UNIFACS
UNIVERSIDADE SALVADOR
LAUREATE INTERNATIONAL UNIVERSITIES

PROGRAMA DE PÓS-GRADUAÇÃO
MESTRADO ACADÊMICO EM SISTEMAS E COMPUTAÇÃO

GEORGE PACHECO PINTO

MDAONTO: UMA ABORDAGEM MDA BASEADA EM ONTOLOGIA

Salvador
2010

GEORGE PACHECO PINTO

MDAONTO: UMA ABORDAGEM MDA BASEADA EM ONTOLOGIA

Dissertação apresentada ao Programa de Pós-Graduação em Sistemas e Computação da Universidade Salvador – UNIFACS, como requisito parcial para obtenção do grau de Mestre.

Orientadora: Profª. Drª. Laís do Nascimento Salvador
Co-orientador: Prof. Dr. Ricardo José Rocha Amorim

Salvador
2010

FICHA CATALOGRÁFICA
Elaborada pelo Sistema de Bibliotecas da UNIFACS Universidade Salvador,
Laureate Internacional Universities

Pinto, George Pacheco

MDAONTO: uma abordagem MDA baseada em ontologia/ George Pacheco Pinto.- Salvador, 2010.

95. : il.

Dissertação (Mestrado) - Universidade Salvador – UNIFACS. Mestrado em Sistemas de Computação, 2010.
Orientador: Profª. Drª. Laís do Nascimento Salvador.
Co-Orientador: Prof. Dr. Ricardo José Rocha Amorim.

1. Desenvolvimento de software. 2. Arquitetura Dirigida a Modelo (MDA). 2. Ontologias. I. Salvador, Laís do Nascimento, orient. II. Amorim, Ricardo José Rocha, co-orient. III. Título.

CDD: 004.6

TERMO DE APROVAÇÃO

GEORGE PACHECO PINTO

MDAONTO: UMA ABORDAGEM MDA BASEADA EM ONTOLOGIA

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre, Universidade Salvador - UNIFACS, pela seguinte banca examinadora:

Laís do Nascimento Salvador – Orientadora _____
Doutora em Engenharia da Computação pela Universidade São Paulo (USP)
Universidade Federal da Bahia – UFBA

José Maria Nazar David _____
Doutor em Engenharia de Sistemas e Computação pela Universidade Federal do Rio de Janeiro (UFRJ)
Universidade Salvador – UNIFACS

Rita Suzana Pitangueira Maciel _____
Doutora em Ciências da Computação pela Universidade Federal de Pernambuco (UFPE)
Universidade Federal da Bahia – UFBA

Salvador, 29 de outubro de 2010

Dedico esse trabalho aos meus pais – Geraldo e Carmen – Amor eterno!

AGRADECIMENTOS

Primeiramente a Deus por mais uma graça alcançada. Pela força e persistência que me fizeram prosseguir e chegar até aqui. Agradeço muito pela família que tenho e por viver rodeado de pessoas que sempre estão dispostos a ajudar.

Aos meus pais Carmen e Geraldo pelo amor incondicional e pela confiança na minha capacidade. Por sempre estarem presentes; incentivando, apoiando todas as minhas decisões. Por serem a base para meu sucesso e meu espelho para vida.

Aos meus irmãos Rogério e Rodrigo sempre presentes, nas horas boas e nas ruins, nas brigas e nas brincadeiras.

As minhas avós Valdelice e Lourdes pelos ensinamentos e valores que escola alguma poderia ter me dado.

A minha orientadora Prof[a]. Laís Salvador pela paciência e ajuda na construção dessa dissertação. Por acreditar e insistir que seria capaz em ter êxito nesse processo.

Ao meu co-orientador Prof. Ricardo Amorim pelas conversas e dicas que iluminaram meus pensamentos nos momentos mais críticos.

A Capes pelo incentivo financeiro necessário para a realização desse trabalho.

A minha amiga Elisângela Rêgo, pelas leituras, dicas, críticas feitas ao trabalho, mas principalmente por sua amizade e sua paciência em me ouvir nas horas mais estressantes.

Aos amigos do Nuperc (Herbert, Cayo, Bruno, Marlo, Bira, Ana Paula, Aline). Galera boa. Longos papos e risadas.

A Taluna e Prof. André Santanchè pelas discussões e troca de ideias sobre a concepção do projeto.

Aos professores Pierre Yves Schobbens e Christina von Flach Garcia Chavez por todo apoio durante a concretização das ideias para o projeto.

Enfim a todos aqueles que direta ou indiretamente contribuíram para realização desse trabalho.

RESUMO

A busca por melhorias no desenvolvimento de software tem impulsionado o surgimento de novas tecnologias, paradigmas, abordagens que elevem a qualidade e a confiabilidade do produto de software. Dessa forma, o desenvolvimento de software baseado em modelos (*Model Driven Development* – MDD), mais especificamente o padrão mantido pelo *Object Management Group* (OMG) – *Model Driven Architecture* (MDA) – surge como uma opção que visa a mudança do foco de um desenvolvimento centrado em código para outra visão centrada em modelos, objetivando-se alcançar níveis maiores de abstração e consequentemente impulsionar o aumento do reuso, produtividade e qualidade do software gerado. Paralelo a esse movimento, a Web Semântica tem tomado corpo e alavancado consigo o uso e criação de ontologias, que representa um mecanismo eficiente de compartilhamento e reuso de conhecimento. Ontologias podem mapear conhecimento consensual sobre um dado domínio. No contexto MDA, um modelo de domínio é conhecido por *Computation Independent Model (*CIM) e apresenta semelhanças conceituais com as ontologias de domínio. Nesse sentido, o presente trabalho tem por objetivo apresentar/estudar uma proposta de interseção entre esses dois mundos, a partir desse ponto de junção (modelo CIM), ou seja, o foco é apresentar/estudar uma abordagem que utilize ontologias como artefato fonte na concepção de modelos de software. Dessa forma, é apresentado *MDAOnto* uma proposta de abordagem MDA, que utiliza ontologias de domínio (escritas em OWL) preexistentes, como fonte inicial na concepção de modelos de software (CIM e posteriormente PIM). Para realização dessa abordagem foi implementada uma ferramenta, cuja concepção também é objeto deste trabalho. Por fim, ainda descreve-se um cenário de uso ilustrativo, apresentando um contexto no qual *MDAOnto* pode se fazer útil.

Palavras-chave: MDA. Ontologia. Poda. CIM. PIM. OWL.

ABSTRACT

The search for improvements in software development has driven the emergence of new technologies, paradigms, approaches that improve quality and reliability of the software product. Thus, Model Driven Development (MDD), specifically Model Driven Architecture (MDA) - a standard maintained by the Object Management Group (OMG) - appears as an option that aims to change the focus of a code-centered development to a model-centered development, aiming to achieve higher levels of abstraction and hence boost the increase of reuse, productivity and quality of the generated software. Parallel to this movement, the Semantic Web has taken shape and brought with it the use and creation of ontologies, which represents an efficient mechanism for sharing and reusing knowledge. Ontologies can map consensual knowledge about a given domain. In a MDA context a domain model is known as Computation Independent Model (CIM) and has conceptual similarities with domain ontologies. In this sense, this work aims to present/examine a proposal for the intersection between these two worlds, from this point of junction (CIM model), i.e., the focus is to present/develop an approach that uses ontologies as source artifacts in the conception of software models. Thus, we present MDAOnto, an MDA approach, a, which uses preexisting domain ontologies (written in OWL), as an initial source in the conception of software models (CIM and PIM later). Finally, this work also describes a tool that implements the proposal, as well as, an illustrative use scenario, presenting a context in which MDAOnto can be useful.

Keywords: MDA. Ontology. Pruning. CIM. PIM. OWL.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta o contexto em que o trabalho está inserido; as justificativas para sua realização; o problema identificado e os objetivos a serem alcançados; a metodologia utilizada e; por fim como está organizado o presente documento.

1.1 CONTEXTO

Desde o surgimento dos computadores, os anseios das pessoas pela facilitação de suas tarefas através de software têm aumentado. Com isso é constante a busca por software cada vez mais complexo e robusto. Frente a esse cenário, a computação tem evoluído, visando acompanhar esse aumento e impulsionando as pesquisas e estudos que levem a métodos mais organizados que possibilitem a criação de soluções mais rapidamente, com maior qualidade e menor custo.

Pensado nisso que, no ano de 1968, em um encontro realizado em Garmisch, Alemanha, foi cunhado o termo Engenharia de Software em resposta à "Crise do Software", visando a criação de uma disciplina similar a outras de engenharia que utilizasse teorias, métodos e ferramentas para auxiliar o processo de desenvolvimento de software (GIBBS, 1994). Desde então, inúmeros esforços são empreendidos na tentativa de aprimorar esse processo.

Assim, a Engenharia de Software tem atuado e buscado transpor, dentre outros, desafios tais como: a tentativa de diminuir os impactos causados por mudanças repentinas tanto relacionadas ao negócio quanto a própria tecnologia, questões de produtividade; interoperabilidade; portabilidade e; documentação consistente (KLEPPE et al, 2003). Notadamente, partes desses esforços têm sido canalizadas na tentativa de aumentar o nível de abstração e reuso alcançados por uma linguagem de programação. Prova disso se tem ao observar os paradigmas de programação existentes, desde os primórdios com a linguagem de máquina até o presente momento com linguagens orientadas a objetos e a aspectos (MELLOR et al, 2004).

Mais um passo no sentido do aumento dos níveis de abstração pode ser visualizado na abordagem de desenvolvimento dirigido por modelos (DDM) que tem como carro chefe a Arquitetura Dirigida por Modelos (MDA) desenvolvida e mantida pelo *Object Management*

Group (OMG). MDA é um exemplo de abordagem, que se baseia na construção de um conjunto de modelos e transformações entre eles, focando o desenvolvimento de software. Sendo assim, o OMG definiu MDA separando seus modelos em camadas diferenciadas pela quantidade de informação de plataforma que elas possuem. Basicamente esses modelos podem ser caracterizados por: *Computation Independent Model* (CIM) representando o sistema independente de detalhes de estrutura e processamento, foca no ambiente e nos requisitos do sistema; *Platform Independent Model* (PIM) representando uma especificação formal da estrutura e função do sistema, suprimindo detalhes técnicos e descrevendo os componentes computacionais e suas interações de forma independente de plataforma e; *Platform Specific Model* (PSM) construído a partir do PIM, através da adição de detalhes específicos da plataforma de destino (OMG, 2003).

Ao mesmo tempo, o *World Wide Web Consortium* (W3C) tem trabalhado em pró da Web Semântica (BERNERS-LEE et al, 2001) e desenvolvido especificações que têm impulsionado o uso e criação de ontologias. Estas que, por sua vez, representam também um tipo de modelo, que para GRUBER (1993a) é "*uma especificação explícita de uma conceituação*". Por conceituação entende-se ser "*objetos, conceitos e outras entidades que se presume existir em uma área de interesse, bem como os relacionamentos entre elas*". Em Noy (2004), uma ontologia é definida como "[*..]. alguma descrição formal de um domínio de discurso, destinado ao compartilhamento entre diferentes aplicações, e expressa em uma linguagem que pode ser usada para o raciocínio*".

Vista tais definições, dada semelhança conceitual, ontologias podem assumir um lugar relevante na criação de modelos de domínio de software, mais especificamente tratando-se de modelos num contexto MDA, representando a camada CIM.

1.2 JUSTIFICATIVA

O presente trabalho foi motivado pela experiência do autor no projeto de pesquisa, "***Uma solução de transformação bidirecional de modelos UML e código baseado na abordagem MDA***", aprovado (2006) no edital para desenvolvimento de soluções inovadoras no campo das Tecnologias da Informação e Comunicação (TIC), financiado pela Fundação de Amparo à Pesquisa do Estado da Bahia (FAPESB). O referido projeto teve por objetivo a concepção de

uma solução para geração de código a partir de especificações em *Unified Modeling Language* (UML) segundo o padrão MDA, com o recurso de engenharia reversa e foco em regras de negócio.

Durante o desenvolvimento da solução percebeu-se que a concepção da camada CIM era estritamente manual e que as ferramentas disponíveis não faziam menção a essa camada, considerando apenas as camadas PIM e PSM. Além disso, raros são os processos de desenvolvimento que contemplam a camada CIM dada dificuldade na sua criação. Considerando a importância de uma modelagem de domínio no processo de desenvolvimento de software e pensando numa possibilidade de automatizar/agilizar a criação de modelos CIM e, consequentemente, dar uma maior importância a esses modelos num contexto MDA, surgiu a ideia de usar ontologias de domínio e, baseado num processo (semi-) automático, apoiar a criação desses modelos.

1.3 PROBLEMA E OBJETIVO

Baseado no que foi dito anteriormente, a concepção da camada CIM era basicamente realizada de forma manual. Dado que essa camada representa o modelo de negócio/domínio do software, sua construção implica em uma análise de requisitos completa, o que consequentemente expõe o processo a todos os problemas inerentes a essa atividade (entendimento incorreto, dificuldade de comunicação, etc.).

Visto tal cenário tem-se um desafio, expresso abaixo:

Seria possível, a partir de uma ontologia de domínio preexistente, chegar-se a modelos de domínio e posteriores modelos independentes de plataforma?

Em busca da resposta para esse questionamento é que surgiu a proposta *MDAOnto*, que culminou em um protótipo de uma ferramenta cujo objetivo é auxiliar o desenvolvimento de modelos de domínio e modelos independentes de plataforma, no contexto MDA (camada CIM e PIM, respectivamente), através da utilização de ontologias de domínio (escritas em OWL) preexistentes.

Sendo assim os objetivos do presente trabalho resumem-se a:

a) Apresentação/estudo de uma proposta que visa integração de MDA com Ontologias na camada CIM;
b) Apresentação de um protótipo desenvolvido baseado na proposta descrita;
c) Descrição de um cenário de uso sobre o qual a proposta pode ser aplicada.

1.4 METODOLOGIA

Esse trabalho foi iniciado a partir dos estudos realizados sobre MDA, bem como as especificações atreladas a esse padrão. Além disso, pesquisas sobre as ferramentas e tecnologias disponíveis também foram feitas com o intuito de se encontrar mecanismos que viabilizassem o projeto.

Em seguida foi realizado um estudo sobre ontologias, focando especificamente no seu uso como facilitador na concepção de modelos de software. Assim, foram levantas informações e encontrados trabalhos correlatos que tratavam a ligação de ontologias com MDA. Além dos trabalhos que se baseavam na poda (exclusão de elementos da ontologia) para a construção de modelos conceituais.

Baseado no levantamento e estudo do estado da arte das tecnologias foi desenvolvida a proposta e definidos os requisitos que seriam implementados no protótipo. Para finalizar criou-se um cenário de uso com o objetivo de vislumbrar um domínio no qual o protótipo se faz útil.

1.5 APRESENTAÇÃO

O restante desse trabalho está organizado em quatro capítulos. O segundo capítulo aborda definições relacionadas a ontologias. Um estudo do referencial teórico foi realizado e levantado os objetivos e benefícios alcançados com seu uso. Além disso, ele trata sobre o uso de ontologias na concepção de modelos de software. Dando maior ênfase à ideia de poda de ontologias, apresentando um algoritmo que realiza coerentemente a referida tarefa.

No capítulo três estão descritos conceitos relativos à Engenharia Dirigida por Modelos (MDE), focando em específico em MDA. Ainda nesse capítulo, também, tratou-se sobre os trabalhos que vislumbravam a possibilidade de fusão de MDA com ontologias.

O capítulo quatro apresenta as características da proposta *MDAOnto*, bem como os detalhes técnicos relativos à implementação do protótipo. Além disso, é descrito um cenário de uso para a proposta.

O quinto capítulo descreve as considerações finais sobre o trabalho realizado e identifica e propõe trabalhos futuros.

2 ONTOLOGIAS

Esse capítulo aborda os conceitos e definições sobre ontologias. Serão descritos os objetivos e benefícios alcançados com sua utilização. Além de um tópico específico que descreve as similaridades de ontologias com modelos de domínio.

2.1 INTRODUÇÃO

Considera-se que pesquisas relacionadas à representação formal do conhecimento data do primeiro milênio A.C., quando do estudo da gramática de *Shastric Sanskrit*, que apresentava uma sintaxe formal e um vocabulário para uma linguagem de propósito geral, além de prover uma análise de sua semântica. Essa gramática é considerada uma das mais remotas sistemáticas de representação do conhecimento.

Entretanto, a representação do conhecimento como se conhece hoje se relaciona aos trabalhos realizados na Grécia antiga, principalmente aos desenvolvidos por Aristóteles (384-322 A.C) na lógica, ciência natural e filosofia metafísica. Especificamente na computação, os primeiros estudos realizados nesse campo focavam basicamente na representação do '*problema*' ao invés da representação do '*conhecimento*'. A ênfase era dada à formulação do problema a ser resolvido, ao invés de formular os recursos disponíveis para o sistema (RUSSEL; NORVIG, 1995).

Esse enfoque dado à representação do problema desencadeou a disseminação de sistemas especialistas que buscavam a representação do conhecimento extraído de um perito e sua formalização numa base de conhecimento. "*Assumia-se que o especialista era uma 'mina' de conhecimento, e que o papel do engenheiro de conhecimento era 'explorar' essa mina*". Tal ideia era por si só falha, já que a utilização apenas do especialista como fonte tornava o processo de extração de conhecimento muito complexo, além de resultar na aquisição de conhecimento superficial (FALBO, 1998).

Nesse mesmo contexto, em 1991, alguns pesquisadores divulgaram um relatório onde descreviam algumas de suas percepções acerca da captura e representação do conhecimento.

Nesse trabalho, relatou-se que, para quase totalidade dos sistemas que eles desenvolviam, uma nova base de conhecimento era construída a partir do zero, muito embora já tivessem desenvolvido diversos sistemas no mesmo domínio. Isso acabava expondo o processo a erros e inconsistências que já poderiam ter sido resolvidos, além de provocar perda de tempo, esforço e consequentemente recursos (NECHES, 1991).

Esse cenário impossibilitava pesquisas e desenvolvimentos devido à ineficiência dos mecanismos de compartilhamento e reuso das bases de conhecimento. A alternativa para '*compartilhar*' e '*reusar*' uma base de conhecimento existente era a adoção por completo do ambiente de programação e representação da base de conhecimento existente (GRUBER, 1991).

Diante desse ambiente, dos custos associados a todo esse processo, surgiu a necessidade do desenvolvimento de uma infraestrutura capaz de preservar, compartilhar e reusar a base de conhecimento existente. Uma infraestrutura que mudasse o foco da tentativa de reproduzir a maneira como os especialistas pensam, para outra, onde a base de conhecimento fosse um produto resultante de uma modelagem. Dessa forma, a modelagem passa a ser o ponto central, e a aquisição de conhecimento um processo construtivo, onde o engenheiro de conhecimento usa todos os tipos de informações disponíveis para concretizar a modelagem. Passou-se a defender a representação do conhecimento, independente ao ponto, que pudesse ser reutilizado em diversas tarefas (GUIZZARDI, 2000).

Desde então, significativos progressos foram alcançados nos métodos de representação e manutenção do conhecimento. Nesse sentido, as ontologias apresentam-se como uma forma particularmente útil de reuso e compartilhamento de conhecimento (CORCHO, 2006).

Esse capítulo tem por objetivo descrever conceitos, benefícios, problemas e a relação de ontologias com modelos de domínio. Inicialmente algumas das definições para o termo serão explicitadas (seção 2.2). Em seguida é feita uma descrição dos tipos/classes de ontologias existentes (seção 2.3). A seção 2.4 descreve OWL, uma linguagem para representação formal de ontologias. Na seção 2.5 é abordado como ontologias podem ser utilizadas como modelo de domínio, bem como sua utilização na construção de novos esquemas conceituais e modelos de domínio. Na seção 2.6 são descritos os benefícios e problemas alcançados a partir do seu uso. E, por fim, algumas considerações sobre o capítulo (seção 2.7).

2.2 DEFINIÇÕES DE ONTOLOGIAS

O termo ontologia surgiu no século XVII como um ramo da Filosofia que tratava da natureza e da organização do ser. Foi introduzido por Aristóteles sendo definido como a "Ciência do Ser" (onto = ser, logia = ciência), a partir da necessidade de se obter respostas para perguntas tais quais: "o que é o ser?", "Quais características comuns a todos os seres?" (MAEDCHE; STAAB, 2001)

No sentido filosófico, uma ontologia é tratada como um sistema particular de categorias, independente de linguagem representacional, que caracteriza certa visão de mundo. Em computação uma ontologia representa um artefato de engenharia constituído por um vocabulário específico, suas definições e axiomas utilizados para criação de novas relações e para restringir as suas interpretações pretendidas (GUARINO, 1998).

Especificamente em relação ao campo da Ciência da Computação, ontologias entraram pela porta aberta na Inteligência Artificial e muito rapidamente ocuparam espaço na vida dos especialistas em análise de domínio. Pesquisas relacionadas ao tema apresentaram um crescimento significativo a partir da década de 1990, principalmente, graças ao surgimento da Web Semântica (RUIZ; HILERA, 2006). Que segundo (BERNERS-LEE, 2001) é considerada uma extensão da Web acrescida de informações que possibilitem um trabalho cooperativo entre as pessoas e os computadores. Nesse contexto, as ontologias assumem um papel significativo provendo um vocabulário controlado de conceitos, explicitamente definidos e com semântica processável pela máquina (MAEDCHE; STAAB, 2001).

Uma das primeiras definições sobre o tema foi dada por (NECHES et al, 1991), tal como segue: "*Uma ontologia define os termos básicos e relações que constituem o vocabulário de uma área, bem como as regras para combinar termos e relações para definir extensões para o vocabulário*".

Considerando essa definição, uma ontologia identifica termos básicos e relações entre eles, identifica regras que combinam termos e provê as definições desses termos e relações. Além

disso, uma ontologia inclui, também, conhecimento que pode ser inferido a partir dos termos e relações explicitamente definidos (GÓMEZ-PÉREZ, 2004).

Tempos depois uma nova definição, amplamente difundida na Ciência da Computação, tratou de caracterizar uma ontologia como sendo "*uma especificação explícita de uma conceituação*". Por "*conceituação*" entende-se serem os objetos, conceitos e outras entidades que podem existir num certo domínio, bem como seus relacionamentos. É uma visão abstrata e simplificada do mundo que se deseja representar. Sendo ela base para a representação formal de conhecimento, já que, toda base de conhecimento, compromete-se com alguma conceituação, seja ela explícita ou implícita. Em relação à outra parte da definição "*especificação explícita*" está relacionada com a representação formal e declarativa dos objetos. Essa representatividade implica na utilização de uma linguagem formal que torne a ontologia legível a computadores (GRUBER, 1993b).

Em 1997, Borst (1997) complementou a definição anterior fornecida por Gruber da seguinte forma: "*Uma ontologia é uma especificação formal de uma conceituação compartilhada*", enfatizando a necessidade de se existir um consenso acerca da conceituação especificada, sob a penalidade de ter sua capacidade de reutilização comprometida.

Guarino (1998), por sua vez, entende uma ontologia como sendo uma especificação parcial e explícita que tenta apenas aproximar-se da representação de mundo definida por uma conceituação. Sendo assim, uma ontologia compromete-se apenas com uma conceituação particular de uma dada realidade e não com a sua completude.

Corcho e outros autores (2006), consideram possível estabelecer um paralelismo entre as definições propostas para o termo ontologia, tanto no sentido filosófico quanto aquelas trazidas pela Ciência da Computação, levando em conta a forma como a realidade é percebida pelas pessoas e pelos computadores. Nesse sentido é possível encontrar três diferenças importantes entre as definições: *i*) para Ciência da Computação, uma ontologia só pode ser considerada uma ontologia se ela for legível a computadores, ou seja, uma ontologia necessita ser codificada numa linguagem interpretável pela máquina; *ii*) do ponto de vista da ciência da computação, uma ontologia é usualmente mais específica que uma ontologia no sentido filosófico e; *iii*) características, tais quais, reusabilidade e compartilhamento tornam-se

essenciais na definição para Ciência da Computação, enquanto no sentido filosófico tais características são irrelevantes.

2.3 TIPOS DE ONTOLOGIAS

Como observado na seção anterior, existem inúmeras e distintas definições para o termo ontologia. Da mesma forma é possível encontrar na literatura diversas classificações/tipificações para o termo, focando em distintos pontos. Nessa seção estão descritas algumas dessas classificações.

Em Guarino (1998) as ontologias são classificadas de acordo com o nível de generalidade, mais especificamente de acordo com o nível de dependência em relação a uma tarefa em particular ou ponto de vista. Dessa forma, quatro tipos foram criados, como pode ser observado na Figura 2.1.

