∃ quartoArco Situação

Classe SubEstruturaAnatômica - é formada pelas subclasses: ArcoAórtico, Folheto, Lacíneas e Parede. Estas classes formam um axioma de cobertura:

ArcoAórtico ⊔ Folheto ⊔ Lacíneas ⊔ Parede

Possui como condição necessária:

∃ éParteDe EstruturaAnatomica

Classe ArcoAórtico - é definida pelas condições:

Condições necessárias e suficientes:

∃ aortaDesenrolada Situação
∃ ectasia Grau
∃ sinaisDeDuploLúmen Situação
∃ temCalcificação Situação

Condições necessárias:

∃ dilataçãoAneurismática Situação
aortaDesenrolada = 1
ectasia = 1
sinaisDeDuploLúmen = 1
temCalcificação = 1
temDilataçãoAneurismáticaEm = 1

Classe Folheto - é formada pelas subclasses FolhetoAnterior e Folheto-Posterior, e é definida pela condição necessária:

∀ temAnormalidadeValvar AnormalidadeValvar

Classe Lacíneas - é definida pela condição necessária:

∀ temAnormalidadeValvar AnormalidadeValvar

Classe Parede - é formada pelas subclasses ParedeSeptal, ParedeInferior, ParedePosterior, ParedeApical e ParedeLateral.

Condições necessárias e suficientes:

ParedeSeptal ⊔ ParedeInferior ⊔ ParedePosterior ⊔
ParedeApical ⊔ ParedeLateral

3. **Classe Exame** - é definida pelas condições necessárias:

dataExame = 1
tipoExame = 1
qualidadeTécnica = 1
∀ solicitadoPor Médico
∃ solicitadoPor Médico
solicitadoPor = 1
∃ órgãoExaminado Órgão

4. **Classe Laudo** - é definida pelas condições necessárias:

∃ temExame Exame
temExame = 1
diâmetroDiastólicoFinalVE = 1
diâmetroRaizAorta = 1
diâmetroSistólicoFinalVE = 1
diâmetroVentricularDireito = 1
espessuraDiastólicaPPVE = 1
espessuraDiastólicaSepto = 1
fracaoEjecaoTeichhoiz = 1
massaVentricularEsquerda = 1
percentEncurtCavidade = 1
relacaoMassa_SuperficieCorporal = 1
relacaoSepto_PPVE = 1
relacaoVolume_Massa = 1
relacaoÁtrioEsquerdo_Aorta = 1
volumeDiastólicoFinal = 1
volumeSistólico = 1
volumeSistólicoFinal = 1
átrioEsquerdo = 1

5. **Classe Órgão** - é formada pelas subclasses Coração e Pulmão.

   Classe Coração - é definida pelas seguintes condições:

   Condições necessárias e suficientes:

   ∀ grauAumento NumeraçãoGrau
   ∃ temÁreaCardíaca Condição

   Condição necessária:

   ∀ proteseValvar Situação
   proteseValvar = 1
   temÁreaCardíaca = 1
   ∀ tipoProteseValvar Válvula
   tipoProteseValvar = 1

6. **Classe Pessoa** - é formada pelas subclasses Médico e Paciente. Possui as seguintes condições necessárias:

   nome = 1
   sexo = 1

   Classe Paciente - possui as seguintes condições necessárias:

   altura = 1
   convenio = 1
   frequênciaCardíaca = 1
   idPaciente = 1
   peso = 1
   pressãoArterial = 1
   ritmoCardíaco = 1

   Classe Teste - possui as seguintes condições necessárias:

   ∀ regurgitação Grau
   regurgitação = 1

4.4 Verificação Axiomática

A ontologia gerada é composta por 69 classes e 77 propriedades (entre *Object Properties* e *Datatype Properties*). Para confirmar que a ontologia é OWL DL utilizamos uma validação disponível no Protégé, através de uma opção para determinar a sublinguagem da ontologia que está sendo editada. Na versão do

Protégé que utilizamos, esta opção é *Determine/Convert OWL Sub-language...* e se encontra no menu OWL.

Uma das características chaves das ontologias OWL DL é que elas podem ser processadas por um raciocinador. O raciocinador executa, entre outras, duas funções importantes: verifica a classificação e a consistência da ontologia. A classificação é verificada através de testes executados sobre todas as classes da ontologia, gerando uma hierarquia de classes inferida. A consistência é verificada através da descrição da classe. Com base nas condições de cada classe, o raciocinador verifica se uma classe pode ou não ter instâncias. Uma classe é julgada como inconsistente se ela não pode ter qualquer instância (HORRIDGE, 2004).

Para verificar a classificação e consistência da ontologia que geramos, utilizamos o raciocinador RacerPro (http://www.racer-systems.com), com licença educacional, que está em conformidade com o DIG. Para utilizar os recursos de raciocínio no Protégé, é necessário executar o raciocinador em paralelo com o ambiente ontológico. Após o raciocinador ter sido inicializado, bastou clicar na opção *Check consitency...*, no menu OWL, para checar a consistência da ontologia. Para verificar a classificação, utilizamos a opção *Classify taxonomy*, do mesmo menu. No Protégé, a hierarquia de classe manualmente definida é chamada hierarquia definida ou *asserted hierarchy*.

A hierarquia de classes automaticamente computada por um raciocinador é chamada hierarquia inferida ou *inferred hierarchy* e no Protégé, aparece em uma nova área, ao lado da hierarquia definida. Se uma classe é reclassificada, seu nome aparece na cor azul, dentro da hierarquia inferida. Quando uma classe é identificada como inconsistente, aparece um círculo vermelho ao redor do ícone da classe.

A vantagem de utilizar um raciocinador pôde ser conferida na prática, em nosso estudo de caso. Utilizando a opção de verificação de consistência foi possível perceber alguns erros. Algumas propriedades tinham *domain* definido mas eram utilizadas na definição de outras classes, diferentes da classe definida como *domain*. Por exemplo, a propriedade `temAnormalidade` possuía como *domain* a classe `SeptoInteratrial`, porém também era utilizada pela classe `Átrio`. Tal fato gerava uma inconsistência, que se propagava para as demais classes que utilizassem as classes inconsistentes na sua definição. O mesmo também ocorreu com algumas definições de *range*.

Outro fato alertado pelo raciocinador é que a transitividade de uma propriedade também deve ser mantida para sua inversa. É o que ocorreu com a propriedade transitiva `éParteDe`, cuja propriedade inversa `temComoParte` não estava marcada como transitiva. Além disso verificamos o excesso de classes definidas, utilizadas pelo raciocinador para efetuar classificação de classes e indivíduos, bem como definições equivocadas de condições como sendo do tipo necessárias e suficientes. Verificamos que tais condições devem ser utilizadas com bastante critério, pois definem o que é necessário um indivíduo ter para ser classificado como membro de determinada classe. Desta forma, algumas classes estavam sendo errôneamente classificadas como subclasses de outras. Com o uso do raci-

ocinador foi possível perceber tal fato e corrigí-lo. Para verificar a classificação, criamos algumas classes de teste que foram corretamente classificadas, de acordo com as condições definidas. Por exemplo, criamos uma classe chamada `Teste`, como subclasse de `owl:Thing` e definimos como condição necessária: $\forall$ regurgitação Grau. Desta forma, a classe foi corretamente classificada como subclasse de `Válvula` já que esta última possui entre as definições de condições necessárias e suficientes, a condição $\forall$ regurgitação Grau.

4.5 Comparação de resultados

Passaremos agora a comparar nossa proposta com outras abordagens, com enfoque em ferramentas. Vale lembrar que utilizamos a ferramenta Poseidon for UML Community Edition versão 4.2 para todo o estudo de caso, sem problemas. Para ter um uso prático do OUP e do XSLT, (GASEVIC; DJURIC; DEVEDZIC, 2006) ressaltam que devemos utilizar uma ferramenta UML que permita:

- Anexar estereótipos para todos os conceitos UML utilizados no OUP. As ferramentas UML atuais raramente permitem que objetos e ligações entre eles e as classes tenham estereótipo.
- Uma forma conveniente para usar valores definidos, ou *tagged values*, e anexá-los a cada elemento UML.
- Criar relações entre conceitos UML. Os autores ressaltam a importância de relações, por exemplo, de dependência entre uma classe UML e um objeto UML.
- Utilizar o padrão XMI para serialização de modelos UML.

Em (GASEVIC; DJURIC; DEVEDZIC, 2006) encontramos a análise de duas ferramentas UML, IBM/Rational Rose e Poseidon for UML, que levou os autores a decidirem pela última porque esta suporta todos os requisitos já mencionados, ao contrário da IBM/Rational Rose que não permite a maior parte deles; a saber, um objeto não pode ter um estereótipo. Além disso, o Poseidon for UML utiliza o repositório MDR do NetBeans para armazenar metamodelos em conformidade com o MOF. Esta característica permite que nos beneficiemos de todas as vantagens do MDA.

A transformação do OUP para OWL é uma extensão prática das ferramentas UML atuais que dá a elas a capacidade de serem utilizadas para desenvolvimento completo de ontologia descrita por uma linguagem da Web Semântica. Segundo (GASEVIC; DJURIC; DEVEDZIC, 2006), é um tipo de ponte entre a engenharia de software e a ontológica, já que implementações em conformidade com o MDA atual estão em um estágio muito imaturo.

Em (GOOD OLD AI Research Group, 2006) encontramos o AIR ODM Eclipse *Plugin*, um ambiente de desenvolvimento de Inteligência Artificial integrado baseado nos conceitos de modelagem da MDA. Ele permite que as tecnologias de software com as quais os usuários estão familiarizados sejam expandidas com novas funcionalidades. Os autores ressaltam que o AIR seria uma primeira implementação do ODM trabalhado pelo grupo ao qual os autores pertencem, o GOOD OLD AI *Research Group*. Este é um grupo informal de pesquisadores de diferentes campos da Inteligência Artificial (AI) que buscam projetar, desenvolver, promover e distribuir componentes de software para construção de sistemas inteligentes. Os campos de interesse do grupo incluem representação do conhecimento, raciocínio inteligente, agentes inteligentes, ensino e aprendizagem inteligente, descoberta de conhecimento e ontologias.