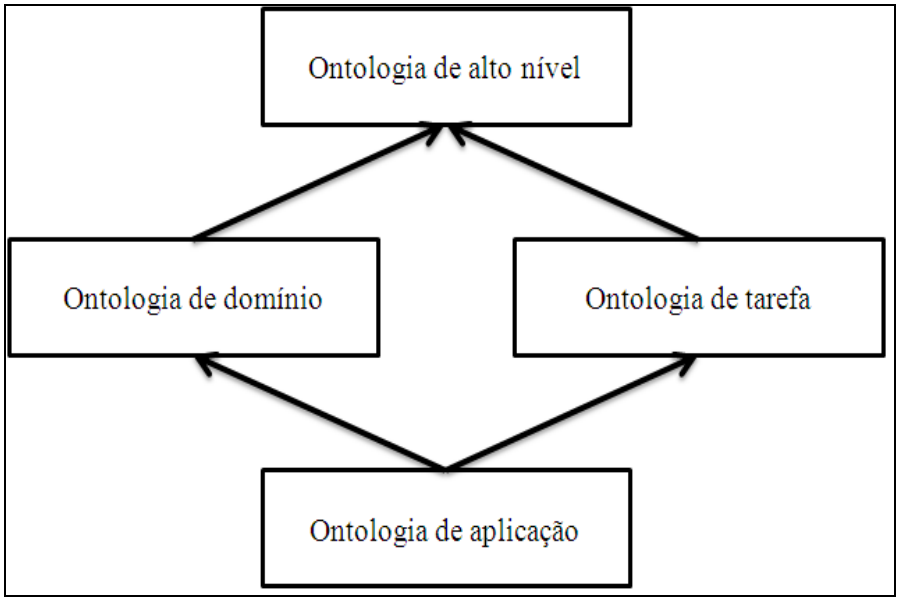

Figura 0.1 - Tipos de ontologias segundo Guarino
Fonte: Guarino (1998).

a) Ontologias de alto nível (*Top-level ontologies*): descrevem conceitos muito gerais, tais como, espaço, tempo, matéria, objeto, etc., que são independentes de um problema ou domínio particular, com o objetivo de unificar critérios entre grandes comunidades de usuários.

b) Ontologias de domínio (*Domain ontologies*): descrevem o vocabulário relacionado a um domínio genérico (Medicina, Direito, etc), por meio da especialização de conceitos definidos nas ontologias de alto nível.
c) Ontologias de tarefas (*Task ontologies*): descrevem o vocabulário relacionado com tarefas ou atividades genéricas, tais como vendas ou diagnose. Também especializa conceitos definidos nas ontologias de alto nível.
d) Ontologias de aplicação (*Application ontologies*): descrevem conceitos dependentes simultaneamente do domínio e da tarefa em questão. Tais conceitos referem-se a papéis desempenhados por entidades do domínio quando da realização de certa atividade.

Fensel (2003) estabeleceu outra tipificação, caracterizando as ontologias como se segue:

a) Ontologias genéricas ou de senso comum (*Generic or common-sense ontologies*): captura conhecimento geral de mundo. Estabelece noções básicas e conceitos para espaço, tempo, estado, etc.
b) Ontologias de representação (*Representational ontologies*): não representam nenhum domínio em particular. No entanto, elas definem conceitos que fundamentam os formalismos de representação de conhecimento.
c) Ontologias de domínio (*Domain Ontologies*): capturam o conhecimento válido para um tipo de domínio particular.
d) Ontologias de métodos e tarefas (*Method and Task Ontologies*): as ontologias de métodos oferecem uma terminologia específica para métodos de resolução de problemas, enquanto as ontologias de tarefas definem termos para especificar tarefas.

Para Ruiz e Hilera (2006) as classificações adotadas por Guarino e Fensel poderiam ser sumarizadas segundo a Figura 2.2, onde as ontologias seriam classificadas pelo seu nível de generalidade, mais especificamente pelo seu nível de dependência em relação ao domínio (dependentes (menos genéricas) e independentes (mais genéricas) de domínio).

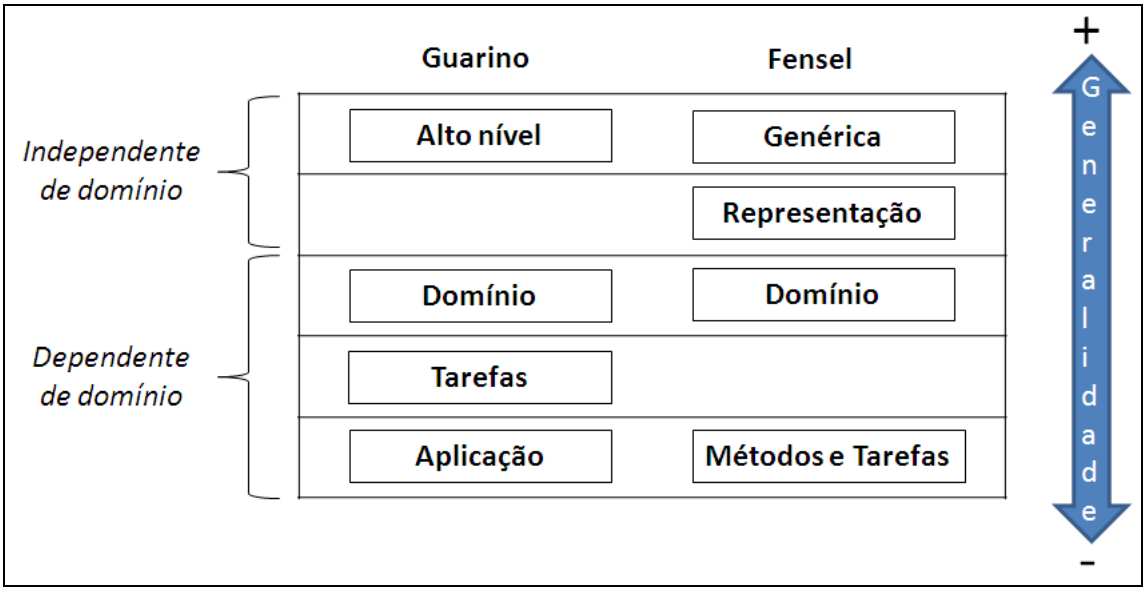

Figura 0.2 - Tipos de ontologias segundo seu nível de generalidade
Fonte: Ruiz e Hilera (2006).

Gómez-Pérez e outros (2004), por sua vez, criaram uma categorização bi dimensional baseada em dois critérios. Eles consideram que a utilização de apenas um critério prejudica uma reflexão adequada da complexidade do problema. Sendo assim, as ontologias podem ser baseadas:

a) Na riqueza de sua estrutura interna:

- Vocabulário controlado (*Controlled vocabularies*): composta por uma lista de termos, tal qual um catálogo.
- Glossário (*Glossaries*): compõe-se de uma lista de termos acrescida de seus significados descritos em uma linguagem natural.
- Tesauros (*Thesauruses*): corresponde a uma lista de termos, porém diferentemente do vocabulário e do glossário, oferece alguma semântica entre os termos, incluindo, por exemplo, sinônimos.
- Hierarquia informal (*Informal hierarchies*): consiste numa hierarquia de termos, porém não corresponde a uma rígida relação de subclasses.
- Hierarquia formal (*Formal hierarchies*): consiste numa hierarquia rígida entre os termos, ou seja, uma rígida relação entre uma classe e sua correspondente superclasse.
- Frames: incluem classes e suas respectivas propriedades, que podem ser herdadas por outras classes hierarquicamente inferiores.

- Ontologias com restrições de valor (*Ontologies with value constraints*): ontologias que permitem restringir valores de suas propriedades, por exemplo, a propriedade *dia* (referente a dias em um mês) não pode ter valor superior a 31.
- Ontologias com restrições lógicas gerais (*Ontologies with generic logical constraints*): refere-se a um tipo de ontologia mais expressivo que permite restrições entre termos de uma ontologia através do uso de lógica de primeira ordem.

b) no objeto da conceituação:

- Ontologias de representação do conhecimento (*Knowledge representation ontologies*): compõem-se de primitivas de representação usada para formalizar conhecimento em um paradigma de representação do conhecimento.
- Ontologias genéricas ou comuns (*Common or generic ontologies*): usadas para representar conhecimento de senso comum, que pode ser reaproveitado entre domínios. Incluem vocabulários relacionados a coisas, eventos, tempo, espaço, etc.
- Ontologias de alto nível (*High-level ontologies*): descrevem conceitos gerais e noções a partir das quais eles podem relacionar-se com os conceitos raiz de todas as ontologias.
- Ontologias de domínio (*Domain ontologies*): reutilizáveis num domínio em particular. Fornece um vocabulário de conceitos de domínio, bem como seus relacionamentos. Os conceitos nesse tipo de ontologias são usualmente especializados de conceitos definidos em uma ontologia de alto nível, assim como seus relacionamentos também o são.
- Ontologias de tarefa (*Task Ontologies*): descrevem o vocabulário relacionado a uma tarefa ou atividade genérica, a partir da especialização dos termos de uma ontologia de alto nível. Provê um vocabulário de termos usados para solucionar problemas que podem ou não pertencer a um mesmo domínio.
- Ontologias de tarefas de domínio (*Domain task ontologies*): ao contrário da ontologia de tarefas, estas são ontologias reusáveis apenas num mesmo domínio.
- Ontologias de métodos (*Method ontologies*): fornece definições de conceitos relevantes e suas relações aplicadas a um processo específico de raciocínio projetado para satisfazer uma tarefa em particular.

- Ontologias de aplicação (*Application ontologies*): contêm todas as definições para modelar o conhecimento necessário de uma aplicação em particular, ou seja, é dependente de aplicação. É possível que elas estendam ou especializem o vocabulário de uma ontologia de domínio ou tarefa com o propósito de usar numa aplicação em particular.

Segundo Ruiz e Hilera, (2006) também é comum se classificar as ontologias pela simplificação do critério da riqueza da estrutura interna. Nesse sentido, as ontologias são classificadas em: *i*) ontologias leves (*lightweight*), que representam basicamente uma taxonomia de conceitos e seus relacionamentos, bem como as propriedades que descrevem tais conceitos e; *ii*) ontologias pesadas (*heavyweight*), apresentam conhecimentos mais aprofundados, além de fornecer maiores restrições sobre a semântica do domínio. Diferenciam-se das ontologias leves a partir da adição de axiomas e restrições que delimitam/esclarecem os significados dos conceitos.

Na Figura 2.3, as ontologias, tal qual classificadas por Gomez-Perez e outros autores (2004), são apresentadas sequencialmente, onde no topo encontram-se as ontologias classificadas como leves: vocabulário controlado, glossário e tesauros. Na base estão classificadas as ontologias pesadas: ontologias com restrições de valor e com restrições lógicas gerais. E centralizado localizam-se as hierarquias informais, hierarquias formais e *frames* que apesar de apresentarem algumas características de ontologias pesadas estas não são consideradas como tal.

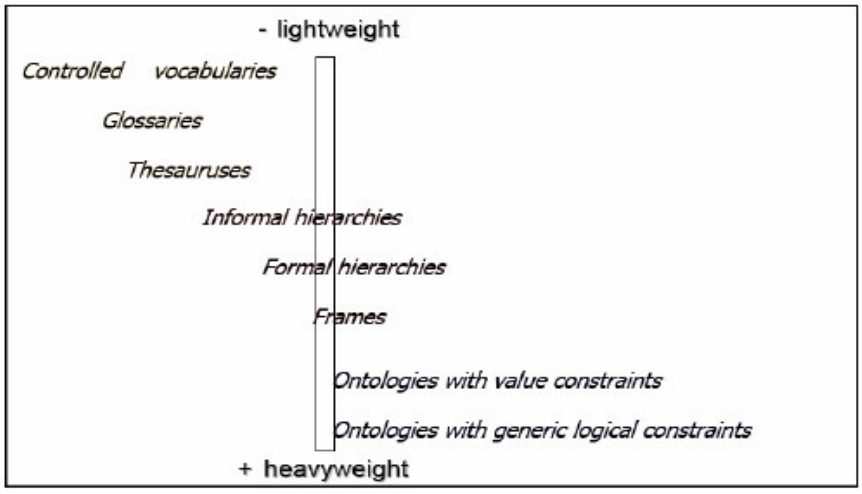

Figura 0.3 - Classificação sequencial das ontologias leves e pesadas.
Fonte: Ruiz e Hilera (2006).

Por fim, Happel e Seedorf (2006) perceberam que mesmo diante de tantas categorizações existentes, basicamente diferenciando as ontologias por níveis de abstração e expressividade, ainda assim, quando se tenta entender como ontologias podem ser aplicadas e quais os benefícios trazidos pelo seu uso na Engenharia de Software tais definições não são tão úteis. Nesse sentido propuseram uma nova classificação que leva em conta duas dimensões: *i*) o papel das ontologias no contexto da Engenharia de Software (o uso em tempo de execução e em tempo de desenvolvimento) e; *ii*) o tipo de conhecimento com qual a ontologia se compromete (domínio e infraestrutura).

A partir dessas duas dimensões chega-se às quatro áreas representas na Figura 2.4 e descritas abaixo:

a) Desenvolvimento Dirigido por Ontologia (*Ontology-driven development* – ODD): inclui o uso de ontologias em tempo de desenvolvimento, descrevendo o domínio do problema. Exemplo disso são as abordagens dentro do contexto de Desenvolvimento Dirigido por Modelos (*Model Driven Develoment* – MDD);
b) Desenvolvimento Permitido por Ontologia (*Ontology-enabled development* – OED): também usa ontologia em tempo de desenvolvimento, mas para auxiliar os desenvolvedores com suas tarefas;
c) Arquitetura baseada em Ontologia (*Ontology-based architectures* – OBA): utiliza ontologias como artefato primário em tempo de execução. Esta representa uma parte central da lógica da aplicação;
d) Arquitetura Permitida por Ontologia (*Ontology-enabled architectures* – OEA): ontologias fornecem uma infra-estrutura de suporte em tempo de execução. Como exemplo tem-se os serviços web semânticos, onde as ontologias são as responsáveis por adicionar o conteúdo semântico.

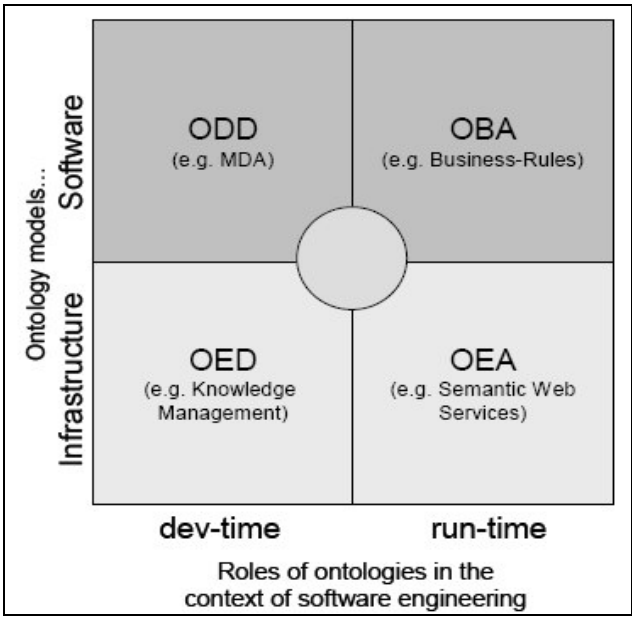

Figura 0.4 - Categorização do uso de ontologias na Engenharia de Software.
Fonte: Ruiz e Hilera (2006).

2.4 OWL – LINGUAGEM PARA REPRESENTAÇÃO DE ONTOLOGIAS

OWL *Web Ontology Language* é uma linguagem sucessora à DAML+OIL, desenvolvida pela W3C como sendo uma especialização de RDFS, que por sua vez é uma extensão de RDF (ANTONIOU; VAN HARMRLEN, 2008). Linguagem utilizada para definição e instanciação de ontologias na Web, que objetiva tornar recursos na Web acessíveis a processos automatizados a partir da adição de informações sobre eles, ou seja, possibilitar que o conteúdo até então criado para consumo humano também seja consumido por máquinas, agentes inteligentes.

OWL é uma linguagem, assim como outras linguagens usadas na Web, que tem sua sintaxe baseada em XML, incluindo um conjunto de elementos e atributos, com significados bem definidos que são usados para descrever termos do domínio e seus relacionamentos numa ontologia (GASEVIC, 2009).

A Web se trata de um ambiente sem restrições, assim, OWL deve prever a hipótese de *open world* considerando que as descrições dos recursos não estão confinadas num único escopo, dessa forma informações definidas em uma ontologia podem ser estendidas por outras (W3C, 2004).

Uma ontologia em OWL é composta basicamente pelos componentes descritos abaixo (W3C, 2004), (ANTONIOU; VAN HARMRLEN, 2008), (OMG, 2009):

Classes: representam conceitos da ontologia. A definição de classes é dada usando o elemento *owl:class*. Toda ontologia escrita em OWL possui pelo menos duas classes conhecidas por *Thing* e *NoThing*.

Propriedades: permitem afirmar fatos gerais sobre os membros das classes e fatos específicos sobre individuais. Uma propriedade é uma relação binária que pode ser classificada em dois tipos**:** *ObjectProperties*, relacionam classes e; *DataTypeProperties*, relacionam classes com tipos de dados. Uma propriedade possui um *domain* e um *range*, caso algum desses valores não tenha sido explicitamente declarado, então a classe *Thing* assume esse papel. Em OWL existem algumas propriedades predefinidas: *i*) propriedades que indicam que diferentes instâncias representam (ou não) o mesmo individual (*sameAs* e *differentForm*); *ii*) propriedades que adicionam informações extra aos elementos da ontologia (*comment*, *label*, *seeAlso* e *isDefinedBy*).

Individuais: podem ser consideradas instâncias de uma classe.

Generalizações: é possível estabelecer relações hieráquicas entre elementos de OWL: *SubPropertyOf*, estabelece relação de hierarquia entre propriedades e; *SubClassOf*, estabelece relação de hierarquia entre classes.

Restrições: OWL possui algumas construções específicas que possibilitam o estabelecimento de restrições: *i*) restrições de cardinalidade sobre propriedades (*FunctionalProperty*, *InverseFunctionalProperty*, *minCardinality*, *maxCardinality* e *cardinality*; *ii*) restrições que delimitam o *range* de propriedades (*allValuesFrom*, *someValuesFrom*); *iii*) restrições que determinam que dois conceitos são equivalentes (*equivalentClass*, *equivalentProperty*, *sameAs*); *iv*) restrições que levam a construção de classes (*intersectionOf*, *unionOf*, *complementOf*, *hasValue*); *v*) restrições de disjunção (*disjointWith*) e; *vi*) outras restrições de propriedades (*TransitiveProperty*, *SymmetricProperty* e *InverseOf*).

2.5 ONTOLOGIAS COMO MODELOS DE DOMÍNIO

Na Engenharia de Software tradicional, um software representa o produto gerado a partir de um conjunto de atividades e resultados, ou seja, processo de software. Em geral, existem quatro atividades fundamentais em processos de software – especificação, desenvolvimento, validação e evolução (SOMMERVILLE, 2007).

Em particular, no desenvolvimento de um novo software, a fase de especificação é normalmente realizada a partir do zero, mesmo embora já se tenha desenvolvido aplicações naquele domínio. Tal circunstância impacta no custo e tempo total do projeto, além de elevar o risco de uma especificação inconsistente. Isso enfatiza a necessidade de se reusar e compartilhar efetivamente o conhecimento de domínio adquirido (GIRARDI; FARIA, 2003).

Nesse contexto entra em cena a Engenharia de Domínio, uma abordagem que objetiva o desenvolvimento de artefatos de software que possam ser reusados. Segundo Arango (1989), ela deve abranger pelo menos três atividades: análise de domínio, especificação da infra-estrutura e implementação da infra-estrutura.

A Análise de Domínio é "*uma tentativa de identificar os objetos, operações e relações entre o que peritos em um determinado domínio percebem como importante*". Em suma, a atividade de análise de domínio responsabiliza-se pela identificação, aquisição e análise de conhecimento de domínio necessário à especificação e construção de software. Essa atividade é considerada equivalente à análise de requisitos convencional, contudo ao invés de explorar requisitos de uma aplicação específica, explora requisitos comuns a uma família de aplicações, ou seja, a um domínio (ARANGO; PRIETO-DIAZ, 1989).

Como produto dessa atividade tem-se um modelo de domínio, considerado um sistema formal de termos e suas relações, regras de composição de termos, regras para raciocínio usando estes termos e regras para mapeamento de itens do domínio do problema para expressões neste modelo e vice-versa. Tal modelo define entidades, operações, eventos e relações que abstraem similaridades em um determinado domínio. Depois de pronto, este modelo deve servir como uma fonte unificada de referência quando ambiguidades surgirem em discussões sobre este domínio, e como um repositório de conhecimento comum, auxiliando de forma

direta a comunicação, o aprendizado e reuso em um nível mais alto de abstração (ARANGO, 1989).

Face à aproximação das definições entre os termos ontologia e modelo de domínio, Falbo e outros (2002) consideram que uma ontologia pode ser usada como modelo de domínio. De tal forma que desenvolveu uma abordagem que usa ontologias de domínio como modelos de domínio e a partir daí deriva objetos.

2.5.1 Construindo modelos conceituais/domínio a partir de ontologias

Além de Falbo e outros (2002), alguns outros autores (OLIVÉ, 2000) e (OMG, 2009) também argumentam a existência de similaridades entre o conceito *ontologia* e os conceitos *esquema conceitual*[1] e *modelo de domínio*. Diante dessas similaridades seria possível construir, a partir de ontologias (refinadas/especializadas), esquemas conceituais e/ou modelos de domínio. Porém, na maioria das vezes, estes são construídos partindo-se do zero, mesmo entendendo-se que o conhecimento, ou partes dele, previamente modelado possa ser reusado e que isso elevaria a qualidade e a produtividade no desenvolvimento.

Nesse contexto, ontologias podem assumir um ou mais dentre três papéis na concepção de esquemas conceituais: i) *bloco de construção*, provendo fragmentos que podem ser reusados pelo esquema conceitual; *ii*) *suporte*, provendo suporte ao projetista no desenvolvimento dos esquemas conceituais e; *iii*) *base*, representa a base sobre a qual o esquema conceitual é derivado, podendo incluir parte ou toda ontologia.

Seguindo nessa linha Conesa e outros (2003) criaram uma abordagem onde ontologias gerais[2] assumem o papel base na concepção de tais esquemas conceituais. Tal abordagem é constituída por três atividades, que são executadas sequencialmente (Figura 2.5), a saber:

a) Refinamento (*Refinement*): normalmente uma ontologia geral (O_G) não incluirá, por completo, o esquema conceitual. Assim, o objetivo dessa atividade é estender

---

[1] Esquema conceitual representa formalmente conhecimentos gerais sobre o domínio da aplicação, ou seja, seus objetos, relações e conceitos (OLIVÉ, 2000).
[2] Conesa e outros (2003) chama de ontologias gerais as ontologias de alto nível, domínio e tarefa.

O_G de forma a contemplá-lo, concebendo O_X. Essa atividade é realizada pelo projetista a partir da análise dos requisitos.

b) Poda (*Pruning*): tem por objetivo excluir elementos da ontologia estendida (O_X) que são irrelevantes para um sistema em particular, concebendo O_P.

c) Refatoração (*Refactoring*): o objetivo é obter um esquema conceitual que seja equivalente a ontologia podada (O_P), mas com uma estrutura melhorada, mais simplificada.

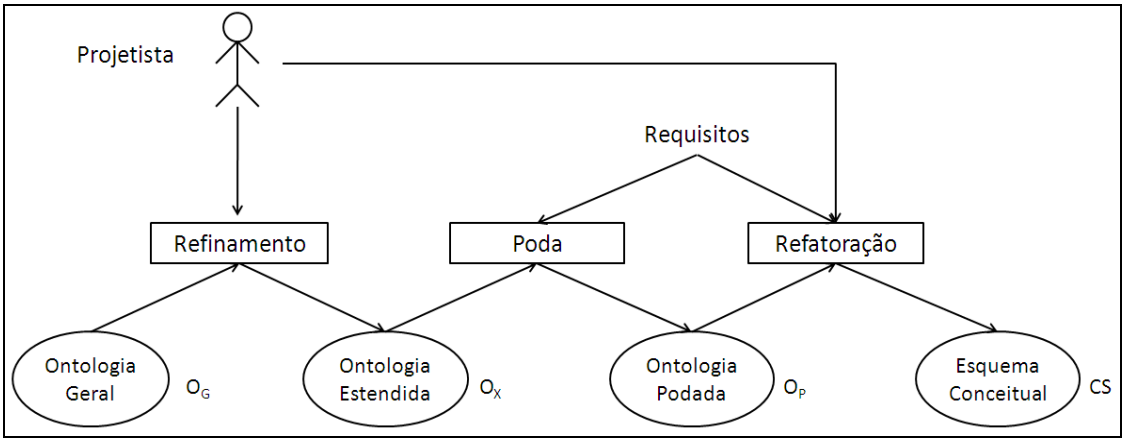

Figura 0.5 - Abordagem para criação de modelos conceituais a partir de ontologias.
Fonte: Conesa e outros (2003).

A atividade de poda é considerada uma das mais importantes nessa abordagem. Em (CONESA; OLIVÉ, 2004) é descrita tal atividade no contexto específico de ontologias escritas em OWL. Por sua complexidade, ela está dividida em duas etapas, como mostra a elipse escurecida na Figura 2.6: *i*) a etapa de Seleção identifica os conceitos da ontologia que são relevantes para a nova ontologia, domínio ou esquema conceitual; e *ii*) a etapa da Poda usa os conceitos escolhidos na etapa anterior para proceder a exclusão dos elementos irrelevantes.