Brockmans et al. (2006) apresentam duas ferramentas desenvolvidas no contexto do ODM submetido pelos autores ao OMG (IBM; Sandpiper Software, 2006): o *Visual Ontology Modeler* (VOM) (Sandpiper Software, 2006) e o *Integrated Ontology Development Toolkit* (IODT) (IBM, 2006). O VOM, desenvolvido pela empresa Sandpiper, é atualmente implementado como um *plugin* para o IBM/Rational Rose. A versão atual é compatível com os metamodelos ODM e o *profile* para RDFS/OWL. VOM suporta engenharia direta e reversa de ontologias RDFS/OWL e importa/exporta do ODM/XMI. A próxima geração do VOM suportará Eclipse e *Eclipse Modeling Framework* (EMF), ambos da IBM. O IODT é um *tookit* para desenvolvimento dirigido à ontologia, que inclui um EMF *Ontology Definition Metamodel* (EODM) baseado no ODM proposto pelos autores. O EODM é um ambiente de engenharia de ontologia baseado no Eclipse e um repositório de ontologia OWL. O *tookit* suporta serialização RDFS/OWL, raciocínio em TBox e ABox e transformação entre RDFS/OWL e outras linguagens de modelagem de dados.

Consideramos válidos os enfoques utilizados por (BROCKMANS et al., 2004) e (BROCKMANS et al., 2006) para um *profile* UML de onde várias idéias podem ser aproveitadas para a definição do *profile* UML oficial (http://www.omg.org). No entanto, a verificação prática da proposta de (BROCKMANS et al., 2006) não foi possível, pois não conseguimos obter os arquivos do VOM para efetuar testes com o Rational Rose, este último obtido com a licença educacional. A Sandpiper através de mensagem eletrônica (vide Apêndice C) nos informou que está trabalhando em uma nova versão, que será disponibilizada em breve. Enquanto, (BROCKMANS et al., 2004) não cita uma ferramenta que implemente sua proposta.

Sobre a proposta de (GASEVIC; DJURIC; DEVEDZIC, 2006), (BROCKMANS et al., 2006) alegam que não apresenta uma implementação completa. Em contrapartida, (GASEVIC; DJURIC; DEVEDZIC, 2006) cita que o problema das transformações entre linguagens de ontologias e linguagens baseadas no MDA não é discutido no documento submetido (IBM; Sandpiper Software, 2006). Acreditamos que é possível a unificação das propostas, aproveitando os pontos de destaques de cada uma.

Tendo em vista todos estes esforços para utilização de um *profile* UML, é natural que se levante uma questão: Há de fato a necessidade de uma *profile* UML para ontologias, já que existe o Protégé? O fato é que muitas ferramentas UML, como o Rational Rose, possuem muitas vantagens tais como: disponibilidade de literatura, cursos, exemplos, tutoriais, suporte da indústria, entre outras. O Protégé pode vir a ter características mais avançadas, mas o uso de padrões de modelagem bem conhecidos continua sendo um argumento para ter *profiles* UML para ontologias (GASEVIC; DJURIC; DEVEDZIC, 2006).

Com o objetivo de compartilhar o conhecimento adquirido e apresentar a proposta sobre a qual trabalhamos, submetemos um artigo, com o título Arquitetura MDA e Ontologia OWL: Análise e Aplicação, para o Simpósio Brasileiro de Engenharia de Software - SBES 2006. Este artigo foi produzido no primeiro semestre de 2006 e basicamente aplica a abordagem de (GASEVIC; DJURIC; DEVEDZIC, 2005) no Sistema de Gerenciamento de Tarefas e Projetos (SISPROJ), utilizado como estudo de caso (SILVA; SANTOS, 2004). O SISPROJ é um sistema que gerencia o fluxo de atividades necessárias para a implementação de solicitações de serviço enviadas ao Centro de Informática de uma empresa pelos usuários dos diversos setores que a compõem. A intenção do artigo era demonstrar que é possível a utilização conjunta das tecnologias da Web Semântica e da Engenharia de Software. A tabela 4.1 descreve as notas recebidas dos três revisores, em cada área de avaliação, que variam de 1 a 5.

	Revisor 1	**Revisor 2**	**Revisor 3**	**Média**
Originalidade e novidade	3	1	2	2
Contribuição Técnica	3	2	1	2
Relevância para a chamada de artigos	4	2	3	3
Leitura, organização e apresentação	4	2	3	3
Total pontos revisor/total máximo	14/20	7/20	9/20	

Tabela 4.1: Avaliação do artigo submetido ao SBES 2006

Como um ponto negativo para não ter sido aceito um dos revisores citou que o artigo apresenta apenas uma proposta inicial e um estudo de caso. Outro revisor afirmou que o trabalho é promissor, desde que seja mostrado que a ontologia gerada pode ser reaproveitada em novos sistemas. Vale ressaltar que o estudo de caso utilizado neste artigo não incluía a formalização completa da ontologia, como o trabalho atual, pois somente incluía *range* e *domain*. Também não foi realizada experiência com o raciocinador para verificação de consistência e classificação da ontologia.

Capítulo 5

Conclusões e trabalhos futuros

Está claro que as vantagens de uma visão ontológica do mundo, como esboçada pela Web Semântica, é atraente para o campo da Engenharia de Software, bem como a utilização de conceitos e ferramentas oriundos da Engenharia de Software, que possui técnicas já consolidadas, permite ampliar os benefícios proporcionados pelas ontologias. O desafio é unir as duas áreas, o que significa que ambos os lados deverão fazer concessões para produzir um enfoque híbrido, combinando elementos da Engenharia de Software com o campo emergente da Engenharia Ontológica, de forma a possibilitar o desenvolvimento e uso de ontologias em aplicações do mundo real. Em direção a esse desafio, do lado da Engenharia de Software encontramos os modelos disponibilizados pelo MDA, tais como MOF, UML e XMI. A Engenharia Ontológica por sua vez disponibiliza as linguagens ontológicas com base formal e bastante poder expressivo, tais como RDF e OWL.