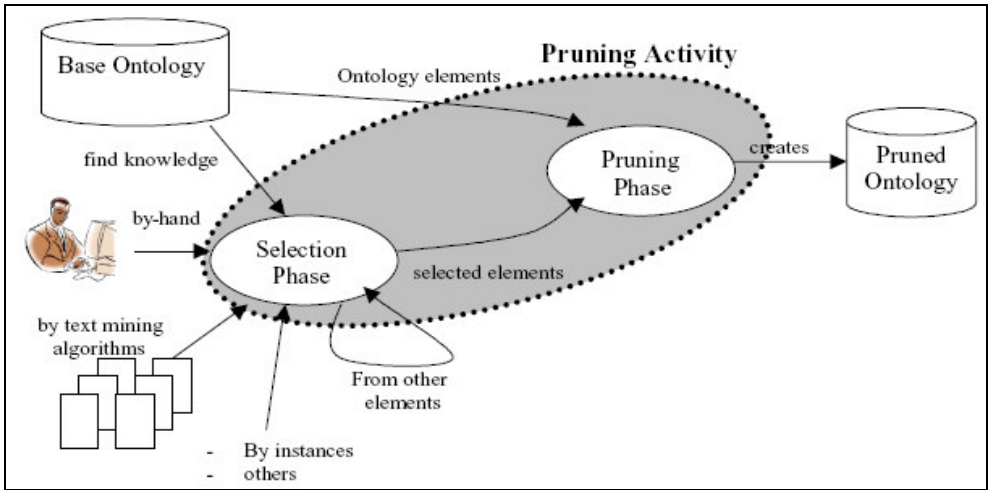

Figura 0.6 - Estrutura da atividade de poda.
Fonte: Conesa e Olivé (2004).

Como dito, a poda só realizar-se-á se estiverem disponíveis os conceitos considerados relevantes, ou seja, aqueles selecionados que deverão estar presentes no produto final dessa atividade. Ao conjunto desses conceitos dá-se o nome de Conceitos de Interesse Direto (*CoI*).

Existem ainda, conceitos que, mesmo não estando em *CoI*, são considerados relevantes, pois generalizam conceitos existentes nele. Assim, ao conjunto formado por *CoI* mais suas generalizações dá-se o nome de *G(CoI)*. Por fim, *CC(O)* forma um conjunto composto por todos os conceitos presentes na ontologia inicial que sofrem alguma restrição.

A partir da seleção dos conceitos relevantes (*CoI*), um algoritmo em quatro passos é aplicado sobre a ontologia inicial (O_0) resultando na ontologia podada (O_P):

Passo 1. Poda de conceitos e restrições irrelevantes, tendo como resultado O_1.

O_0 pode e provavelmente incluirá conceitos irrelevantes para a aplicação. Nesse passo todos os conceitos que não estão presentes em *G(CoI)* serão excluídos. Isso implica na poda de todas as relações de generalização e classificação em que esses conceitos participam. Da mesma forma, podam-se restrições em O_0 que não são relevantes para a ontologia final, ou seja, aquelas que restringem um ou mais conceitos que não estão presentes em *G(CoI)*.

Passo 2. Poda de parentes desnecessários, tendo como resultado O_2.

Os conceitos existentes em O_1 são exatamente *G(CoI)*, entretanto nem todos são necessários em *Op*. Os conceitos realmente necessários (*NeededConcepts*) são definidos a partir da união de *CoI* e dos conceitos afetados pelas restrições restantes. Dessa forma, os demais conceitos

são potencialmente desnecessários, podendo ser excluídos. Assim é possível podar os pais dos *NeededConcepts* que não são filhos de algum outro conceito presente neste mesmo grupo.

Passo 3. Poda de caminhos de generalização desnecessários, resultando em O_3.

Esse passo objetiva a eliminação de caminhos de generalização, entre dois conceitos (C_1 e C_N), desnecessários. Um caminho de generalização entre C_1 e C_N é considerado potencialmente redundante se nenhum dos conceitos intermediários C_2,..., C_{N-1} for membro de *CoI* unido com $CC(O_2)$, ou for pai/filho de outra relação de generalização. Um caminho potencialmente redundante é considerando redundante se existir outro caminho de generalização entre eles. Assim os C_2,..., C_{N-1} e todas as suas relações de generalização são podadas.

Passo 4. Poda de individuais órfãos, tendo como resultado O_P.

Essa etapa é responsável por podar as instâncias da ontologia cujos classificadores (classes e propriedades) já tenham sido excluídos em etapas anteriores.

2.6 ONTOLOGIAS: BENEFÍCIOS E PROBLEMAS

O interesse por ontologias apresenta um crescimento significativo, graças aos benefícios alcançados com sua utilização, principalmente na Web Semântica (BERNERS-LEE et al, 2001). Nesse sentido entende-se, de forma geral, que estas apresentam três primordiais benefícios em seu uso: *i*) esclarece a estrutura do conhecimento, ou seja, esclarece a natureza dos conceitos e termos utilizados na representação do conhecimento; *ii*) reduz a ambiguidade conceitual e terminológica, fornecendo um mecanismo de unificação entre as pessoas com diferentes necessidades e/ou pontos de vistas num contexto particular e; *iii*) permite o compartilhamento de conhecimento, ou seja, a partir de uma análise ontológica obtém uma conceituação que adequadamente expressa e codificada numa ontologia pode ser compartilhada entre pessoas e sistemas que tenham as mesmas necessidades num mesmo domínio.

Mesmo apresentando maior uso por parte da Web Semântica, ontologias podem ser úteis também no desenvolvimento de software em geral. Especificamente em relação à Engenharia de Software, as ontologias de domínio assumem um papel importante quando se trata a questão de desenvolvimento de software reutilizável e de alta qualidade. Elas fornecem uma

terminologia sem ambiguidades que pode ser compartilhada em todo processo de desenvolvimento (RUIZ; HILERA, 2006).

Jasper e Uschold (1999) entendem que ontologias são úteis para melhorar a comunicação entre homens e máquinas. Nesse sentido, os benefícios com seu uso podem ser agrupados em três grupos:

a) Comunicação: possibilita o compartilhamento de conhecimento e facilita a comunicação entre pessoas/sistemas, mesmo quando estes possuem necessidades e visões diferentes. Fornecem um mecanismo de unificação com redução de ambiguidades e aumento da consistência. Nesse caso uma ontologia informal e sem ambiguidades pode ser suficiente.
b) Interoperabilidade: conceito que ganha destaque quando se desejam realizar troca de dados ou tradução entre diferentes métodos de modelagem, paradigmas, linguagens e ferramentas. Ontologias podem ser utilizadas como um mecanismo de intercâmbio, suportando traduções entre diferentes linguagens e representações.
c) Engenharia de software/sistemas: consideram-se os papéis que as ontologias podem desempenhar para suportar o projeto e o desenvolvimento de softwares, a saber:
   - Reusabilidade: ontologias são uma base formal para codificação de entidades importantes, atributos, processos e seus relacionamentos num domínio. Dessa forma elas podem ser um componente de software reusável ou compartilhado, que agrupadas em bibliotecas podem ser reusadas e adaptadas em diferentes classes de problemas e ambientes.
   - Pesquisa: ontologias podem ser usadas como metadados, servindo como um índice dentro de um repositório de informações.
   - Confiabilidade: a partir de uma ontologia formal é possível realizar uma checagem de consistência, seja ela automática ou não, o que resulta em um software mais confiável.
   - Especificação: podem auxiliar no processo de identificação de requisitos e no entendimento das relações entre os componentes de um sistema de software.
   - Manutenção: a realização de manutenção de software exige um esforço significativo no entendimento das suas funcionalidades. O uso de ontologias, tanto no desenvolvimento quanto como parte do software, pode gerar uma redução nos custos de manutenção, dado que elas permitem uma melhora na qualidade da documentação gerada.

- Aquisição de Conhecimento: o processo de desenvolvimento de sistemas pode ter sua velocidade e confiabilidade aumentada, através do uso de ontologias existentes como artefato fonte e guia para a aquisição de conhecimento.

Para Falbo (1998) a principal vantagem na utilização de ontologias é que estas representam um meio prático e objetivo de reutilização e compartilhamento de bases de conhecimento, além de fornecer diretrizes que orientam a aquisição de conhecimento específico a uma aplicação. Sendo assim, o esforço aplicado na aquisição do conhecimento necessário para construção de novas aplicações é reduzido, ao invés de eliminado, já que uma ontologia, por mais completa que seja, não é capaz de prever todo conhecimento necessário a uma aplicação específica. A vantagem advém justamente do seu (re) uso em projetos e desenvolvimento de bases de conhecimento para diferentes aplicações.

Outra vantagem descrita por Falbo (1998) é que as ontologias são úteis na compreensão e interação entre as pessoas. Elas servem como uma forma de unificação e comunicação entre as comunidades, dada capacidade de representação de conhecimento consensual em um domínio.

Por fim, Pisanelli (2002), sumariza os benefícios alcançados com as ontologias em oito tópicos:

a) Semântica clara;
b) Taxonomia explícita;
c) Clara ligação entre conceitos, seus relacionamentos e teorias gerais;
d) Ausência de ambiguidades dentro de um determinado contexto formal;
e) Modularização de contexto;
f) Mínimo de axiomatização para detalhar diferenças entre conceitos irmãos;
g) Boa política de escolha de nomes e;
h) Uma rica documentação.

Mesmo diante das vantagens apresentadas, o uso de ontologias também apresenta alguns problemas. O'Leary (1997) identificou alguns impedimentos acerca do uso de ontologias: *i*) a escolha de uma ontologia é um processo político que requer negociação, já que nenhuma ontologia irá satisfazer ou se adequar a todos envolvidos no processo; *ii*) ontologias não são

estáticas e necessitam evoluir; *iii*) ontologias são, normalmente, estruturadas de maneira precisa, tornando-as vulneráveis a extensões, devido a relação entre complexidade e precisão das definições; *iv*) ontologias, mesmo podendo ser desenvolvidas independentemente, podem ser agrupadas em bibliotecas, possibilitando um uso em conjunto. Entretanto, ontologias desenvolvidas independentemente podem não se integrar com outras, por razões tais quais similaridade de vocabulários e visões conflitantes de mundo.

Falbo (1998) considera como sendo um problema, a falta de consenso com relação a avaliação da qualidade de ontologias, o que leva a uma incerteza da capacidade de uma biblioteca de ontologias elevar a produtividade do desenvolvimento de aplicações.

2.7 CONSIDERAÇÕES

Tendo em vista a necessidade de se representar o conhecimento formalmente, possibilitando que este seja reusado e compartilhado, as ontologias surgiram como uma forma particularmente útil para atingir esses propósitos. Elas representam um meio coerente para compreensão em um dado universo de discurso, fornecendo um meio de universalização e entendimento consensual do conhecimento, facilitando a comunicação entre pessoas/sistemas, mesmo que estes apresentem necessidades e percepções distintas.

Como visto, esse capítulo dedicou-se a tratar sobre importância das ontologias na Ciência da Computação. Foram discutidas suas definições e características comumente aceitas pela comunidade. Foi mostrado, também, que existem diferentes tipificações, cada uma apresentando um ponto específico para classificação, o que representa falta de consenso sobre o tema.

Destacou-se o fato de ontologias, mais especificamente ontologias de domínio, poderem ser comparadas a esquemas conceituais e modelos de domínio, podendo-se utilizá-las como tal. Além disso, elas podem servir como base para criação de novos modelos. Nesse contexto foi apresentada uma abordagem composta por três atividades para geração de esquemas conceituais, destacando-se principalmente, dada sua relevância, a atividade de poda.

Por fim, outro ponto abordado foram os principais benefícios obtidos com sua utilização, principalmente por parte da Engenharia de Software, bem como alguns problemas associados.

3 ARQUITETURA DIRIGIDA A MODELOS (MDA)

Neste capítulo será tratado em nível geral o que é MDA e qual a importância, benefícios e utilidade de seus modelos. Além disso, será abordado como encaixar esse padrão de desenvolvimento com o uso de ontologias.

3.1 INTRODUÇÃO

Para suprir a demanda, cada vez mais crescente por softwares, é necessário que existam tecnologias, metodologias, processos, ferramentas que apoiem e otimizem o desenvolvimento deles. Tais elementos caracterizam objetos de estudo da Engenharia de Software, que tem atuado e buscado encontrar meios eficientes para o desenvolvimento de software.

Nesse sentido MDA (*Model Driven Architecture*) (OMG, 2003) apresenta-se como mais um passo em direção ao aumento dos níveis de abstração. Trata-se de uma especialização de MDE (*Model Driven Engeneering*), cuja principal característica é a mudança de foco de um desenvolvimento centrado em código para um centrado em modelos, expandindo dessa forma os limites de utilidade dos modelos para além de simples documentação. MDA é mantido pelo OMG (*Object Management Group*) e composto por diversos padrões – MOF (*Meta-Object Facility*) (OMG, 2006a), UML (*Unified Modeling Language*) (OMG, 2010), OCL (*Object Constraint Language*) (OMG, 2006b), XMI (*XML Metadata Interchange*) (OMG, 2007) – desse mesmo grupo.

Nas próximas seções serão tratadas sobre o desenvolvimento dirigido por modelos de forma geral, apresentando a teoria que rege essa abordagem (seção 3.2). Em específico será descrita a abordagem MDA juntamente com sua representação de modelos, transformações e benefícios obtidos com a sua adoção (seção 3.3). Finalmente será feita uma descrição de trabalhos que buscam agregar os benefícios de MDA com o desenvolvimento e utilização de ontologias (seção 3.4). Na seção 3.5 encontram-se as considerações finais do capítulo.

3.2 ENGENHARIA DIRIGIDA POR MODELO (MDE)

MDE é uma abordagem concebida com o intuito de lidar com a complexidade de plataformas, apresentando um uso diferente para modelos, que até então eram tratados simplesmente como um artifício de documentação ou meros esboços das ideias de projeto, passando a ser considerados como artefatos de primeira classe. Dessa forma o foco do desenvolvimento de software desvia-se da atividade de codificação para a de modelagem juntamente com a realização de um conjunto de transformações sucessivas entre eles (BEZIVIN, 2005).

Por modelagem entende-se ser o uso de algo simples, seguro e barato para abstrair-se a realidade, de forma que se possa lidar com o mundo de maneira simplificada, evitando-se a sua complexidade, ou seja, abstração da realidade a partir do uso de modelos (ROTHENBERG, 1989).

No contexto da ciência da computação, modelos são abstrações de sistemas que fornecem aos *stakeholders* um entendimento sobre o domínio do problema, suprimindo detalhes irrelevantes. Estes podem ser utilizados como mecanismo facilitador da comunição entre os *stakeholders*, bem como fonte de conhecimento sobre funcionalidades do software durante sua manutenção.

A modelagem pode ser caracterizada de acordo com o grau de sincronismo existente entre os modelos e o seu código representativo. Essa caracterização é exibida na Figura 3.1, na qual cada categoria representa uma forma de utilização de modelos na construção de software, a saber: *code only*: abordagem em que não se tem a presença de modelos definidos, sendo estes informais e intuitivos, e muitas vezes presentes apenas na cabeça do desenvolvedor; *code visualization*: essa abordagem implica a utilização de modelos como forma de visualização e entendimento do código, ou seja, o modelo está fortemente acoplado com o código; *roundtrip engineering*: nesta, tanto modelo quanto código possuem grande importância. Eles coexistem e qualquer alteração em um dos artefatos é refletida no outro; *model-centric*: os modelos são criados com nível de completude suficiente que se possa gerar a implementação completa do sistema. O desenvolvimento de software dirigido por modelos encontra-se nessa categoria; *model only*: os desenvolvedores usam apenas modelos, os quais são fontes de entendimento, base para discussões, comunicação, dentre outros (BROWN et al, 2005).

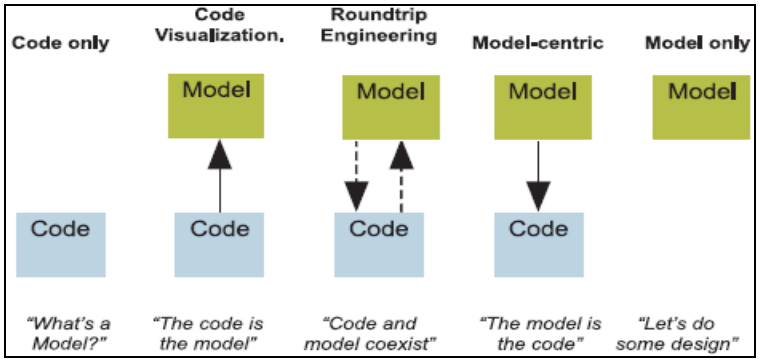

Figura 0.1 - Caracterização da modelagem.
Fonte: Brow (2004).

Enquanto modelos descrevem uma abstração da realidade, metamodelos definem a modelagem propriamente dita, ou seja, um metamodelo representa um modelo de uma linguagem de modelagem. É uma representação da estrutura, semântica e restrições de um grupo de modelos, sendo o responsável por descrever uma determinada plataforma (Java, .Net, Windows, Linux, dentre outras). A utilização de metamodelos agrega o benefício da facilidade de comunicação entre distintos modelos, além de possibilitar a transformação entre eles (MELLOR, 2004).

Abordagens que utilizam o conceito de metamodelos são baseadas numa arquitetura de níveis de modelos e metamodelos, onde a estrutura de modelos em baixo nível é explicada através de metamodelos em alto nível. Como pode ser observado na Figura 3.2 um sistema pode ser "*representado por*" um modelo que por sua vez está em "*conformidade com*" um metamodelo.

O fato de ser "*representado por*" indica que através do uso de um modelo em um nível mais alto de abstração representa-se algo do mundo real. Por outro lado a relação *conformsTo* diz que o modelo gerado está de acordo com alguma especificação, que dá algum tipo de significado aquele modelo (BEZIVIN, 2005).

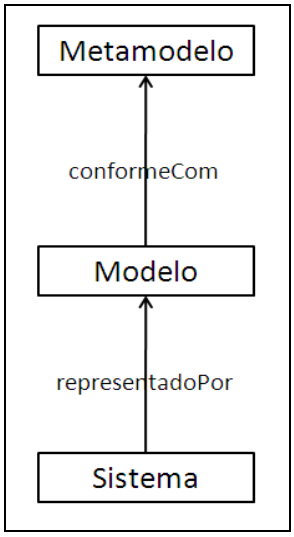

Figura 0.2 - Relações básicas MDE.
Fonte: Bezivin (2005).

3.4 MDA

Em 2001, o OMG, definiu o termo *Model Driven Architecture* (MDA) representando uma abordagem de desenvolvimento de software orientado a modelos, configurando uma versão específica de MDE. Este é formado por diversos outros padrões OMG (por exemplo, MOF, UML, OCL, XMI) (OMG, 2003).

MDA é uma abordagem no qual o código é (semi-) automaticamente gerado a partir de modelos mais abstratos. E para descrição desses modelos, bem como das transformações entre eles, são utilizadas linguagens de especificação padrão (BROWN, 2004).

Ele provê uma estrutura para: *i*) especificar um sistema independente de plataforma; *ii*) especificar plataforma; *iii*) escolher uma plataforma específica para o sistema e; *iv*) transformar a especificação do sistema em uma plataforma específica. Segundo (OMG, 2003), com a adoção e adequação desse padrão esperar-se que três objetivos básicos sejam alcançados: portabilidade, interoperabilidade e reusabilidade.

3.4.1 Modelos em MDA

Assim como em MDE, o ponto chave do padrão MDA também está na utilização de modelos como elemento chave no processo de desenvolvimento de software. MDA separa a especificação das funcionalidades dos detalhes de implementação, definindo para isso uma arquitetura que possibilita a estruturação dessas especificações na forma de modelos. Tais modelos representam a função, estrutura e comportamento do sistema e estão em conformidade com metamodelos (OMG, 2001).

MDA foi constituída a partir de uma arquitetura de metamodelagem em quatro camadas que é exibida na Figura 3.3. Ela baseia-se na arquitetura universal representativa das relações MDE (Figura 3.2), e foi criada com o objetivo de estabelecer uma terminologia padrão de comunicação entre os modelos. Essa separação arquitetural provê um entendimento das relações entre os diversos padrões OMG, facilitando tanto a acomodação de novos padrões de modelagem e metamodelagem, quanto permitindo sua integração e gerenciamento (ATKINSON; KUHNE, 2003).

No nível mais baixo dessa arquitetura encontra-se a camada M0, representando elementos do mundo real. Estes são categorizados através de modelos definidos no nível M1. Em M2 localizam-se os metamodelos que capturam as linguagens de modelagem da camada M1, e que por sua vez está em conformidade com o meta-metamodelo da camada M3, representado no universo MDA pelo MOF. Em M3 são descritas as propriedades que um metamodelo possui. É o nível sobre o qual linguagens de modelagem e metamodelagem operam, possibilitando intercâmbio entre ferramentas (MELLOR, 2004).

MOF é uma linguagem abstrata e que auto se descreve. Ela define um *framework* para especificar, construir e gerenciar metamodelos. É de fundamental importância para definição de qualquer linguagem de modelagem que queira se adequar ao padrão MDA. Todos metamodelos, padrões e customizações que são definidas em MOF localizam-se no nível M2. UML é um desses padrões que representa uma linguagem para especificar, visualizar e documentar software. Os conceitos básicos de UML podem ser estendidos a partir do uso de perfis UML conferindo novas características à linguagem. Outros metamodelos tais como QVT e ATL utilizados para representação de transformações entre modelos; OCL,

representando uma extensão da UML que confere maior precisão aos modelos gerados, também fazem parte do nível M2.

Além dos padrões descritos, MDA está baseada também em XMI, que define mapeamentos de modelos, metamodelos e meta-metamodelos para documentos XML e *schemas* XML, possibilitando melhor troca e compartilhamento deles, já que muitas ferramentas de software suportam o uso de XMI (GASEVIC, 2009).

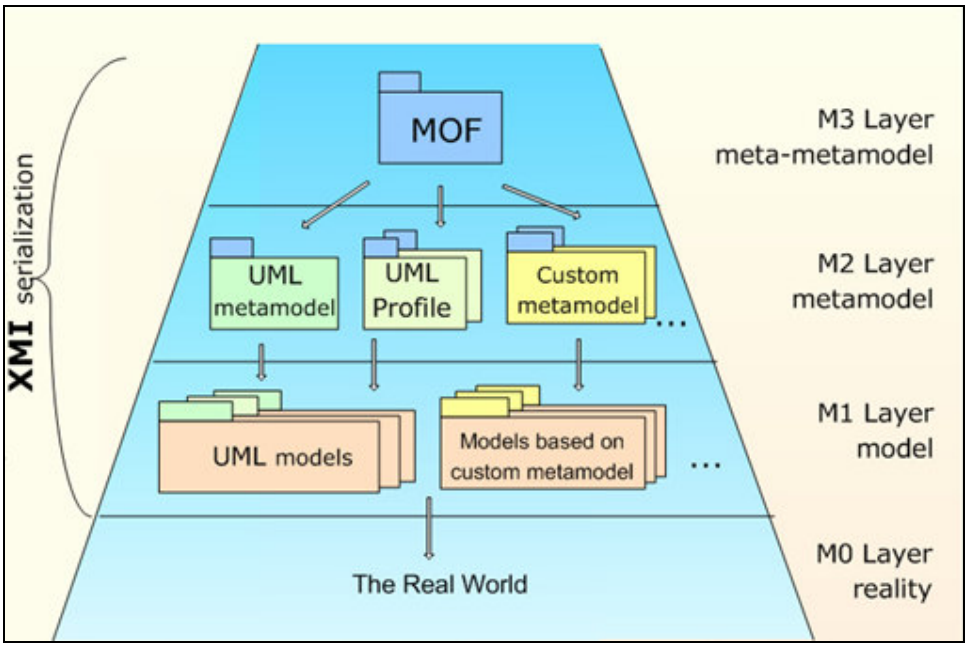

Figura 0.3 - Arquitetura modelagem em quatro camadas.
Fonte: Djuric (2005).

É comum se criar caracterizações de modelos baseadas em critérios de abstração[3], dando origem a modelos que são diferenciados de acordo com algum ponto de vista (v*iewpoint*), focando em requisitos particulares de um sistema (OMG, 2001). O padrão MDA analisa sistemas a partir de três *pontos de vista*, os quais são relacionados a seguir: *i*) *Computation Independent Viewpoint* (Ponto de Vista Independente de Computação), foca no ambiente do sistema e nos requisitos, detalhes de estrutura e processamento são suprimidos; *ii*) *Platform Independent Viewpoint* (Ponto de Vista Independente de Plataforma), foca na operação do sistema, no entanto suprimindo os detalhes de plataformas específicas. Sua concepção

[3] Entenda-se por abstração o ato de suprimir detalhes irrelevantes, com o intuito de se gerar um modelo mais simplificado (OMG, 2003).

pressupõe que parte da especificação do sistema não muda de uma plataforma para outra e; *iii*) *Platform Specific Viewpoint* (Ponto de Vista Específico de Plataforma) adiciona detalhes específicos de plataforma (OMG, 2003).