As ferramentas para criação e manutenção de ontologias disponibilizadas no mercado, tal como Protégé, proporcionam condições para definição e formalização das informações inerentes à ontologia, além de permitir o uso de um raciocinador para verificar a consistência e classificação da ontologia. Desta forma, a identificação e correção de erros é facilitada. Padrões de interoperabilidade disponibilizados pelo W3C, como XSLT, permitem a transformação das informações para diversos formatos, de acordo com a necessidade. Este seria talvez o principal benefício que a Web Semântica pode trazer para a Engenharia de Software. Os modelos de domínios também podem ser compartilhados entre as aplicações em um ambiente aberto proporcionando reuso e interoperabilidade, tal como está previsto na Web Semântica.

Além disso, esforços para definição de modelos que aproveitam as tecnologias já consolidadas na área de Engenharia de Software para serem utilizadas em conjunto com as tecnologias da Web Semântica, como é o caso do ODM, já estão sendo levados adiante pela OMG e estão cada vez mais próximos da realidade através da formalização dos mapeamentos necessários. É neste cenário que encontra-se este trabalho, onde estudamos e aplicamos técnicas para integração de sistemas computacionais utilizando ontologias.

Como trabalhos futuros citamos o acompanhamento do processo de utilização

da ontologia na implementação de um novo software, iniciando pelo Projeto Raio X, cujos conceitos já estão incorporados na ontologia que geramos, e incluindo os sistemas para geração de laudos dos exames de Cintilografia Cardíaca (CINT), Eletrocardiograma (ECG), Tomografia Computadorizada (CT) e Ressonância Magnética Nuclear (RMN), a partir da definição do diagrama de classes. As ontologias geradas através da nossa abordagem podem ser mescladas utilizando diversos trabalhos de *matching* semântico, por exemplo, (LOPES, 2005).

Também como trabalhos futuros podemos citar a evolução da ontologia, com reuso de ontologias existentes, e os testes com a ferramenta Rational Rose em conjunto com o plugin VOM, o qual a empresa Sandpiper por mensagem eletrônica (vide Apêndice C) prontificou-se a disponibilizá-lo. Desta forma será possível comparar na prática as duas abordagens mais importantes, atualmente existentes na literatura.

Apêndice A

XSLT

Incluímos a seguir parte do código da XSLT utilizada neste trabalho para transformação do arquivo xmi em uma ontologia OWL, sendo que o código completo está disponível no site do pesquisador Gasevic (http://www.sfu.ca/∼dgasevic).

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:UML="org.omg.xmi.namespace.UML" xmlns:UML2 =
'org.omg.xmi.namespace.UML2' xmlns:owl=
"http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
exclude-result-prefixes="UML" xmlns="http://owl.protege.stanford.edu#">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:variable name="owlPrefix">
<xsl:text>http://www.w3.org/2002/07/owl#</xsl:text>
</xsl:variable>
<xsl:variable name="xsdPrefix">
<xsl:text>http://www.w3.org/2001/XMLSchema#</xsl:text>
</xsl:variable>
<xsl:key name="stereotypeID" match="UML:Stereotype" use="@xmi.id"/>
<xsl:key name="stereotypeName" match="UML:Stereotype" use="@name"/>
<xsl:key name="classID" match="UML:Class" use="@xmi.id"/>
<xsl:key name="classStereotypeID" match="UML:Class"
use= "UML:ModelElement.stereotype/UML:Stereotype/@xmi.idref"/>
<xsl:key name="linkEndID" match="UML:LinkEnd" use="@xmi.id"/>
<xsl:key name="linkLinkEndID" match="UML:Link"
use="UML:Link.connection/UML:LinkEnd/@xmi.id"/>
<xsl:key name="linkID" match="UML:Link" use="@xmi.id"/>
<xsl:key name="objectID" match="UML:Object" use="@xmi.id"/>
<xsl:key name="tagDefinitionID" match="UML:TagDefinition" use="@xmi.id"/>
```