Brown (2004) expôs que MDA está subordinada a quatro princípios básicos, a saber: *i*) modelos em notação bem definida são a base para o entendimento de sistemas; *ii*) a construção de sistemas pode ser baseada em modelos e nas transformações ocorridas entre eles; *iii*) suporte formal na descrição de modelos através de metamodelos, facilitando a integração e transformação entre eles; *iv*) larga adoção e aceitação dessa abordagem requer padrões industriais.

Para suportar esses princípios e baseando-se nos pontos de vista tratados anteriormente, MDA caracteriza seus modelos de três formas (Figura 3.4):

a) *Computation Independent Model* (CIM) representa uma visão do sistema do ponto de vista Independente de Computação, ou seja, independente de detalhes de estrutura e processamento, focando no ambiente e nos requisitos do sistema, ou seja, descreve conceitos e seus relacionamentos num dado domínio e ajuda a comunicar precisamente o que o sistema deve fazer. Por tal motivo é também conhecido como modelo de negócio ou domínio, que segundo (ARANGO; PRIETO-DIAZ, 1989) pode ser tratado como um sistema formal de termos, relações entre tais termos, regras de composição de termos, regras para raciocínio usando estes termos e regras para mapeamento de itens do domínio do problema para expressões neste modelo. Esse tipo de modelo é o produto do trabalho de especialistas em domínio.

b) *Platform Independent Model* (PIM), que apresenta uma visão do sistema do ponto de vista Independente de Plataforma, provendo uma especificação formal da estrutura e função do sistema, suprimindo detalhes técnicos e descrevendo os componentes computacionais e suas interações de forma independente de plataforma. O PIM pode ser considerado um modelo para ser executado sobre uma máquina virtual independente de tecnologia (GASEVIC, 2009).

c) *Platform Specific Model* (PSM) é uma visão do sistema do ponto de vista Específico de Plataforma formado através do complemento do PIM com informações de uma plataforma específica (OMG, 2003).

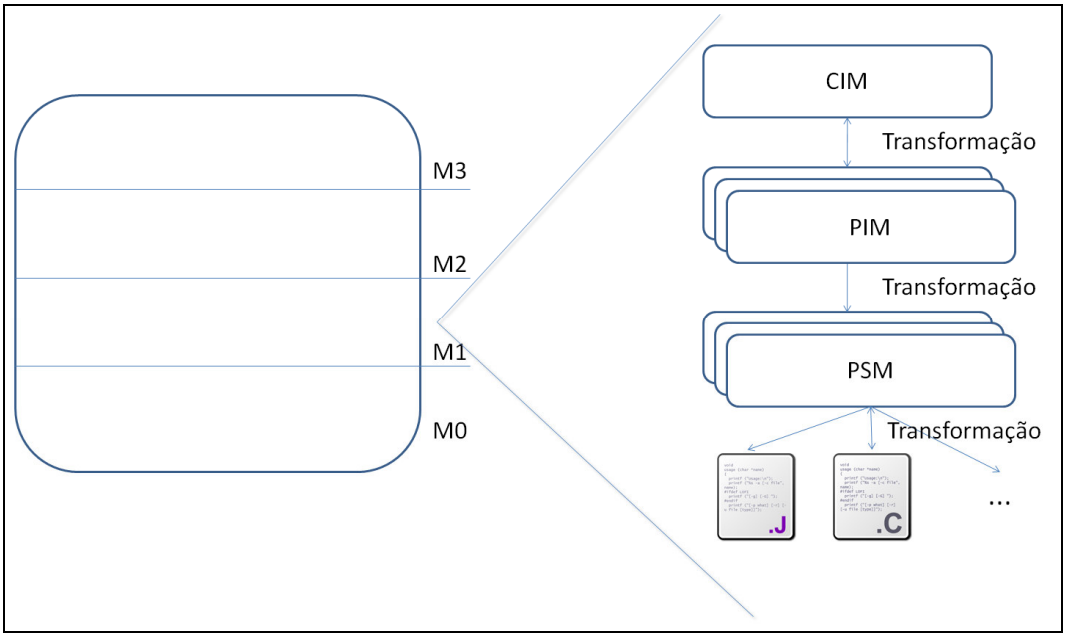

Figura 0.4 - Caracterização de modelos MDA
Nota: Elaboração do autor.

Com essa separação de modelos desvia-se o foco dos desenvolvedores do PSM para o PIM e CIM. E a partir daí, detalhes específicos de plataforma e código são gerados automaticamente através do uso de transformações (Figura 3.5). Configurando esta capacidade como uma das principais vantagens de MDA.

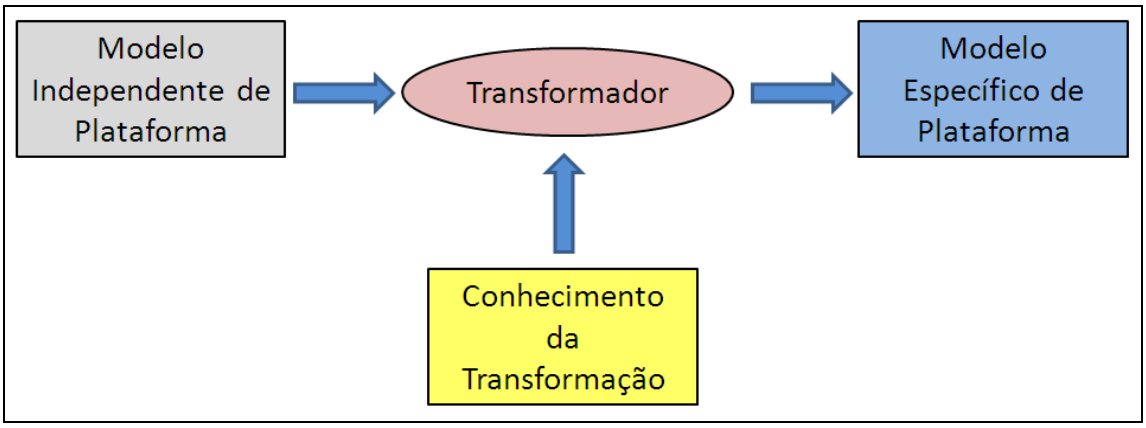

Figura 0.5 - Visão geral transformação PIM para PSM
Fonte: Gasevic (2009).

3.4.2 Mapeamentos e Transformações

Mapeamento é um conjunto de regras e técnicas utilizadas nas transformações entre modelos. MDA prevê dois tipos básicos de mapeamento: *i*) Mapeamento de Tipos de Modelo, o qual especifica um mapeamento entre os elementos descritos por uma linguagem no PIM, para elementos de uma linguagem do PSM. Um exemplo disso é o mapeamento através de metamodelos, onde os elementos do modelo tanto PIM quanto PSM são descritos como metamodelos MOF; *ii*) Mapeamento de Instância de Modelo, descreve um mapeamento pelo qual elementos específicos do PIM devem ser transformados de uma forma particular. Este caracteriza-se pela definição de marcas, as quais representam elementos do PSM e são aplicadas a elementos do PIM, guiando a transformação (OMG, 2003).

O grande desafio, não só de MDA, mas como de qualquer outra abordagem de desenvolvimento de software dirigido por modelos são as transformações. Transformação de modelos é um processo no qual se tem um modelo fonte como entrada e se produz um modelo destino como saída, respeitando as regras de mapeamento. Para a realização dessa tarefa é necessário conhecimento da sintaxe abstrata e semântica dos modelos fonte e destino, que pode ser conseguida usando justamente a metamodelagem.

MDA prevê a existência das seguintes transformações: PIM para PIM, PIM para PSM, PSM para PSM e PSM para PIM (OMG, 2001). Elas podem ser realizadas manualmente, com auxilio do computador ou automaticamente. Dessa forma as ferramentas devem suportar a sua automatização, permitindo não somente realizar transformações previamente definidas, mas também permitir que os usuários definam novas (SENDALL; KOZACZYNSKI, 2003).

A transformação de PIM para PIM tem por objetivo o refinamento do modelo, ou seja, o modelo é melhorado e adequado às necessidades do sistema. As transformações PIM para PSM permitem que a partir de um único PIM sejam gerados vários PSM's (Figura 3.5), ou seja, especializando-se para uma determinada plataforma.

MDA define quatro abordagens de transformação PIM em PSM, a saber (OMG, 2003):

a) *Manual Transformation* (Transformação Manual), essa abordagem não difere muito do que se tem feito hoje com relação a projeto de software. MDA apenas sugere que

exista uma distinção explícita entre PIM e o PSM e que se mantenha um registro da transformação.

b) *Transforming a PIM Prepared Using a Profile* (Transformação do PIM Usando Perfil UML), essa abordagem pressupõe a utilização de um PIM marcado através da utilização de perfis UML que representam uma plataforma específica. Este PIM marcado será submetido a uma transformação, gerando um PSM expresso com outro perfil UML.
c) *Transforming Using Patterns and Markings* (Transformação Usando Padrões e Marcas), os padrões são usados na especificação de mapeamentos, enquanto as marcas são utilizadas para preparar o PIM que posteriormente será transformado em PSM através do uso dos padrões.
d) *Automatic Transformation* (Transformação Automática), em certas situações é possível que se tenha um PIM completo o suficiente que possibilite a geração de código sem a necessidade da existência do PSM.

Por fim as transformações: PSM para PSM relacionada ao refinamento do modelo dependente de plataforma e; PSM para PIM onde se realiza transformação de um modelo em tecnologia particular para um modelo independente de plataforma, possibilitando a manutenção da consistência dos modelos em todos os níveis.

3.4.3 Benefícios Associados à MDA

A utilização de MDA agrega alguns benefícios ao processo de desenvolvimento de software, dentre os quais se destacam (KLEPPE, 2003):

a) Produtividade: O ganho em produtividade ocorre devido ao advento da camada PIM, já que o os desenvolvedores direcionam os seus esforços para a sua construção. Dessa forma a preocupação com detalhes específicos e com o código é transferida para construção de modelos em alto nível, acarretando uma maior atenção para o problema a ser resolvido e consequentemente um software em maior conformidade com os interesses dos usuários. O foco no PIM leva a supressão de informações que são necessárias na camada PSM. Informações que são acrescidas a partir das transformações realizadas entre PIM e PSM. A partir do uso dessas transformações, mesmo com os grandes esforços necessários para sua criação, se tem outro ganho de

produtividade, já que estas são descritas apenas uma vez e reaproveitadas em diversos outros projetos.

b) Interoperabilidade: Tal benefício é conseguido a partir da utilização de *bridges* (pontes), ou seja, uma forma de interligar as informações contidas em PSMs distintos gerados pelo mesmo PIM. Assim, caso seja possível realizar uma transformação de um PIM em outros PSMs, também deverão estar disponíveis as informações para construir essa ponte.

c) Portabilidade: Mais uma vez destaca-se a presença do PIM. Com o foco em sua construção e como por definição ele é independente de plataforma, então tudo que seja construído nele é portável.

d) Documentação e Manutenção: Considerando-se que o foco dos desenvolvedores é desviado para os modelos e para as transformações sucessivas até o código, implica que o modelo construído é a representação fiel do código. Dessa forma tais modelos podem ser considerados uma forma de documentação de alto nível, consequentemente um apoio importante na manutenção. Como se pode observar, a maioria dos benefícios apresentados deve-se ao fato da existência do PIM. Por tal motivo que se considera a sua existência como sendo um dos diferenciais trazidos por MDA.

3.5 MDA X ONTOLOGIAS

Como visto no capítulo anterior, a adoção de ontologias, em particular ontologias de domínio, possibilitam o compartilhamento e o reuso de conhecimento entre inúmeras aplicações num mesmo domínio. Mesmo observada as vantagens no seu uso, técnicas empregadas na construção e utilização de ontologias eram provenientes da Inteligência Artificial, o que de certo modo dificultava sua adoção por parte da comunidade de Engenharia de Software. Tendo em vista esse problema e o anseio por aproximar os profissionais de engenharia com o mundo das ontologias, propostas têm surgido almejando fornecer um aporte ferramental e metodológico que permita a integração dessas duas abordagens (DJURIC, 2003).

Nesse sentido, Cranefield, (2001) desenvolveu uma proposta que visou a utilização de UML para modelagem de ontologias. Nesse trabalho se apresentou semelhanças – classes, relações, herança, etc. – e diferenças entre os conceitos de UML e ontologias. A principal diferença relatada referia-se às propriedades, que em ontologias representam elementos de primeira

classe, enquanto que em UML sua existência depende de uma classe. Basicamente a ideia desenvolvida estava relacionada ao: uso de diagramas de classes para modelar a taxonomia dos conceitos e relações; uso de diagramas de objetos na modelagem das instâncias e; uso de OCL para especificar restrições. Entretanto, o grande problema encontrado diz respeito ao fato dos modelos UML não apresentarem uma definição formal, porém o seu uso não necessariamente se apresenta mais propenso a erros de interpretação que o uso de RDF.

Seguindo na mesma direção, Baclawski (2001) desenvolveram uma abordagem que tinha como pressuposto a extensão do metamodelo UML para abarcar construções específicas à RDF e DAML+OIL (Quadro 3.1). Nesse contexto o trabalho abordou as semelhanças e distinções entre as linguagens e criou, quando possível, um mapeamento entre suas referidas construções. Uma das conclusões desse mesmo trabalho é que definir um mapeamento entre as linguagens que preserve a semântica é praticamente impossível dado suas particularidades, ou seja, linguagens de ontologias são ditas monotônicas, o que significa dizer que, qualquer informação adicionada à ontologia nunca contradiz ou invalida outras previamente definidas, enquanto UML não apresenta tal característica.

DAML Concept	Similar UML Concepts
Ontology	Package
Class	Class
As Sets (union, intersection, etc.)	Not Supported
Hierarchy	Class Generalization Relations
Property	Aspects of Attributes, Associations and Classes
Hierarchy	None for Attributes, limited Generalization for Associations, Class Generalization Relations
Restriction	Constrains Association Ends, including multiplicity and roles. Implicitly as class containing the attribute
Data Types	Data Types
Instances and Values	Object Instances and Attribute Values

Quadro 0.1 - Mapeamento de conceitos em alto nível de DAML para UML
Fonte: Baclawski (2001).

Mesmo considerando a extensão do metamodelo UML uma boa técnica, isso não implica dizer que se trata de uma boa atitude a ser tomada, já que isso torna necessária a criação de ferramentas específicas que abarquem tais modificações. Tal empecilho levou a outra opção: adotar como solução a criação de um perfil UML e dessa forma possibilitar a utilização de ferramentas disponíveis no mercado.

Um fato a ser considerado é que UML é uma linguagem baseada nas características do paradigma da orientação a objetos e como citado anteriormente apresenta limitações em relação ao uso com ontologias. Tais limitações podem ser superadas através da utilização casada com outros padrões e/ou abordagens. Uma opção é o uso da abordagem MDA, que, nos últimos anos, tem ocupado cada vez mais um lugar de destaque na Engenharia de Software, graças a sua capacidade de diminuição da complexidade do software baseada na divisão em diferentes níveis de abstração (GASEVIC, 2009).

MDA possibilita a definição de metamodelos para linguagens, surgindo como uma boa opção para definir uma linguagem de modelagem de ontologias baseadas no MOF. Os esforços despendidos na união desses espaços tecnológicos – OWL/RDF(S) (ontologias) e MOF (MDA) – gerou uma infraestrutura representada na Figura 3.6. Esta se constitui por: ODM (*Ontology Definition Metamodel*), OUP (*Ontology UML Profile*) e mapeamentos bidirecionais entre OWL e ODM; ODM e outros metamodelos; ODM e OUP e; OUP e outros perfis UML (DJURIC, 2005).

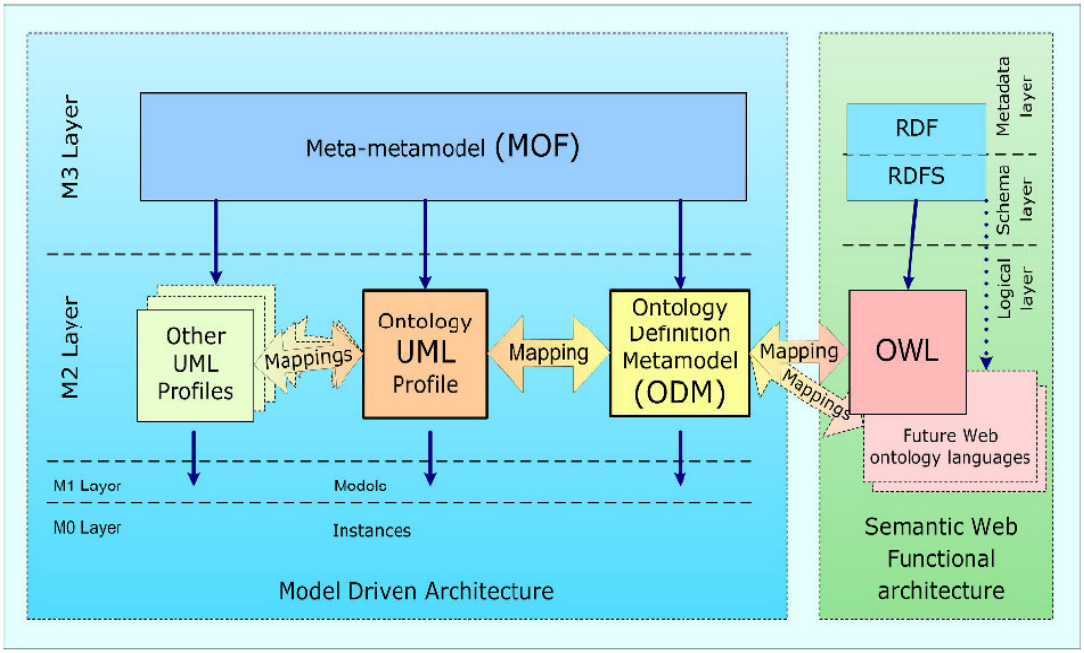

Figura 0.6 - Infraestrutura de modelagem de ontologias no contexto MDA
Fonte: Djuric (2005).

ODM (*Ontology Definition Metamodel*) (OMG, 2009) é um esforço conjunto em resposta a uma RFP – *Request For Proposal* – da OMG, que desponta como uma proposta proeminente de casamento MDA com ontologias. Essa abordagem define um conjunto de metamodelos

baseados no MOF para definição de ontologias dentro do framework MDA (Figura 3.7). ODM está composto por alguns metamodelos, sendo dois principais (OWL e RDFS) que representam linguagens de ontologias. Além disso, um metamodelo para uma linguagem de expressões lógicas (*Common Language* – CL), um para suporte de outros padrões de linguagens de ontologias (*Topic Maps* – TP) e outro para representação de lógica de descrições (*Description Logic* – DL) fazem parte da proposta.

Ainda faz parte da proposta um metamodelo para representar OUP, que adiciona ao ODM a capacidade de modelagem gráfica de ontologias usando diagramas UML. Além de proporcionar o reaproveitamento de ferramentas CASE UML disponíveis no mercado.

Por fim e não menos importante, complementando a proposta definiu-se um conjunto de transformações bidirecionais entre os diferentes metamodelos produzidos e UML. Para isso, tanto os modelos UML quanto os modelos ODM são passíveis de serialização no formato XMI viabilizando a criação de tais transformações (GASEVIC, 2009).

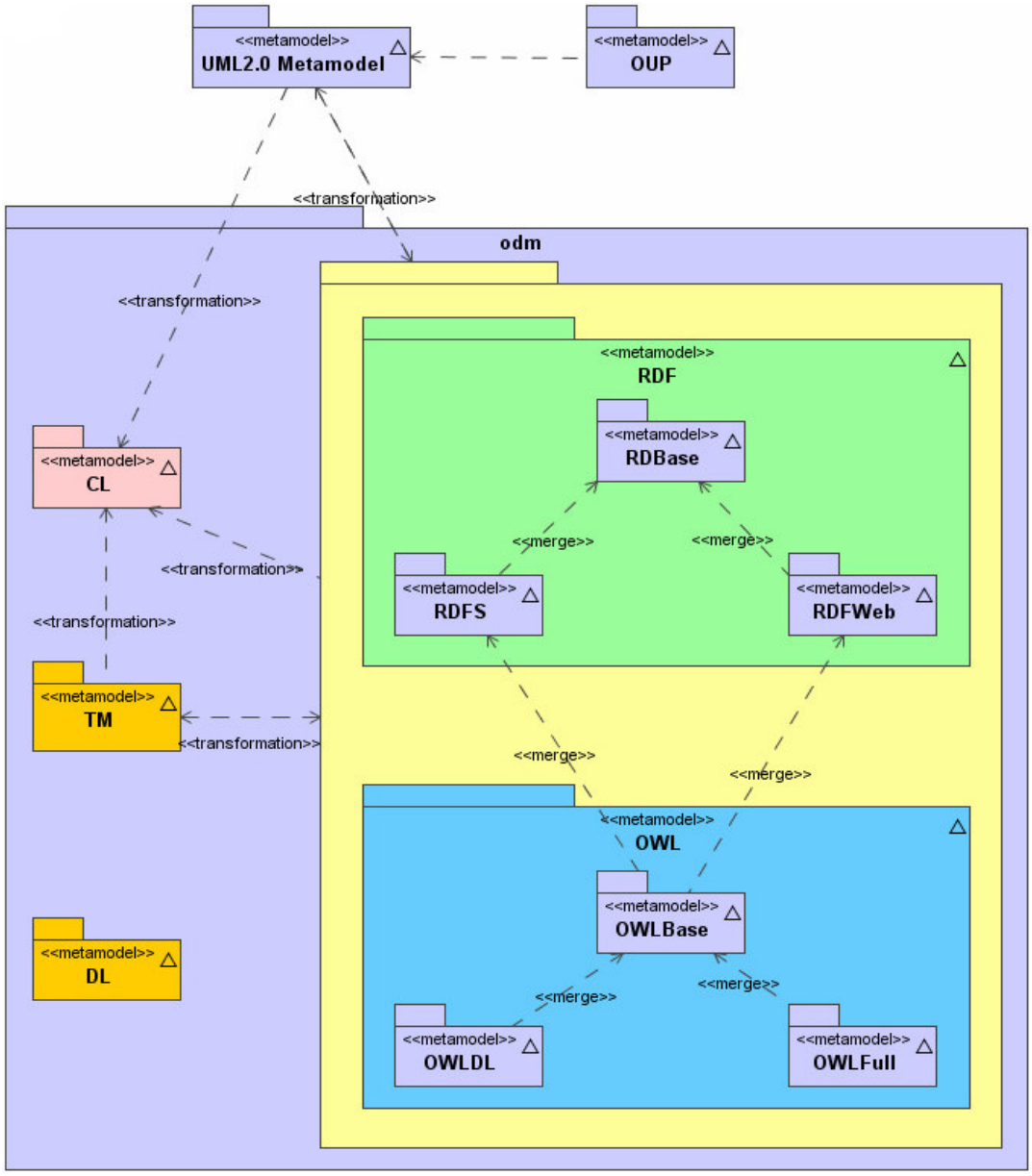

Figura 0.7 - Metamodelos ODM
Fonte: Gasevic (2009).

No mesmo contexto, em seu estudo, Atkinson e Kiko (2008) perceberam que apesar de existirem algumas possibilidades que relacionam MDA com ontologias, nenhuma delas apresentava uma comparação detalhada em termos de seus espaços tecnológicos, mais especificamente, uma comparação entre OWL e UML. Nesse trabalho, então, foi desenvolvida uma comparação e um mapeamento entre essas duas linguagens, sendo que do lado UML foi considerado apenas o diagrama de classes, complementado através do uso da linguagem OCL, possibilitando o alcance de maior precisão e semântica dos diagramas.

Até então, as abordagens descritas, focavam nas linguagens de representação e nos espaços tecnológicos de cada comunidade, diferentemente, Abmann (2006) tratam a junção de MDA e ontologias em termos conceituais. A abordagem visa encontrar um papel específico para o uso de ontologias num contexto MDA.

Dessa forma, a camada CIM apresenta-se como protagonista dessa junção. Ela contém informações sobre o sistema do ponto de vista do usuário (pode representar um modelo de análise), podendo subdividir-se em: *i*) um modelo de domínio, com conceitos e suas relações num dado domínio; *ii*) modelo de negócio, com regras particulares à empresa e; *iii*) requisitos específicos ao software em desenvolvimento.

Para Abmann (2006) as ontologias descrevem conceitos de domínio similares ao CIM. Assim, tanto o modelo de domínio, quanto o de negócio podem ser representados através de ontologias de domínio. Sendo assim, o desenvolvimento dirigido por modelos pode iniciar-se a partir de ontologias de domínio, refiná-las e especializá-las até se atingir modelos de sistemas úteis.

Com isso, Abmann (2006) diz que, considerar-se ontologias como modelos de análise aumenta a confiabilidade do software em desenvolvimento e reduz os riscos atrelados a uma análise de domínio completa, já que elas representam modelos bem construídos e frequentemente usados. Complementando, ontologias fornecem vocabulário comum que facilita a comunicação entre os *stakeholders*, além de mecanismos que favorecem a interoperabilidade entre aplicações. Além disso, uma mesma ontologia pode ser reusada em diversos softwares, num mesmo contexto.

3.6 CONSIDERAÇÕES

Esse capítulo foi dedicado ao desenvolvimento de software dirigido por modelos. Inicialmente a abordagem foi tratada de modo geral, descrevendo-se seus principais conceitos e, em seguida, especificada para MDA, padrão criado e mantido pelo OMG.

Foi visto que MDA versa sobre um padrão que objetiva desviar o foco dos desenvolvedores para construção de modelos, para que com o suporte de transformações automáticas seja gerado código. Dessa forma se consegue aumentar a produtividade, produzir software

interoperável e portável. Tem-se também documentação sempre atualizada, aumento no reuso e na compreensão do software, possibilitando que manutenções e adaptações sejam menos traumáticas.

Complementando o capítulo, foi abordada a ideia do uso de ontologias – como mecanismo de compartilhamento e reuso de conhecimento – na Engenharia de Software. Nesse contexto tratou-se, especificamente, sobre a adoção de ontologias num cenário de desenvolvimento dirigido por modelos. Assim, algumas propostas, que visam a aproximação desses dois mundos foram relatadas, como por exemplo, (ABMANN, 2006), (ATKINSON; KIKO, 2008) e (OMG, 2009).

4 *MDAONTO*: FERRAMENTA MDA BASEADA EM ONTOLOGIAS

Neste capítulo é descrita *MDAOnto*, uma abordagem MDA que se baseia no uso de ontologias de domínio preexistentes, escritas em OWL, como subsídio para concepção inicial de seus modelos. Nas próximas seções serão descritas: o contexto, as características da abordagem proposta; os detalhes técnicos do protótipo construído e; um cenário de uso exemplificando a utilização do protótipo.

4.1 CONTEXTO

Um dos desafios enfrentados pela Engenharia de Software está ligado ao processo de elicitação de requisitos, no qual analistas interagem com diferentes fontes de informação (*stakeholders*, documentação sobre o domínio, aplicações existentes, entre outras) com o intuito de descobrir as funcionalidades necessárias ao Sistema em Desenvolvimento (SD). Trata-se de um processo complexo já que as pessoas envolvidas possuem visões, necessidades e interesses distintos (SOMMERVILLE, 2007).

Visando melhorias no processo de desenvolvimento de software novas tecnologias, paradigmas, abordagens surgem atacando os diferentes pontos vulneráveis nesse processo. Nesse sentido, MDA apresenta-se como exemplo de abordagem, que eleva o desenvolvimento de software à construção de um conjunto de modelos e transformações entre eles. Tais modelos são concebidos com níveis distintos de abstração, seguindo a ideia de separação entre as características operacionais dos sistemas e como estes utilizam, interagem com a plataforma de destino (OMG, 2003).

Paralelo a esse movimento, a Web Semântica tem se desenvolvido e alavancado consigo o uso de ontologias. Estas (como discutido na seção 2.5 deste documento), mesmo sendo usadas frequentemente na Web Semântica, também podem ser inseridas num processo de desenvolvimento de software em geral. De tal forma que tem ocupado lugar de destaque como um elemento fonte na criação de modelos de domínio de software, mais especificamente tratando-se de modelos num contexto MDA (ver seção 3.4) e (REGO, 2007).

Na próxima seção será apresentada a proposta para *MDAOnto*, através da descrição das características da abordagem. Na seção 4.3 são detalhadas informações técnicas acerca da implementação do protótipo. Na seção 4.4 são descritos alguns trabalhos relacionados à presente abordagem. Seção 4.5 apresenta um cenário de uso para a ferramenta usando a ontologia IMS DL (AMORIM, 2006). Por fim, a seção 4.6 trata das considerações sobre o capítulo.

4.2 MDAONTO – A PROPOSTA

*MDAOnto – Ferramenta **MDA** baseada em **Onto**logias* – é o acrônimo escolhido para nomear uma proposta de abordagem cujo objetivo é auxiliar o desenvolvimento de modelos de domínio e modelos independentes de plataforma, no contexto MDA (camada CIM e PIM, respectivamente), através do subsídio de ontologias de domínio (escritas em OWL) preexistentes.

Na Figura 4.1 é exibido um diagrama representativo de *MDAOnto* sob a luz das camadas MDA. A ideia por trás desse trabalho é: apoiado em uma ontologia de domínio preexistente chegar-se a um modelo integrante da camada CIM baseando-se na poda de conceitos irrelevantes ao contexto em que se irá trabalhar. Em seguida essa ontologia podada passa por uma transformação (OWL2UML) gerando um modelo UML, especificamente diagrama de classes, compondo a camada PIM. Nesse ponto o usuário poderá refinar seu modelo, adicionando ou até mesmo eliminando outros conceitos. A partir daí seguem-se as transformações necessárias para se chegar ao PSM e posteriormente ao código. Vale salientar que neste trabalho estão sendo consideradas apenas as funcionalidades delimitadas pela linha tracejada, ou seja, as transformações PIM – PSM e PSM – Códigos são objetos de trabalhos futuros.

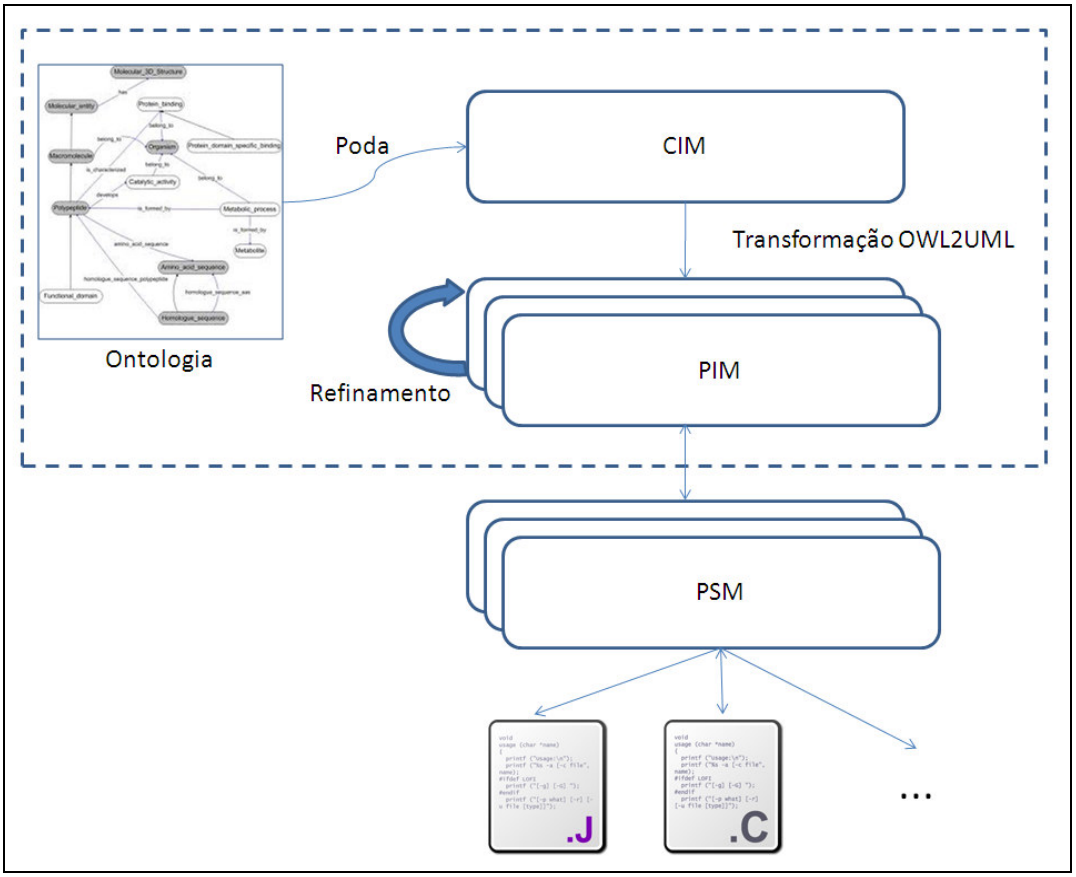

Figura 0.1 - *MDAOnto* sob a visão das camadas MDA
Nota: Elaboração do Autor.

Para satisfazer os objetivos pretendidos, *MDAOnto* compõe-se de quatro atividades, como pode ser observado na Figura 4.2:

1. Seleção de Conceitos Relevantes
2. Poda da Ontologia
3. Transformação OWL2UML
4. Refinamento do Modelo UML

Em cada uma dessas atividades são realizadas tarefas específicas e gerados produtos que servem de subsídio para a realização da atividade seguinte. Cada uma dessas atividades será examinada a seguir.

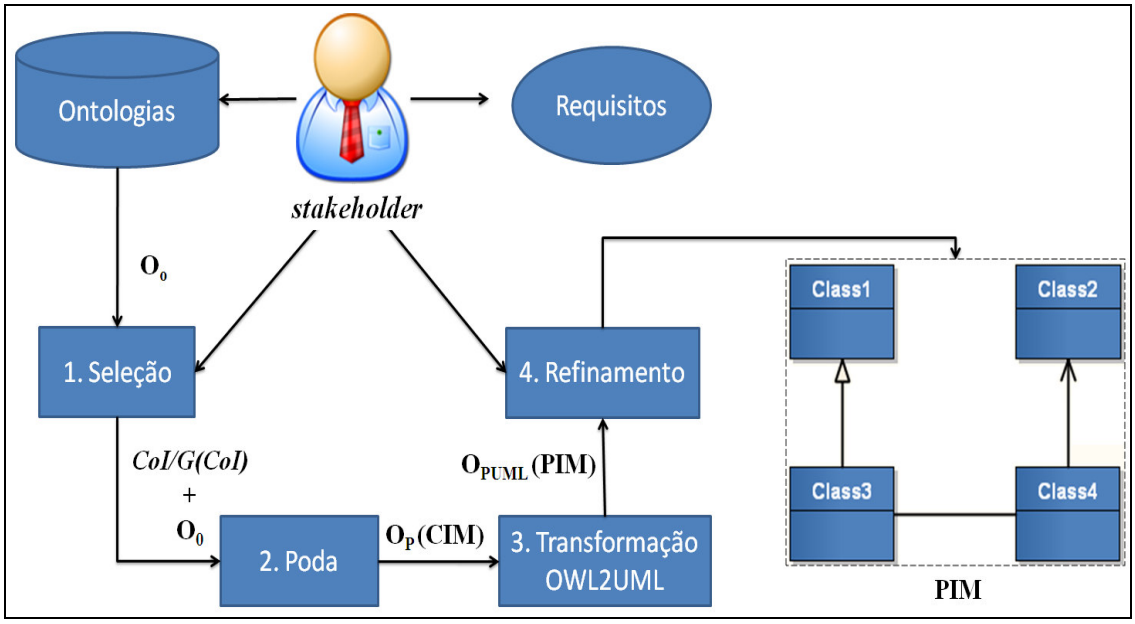

Figura 0.2 - Atividades *MDAOnto*
Nota: Elaboração do Autor.

4.2.1 Seleção de conceitos relevantes

Ontologias podem expressar mais conhecimentos que o necessário para o SD. Deste modo, a mesma ontologia pode ser (re) usada para a criação de diversos softwares dentro de um mesmo domínio.

Considerando tal fato, essa primeira atividade tem por objetivo selecionar, dentre todos os elementos presentes na ontologia, aqueles considerados relevantes, isto é, aqueles cuja presença seja necessária no modelo final do SD. Para tanto, o ato de selecionar os conceitos relevantes está baseada na abordagem de seleção por composição sequencial, que segundo (CONESA; OLIVÉ, 2004), define uma sequência de seleção, na qual a saída de uma etapa torna-se a entrada para uma próxima. Nesse trabalho, parte da seleção é realizada manualmente, outra parte, automaticamente.

Para a realização dessa seleção, assume-se a existência de uma ontologia escrita em OWL relacionada ao domínio do SD. Estas ontologias podem estar disponíveis em repositórios na web ou internos à própria empresa, cabendo ao *stakeholder* importá-las para o projeto do SD.

Após a importação, a ontologia é automaticamente processada por *MDAOnto*, criando-se uma estrutura simplificada, composta por um conjunto de classes e seus respectivos comentários (se existirem). Essa visão fornece ao *stakeholder* uma ideia geral de todos os conceitos modelados na ontologia, bem como um breve esclarecimento (descrição/comentário) sobre os seus significados. A partir daí, ele seleciona apenas as classes que são consideradas fundamentais para o SD.

Nesse ponto é importante destacar que a seleção dos conceitos relevantes ao SD, naturalmente, não é feita ao acaso. Da mesma forma que acontece em uma modelagem de software comum, a seleção está baseada na experiência do *stakeholder* com o domínio e principalmente nos requisitos levantados junto ao cliente ou qualquer outra fonte de consulta/apoio.

Feita a seleção, *MDAOnto* produz dois conjuntos de dados: *i*) o *CoI* – conceitos de interesse direto – e o *ii*) *G(CoI)* – conceitos de interesse e suas generalizações. Em *MDAOnto*, o *CoI* é definido a partir da seleção do *stakeholder* e pela seleção automática de todas as propriedades existentes na ontologia, cujos *domains* e *ranges* estejam presentes na seleção realizada inicialmente. Já o *G(CoI)* é composto pelas classes selecionadas mais as suas generalizações, que são localizadas a partir de um processamento automático.

4.2.2 Poda da ontologia

Para essa atividade tem-se como entrada o produto gerado pela atividade anterior – os conjuntos *CoI* e *G(CoI)*, ver Figura 4.2. O objetivo é tornar a ontologia original (O_0) uma versão reduzida e específica ao SD, a partir da exclusão de elementos irrelevantes, ou seja, elementos que não estejam presentes em nenhum dos dois conjuntos citados anteriormente.

Entretanto, não basta apenas excluir os demais elementos que não foram selecionados e que a priori, o usuário não os consideram relevantes. Pois, estes podem possuir algum tipo de relacionamento indispensável para com esses elementos relevantes (como por exemplo, numa hierarquia), onde sua exclusão pode vir a causar algum tipo de inconsistência ou perda de informação importante. Para tanto, essa atividade está baseada no algoritmo de poda descrito na seção 2.5.1, que elimina os elementos irrelevantes coerentemente.

A partir do *CoI* e *G(CoI)* (Figura 4.2), o algoritmo realiza a poda O_0 resultando na ontologia podada (O_P), passando por ontologias intermediárias (O_1, O_2, O_3). Esse processo é ilustrado na Figura 4.3 através de um diagrama de atividades.

Figura 0.3 - Algoritmo de poda de *MDAOnto*
Nota: Elaboração do Autor.

Em *MDAOnto*, os passos desse algoritmo, está implementado tal qual definido na seção 2.5.1, com exceção do último passo, que diferentemente do algoritmo original, elimina todas e quaisquer instâncias, mesmo que seus classificadores ainda existam em O_3. Essa modificação foi realizada, já que a proposta atual considera apenas diagramas de classes como mecanismo para representação de modelos independentes de plataforma, de tal forma as instâncias não serão necessárias. Tal decisão de projeto está atrelada ao fato de que, *MDAOnto* está alinhada com outras propostas (OMG, 2009) e Atkinson e Kiko (2008) que focam a integração de MDA (UML) com ontologias (OWL), mas só consideram apenas a estrutura estática de UML (representada por diagramas de classes) na comparação/mapeamento dos espaços tecnológicos.

As instâncias, tanto em UML quanto em OWL, podem ser representadas. Entretanto, o propósito e a interpretação diferem entre as linguagens. Enquanto, em OWL é permitido que instâncias sejam indeterminadas, isto é, sem um tipo definido e desse modo classificada por algum um tipo universal, em UML isso não é possível, sendo indispensável que instâncias possuam um classificador concreto definido pelo usuário. Além disso, outro ponto é que UML requer que um nome utilizado para uma instância seja distinto/único, enquanto OWL permite que distintas descrições de instâncias representem uma única instância (representação de sinônimos) (ATKINSON; KIKO, 2008). Devido essas dificuldades, a abordagem inicial de *MDAOnto* considera apenas diagramas de classes UML para representação de modelos independentes de plataforma, excluindo a representação de instâncias.

Ao final dessa atividade tem-se uma ontologia podada, composta apenas pelos elementos relevantes ao SD, ou seja, todos aqueles selecionados pelo *stakeholder* mais aqueles necessários à sua complementação, ao seu entendimento. Dessa forma, a ontologia podada, representa o CIM relativo ao SD.

4.2.3 Transformação OWL2UML

UML é uma linguagem para modelagem de software amplamente difundida tanto na academia quanto na indústria, o que leva a crer que a maioria dos envolvidos no desenvolvimento de software tem domínio/conhecimento sobre ela. Contrapondo-se a esse fato, o mesmo público não apresenta tamanha desenvoltura com ontologias e consequentemente com a linguagem de representação atualmente mais usada, OWL. Além disso, a forma de estruturação do conhecimento numa ontologia difere da forma de representação através de UML. Por exemplo, diferente de um modelo UML, uma propriedade (atributo) numa ontologia representa um elemento de primeira classe, ou seja, sua existência não está condicionada a uma determinada classe.

Levando-se em consideração tal fato, *MDAOnto* prevê a realização da transformação da ontologia podada descrita em OWL para um modelo equivalente em UML e OCL, mais especificamente para um Diagrama de Classes acrescido de restrições escritas em OCL, aproximando o uso de ontologias por parte desse público, já que UML é uma linguagem

difundida e existem inúmeras ferramentas CASE disponíveis no mercado. Além de trazer o conhecimento para um contexto no qual os desenvolvedores tem contato.
Outro fator preponderante para a realização dessa transformação é o simples fato de *MDAOnto* ser parte de uma solução MDA, que baseia-se em transformações sucessivas para especialização dos modelos até se alcançar o código e que tem na UML a sua principal linguagem de modelagem. Nesse contexto, essa atividade representa uma transformação do tipo CIM – PIM.

Essa atividade baseia-se nos trabalhos realizados em (OMG, 2009) e Atkinson e Kiko (2008), os quais definiram regras de mapeamento de OWL para UML e OCL e vice-versa.

4.2.4 Refinamento do Modelo UML

Após a transformação OWL2UML (CIM – PIM), o modelo gerado por *MDAOnto* contém, naturalmente, apenas informações que estavam presentes na ontologia original (O_0). Como dito anteriormente, uma ontologia poderá expressar mais conhecimento que o necessário para o SD. Todavia, a mesma ontologia provavelmente não conterá todo conhecimento necessário ao SD. Por exemplo, uma ontologia pode possuir dezenas de conceitos, o que torna possível que alguns desses conceitos não sejam de interesse ao SD, porém é possível que alguns outros conceitos sejam necessários, mas que, por sua vez, não estão presentes na ontologia. Por isso, o modelo gerado ainda não pode ser considerado completo.

Assim sendo, a atividade de refinamento é a responsável por garantir a completude do modelo PIM. Durante essa atividade, o *stakeholder* irá refinar o modelo produzido na atividade anterior, através da adição das informações capturadas dos requisitos definidos para o SD. Ao final dessa atividade, o modelo PIM estará criado e pronto para prosseguir para próximas etapas num processo MDA.

Essa atividade foi posta como sendo um último passo, focando em um dos objetivos de *MDAOnto*, que é de aproximar *stakeholders*, sem grandes aptidões, do uso de ontologias. Dessa forma é possível que tais *stakeholders* utilizem o conhecimento modelado nas ontologias e complementem o modelo PIM através do uso da UML.

Nesse ponto é importante atentar ao fato que as classes provenientes da ontologia poderão ser rastreadas/identificadas de forma que se possa diferenciá-las das outras que venham ser adicionadas, para isso *MDAOnto* acrescenta a essas classes o atributo URI mantendo uma ligação dela com a ontologia. Com isso, além da adição de semântica ao modelo e posteriormente ao código, através, por exemplo, de anotações semânticas (COSTA, 2010), espera-se possibilitar, também a validação de consistência dos modelos construídos e uma melhor documentação do software, facilitando manutenções futuras.

Outro ponto é que, nesse trabalho, não existe o intuito de modificar (adicionar ou excluir elementos) a ontologia original, dada a complexidade para realização da tarefa. Seria necessário, para uma reflexão das informações, garantir que os fatos adicionados/alterados/excluídos não contradigam outros existentes. Além disso, informações adicionadas, não necessariamente fazem parte de um contexto geral do domínio, podem representar apenas situações específicas a um dado software ou empresa, não sendo, portanto um consenso na comunidade. Para tanto, as modificações geradas nessa atividade não serão propagadas para a mesma, isto é, refletidas na ontologia original.

4.5 MDAONTO – DETALHES TÉCNICOS

Para a abordagem descrita anteriormente foi desenvolvido um protótipo de ferramenta MDA escrito em Java e projetado como um plug-in para o Eclipse. Nesse sentido, todas as características disponíveis no Eclipse (tais quais, *wizards*, editores, perspectivas, etc.) puderam ser reusadas, bem como alguns dos projetos mantidos por ele, em especial o EMF (ECLIPSE, 2010b).

O EMF é um *framework* de modelagem e facilitador para geração de código, que visa à construção de ferramentas e aplicações baseadas na estrutura e modelos de dados do Eclipse. O núcleo desse *framework* inclui: *i*) um metamodelo conhecido por Ecore, que, essencialmente representa uma variante do MOF similar à UML. Ele é utilizado para definição de modelos independentes de plataforma, concentrando-se basicamente nas informações essenciais, definidas através de modelos estruturais, tais como diagramas de classe (BUDINSKY, 2003); *ii*) suporte à serialização de objetos em documentos XMI,

permitindo o intercâmbio entre diferentes ferramentas e aplicações e; *iii*) uma API para manipulação desses objetos.

Nesse sentido, considerando que *MDAOnto* é uma solução baseada no Eclipse e diante das ferramentas e facilidades providas por ela, foi escolhido o metamodelo Ecore como o representante da camada PIM ao invés da criação e adaptação de um metamodelo puramente UML. Para possibilitar a manipulação (criação, edição e manutenção) dos modelos Ecore utiliza-se o componente *Ecore Tools* (ECLIPSE, 2010c) que fornece facilitadores como, por exemplo, editores gráficos.

Outro projeto utilizado na construção do protótipo foi o ATL (ECLIPSE, 2010a), que compreende de linguagem e ferramentas para realização de transformação entre modelos. Ele provê mecanismos para criação de modelos destino a partir de modelos fonte. De forma que, como facilitador do desenvolvimento, *MDAOnto* utiliza a API EMFTriple (EMFTRIPLE, 2010) que tem por objetivo aproximar tecnologias da Web Semântica com o EMF. Uma dessas aproximações diz respeito às transformações de modelos escritos em OWL para Ecore usando ATL.

Além desses projetos, *MDAOnto* também utiliza OWL API (OWL API, 2010) para manipular e serializar ontologias escritas em OWL. Essa API realiza *parser* em arquivos OWL/XML e cria, baseado no metamodelo OWL implementado internamente, uma representação em memória da estrutura da ontologia e viabiliza sua manipulação através do uso de objetos Java. Essa API tem um papel muito importante no protótipo. Com o seu uso foi possível desenvolver com maior facilidade, para *MDAOnto*, um importador e exportador de ontologias escritas em OWL, sem a necessidade de criação de um *parser* OWL próprio. Como pode ser observado no Quadro 4.1.

```
// Importa a ontologia
public OWLOntology loadOntology(URI uri) throws OWLOntologyCreationException {
  ontology = manager.loadOntologyFromPhysicalURI(uri);
  return ontology;
}
// Exporta a ontologia
public void saveOntology (URI uri) throws UnknownOWLOntologyException,
OWLOntologyStorageException{
  manager.saveOntology(ontology, new RDFXMLOntologyFormat(), uri);
}
```

Quadro 0.1 - Importação e exportação de ontologias usando OWL API
Nota: Elaboração do autor.

A arquitetura de *MDAOnto* é formada por quatro componentes como exibido na Figura 4.4. Esses componentes conferem a *MDAOnto* a capacidade de importação de ontologias, seleção de conceitos relevantes, poda dos conceitos desnecessários, transformação OWL para Ecore e posterior refinamento desse modelo através da manipulação visual do diagrama de classes gerado. É importante destacar que estes quatro componentes implementam as atividades descritas na seção anterior.