```
<xsl:key name="dataTypeID" match="UML:DataType" use="@xmi.id"/>
<xsl:key name="dependencyID" match="UML:Dependency" use="@xmi.id"/>
<xsl:key name="dependencyClientID" match="UML:Dependency"
use="UML:Dependency.client/UML:Class/@xmi.idref"/>
<xsl:key name="dependencySupplierID" match="UML:Dependency"
use="UML:Dependency.supplier/UML:Class/@xmi.idref"/>
<xsl:key name="association" match="UML:Association" use="."/>
<xsl:key name="generalizationID" match="UML:Generalization"
use="@xmi.id"/>
<xsl:key name="associationStereotypeID"
match="UML:Association"
use="UML:ModelElement.stereotype/UML:Stereotype/@xmi.idref"/>
<xsl:key name="associationEnd1" match="UML:Association"
use="UML:Association.connection/UML:AssociationEnd[1]/UML:
AssociationEnd.participant/UML:Class/@xmi.idref"/>
<xsl:key name="associationEnd2" match="UML:Association" use="UML:
Association.connection/UML:AssociationEnd[2]/UML:
AssociationEnd.participant/UML:Class/@xmi.idref"/>
<xsl:template match="/">
<xsl:apply-templates select="XMI"/>
</xsl:template>
<xsl:template match="XMI">
<xsl:apply-templates select="XMI.content/UML:Model"/>
</xsl:template>
<xsl:template match="XMI.content/UML:Model">
<xsl:apply-templates select="UML:Namespace.ownedElement"/>
</xsl:template>
<xsl:template match="UML:Namespace.ownedElement">
<rdf:RDF>
<xsl:attribute name =
"xml:base">http://owl.protege.stanford.edu</xsl:attribute>
<xsl:apply-templates select="UML:Package/UML:Namespace.ownedElement"/>
</rdf:RDF>
</xsl:template>
<xsl:template match="UML:Package/UML:Namespace.ownedElement">
<xsl:if test="(key('stereotypeID', ../UML:
ModelElement.stereotype/UML:Stereotype/@xmi.idref)/@name = 'ontology') and
(key('stereotypeID',
../UML:ModelElement.stereotype/UML:Stereotype/@xmi.idref)/
UML:Stereotype.baseClass = 'Package')">
<xsl:apply-templates select="UML:Class"/>
<xsl:apply-templates select="UML:Object"/>
</xsl:if>
</xsl:template>
```

Apêndice B

Sintaxe OWL do Protégé

Elemento	**Símb.**	**Exemplo**	**Significado**
allValuesFrom	$\forall$	$\forall$ criança Masculino	Toda criança deve ser do tipo Masculino
someValuesFrom	$\exists$	$\exists$ criança Advogado	Pelo menos uma criança deve ser do tipo Advogado
hasValue	$\ni$	rico $\ni$ verdadeiro	A propriedade rico deve ter valor verdadeiro
cardinality	$=$	criança $= 3$	Deve haver exatamente 3 crianças
minCardinality	$\geq$	criança ≥ 3	Deve haver pelo menos 3 crianças
maxCardinality	$\leq$	criança ≤ 3	Deve haver no máximo 3 crianças
complementOf	$\neg$	$\neg$Pai	Qualquer coisa que não é do tipo Pai
intersectionOf	$\sqcap$	Humano $\sqcap$ Masculino	Todos os Humanos que são Masculinos
unionOf	$\sqcup$	Doutor $\sqcup$ Advogado	Qualquer coisa que é Doutor ou Advogado
enumeration	{...}	{masculino feminino}	Indivíduos masculino ou feminino

Tabela B.1: Sintaxe OWL do Protégé

Apêndice C

Mensagem Eletrônica da Sandpiper

From: "Elisa F. Kendall" <ekendall@sandsoft.com>
To: eluzai@unb.br
Sent: Tuesday, November 28, 2006 7:35 PM
Subject: Inquiry regarding Sandpiper's VOM tool

Hi Eluzai,

We received your email and would be happy to have you use our tool in your research. We are in the process of finalizing our latest release, which will have owl reverse engineering capabilities, so it might be best if you wait for this, which I anticipate will be a week or two from now. It's a multi-pass process, thus can be a little slow for fairly large ontologies, fyi, but the results are good based on our testing to date.

We do not resell IBM Rational Rose, so you'll need to obtain a separate license for that. The VOM works with any IBM Rational Rose Enterprise product from 2002 or later, and with Rose Data Modeler or other versions that are newer than 2004 (any that support add-ins generally).

Also, it would be helpful to understand your project:

- what domain area are you working in

- what kinds of applications/target use cases for the ontology(ies) do you have?

- what questions are you attempting to answer through the research process

- what timeframe do you need to complete the project by

We would like the option of reviewing and possibly citing your paper if you use our software free of charge, so please give that some thought as well.

If you are willing to share your research and work with us to provide feedback, then we would be delighted to have you try the software and let us know what you

find. We'll also support you as resources permit, but we're a very small business with key customers to support, so please remind us if we haven't gotten back to you in a day or two.

Let us know what you would like to do, and we'll look forward to working with you.

Best regards,

Elisa Kendall.

Referências

ALMEIDA, M. B. Uma Visão Geral sobre Ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção. *Ciência da informação*, v. 32, n. 3, p. 7–20, 2003. Disponível em: `http://www.ibict.br/cienciadainformacao/include/getdoc.php?id=223article=36mode=pdf`. Acesso em: abril de 2005.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML - Guia do Usuário*. Rio de Janeiro, RJ: Campus, 2005.

BPTRENDS, B. P. T. *UML 2.0 Adopted*. 2003. Site. Disponível em: `http://www.bptrends.com/`. Acesso em: julho de 2006.

BREITMAN, K. K. *Web Semântica: a Internet do Futuro*. Rio de Janeiro, RJ: LTC - Livros Técnicos e Científicos Editora, 1999.

BREITMAN, K. K.; CASANOVA, M. A. Desenvolvimento de Ontologias para Engenharia de Software e Banco de Dados: Um Tutorial Prático. In: *Anais do 19º Simpósio Brasileiro de Engenharia de Software*. Uberlândia, MG: [s.n.], 2005. Disponível em: `http://www.sbbd-sbes2005.ufu.br`. Acesso em: dezembro de 2005.

BROCKMANS, S. et al. A Model Driven Approach for Building OWL DL and OWL Full Ontologies. In: *The 5th International Semantic Web Conference*. Athens/GA, USA: Springer, 2006. (LNCS). Disponível em: <`http://www.aifb.uni-karlsruhe.de/WBS/sbr/publications/omgodm.pdf`>.

BROCKMANS, S. et al. Visual modeling of owl dl ontologies using uml. In: *Proceedings of the 3th International Semantic Web Conference*. Hiroshima, Japan: Springer-Verlag, 2004. p. 198–213.

BROWN, A. *An introduction to Model Driven Architecture Part I: MDA and todays systems*. February 2004. IBM developerWorks. Disponível em: `http://www-128.ibm.com/developerworks/rational/library/3100.html`. Acesso em: setembro de 2005.

BRUGGEMANN, E. S.; PORTO, R. M. Uma Proposta Ontológica para um Sistema de Gestão de Versionamento do Modelo ITIL. Undergraduate Work at University of Brasília. July 2006.

CANTELE, R. C. et al. Reengenharia e Ontologias: Análise e Aplicação. In: LIMA, F. (Ed.). *WWS'2004: Proceedings of the 1º Workshop de Web Semântica*. Brasília: SBC, 2004. Disponível em: `http://www.lti.pcs.usp.br/publicacoes/Cantele-etal04-WWS.pdf`. Acesso em: junho de 2005.

CARVALHO, H. S.; JR, C. J. N. C.; HEINZELMAN, W. B. Gerenciamento de Informações Médicas do Paciente (Projeto GIMPA). In: *Proceedings of VIII Congresso Brasileiro de Informática em Saúde*. Natal/RN, Brasil: [s.n.], 2002.

CHAVES, M. S. Uso de Ontologias para Gerenciamento e Acesso a Documentos na Web. In: *Anais da V Oficina de Inteligência Artificial*. Pelotas, RS: [s.n.], 2001. p. 75–87.

CRANEFIELD, S. UML and the Semantic Web. In: *SWWS: Proceedings of the International Semantic Web Working Symposium*. [S.l.: s.n.], 2001.

DECKER, S. et al. The Semantic Web: The Roles of XML and RDF. *IEEE Internet Computing*, v. 4, n. 5, p. 63–74, 2000.

DEVEDZIC, V. Understanding Ontological Engineering. *Commun. ACM*, ACM Press, New York, NY, USA, v. 45, n. 4, p. 136–144, 2002. ISSN 0001-0782.

DJURIC, D.; GASEVIC, D.; DEVEDZIC, V. A MDA-based Approach to the Ontology Definition Metamodel. In: *Proceedings of the 4^{th} International Workshop on Computational Intelligence and Information Technologies*. Nis, Serbia and Montenegro: [s.n.], 2003. p. 51–54.

DJURIC, D.; GASEVIC, D.; DEVEDZIC, V. Ontology Modeling and MDA. *Journal of Object Technology*, v. 4, n. 1, p. 109–128, 2005. Disponível em: `http://www.jot.fm/issues/issue_2005_01/article3`. Acesso em: setembro de 2005.

DSTC. *Ontology Definition Metamodel (ODM) Initial Submission*. August 2003. Disponível em: `http://www.omg.org/docs/ad/03-08-01.pdf`.

DUDDY, K. UML2 Must Enable a Family of Languages. *Commun. ACM*, ACM Press, New York, NY, USA, v. 45, n. 11, p. 73–75, 2002. ISSN 0001-0782.

EHRIG, S. H. M. et al. *The Karlsruhe View on Ontologies*. [S.l.], 2003.

GASEVIC, D.; DJURIC, D.; DEVEDZIC, V. Bridging MDA and OWL Ontologies. *Journal of Web Engineering*, v. 4, n. 2, p. 118–143, 2005.

GASEVIC, D.; DJURIC, D.; DEVEDZIC, V. MDA-based Automatic OWL Ontology Development. *International Journal on Software Tools for Technology Transfer (STTT)*, Springer Berlin-Heidelberg, 2006. ISSN 1433-2779.

GASEVIC, D. et al. A UML Profile for OWL Ontologies. In: *Proceedings of Model-Driven Architecture: Foundations and Applications - MDAFA*. Linkoping, Sweden: [s.n.], 2004. p. 138–152.

GASEVIC, D. et al. Converting UML to OWL ontologies. In: *WWW Alt. '04: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*. New York, NY, USA: ACM Press, 2004. p. 488–489. ISBN 1-58113-912-8.