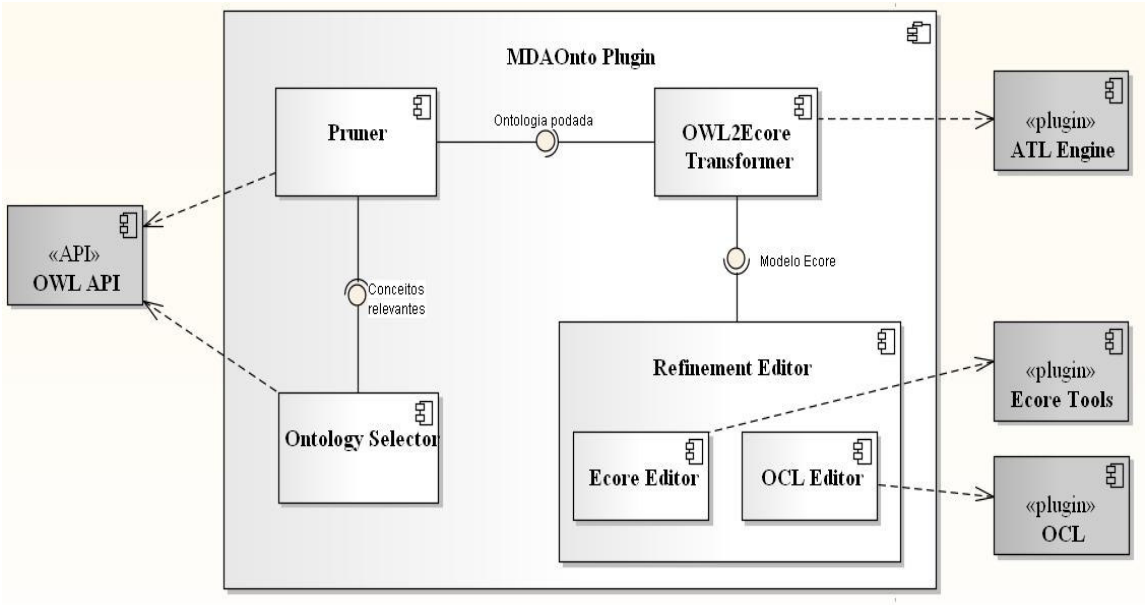

Figura 0.4 - Arquitetura *MDAOnto*
Nota: Elaboração do Autor.

4.3.1 Ontology Selector

Este componente implementa a atividade de seleção. Para isso ele é apoiado pelo protótipo como pode ser observado na Figura 4.5. Nela é exibido o *wizard* para importação de ontologias baseado no *Wizard* e *Wizard Page* padrão do Eclipse. Para sua construção foi necessário criar uma extensão do Wizard padrão do Eclipse – *MDAOnto Wizard* (Figura 4.5b) – conferindo as características necessárias ao protótipo. Características essas representadas através de duas páginas (*Wizard Pages*): *SelectOntologyPage* (Figura 4.5c) criada com a função específica de localização e importação de arquivos .OWL (arquivo XML que representa a ontologia) e; *ShowOntologyClassesPage* (Figura 4.6) exibe, após o processamento da ontologia, uma árvore de *checkboxes* com as classes provenientes da

ontologia, além de uma caixa de texto contendo comentários/descrições (caso existam) acerca de cada classe. Para auxiliar a composição dessa página foi criada a classe *OWLImporterManager* que utiliza-se da OWLAPI para realização do carregamento, *parser* e posterior coleta das classes e comentários presentes na ontologia.

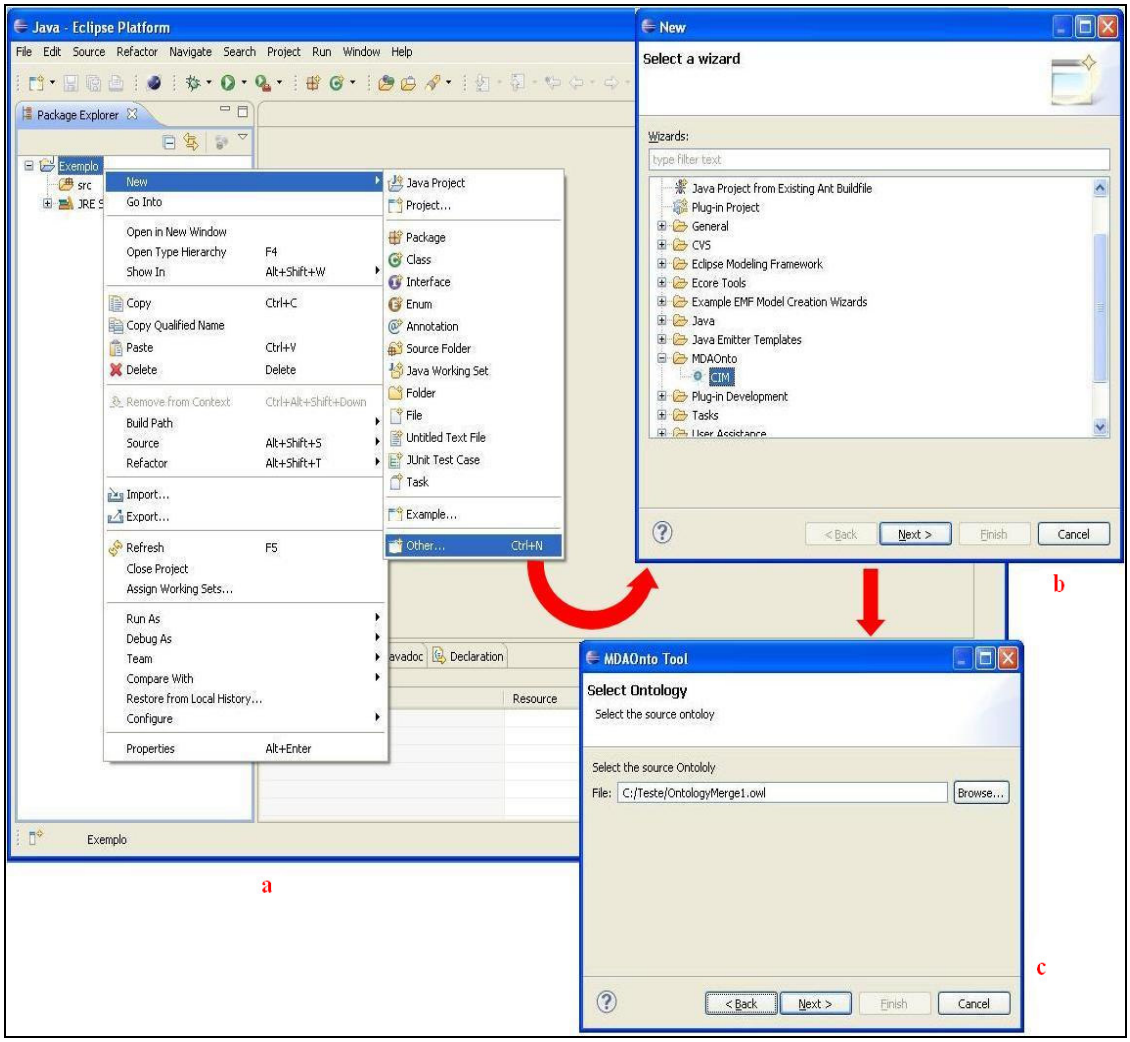

Figura 0.5 - *MDAOnto* Wizard
Nota: Elaboração do Autor.

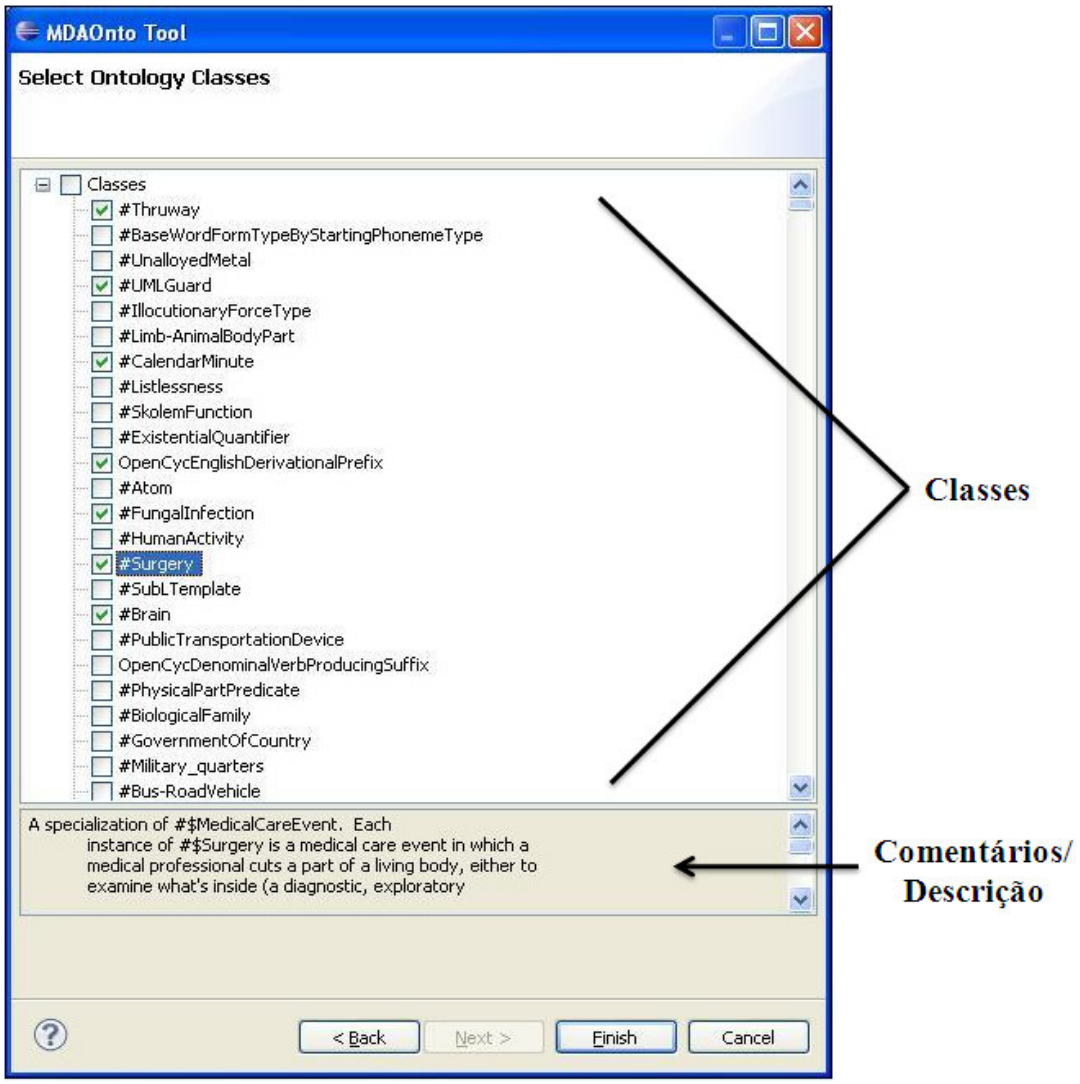

Figura 0.6 - Seleção de Conceitos
Nota: Elaboração do Autor.

Nesse cenário, quando o *stakeholder* determinar o caminho para a ontologia (XML representante) a ser importada e clicar no botão *Next*, internamente será disparado o seu processamento, ou seja, usando a OWL API, será realizado o *parser* no arquivo e as classes existentes serão localizas e separadas para, em seguida, serem exibidas ao *stakeholder*. Dentre essas classes serão escolhidas as relevantes ao SD, que, quando finalizado o *wizard*, servirão como subsídio para formação do *CoI* e do *G(CoI)*. Esses, por sua vez, servirão como parâmetro de entrada para a realização de todo processo de poda e posteriores transformações.

4.3.2 Pruner

Componente responsável por realizar a poda de conceitos irrelevantes ao SD. *Pruner* requer como entrada os conceitos relevantes – *CoI* e *G(CoI)* – que são representados internamente

através de vetores de objetos. A partir daí utilizou-se o algoritmo de poda descrito na seção 2.5.1 e implementou-se uma versão para *MDAOnto* através da linguagem Java.

As classes e métodos desse componente implementam os passos desse algoritmo de poda. Esse componente está apoiado pela OWL API, que além de possuir uma representação interna do metamodelo OWL, também permite que se manipule (leitura, escrita e exclusão) elementos da ontologia. Com essas características, esse componente fornece como saída um arquivo OWL representando a ontologia podada.

No Quadro 4.2, abaixo, é exibido o código que realiza a poda de ontologia, onde pode-se observar a indicação na linha 1 para a realização da poda dos conceitos irrelevantes. A linha 2 realiza a poda dos pais desnecessários. Enquanto que na linha 3 as generalizações duplicadas são excluídas e, por fim, na linha 4 todos os individuais são eliminados.

```
    // Exclui os conceitos irrelevantes da ontologia passada por parâmetro, baseado nos
    conceitos relevantes e suas generalizações gCoi.
1.  OWLPruning.getInstance().pruningIrrelevantConcepts(gCoi,
    OWLImporterManager.getInstance().getOntology());

    //Poda das generalizações desnecessárias. Para isso ele recebe como parâmetro um vetor de
    conceitos necessários (neededConcepts), que representam a união dos conceitos
    selecionados pelo usuário (CoI) mais os conceitos restringidos pelas restrições que
    restaram após os passos anteriores.

2.  OWLPruning.getInstance().pruningUnnecessaryParents(neededConcepts, gCoi,
    OWLImporterManager.getInstance().getOntology());

    // Poda dos caminhos de generalizações duplicados.
3.  OWLPruning.getInstance().pruningDuplicatedPaths(this.getCOI(), ontology, new
    ArrayList(ontology.getReferencedClasses()));

    // Poda de todos individuais.
4.  OWLPruning.getInstance().pruningAllIndividuals(OWLImporterManager.getInstance().ge
    tOntology());
```

Quadro 0.2 - Código para poda de ontologia
Nota: Elaboração do autor.

4.3.3 OWL2Ecore Transformer

Esse componente não apresenta qualquer característica gráfica. Seu objetivo é realizar a transformação de ontologia escrita em OWL para Ecore. Para esse propósito utiliza a API

EMFTriple que se baseia em ATL para realização das transformações. Como entrada ele requer um arquivo .OWL em conformidade com o metamodelo da linguagem e fornece como saída um arquivo .Ecore com a representação da ontologia através do metamodelo Ecore. Esse é um componente importante quando se pensa numa ferramenta MDA, visto que transformações representam um ponto de grande complexidade e dificuldade nesse contexto. Além disso, ele cumpre com um dos objetivos do trabalho que é prover facilitação no uso de ontologias para concepção de modelos de software.

4.3.4 Refinement Editor

Esse último componente possibilita que o *stakeholder* refine seu modelo, adicionando, alterando ou excluindo informações. Para esse propósito, *MDAOnto* usa essencialmente características inerentes ao Eclipse, através do *plug-in Ecore Tools*, que possibilita a manipulação desses modelos tanto graficamente quanto através de uma estrutura hierárquica, como pode ser observado na (Figura 4.7).

Outro ponto está relacionado ao editor OCL que nesse protótipo não foi implementado, porém como *MDAOnto* está baseada no conceito de plug-ins é possível no futuro adicionar novas funcionalidades, inclusive o próprio editor OCL.

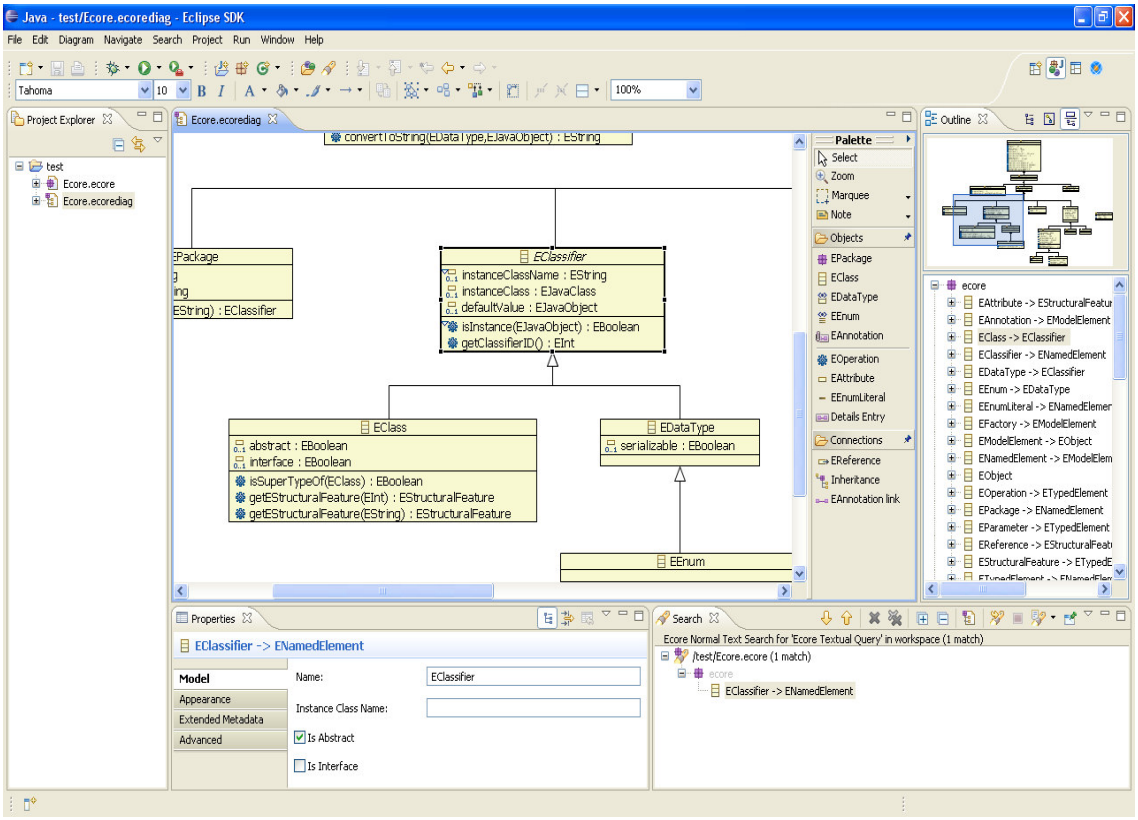

Figura 0.7 - *Ecore Tools*
Fonte: ECLIPSE (2010c).

4.4 COMPARAÇAO COM TRABALHOS CORRELATOS

O presente trabalho está alinhado com algumas propostas existentes, que implementam esforços na integração de MDA com ontologias e que foram descritos na seção 3.4 do presente documento.

Um trabalho muito importante é o ODM (OMG 2009), que define um conjunto de metamodelos, baseados no MOF (*Meta Object Facility*), tais como, RDF, RDFS, OWL, UML, entre outros, bem como um conjunto de relações entre eles. Esses metamodelos representam o ponto de partida nessa integração, já que torna possível um mapeamento entre modelos desses dois mundos.

Em Atkinson e Kiko (2008) os autores realizaram uma investigação completa dos espaços tecnológicos (ontologia e MDA) a partir de OWL e UML (diagrama de classes), mostrando quando possível, como as construções de OWL são mapeadas para UML. Para isso a

linguagem OCL também é utilizada, já que ela adiciona maior precisão aos modelos UML construídos.

De acordo com Abmann (2006), uma ontologia representa um modelo que descreve conceitos de domínio, similar ao CIM, sendo usada principalmente na Web Semântica, porém podendo ser útil no desenvolvimento de software em geral. Nesse sentido, os autores propõem papéis para ontologias, modelos e metamodelos dentro do contexto MDA.

Conesa e Olivé (2004) propuseram um algoritmo para poda de informações irrelevantes de ontologias, com o objetivo de se construir modelos conceituais.

Em Li e Chen (2008) é descrita uma abordagem MDA, na qual desenvolvedores podem usar conhecimento existente nas ontologias para auxiliar na implementação parcial de serviços Web semânticos. A ideia é usar ontologias de domínio existentes como base para a análise de requisitos e de sistema. A proposta considera a realização de uma transformação das ontologias em modelos independentes de plataforma em conformidade com o metamodelo Ecore, que após alguns refinamentos, podem ser usados para construção de modelos dependentes de plataforma (serviços Web semânticos). É um trabalho que foi implementado, também, na forma de plug-in para o Eclipse e que usa os projetos EMF e EODM na sua concepção.

MDAOnto alinha-se com o primeiro trabalho correlato, já que utiliza, também, metamodelos (RDF/OWL e Ecore) para representar seus modelos, possibilitando que esses dois mundos (MDA e Ontologias) se comuniquem. Nesse sentido, o segundo trabalho, foi extremamente útil para a proposta apresentada no momento da realização das transformações entre esses modelos.

Assim como no trabalho Abmann (2006), *MDAOnto* também considera ontologias como parte do CIM. Contudo, diferentemente do trabalho citado, *MDAOnto* apresenta a particularidade de considerar apenas parte da ontologia como representante do CIM, dando ao usuário a possibilidade de seleção, com o apoio automatizado da ferramenta, de quais partes da ontologia farão parte do CIM. Outro ponto de diferenciação é que inicialmente *MDAOnto* está focada em aplicações educacionais, como por exemplo, ferramentas de autoria de UAs.

O trabalho de Conesa e Olivé (2004) foi utilizado em *MDAOnto* para excluir elementos irrelevantes da ontologia antes de ser realizada a transformação para UML, isto é, evita-se que o modelo gerado seja confuso/ilegível, repleto de informações desnecessárias, o que dificultaria o entendimento e a manipulação deles.

Por fim, como visto nas seções anteriores, diferentemente de Li e Chen (2008) *MDAOnto* não considera como necessária toda ontologia, já que nem sempre todo conhecimento presente nela é útil para um software específico. Dessa forma é dada ao *stakeholder* a opção de escolher o que realmente interessa naquele contexto, visto que, uma transformação completa pode acabar gerando mais dificuldade de entendimento do que propriamente facilidades.

4.5 CENÁRIO DE USO

Nessa seção será apresentado o cenário de uso. Inicialmente o contexto em que ele está inserido, em seguida o problema encontrado e como *MDAOnto* pode colaborar para a otimização do processo.

4.5.1 Contexto

O avanço e a popularidade da Internet vivenciada nos últimos anos desencadearam inúmeros esforços de pesquisa, dedicados a encontrar alternativas que facilitem a criação, estruturação, acesso e distribuição de conteúdos para uso na educação. Daí surgiu o conceito de Objetos de Aprendizagem (OA), que é dito ser "*qualquer entidade, digital ou não, que pode ser utilizada e reutilizada durante o processo de aprendizagem que utilize tecnologia*". Tais objetos podem ter conteúdo hipermídia, conteúdo instrucional, outros objetos de aprendizagem e software de apoio, como por exemplo, exercícios, casos, tarefas de estudo, apresentações, etc. (KOPER; MANDERVELD, 2004). A partir daí surgiu a necessidade de criação de especificações para descrição de OAs com o foco em questões de tecnologia, reuso e interoperabilidade entre plataformas. Assim, neste contexto, emergiram especificações de metadados educacionais (IEEE LTSC, 2002), (ADL, 2004) e (IMS LD, 2003) orientadas a aprendizagem on-line.

OAs representam uma parte de todo processo de ensino-aprendizagem. Portanto, eles formam, juntamente com o *design* didático, Unidades de Aprendizagem (UA), que representam

"*qualquer pedaço delimitado de educação ou treinamento*" como, por exemplo, cursos, workshop, uma lição, etc. Elas representam não apenas um conjunto de recursos (OAs) ordenados, mas também atividades prescritas (soluções de problema, pesquisa, discussão, etc.), avaliações, serviços e instalações de apoio. Quais atividades, recursos, papéis e fluxo de trabalho são definidos através do *design* didático ou da aprendizagem (*learning design – LD*) da UA (IMS LD, 2003).

Nesse âmbito, a especificação IMS *Learning Design* (IMS LD) propõe-se ser um metamodelo pedagógico, com o foco na interoperabilidade e reutilização de materiais, que intenciona representar o *LD* de uma UA formalmente, semanticamente e que seja interpretável por máquina. IMS LD é um padrão *de facto* que permite a descrição do processo de ensino-aprendizado juntamente com os OAs associados em pacotes de conteúdo, representando UAs.

As ferramentas que operam sobre o IMS LD são divididas entre autoria e execução, projetadas para a modelagem e uso, respectivamente, das UAs. Em geral, poucos são os softwares que implementam IMS LD de forma efetiva. Dentre estes, os softwares de edição e os *players* não tem um foco em alguma técnica pedagógica específica e são mais orientados a usuários especialistas em IMS LD que a professores.

4.5.2 Problemas

O uso de especificações baseadas em metadados (tal qual IMS LD) é útil na descrição de recursos educacionais, facilitando a interoperabilidade e o reuso, pois representam o vocabulário que descrevem o LD. Em contrapartida, a semântica da especificação é usualmente expressa em linguagem natural, que embora mais fácil para o entendimento humano, é de difícil processamento por máquina. Outro fator é que IMS LD tem sido formalmente modelado com XMLS para facilitar a interoperabilidade entre os sistemas de software. No entanto, um dos grandes problemas da sua implementação por um Ambiente Virtual de Aprendizagem (AVA) refere-se à dificuldade em se obter uma UA totalmente consistente com o modelo de conhecimento desta especificação (AMORIM, 2007).

Em um estudo sobre a criação de UA(s) com o XMLS de IMS LD, verificou-se que podem ocorrer erros de consistência, mesmo utilizando-se ferramentas de autoria bastante relevantes

como Reload (RELOAD, 2010) e Coppercore (COPPERCORE, 2010). Em parte, estas inconsistências se devem ao fato da maioria do conhecimento descrito nos documentos IMS LD ser de complexo entendimento e implementação por parte dos desenvolvedores de software. Tal fato, pode levar a entendimentos incorretos que podem ser implementados em diferentes sistemas, dificultando a interoperabilidade. Por exemplo, o conhecimento declarado e não representado em XMLS tem que ser interpretado pelo programador da aplicação: neste caso, erros de interpretação podem ser facilmente cometidos. Além do mais, XMLS não é expressiva suficiente para descrever toda semântica associada aos elementos de IMS LD (AMORIM, 2007).