GENTLEWARE. *Ontology Definition Metamodel (ODM) Proposal*. August 2003. Disponível em: `http://www.omg.org/docs/ad/03-08-09.pdf`.

GENTLEWARE. *Poseidon for UML*. 2006. Disponível em: `http://gentleware.com/`. Acesso em: Agosto de 2006.

GOOD OLD AI Research Group. *AIR ODM Eclipse Plugin*. 2006. Disponível em: `http://www.sfu.ca/~dgasevic/projects/AIR/`.

GRUBER, T. *What is an Ontology?* 2005. Site Tom Gruber. Disponível em: `http://www-ksl.stanford.edu/kst/what-is-an-ontology.html`. Acesso em: agosto de 2005.

GRÜNINGER, M.; FOX, M. Methodology for the Design and Evaluation of Ontologies. In: *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*. Montreal: [s.n.], 1995.

GUARINO, N. Formal Ontology and Information Systems. In: *FOIS'98: Proceedings of the International Conference On Formal Ontology In Information Systems*. Trento, ITALY: Amsterdam, IOS Press, 1998. p. 3–15.

GUARINO, N.; WELTY, C. Evaluating Ontological Decisions with OntoClean. *Commun. ACM*, ACM Press, New York/NY, USA, v. 45, n. 2, p. 61–65, 2002. ISSN 0001-0782.

GóMEZ-PéREZ, A. Ontological Engineering: A State Of The Art. *Expert Update*, British Computer Society, v. 2, n. 3, p. 33–43, 1999.

GóMEZ-PéREZ, A.; CORCHO, O. Ontology Languages for the Semantic Web. *IEEE Intelligent Systems*, v. 17, n. 1, p. 54–60, 2002. ISSN 1541-1672.

HOLMAN, G. K. *What is XSLT?* 2000. Disponível em: `http://www.xml.com/pub/a/2000/08/holman/`. Acesso em: outubro de 2005.

HORRIDGE, M. *A Practical Guide To Building OWL Ontologies Using The Protege-OWL Plugin and CO-ODE Tools*. 2004. Disponível em: `http://www.co-ode.org/`. Acesso em: Setembro de 2006.

HORROCKS, I. *OWL: A Description Logic Based Ontology Language*. 2006. Disponível em: `http://www.cs.man.ac.uk/~horrocks/Slides/`. Acesso em: Outubro de 2006.

IBM. *Integrated Ontology Development Toolkit*. 2006. Disponível em: `http://www.alphaworks.ibm.com/tech/semanticstk`.

IBM; Sandpiper Software. *Ontology Definition Metamodel (ODM)*. April 2006. Sixth Revised Submission to OMG/RFP ad/2003-03-40. Disponível em: `http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf`. Acesso em: novembro de 2006.

IBM; SOFTWARE, S. *Ontology Definition Metamodel (ODM) Proposal*. August 2003. Disponível em: `http://www.omg.org/docs/ad/03-07-02.pdf`. Acesso em: agosto de 2005.

ISAAC, A.; TRONCY, R. *DOE - The Differential Ontology Editor*. July 2006. Disponível em: `http://homepages.cwi.nl/~troncy/DOE/`. Acesso em: julho de 2006.

JACOBS, I. *About the World Wide Web Consortium (W3C)*. 2005. Disponível em: `http://www.w3.org/Consortium/Overview`. Acesso em: dezembro de 2005.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. *The Unified Software Development Process*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

JUDE. *Java and UML Developers Environment*. 2006. Disponível em: `http://jude.change-vision.com/jude-web/index.html`. Acesso em: Agosto de 2006.

JUNQUEIRA JR, L. F. *Considerações Básicas sobre a Organização Estrutural e a Fisiologia do Aparelho Cardiovascular*. 2006. Disponível em: `http://www.unb.br/fs/clm/labcor/resfisio.htm`. Acesso em: Setembro de 2006.

KEPLER, F. et al. Classifying Ontologies. In: *Proceedings of the 2^{nd} Workshop on Ontologies and their Applications*. Ribeirão Preto, Brazil: [s.n.], 2006.

KNUBLAUCH, H. Ontology-Driven Software Development in the Context of the Semantic Web: An Example Scenario with Protege/OWL. In: *Proceedings of the International Workshop on the Model-Driven Semantic Web*. Monterey, CA: [s.n.], 2004. Disponível em: `http://www.knublauch.com/publications.html`. Acesso em: setembro de 2005.

KNUBLAUCH, H. et al. *A Semantic Web Primer for Object-Oriented Software Developers*. December 2005. Editors' Draft. Disponível em: `http://www.w3.org/2001/sw/BestPractices/SE/ODSD/20051217/`. Acesso em: janeiro de 2006.

KOGUT, P. et al. UML for Ontology Development. *The knowledge Engineering Review*, v. 17, n. 1, p. 61–64, 2002.

KURTEV, I.; BéZIVIN, J.; AKSIT, M. Technological Spaces: an Initial Appraisal. In: *CoopIS'02: Proceedings of the Confederated Internacional Conferences - DOA*. Irvine, CA, USA: Industrial track, 2002.

LOPES, J. G. R. C. *Matching Semântico de Recursos Computacionais em Ambientes de Grade com Múltiplas Ontologias*. Dissertação (Mestrado) — Departamento de Ciência da Computação, Universidade de Brasília, Brasília - DF, 2005.

LóPEZ, F. Overview of the Methodologies for Building Ontologies. In: *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5)*. Stockholm, Sweden: [s.n.], 1999. Disponível em: <citeseer.ist.psu.edu/lpez99overview.html>.