Para resolver esse problema, ontologias surgem como um mecanismo para descrever formalmente e explicitamente a estrutura e a semântica dos conceitos. Por esse motivo, (AMORIM, 2006) descreveram IMS LD como uma ontologia OWL (Figura 4.8). Com ela, a semântica de IMS LD é descrita explicitamente e o desenvolvimento de um software que permita o projeto e execução de UA pode ser amplamente facilitado prevenindo entendimentos incorretos dos conceitos da especificação.

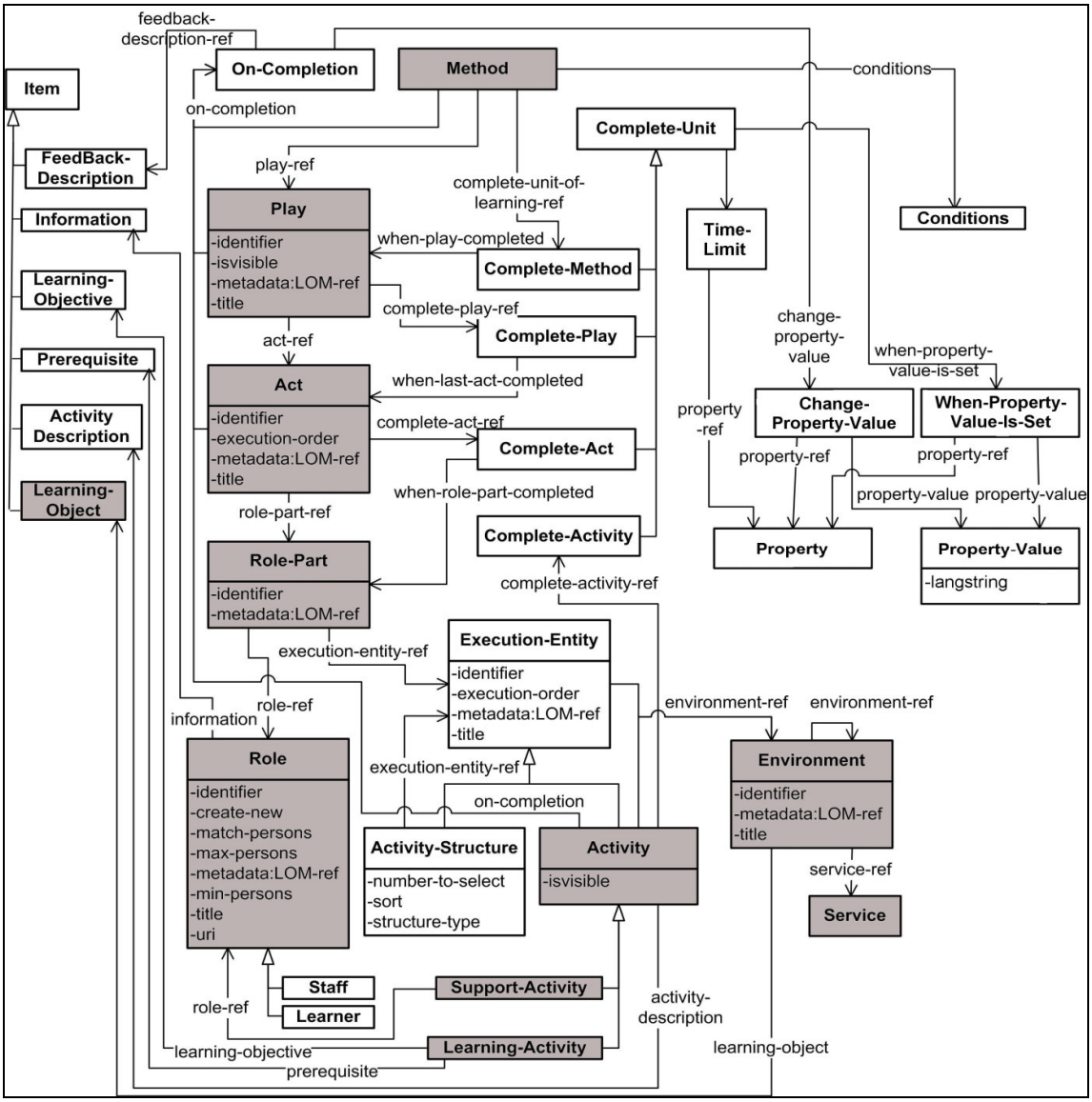

Figura 0.8 – Núcleo da ontologia IMS LD
Fonte: Amorim (2006).

No entanto, a criação, manutenção e adaptação de ferramentas (autoria e execução) compatíveis com IMS LD é uma tarefa trabalhosa e que consome tempo. Nesse contexto *MDAOnto* surge como uma opção facilitadora dessa atividade. Assim, o objetivo da seção seguinte é descrever como *MDAOnto* apoia a implementação de ferramentas (autoria e execução) compatíveis com a ontologia IMS LD, a partir de um cenário de uso com a plataforma Amadeus.

O Amadeus é um LMS livre (licença GNU-GPL) e desenvolvido em Java. Trata-se de um sistema capaz de integrar diferentes mídias digitais sob o conceito de extensões e micro-

mundos todos integrados através da camada de *middleware*, conhecida por AMADeUs-MM (LOBATO et al, 2009). Esse sistema caracteriza-se como um LMS de segunda geração baseado no conceito de *blended learning* para suportar educação a distância e presencial. O conceito de *blended learning* possibilitou a extensão dos estilos de interação possíveis entre os usuários e o sistema, os conteúdos e os demais usuários (GOMES; TEDESCO, 2002).

A escolha do Amadeu para o cenário de uso e um posterior caso de uso deveu-se a propensão à adoção/adaptação da ferramenta para tornar-se compatível com IMS LD e futuramente com a ontologia educacional IMS LD.

4.5.3 Exemplo de Utilização

Nessa subseção é descrito um exemplo de criação de modelos de uma ferramenta de autoria, a ser incorporada no Amadeus, com o suporte de *MDAOnto* e a ontologia IMS LD. Dessa forma, esse exemplo tem o objetivo de demonstrar como, a partir de uma ontologia de domínio, consegue-se chegar a um modelo CIM e posterior PIM usando *MDAOnto*.

No cenário implementado, e segundo os passos de execução de *MDAOnto* definidos anteriormente, tem-se inicialmente a importação da ontologia IMS LD para o projeto (*MDAOntoExample*) (Figura 4.9a). Nesse momento ocorre um processamento inicial, onde todas as classes da ontologia são coletadas e, em seguida, exibidas ao *stakeholder* para que ele possa selecionar as que considera relevantes (Figura 4.9b). Para esse exemplo, com o objetivo de construção de uma ferramenta de autoria, foram selecionadas as seguintes classes[4]: *Method, Play, Act, Role-Part, Role, Activity, Learning Activity, Support Activity, Environment, Learning Object, Service*. Na Figura 4.8 podem ser observadas essas classes (em destaque) e suas relações.

O *Method* relaciona-se com a dinâmica do processo de ensino/aprendizagem, desenvolvido para satisfazer os objetivos de aprendizagem. Um *Method* tem um ou mais *Plays* que executam-se em paralelo. Em um *Play* são especificadas quais *Roles* executam quais

[4] As classes escolhidas para o cenário, por motivo de simplificação, foram classes pertencentes ao nível A de IMS LD.

Activities e em qual ordem isso acontece. Cada *Play* consiste de um ou mais *Acts* que executam-se sequencialmente. O *Act* pode ser entendido como um estágio em um curso ou módulo. Cada *Act* consiste de um ou mais *Role-Parts* executada em paralelo, que associa um *Role* e uma *Activity.* Activity (*Learning* ou *Suport*) descreve as tarefas que um *Role* tem que realizar dentro de um ambiente *Environment* específico composto por objetos de aprendizagem *Learning Objects*, *Services* e *sub-Environments*.

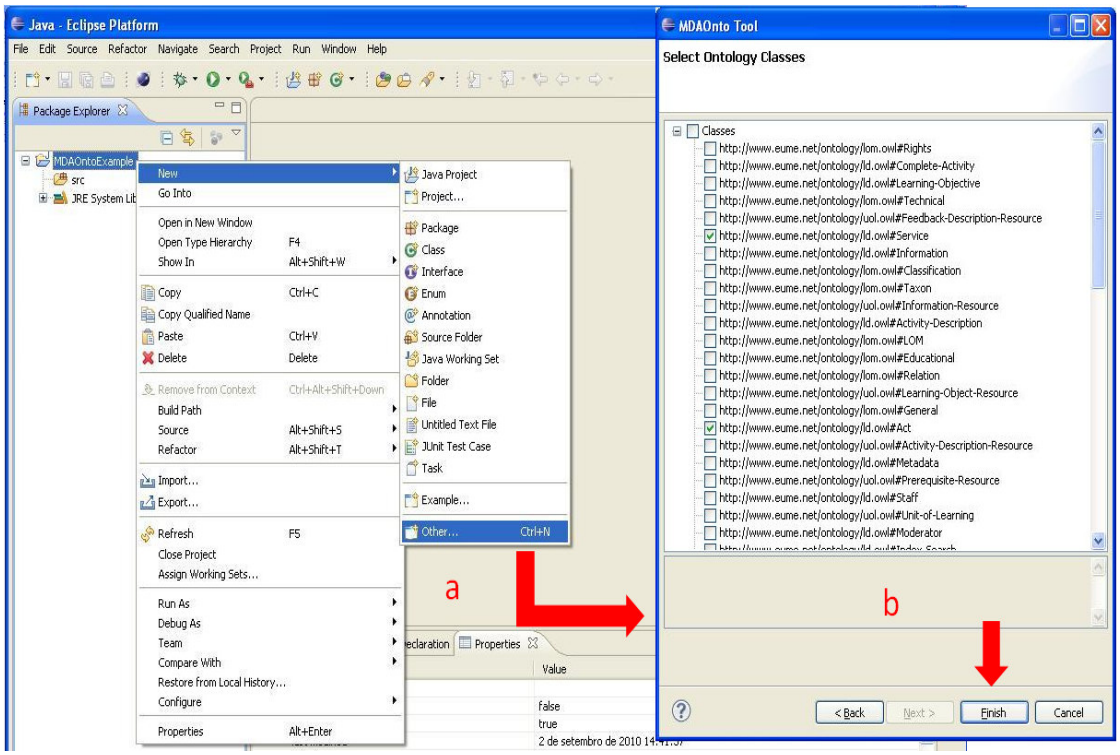

Figura 0.9 - *MDAOnto*: importação e seleção de conceitos relevantes da ontologia
Nota: Elaboração do Autor.

Feita a seleção de todas as classes relevantes, o *stakeholder* finaliza o *wizard*, que dispara o processamento que realiza duas etapas: primeiramente são coletadas todas as classes selecionadas pelo *stakeholder* e postas em um conjunto juntamente com as propriedades que tenham ligações com estas, ou seja, propriedades cujos *domains* e *ranges* estejam dentre as classes escolhidas. A partir disso é a realizada poda da ontologia e gerado um arquivo .owl (*MDAOntoExample.owl*) que representa essa ontologia podada. Em seguida, acontece a transformação OWL2Ecore, que baseado em um conjunto de regras de mapeamento, torna a ontologia podada (em OWL) num diagrama de classes Ecore (*MDAOntoExample.ecore*). Esses dois arquivos são criados e postos dentro do diretório *models* ficando disponíveis para

visualização e manipulação, como pode ser observado, em destaque (com uma elipse), na Figura 4.10.

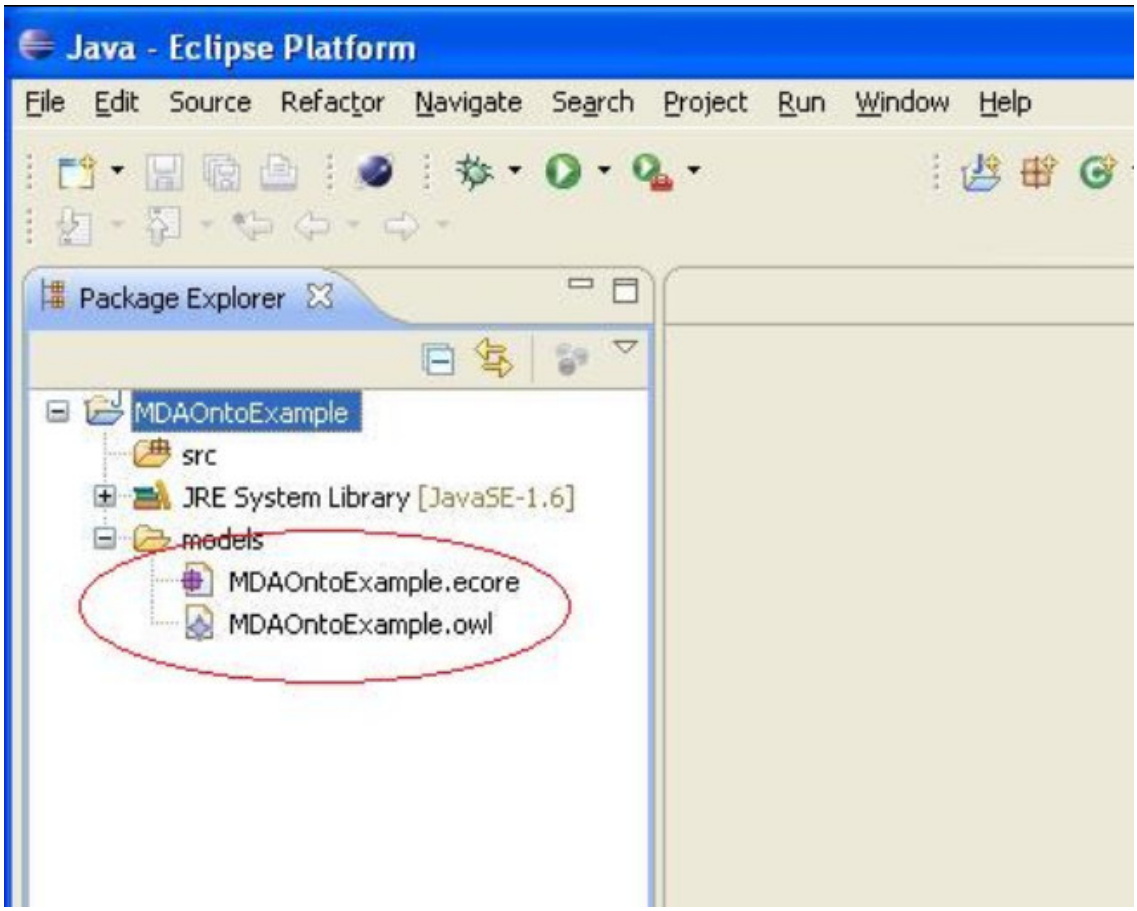

Figura 0.10 - Arquivos gerados por *MDAOnto*
Nota: Elaboração do Autor.

Com o modelo transformado e apoiado nos requisitos levantados, o próximo passo é realizar o refinamento (adicionando ou excluindo elementos). Para essa tarefa, *MDAOnto* conta com o apoio dos recursos disponibilizados pelas ferramentas do *plug-in* Ecore Tools e disponibiliza para o *stakeholder* duas possibilidades para realização desse trabalho: a primeira trata-se da manipulação do modelo através de uma estrutura hierárquica (Figura 4.11), na qual ele poderá adicionar, alterar ou excluir qualquer elemento daquele modelo. Por exemplo, pode-se adicionar um novo atributo à classe *Support_Activity* (elipse Figura 4.11).

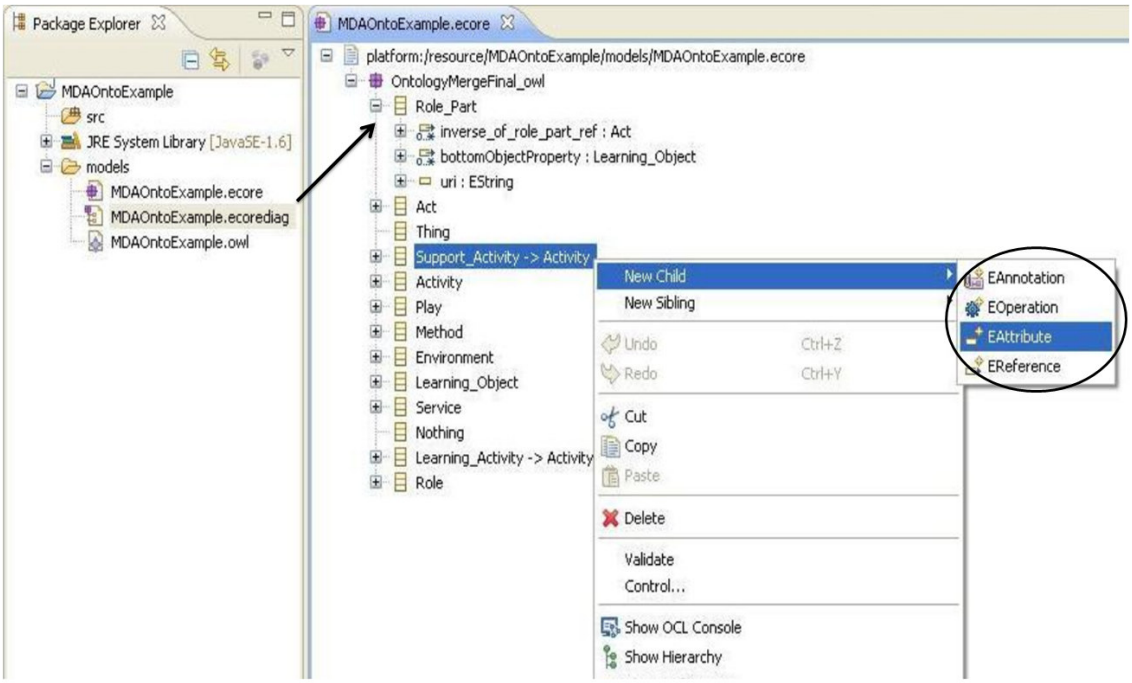

Figura 0.11 - Atividade de refinamento através da estrutura hierárquica
Nota: Elaboração do Autor.

A segunda possibilidade permite que o *stakeholder* manipule graficamente o modelo gerado. Para isso é necessário instanciar um diagrama ecore com base no modelo existente. Na Figura 4.12 (com a seta escura) é mostrado que um novo arquivo foi criado (*MDAOntoExample.ecorediag*). A partir disso o *stakeholder* poderá manipular os elementos do modelo usando a interface gráfica e dispor de recursos tais como "*drag and drop*", paleta com componentes (elipse na Figura 4.12), seção de propriedades, dentre outros.

Ainda é possível observar em destaque na Figura 4.12, que existe uma classe chamada *Export*. Tal classe não existe na ontologia, porém foi adicionada, exemplificando a última atividade (refinamento) de *MDAOnto*. Com essa adição espera-se, por exemplo, possibilitar que a ferramenta a ser construída possua um mecanismo de exportação das UAs. Assim como foi possível essa adição, o *stakeholder* poderá realizar qualquer adição/exclusão/alteração que lhe convier.

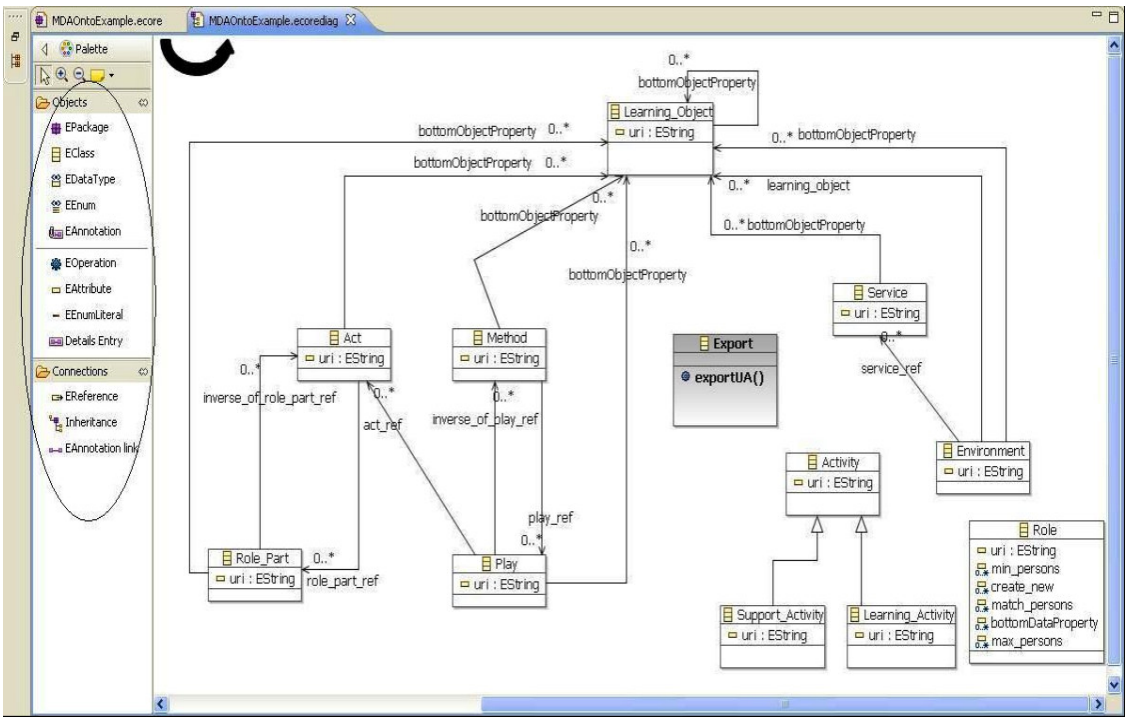

Figura 0.12 - Atividade de refinamento realizada graficamente
Nota: Elaboração do Autor.

Um ponto a ser destacado é que as classes provenientes da ontologia possuem um atributo chamado *URI*, como pode ser observado no recorte do modelo gerado (Figura 4.13), que as identifica/distingue de outras que venham ser adicionadas. Configurando um mecanismo de rastreamento dentre as classes presentes no SD e a ontologia fonte. Com isso, no futuro, além da adição de semântica ao modelo e posteriormente ao código, espera-se possibilitar, também a validação de consistência dos modelos construídos. Dessa forma, por exemplo, toda informação referente a um curso, como atividades, materiais, etc., seriam armazenadas/representadas através da ontologia. Assim, seria possível, durante a autoria de uma UA, se realizar de forma automática uma validação dos dados incluídos para garantir que estão de acordo com as relações e restrições definidas pela ontologia, o que levaria a criação de uma UA condizente com IMS LD (PINTO, 2010).

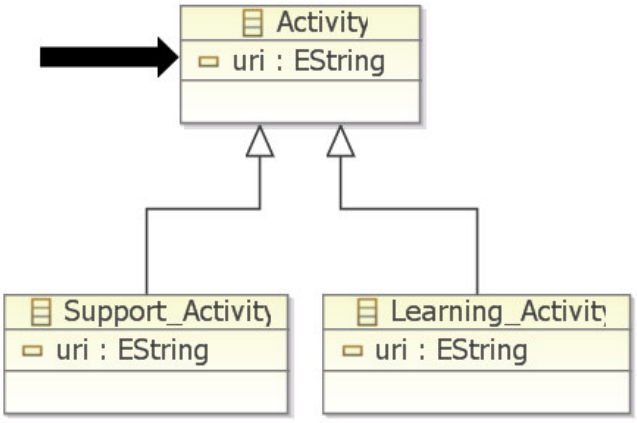

Figura 0.13 - Atributo URI adicionado às classes provenientes da ontologia
Fonte: Elaboração do Autor.

4.6 CONSIDERAÇÕES

Nesse capítulo, *MDAOnto* foi descrita, uma abordagem MDA, cujo objetivo é auxiliar o desenvolvimento de modelos de software (CIM e PIM), a partir do uso de ontologias de domínio preexistentes escritas em OWL.

MDAOnto apoia-se em quatro atividades: *i*) seleção de conceitos relevantes; *ii*) poda da ontologia; *iii*) transformação OWL2UML e; *iv*) refinamento.

A adoção de *MDAOnto* visa alcançar alguns objetivos. Estes estão atrelados à adoção de cada uma das atividades da proposta. Então, a partir:

a) Da seleção dos conceitos relevantes um dos benefícios que se vislumbra à *MDAOnto* é o fato que, de certo modo, a ferramenta consegue abstrair a complexidade inerente ao entendimento de uma ontologia, a partir do momento que simplifica a visão dos *stakeholders*, através da exibição de um conjunto de classes (conceitos) em alto nível, juntamente com seus comentários/descrições, escondendo, dessa forma, os diferentes elementos que nesse ponto dificultaria o seu entendimento e utilização. Além disso, não é necessária a utilização de outra ferramenta para manipulação da ontologia num contexto de desenvolvimento de software, já que *MDAOnto* está integrada ao próprio ambiente de desenvolvimento.
b) Da poda da ontologia, o benefício vislumbrado à *MDAOnto* está diretamente ligado ao uso de conhecimento consensual e difundido como fonte no desenvolvimento de

modelos de software, principalmente pelo fato da utilização de forma automatizada. Assim, espera-se alcançar redução de custos, tempo e riscos de uma análise de domínio completa partindo-se do zero.

c) Da transformação OWL2UML, vislumbra-se à *MDAOnto* o benefício da facilitação na manipulação e reutilização do conhecimento modelado em ontologias diretamente com UML, aproximando ainda mais as ontologias da Engenharia de Software.
d) Do refinamento busca-se reforça o objetivo de se aproximar *stakeholders* sem grandes aptidões do uso de ontologias. Dessa forma é possível que tais *stakeholders* utilizem o conhecimento modelado nas ontologias e complementem tal modelo com o que for necessário ao SD através do uso da UML.