MANCHESTER, U. of. *OilEd*. July 2006. Disponível em: `http://oiled.man.ac.uk/`. Acesso em: julho de 2006.

MCGUINNESS, D. L.; HARMELEN, F. van. *OWL Web Ontology Language Overview*. February 2004. Editors' Draft W3C. Disponível em: `http://www.w3.org/tr/2004/REC-owl-features-20040210/`. Acesso em: dezembro de 2005.

MILLER, E. et al. *Semantic Web*. 2005. Disponível em: `http://www.w3.org/2001/sw/`. Acesso em: dezembro de 2005.

MIZOGUCHI, R. Tutorial on Ontological Engineering - Part 2: Ontology Development, Tools and Languages. *New Generation Computing OhmSha & Springer*, v. 22, n. 1, p. 61–96, 2005. `http://www.ei.sanken.osaka-u.ac.jp/japanese/tutorial-j.html`.

NARDI, D.; BRACHMAN, R. J. An Introduction to Description Logics. In: BAADER, F. et al. (Ed.). *The Description Logic Handbook*. [S.l.]: Cambridge University Press, 2002. p. 5–44.

NIEMANN, B. et al. *Introducing Semantic Technologies and the Vision of the Semantic Web*. February 2005. Semantic Interoperability (XML Web Services) Community of Practice (SICoP). Disponível em: `http://web-services.gov/`. Acesso em: março de 2005.

NOY, N. F.; MCGUINNESS, D. L. *Ontology Development 101: A Guide to Creating Your First Ontology*. [S.l.], March 2001.

OMG, O. M. G. *Ontology Definition Metamodel - Request for Proposal*. 2003. OMG Document. Disponível em: `http://www.omg.org/cgi-bin/doc?ad/2003-03-40`.

OMG, O. M. G. *UML 2.0 Infrastructure Final Adopted Specification*. October 2004. Disponível em: `http://www.omg.org/cgi-bin/doc?ptc/2003-09-15`. Acesso em: julho de 2005.

OMG, O. M. G. *Introduction to OMG's Unified Modeling Language*. July 2005. Disponível em: `http://www.omg.org/gettingstarted/what_is_uml.htm`. Acesso em: julho de 2005.

ONTOPRISE. *OntoStudio*. July 2006. Disponível em: http://www.ontoprise.de. Acesso em: julho de 2006.

POOLE, J. D. Model-Driven Architecture: Vision, Standards And Emerging Technologies. In: *ECOOP 2001: Proceedings of the Workshop on Metamodeling and Adaptive Object Models*. [S.l.: s.n.], 2001. Disponível em: http://www.omg.org/mda/mda_files/Model-Driven_Architecture.pdf. Acesso em: agosto de 2005.

PRESSMAN, R. S. *Engenharia de Software*. Rio de Janeiro, RJ: McGraw-Hill, 2002.

REILLY, T. O. What is web 2.0? *Commun. ACM*, ACM Press, v. 45, n. 2, p. 61–65, 2002.

ROCHA, B. *O Corpo Humano*. October 2006. Disponível em: http://www.corpohumano.hpg.ig.com.br/circulacao/coracao/coracao_2.html.

Sandpiper Software. *Visual Ontology Modeler*. 2006. Disponível em: http://www.sandsoft.com/products.html.

SELIC, B. UML 2: A Model-driven Development Tool. *Journal of Object Technology*, v. 45, n. 3, 2006. Disponível em: http://www.ibm.com. Acesso em: julho de 2006.

SILVA, E. V.; SANTOS, E. S. dos. SISPROJ - Sistema de Gerenciamento de Tarefas e Projetos. Undergraduate Work at University Catholic of Brasília. December 2004.

SMITH, M. K.; WELTY, C.; MCGUINNESS, D. L. *OWL Web Ontology Language Guide*. February 2004. W3C Recommendation. Disponível em: http://www.w3.org/TR/owl-guide. Acesso em: agosto de 2006.

SOFTWARE, S.; LABORATORY, S. U. K. S. *UML for Knowledge Representation. A Layered, Component-Based Approach to Ontology Development*. March 2003. Disponível em: http://www.omg.org/docs/ad/03-08-06.pdf.

STANFORD, U. of. *The Chimaera Ontology Environment*. 2006. Disponível em: http://www.ksl.stanford.edu/software/chimaera/. Acesso em: julho de 2006.

STEVENS, R. et al. Using OWL to Model Biological Knowledge. *IJHCS: International Journal of Human and Computer Studies*, IN PRESS.

SU, X.; ILEBREKKE, L. A Comparative Study of Ontology Languages and Tools. *Lecture Notes In Computer Science*, p. 761–765, 2002. Disponível em: http://www.idi.ntnu.no/~xiaomeng/paper/caise02WorkshopCRC.pdf. Acesso em: julho de 2005.

TETLOW, P. et al. *Ontology Driven Architectures and Potential Uses of the Semantic Web in Software Engineering*. May 2005. Editors' Draft W3C. Disponível em: `http://www.w3.org/2001/sw/BestPractices/SE/ODA/`. Acesso em: junho de 2005.

VOLZ, R. *OWL for Practitioners*. April 2004. 1^{st} F2F Meetings of the SDK Cluster Working Group on Ontologies. Disponível em: `http://www.sdk-cluster.org/presentations/2004-04-06\%20OWL\%20for\%20practitioners2.ppt`. Acesso em: agosto de 2006.

W3C. *DAML+OIL (March 2001) Reference Description*. 2001. Disponível em: `http://www.w3.org/TR/daml+oil-reference`. Acesso em: dezembro de 2005.

W3C. *World Wide Web Consortium*. 2005. Disponível em: `http://www.w3.org`. Acesso em: novembro de 2005.