Tal abordagem tem o intuito, também, de permitir que usuários envolvidos com desenvolvimento de software, mas que não possuem familiaridade com ontologias possam se aproveitar do conhecimento modelado nelas, já que *MDAOnto* busca o quanto possível manipulá-los de forma simplificada utilizando tecnologias amplamente difundidas como Ecore (uma simplificação da UML).

Especificamente, *MDAOnto* foi desenvolvida usando a linguagem Java e projetada como um *plug-in* para o Eclipse. Sendo assim, características disponíveis nessa IDE puderam ser reaproveitas, além do uso de alguns dos projetos como EMF, *Ecore Tools*.

Por fim, um cenário de uso foi criado, demonstrando um possível domínio de uso da abordagem. Nesse sentido, usou-se a ontologia IMS LD no contexto de criação de UAs, como subsídio para a construção dos modelos CIM e PIM de uma ferramenta de autoria. Dessa forma, espera-se uma facilitação/redução do esforço de desenvolvimento desse tipo de aplicação.

5 CONCLUSÃO

Como visto nesse trabalho, muitas pesquisas e estudos têm sido desenvolvidas visando à produção de softwares que satisfaçam aos desejos das pessoas num tempo hábil, com qualidade e naturalmente custo reduzido.

Dentre esses estudos, esse trabalho destacou a existência de dois em especial: MDA, que propõe a mudança do foco no desenvolvimento centrado em código para um centrado em modelos e; as ontologias, uma forma de representação de conhecimento, que pode ser utilizada, por exemplo: para representar um consenso em um determinado domínio; como artefato fonte de consulta; como mecanismo facilitador da comunicação.

Seguindo nessa linha, esse trabalho apresentou uma abordagem que propôs a interseção entre MDA e ontologias. Assim surgiu *MDAOnto*, uma proposta, que visa auxiliar, no contexto MDA, o desenvolvimento de modelos de domínio de software e modelos independentes de plataforma (camadas CIM e PIM), subsidiado pelo uso de ontologias de domínio preexistentes (escritas em OWL) como artefato chave na sua concepção.

Foram descritas as características da abordagem, bem como os detalhes técnicos envolvidos na implementação de um protótipo. Além disso, um cenário de uso fora criado como forma de demonstrar o funcionamento de *MDAOnto* e para isso utilizou-se uma ontologia relacionada ao domínio educacional (IMS LD), onde o (re) uso e a interoperabilidade de material de aprendizado e a modelagem do processo educacional são aspectos chave.

Assim, usando *MDAOnto* busca-se alcançar o seguintes benefícios:

a) Reduzir a complexidade da modelagem de domínio, ou seja, parte-se do pressuposto que não será necessário realizar toda modelagem (a cada novo projeto) a partir do zero, já que o conhecimento existente na ontologia poderá ser (re) usado;
b) Portabilidade, reuso, interoperabilidade, produtividade e documentação consistente, graças ao uso do padrão MDA;
c) Aumentar a confiabilidade do produto de software devido o uso de ontologias. Considerando que as referidas ontologias tenham sido bem construídas e sejam amplamente utilizadas;

d) Reuso, já que com a mesma ontologia pode-se criar diferentes software num mesmo domínio e;
e) Possibilitar que os profissionais envolvidos no desenvolvimento de software, que não possuem familiaridade com a manipulação de ontologias, possam beneficiar-se do conhecimento modelado nelas.

5.1 LIMITAÇÕES DO TRABALHO

Dentre as limitações do trabalho, destaca-se a dificuldade em se estabelecer mecanismos para avaliar os benefícios associados à utilização da abordagem num contexto de desenvolvimento de software. Muito disso se deve ao fato de que o trabalho restringiu-se a criação de um cenário de uso, que ilustrasse uma situação em que *MDAOnto* pudesse se fazer útil, já que, para a realização de um estudo de caso real seria necessário uma complementação do protótipo. Isto é, necessita-se de um protótipo que, além de apresentar as funcionalidades implementadas nesse trabalho, apresente uma extensão das funções, que leve a geração de código a partir desses modelos iniciais. Daí supõe-se que seria possível coletar a percepção dos usuários com relação à abordagem proposta.

Outra limitação relaciona-se ao fato de que, o protótipo criado não possui implementadas as transformações que tornam os axiomas da ontologia restrições em OCL e nem um editor que possibilite o usuário adicionar novas restrições. Mesmo que isso estivesse implementado, ainda assim, alguns conceitos da ontologia não poderiam ser transformados, já que em determinadas situações não existe um mapeamento entre eles (ATKINSON; KIKO, 2008).

Outro ponto limitador do trabalho diz respeito às instâncias das ontologias, que nesse trabalho foram desconsideradas para efeito de transformação. Porém, como foi visto anteriormente, segundo a literatura é possível representar as instâncias tanto no cenário de ontologias quanto MDA, no entanto a semântica do conceito é diferente nos dois ambientes.

5.2 TRABALHOS FUTUROS

Diante da complexidade esses dois mundos – MDA e ontologias – *MDAOnto* é um trabalho que apresenta muitas possibilidades de melhorias e extensões. Nesse sentido alguns trabalhos futuros foram identificados:

a) Complementação do protótipo através da adição das transformações PIM-PSM e PSM código. Uma opção seria uma adaptação à ferramenta produzida no projeto de pesquisa "***Uma solução de transformação bidirecional de modelos UML e código baseado na abordagem MDA***" no qual o autor do presente trabalho fez parte;
b) Melhorar o cenário de uso com a intenção de se criar um estudo de caso, que possibilite a validação e a descoberta de novos requisitos para a *MDAOnto*. Para isso será necessário ter um protótipo completo suficiente que se possa fechar um ciclo de desenvolvimento, ou seja, que possibilite a geração de código a partir de ontologias.
c) Estudo da viabilidade do mapeamento de instâncias (*individuals*) das ontologias para diagramas de objetos e, caso possível, posterior implementação;
d) Estender a transformação OWL2Ecore implementada no protótipo, para permitir que os axiomas descritos na ontologia possam ser mapeados, quando possível, para restrições representadas em OCL;
e) Estender *Refinement Editor* implementado no protótipo através da adição de um editor específico para manipulação de código OCL. Possibilitando que os *stakeholders* acrescentem suas regras de negócio diretamente nos modelos.

REFERÊNCIAS

ABMANN, U.; ZSCHALER, S.; WAGNER, G. Ontologies, meta-models, and the model-driven paradigm. In: CARELO, C.; RUIZ, F.; PIATTINI, M. (Ed.). **Ontologies for software engineering and software technology**. Berlin: Springer, 2006.

ADVANCED DISTRIBUTED LEARNING (ADL). **Sharable Content Object Reference Model - SCORM – Overview**, 2004. Disponivel em: <http://www.adlnet.gov/Technologies/scorm/SCORMSDocuments/2004%204th%20Edition/Overview.aspx>. Acesso em: 28 set. 2010.

AMORIM, R. et al. A Learning design ontology based on the IMS specification. **Educational Technology & Society**, v.9, n.1, 2006. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.109.451>. Acesso em: 28 set. 2010.

AMORIM, R. et al. Representing and executing units of learning on the basis of a learning design ontology. **Inteligência Artificial**, v.11, n.33, 2007. Disponível em: <http://erevista.aepia.org/index.php/ia/article/view/527>. Acesso em: 28 set. 2010.

ANTONIOU, G.; VAN HARMRLEN, F. **A semantic web primer**. 2. ed. Londres: The MIT Press, 2008. 288 p.

ARANGO, G.; PRIETO-DIAZ, R. Domain analysis: concepts and research directions. In: PRIETO-DIAZ, R.; ARANGO, G. **Domain analysis:** acquisition of reusable information for software construction. [S.l.]: IEEE Computer Society Press, 1989.

ARANGO, G. Domain analysis: from art form to engineering discipline. **ACM SIGSOFT Software Engineering Notes**, New York, v.14, n.3, p.152-159, may 1989.

ATKINSON, C.; KIKO, K. A detailed comparison of UML and OWL. Technical Report / Department for Mathematics and Computer Science, 2008, p. 2-58. Disponível em: <http://madoc.bib.uni-mannheim.de/madoc/frontdoor.php?source_opus=1898&show_connotea=1>. Acesso em 28 set. 2010.

ATKINSON, C.; KUHNE, T. Model-driven development: a metamodeling foundation. IEEE Software. **Los Alamitos,** v.20, n.5, p.36-41, set. 2003. Disponível em: <http://portal.acm.org/citation.cfm?id=942704>. Acesso em 28 set. 2010.

BACLAWSKI, K. Extending UML to Support Ontology Engineering for the Semantic Web. In: INTERNATIONAL CONFERENCE ON THE UNIFIED MODELING LANGUAGE, MODELING LANGUAGES, CONCEPTS, AND TOOLS, 4., 2001, Toronto. **Anais eletrônicos**… London: LNCS 2185 Springer, 2001. Disponível em: <http://portal.acm.org/citation.cfm?id=719462&dl=GUIDE&coll=GUIDE&CFID=103578197&CFTOKEN=21941602>. Acesso em: 28 set. 2010.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. **Scientific American,** New York, maio 2001.

BEZIVIN, J. On the unification power of models. In: SOFTWARE AND SYSTEM MODELING, 4., 2005. **Anais eletrônicos**… Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.105.8513>. Acesso em 26 set. 2010.

BORST, W. **Construction of engineering ontologies, graduate school for information and knowledge systems**. 1997. Disponível em: <http://www.ub.utwente.nl/webdocs/inf/1/t0000004.pdf>. Acesso em: 26 set. 2010.

BROWN, A.; CONALLEN, J.; TROPEANO, D. Introduction: models, modeling, and model-driven architecture (MDA). In: BEYDEDA, S.; BOOK, M.; GRUHN, V. **Model-Driven Software Development**. Springer, 2005. p 1-16.

BROWN, A. Model driven architecture: principles and practice. **Software and Systems Modeling**. Berlin, v.3, n.4, p.314-327, dez. 2004.

BUDINSKY, F. et al. **Eclipse modeling framework:** a developer's guide. [S.l.]: Addison-Wesley Professional, 2003. 720p.

CONESA, J.; DE PALOL, X.; OLIVÉ, A. Building conceptual schemas by refining general ontologies. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 14., 2003, Praga. **Anais**… Praga: LNCS 2736, Springer, p. 693-702.

CONESA, J.; OLIVÉ, A. A General Method for Pruning OWL Ontologies. In: ON THE MOVE TO MEANINGFUL INTERNET SYSTEMS: COOPIS, DOA, AND ODBASE, 2004, Cyprus. **Anais**… Cyprus: LNCS 3291, Springer, p. 981-998.

COPPERCORE. **The IMS Learning Design Engine**. Disponível em: <http://coppercore.sourceforge.net/>. Acesso em: 28 set. 2010.

CORCHO, O.; FERNÁNDEZ-LÓPEZ, M.; GÓMEZ-PÉREZ, A. Ontological engineering: principles, methods, tools and languages. In: CALERO, Coral; RUIZ, Francisco; PIATTINI, Mario. **Ontologies for software engineering and software technology**. Berlin: Springer, 2006. p. 1-48.

COSTA, T. M. D. **Utilizando padrões de anotação de código para suporte ao desenvolvimento de software semântico**. 2010. Dissertação (Mestrado)- Sistemas e Computação. Universidade Salvador – UNIFACS, 2010.

CRANEFIELD, S. Networked knowledge representation and exchange using UML and RDF. **Journal of Digital Information**, v.1, n.8, 2001. Disponível em: <http://journals.tdl.org/jodi/article/viewArticle/30/31>. Acesso em: 28 set. 2010.

DJURIC, D.; GASEVIC, D.; DEVEDZIC, V. A MDA-based approach to the ontology definition metamodel. In: WORKSHOP ON COMPUTATIONAL INTELLIGENCE AND INFORMATION TECHNOLOGIES, 4., 2003, Serbia. **Anais eletrônicos**… Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.2421>. Acesso em: 28 set. 2010.

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/article3.pdf>. Acesso em: 28 set. 2010.

ECLIPSE. **ATL - a model transformation technology**. 2010a. Disponível em: <http://www.eclipse.org/atl/>. Acesso em: 28 set. 2010.

ECLIPSE. **Eclipse Modeling Framework Project**. 2010b. Disponível em: <http://www.eclipse.org/modeling/emf/>. Acesso em: 28 set. 2010.

ECLIPSE. **Eclipse Modeling – EMFT – ECORE TOOLS**. 2010c. Disponível em: <http://www.eclipse.org/modeling/emft/downloads/?project=ecoretools>. Acesso em: 28 set. 2010.

EDUTOOLS. **Archived Course Management Systems Reviews**. Comparação entre sistemas de gestão de aprendizagem. 2010. Disponível em: <http://www.edutools.info/compare.jsp?pj=8&i=263,276,299,358,366,386,387>. Acesso em: 28 set. 2010.

EMFTRIPLE. **(Meta)Models on the Web of Data**. 2010. Disponível em: <http://code.google.com/p/emftriple/>. Acesso em 28 set. 2010.

FALBO, R.; GUIZZARDI, G.; DUARTE, K. C. An ontological approach to domain engineering. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, 14., 2002, Ischia. **Anais eletrônicos**… New York:

ACM, 2002. Disponível em <http://portal.acm.org > Acesso em: 26 set 2010.

FALBO, R. Integração de conhecimento em um ambiente de desenvolvimento de software, 1998. 203 f. Tese (Doutorado) – Universidade Federal do Rio de Janeiro / COPPE, 1998.

FENSEL, D. **Ontologies:** a silver bullet for knowledge management and electronic commerce**.** 2. ed. Berlin: Springer-Verlag, 2003. 162p.

GASEVIC, D.; DJURIC, D.; DEVEDZIC, V. **Model driven engineering and ontology development**. 2. ed. Berlin: Springer, 2009. 378 p.

GIBBS, W. W. Software's chronic crisis. **Scientific American**, p. 86-95, set. 1994. Disponível em: <http://www.cis.gsu.edu/~mmoore/CIS3300/handouts/SciAmSept1994.html> Acesso em: 22 set. 2010.

GIRARDI, R.; FARIA, C. G. A Generic Ontology for the Specification of Domain Models. In: INTERNATIONAL WORKSHOP ON COMPONENT ENGINEERING METHODOLOGY, 1., 2003, Erfurt. **Anais eletrônicos**… Erfurt, 2003. Disponível em: <http://wi2.wiwi.uni-augsburg.de/downl/gi-files/WCEM/girardi.pdf > Acesso em: 26 set 2010.

GOMES, A. S.; TEDESCO, P. A. Amadeus: a framework to constructivist support based on projects and multi-dimensional learner evaluation. In: WORLD CONFERENCE ON E-LEARNING IN CORPORATE, GOVERNMENT, HEALTHCARE AND HIGHER EDUCATION, 2002, Montreal. **Anais eletrônicos**… Montreal, 2002. Disponível em: <http://www.cin.ufpe.br/~asg/home.php?p=publications>. Acesso em: 28 set. 2010.

GÓMEZ-PÉREZ, A.; FERNÁNDEZ-LÓPEZ, M.; CORCHO, O. **Ontological engineering,** 2nd., British: Springer-Verlag London Limited, 2004. 403p.

GRUBER, T. R. **A translation approach to portable ontology specifications**. Knowledge Acquisition, p. 199-220, abril 1993a. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.101.7493> Acesso em: 22 set. 2010.

GRUBER, T. R. The Role of common ontology in achieving sharable, reusable knowledge bases. In: PRINCIPLES OF KNOWLEDGE REPRESENTATION AND REASONING, 2, 1991, San Mateo, CA. **Anais eletrônicos**… San Mateo, 1991. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.1743>. Acesso em: 28 set. 2010.

GRUBER, T. R. Toward Principles for the design of ontologies used for konowledge sharing. In: FORMAL ontology in conceptual analysis and knowledge representation. [S.l]: Kluwer Academic Publishers, 1993b.

GUARINO, N. Formal ontology and information system, In: INTERNATIONAL CONFERENCE ON FORMAL ONTOLOGY AND INFORMATION SYSTEMS, 1., 1998, Trento, Italy. **Anais eletrônicos**... Trento, Italy, 1998. Disponível em: <http://www.loa-cnr.it/Papers/FOIS98.pdf>. Acesso em: 26 set. 2010.

GUIZZARDI, G. **Desenvolvimento para e com reuso**: um estudo de caso no domínio de vídeo sob demanda. 2000. 202 f. Dissertação (Mestrado) – Universidade Federal do Espírito Santo, 2000.

HAPPEL, H.; SEEDORF, S. Applications of ontologies in software engineering. In: INTERNATIONAL WORKSHOP ON SEMANTIC WEB ENABLED SOFTWARE ENGINEERING, 2, 2006, **Anais…** 2006.

IEEE LTSC. **Draft standard for learning object metadata**. New York, 2002, p.44. Disponível em: <http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf>. Acesso em: 28 set. 2010.

IMS LEARNING DESIGN – IMS LD. **Information model, best practice and implementation guide**. Version 1.0 Final Specification IMS Global Learning Consortium Inc, 2004. Disponível em: <http://www.imsglobal.org/learningdesign/>. Acesso em: 28 set. 2010.

JASPER, R.; USCHOLD, M. A framework for understanding and classifying ontology applications. In: WORKSHOP ON KNOWLEDGE ACQUISITION MODELING AND MANAGEMENT, 12., 1999, Banff. **Anais eletrônicos**… Banff, 1999. Disponível em: <http://www.citeulike.org/user/mbrambil/article/7749652>. Acesso em: 26 set. 2010.

KLEPPE, A.; WARMER, J.; BAST, W. **MDA explained the model driven architecture:** practice and promise. [S.l.]: Part of the Addison-Wesley Object Technology Series, 2003. 170 p.

KOPER, R.; MANDERVELD, J. Educational modelling language: modelling reusable, interoperable, rich and personalised units of learning. **British Journal of Educational Technology**, v. 35, n. 5, 2004, p. 537-552. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.65.7845>. Acesso em: 28 set. 2010.

KOPER, R. **Introduction to IMS Learning Design**. Open University of the Netherlands, Berlin, 2005. Disponível em: <http://dspace.ou.nl/handle/1820/476>. Acesso em: 28 set. 2010.

LI, T; CHEN, C. **From ontology to semantic web service via model-driven system development**. 2008. Disponível em: <http://dspace.lib.fcu.edu.tw/handle/2377/10829>. Acesso em: 13 out. 2010.

LOBATO, L. L. et al. AMADeUs-MM: Rede educacional com integração de serviços multimedia. In: INTERNATIONAL CONFERENCE ON SYSTEMS INTEGRATION, 4., 2007, Brasília. **Anais eletrônicos**... Brasília, 2007. Disponível em: <http://www.cin.ufpe.br/~ccte/publicacoes/Artigo_AmadeusMM_ICSI.pdf>. Acesso em: 28 set. 2010.

MAEDCHE, A.; STAAB, S. Ontology Learning for the Semantic Web. **IEEE Intelligent Systems**, Piscataway, v.16, n.2, p. 72-79, mar. 2001.

MELLOR, S. J. et al. **MDA distilled:** principles of model-driven architecture. Boston: Addison-Wesley, 2004. 176 p.

NECHES, R. et al. Enabling technology for knowledge sharing. **AI Magazine**, v.12, n. 3, 1991. Disponível em: <http://portal.acm.org/citation.cfm?id=123775>. Acesso em: 28 set. 2010.

NOY, N. F. Semantic integration: a survey of ontology based approaches. dez 2004. **ACM Press**, n. 33, 65-70. Disponível em: <http://portal.acm.org/citation.cfm?id=1041410.1041421> Acesso em: 22 set. 2010.

O'LEARY, D. E. Impediments in the use of explicit ontologies for KBS development. **International Journal of Human-Computer Studies**, v.46, n.3, mar 1997. Disponível em: <http://portal.acm.org/citation.cfm?id=250546>. Acesso em: 26 set. 2010.

OLIVÉ, A. An introduction to conceptual modeling of information systems. In: PIATTINI, M.; DÍAZ, O. **Advanced database technology and design**. [S.l.]: Artech House, 2000.

OMG. OBJECT MODELING GROUP. **MDA guide version 1.0.1.** 2003. Disponível em: <http://www.enterprise-architecture.info/Images/MDA/MDA%20Guide%20v1-0-1.pdf>. Acesso em: 22 set. 2010.

OMG – OBJECT MODELING GROUP. **Meta Object Facility (MOF) Core Specifications,** 2006a. Disponível em: <http://www.omg.org/spec/MOF/2.0/PDF/>. Acesso em: 26 set. 2010.

OMG, OBJECT MODELING GROUP. **Model driven architecture (MDA**). 2001. Disponível em: <http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01>. Acesso em: 28 set. 2010.

OMG – OBJECT MODELING GROUP. **Object constraint language**, 2006b. Disponível em: <http://www.omg.org/spec/OCL/2.2/> Acesso em: 26 set. 2010.

OMG, OBJECT MODELING GROUP – OMG. **Ontology Definition Metamodel**, 2009. Disponível em: <http://www.omg.org/spec/ODM/1.0/PDF/> Acesso em: 26 set 2010.

OMG – OBJECT MODELING GROUP. UML 2.3 superstructure specification, 2010. Disponível em: <http://www.omg.org/spec/UML/2.3/Superstructure/PDF/>. Acesso em: 26 set. 2010.

OMG – OBJECT MODELING GROUP. **XMI mapping**, 2007. Disponível em: <http://www.omg.org/spec/XMI/2.1.1/> Acesso em: 26 set. 2010.

OWL API. **The OWL API**. Disponível em: <http://owlapi.sourceforge.net/index.html>. Acesso em: 28 set. 2010.

PISANELLI, D. M.; GANGEMI, A.; STEVE, G. Ontologies and information systems: the marriage of the century. In: LYEE WORKSHOP, 2002, Paris. **Anais eletrônicos**… Paris, 2002. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.103.9079>. Acesso em: 26 set. 2010.

PINTO, G. P.; SALVADOR, L; AMORIM, R. Uma solução MDA baseada em ontologias na construção de ferramentas de autoria para unidades de aprendizagem. In: BRAZILIAN WORKSHOP ON SEMANTIC WEB AND EDUCATION, 3., 2010, João Pessoa. **Anais...** João Pessoa.

REGO, E.; PINTO, G. P.; SALVADOR, L.; CHAVEZ, C.; SANTOS, W. Extração de visões centradas em stakeholders a partir de ontologias: uma abordagem MDA. In: WORKSHOP ON ONTOLOGIES AND METAMODELING IN SOFTWARE AND DATA ENGINEERING 2., 2007, João Pessoa. **Anais eletrônicos**... João Pessoa, 2007. Disponível em: <http://www.lbd.dcc.ufmg.br:8080/colecoes/womsde/2007/002.pdf>. Acesso em: 19 out. 2010.

RELOAD. **Reusable eLearning object authoring e delivery**. 2010. Disponível em: <http://www.reload.ac.uk/>. Acesso em: 28 set. 2010.

ROTHENBERG, J. The nature of modeling. In: WIDMAN, L.; LOPARO, K.; NIELSON, N. **Artificial Intelligence, Simulation, and Modeling.** New York: Wiley, 1989. p. 75-92.

RUIZ, F.; HILERA, J. R. Using ontologies in software engineering and technology. In: CALERO, Coral; RUIZ, Francisco; PIATTINI, Mario. **Ontologies for software engineering and software technology**. Berlin: Springer, 2006. p. 49-102.

RUSSEL, S.; NORVIG, P. Building a Knowledge Base. In: _____. **Artificial intelligence**: a modern approach. New York: Prentice-Hall, 1995, p. 217-264.

SENDALL, S.; KOZACZYNSKI, W. Model Transformation – the Heart and Soul of Model-Driven Software Development. **IEEE Software**, v.20 n.5, p.42-45, set. 2003. Disponível em: <http://portal.acm.org/citation.cfm?id=942589.942713 >. Acesso em: 28 set. 2010.

SOMMERVILLE, I. **Engenharia de software**. 8. ed. São Paulo: Pearson Addison Wesley, 2007. 592 p.

W3C. **OWL WEB ontology language guide**. 2004. Disponível em: <http://www.w3.org/TR/owl-guide/>. Acesso em: 26 set 2010.

WEBCT. **LMS Options and Comparisons**. Opções e comparações entre sistemas de gestão de aprendizagem. 2010. Disponível em: <http://kumu.brocku.ca/webct/LMS_Options_and_Comparisons>. Acesso em: 28 set. 2010.