Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Programa de Pós-graduação em Engenharia de Eletricidade

Desenvolvimento de um Sistema de Informação Médica com Web Services e MDA

Simone Azevedo Bandeira de Melo

São Luís
2007

Simone Azevedo Bandeira de Melo

Desenvolvimento de um Sistema de Informação Médica com Web Services e MDA

Dissertação de Mestrado submetida à Coordenação do Curso de Pós-graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como parte dos requisitos para obtenção do título de Mestre em Engenharia de Eletricidade, Área de Concentração: Ciência da Computação.

Orientador: Zair Abdelouahab

Ph.D. em Ciência da Computação – UFMA

São Luís
2007

Melo, Simone Azevedo Bandeira de.

Desenvolvimento de um Sistema de Informação Médica com Web Services e MDA / Simone Azevedo Bandeira de Melo. - 2007.

125 f.

Orientador: Zair Abdelouahab.

Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia de Eletricidade, Universidade Federal do Maranhão, São Luis, 2007.

1. Informática na Medicina. 2. Sistema de Informação Médica. 3. *Web Services*. 4. Arquitetura Dirigida por Modelos I. Abdelouahab, Zair, orientador. II. Título.

CDU: 004.61

Simone Azevedo Bandeira de Melo

Desenvolvimento de um Sistema de Informação Médica com Web Services e MDA

Dissertação de Mestrado submetida à Coordenação do Curso de Pós-graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como parte dos requisitos para obtenção do título de Mestre em Engenharia de Eletricidade, Área de Concentração: Ciência da Computação.

Aprovado em 02 de maio de 2007

BANCA EXAMINADORA

_____

Prof. Zair Abdelouahab, Ph.D.
(Orientador)

_____

Prof. Omar Andres Carmona Cortes, Dr.
(Membro da Banca Examinadora)

_____

Prof. Sofiane Labidi, Dr.
(Membro da Banca Examinadora)

_____

Prof. Denivaldo Cícero Pavão Lopes, Dr.
(Membro da Banca Examinadora)

Aos meus pais Walter e Maria Onilda Bandeira de Melo

Agradecimentos

A Deus pela proteção e bênçãos recebidas.

Aos meus pais, Walter e Onilda pela torcida, compreensão, carinho e preocupação.

Ao Professor e Orientador, PhD. Zair Abdelouahab pela orientação, apoio e incentivo a mim despendidos.

Ao Professor Dr. Denivaldo Lopes um agradecimento especial pelo esclarecimento de dúvidas fundamentais que fizeram este trabalho ser realizado da melhor forma possível, pelos aconselhamentos e direcionamentos fornecidos para elaboração desta dissertação.

A Professora PhD. Maria da Guia da Silva, pelo apoio e estímulo.

Ao meu grande amigo Mauro, pela ajuda, com palavras incentivadoras, apoio, colaboração e entendimento, que foi de fundamental importância na conclusão deste trabalho.

Ao meu amigo Carlos Afonso, por ter sempre me apoiado e incentivado.

Aos meus companheiros de Mestrado Lindonete, Mauro, Osvaldo, Emerson, Ricardo, Francisco, Johnneth, Adriano, Irlandino, André, Rafael, Aline e Helaine que acompanharam e contribuíram no desenvolvimento deste trabalho, pelo apoio e pela amizade.

Ao meu namorado Everleno, pelas palavras de incentivo e pelo ombro amigo.

A minha amiga Poliana por ter me proporcionado momentos de alegria e descontração nas horas mais triste.

A todos os meus companheiros de trabalho pela ajuda e compreensão nos momentos difíceis de conciliação de estudo e trabalho.

A todos os meus familiares e amigos pela compreensão nos momentos da minha ausência.

A FAPEMA pelo financiamento parcial deste trabalho.

A todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

Que Deus lhes abençoe sempre!

"Algo só é impossível até que alguém duvide e acabe provando o contrário”
Albert Einsten

Resumo

Neste trabalho, desenvolvemos um sistema de informação médica de auxílio no diagnóstico médico, baseado em uma abordagem orientada a modelos, no qual o sistema permite o compartilhamento de informações entre especialistas fisicamente dispersos. O desenvolvimento do sistema de informação médica foi feito usando MDA (Arquitetura Dirigida a Modelo) em que o PIM (Modelo Independente da Plataforma) foi feito conforme a UML (Linguagem de Modelagem Unificada), e os PSMs (Modelos Específico da Plataforma) conforme as Plataformas dos *Web Services*. Para implementar este sistema de informação médica, provemos metamodelos para os *Web Services*, JWSDP (*Java Web Services Developer Pack*) e WSOracle. Assim, provemos definições de transformação de UML para os *Web Services*, JWSDP e WSOracle. O desenvolvimento do sistema coloca em evidência o processo de transformação de PIM para PSM, utilizado em MDA.

Palavras-chave: Sistema de Informação Médica, *Web Services*, Arquitetura Dirigida por Modelos (MDA), Transformação de Modelos, Mapeamento.

Abstract

In this work, we develop a medical information system to support medical diagnosis based on an approach oriented to models, in which the system allows sharing of information between physically scattered specialists. The development of a medical information system is done using MDA (Model Drive Architecture) in which the PIM (Platform Independent Model) is created with UML (Unified Modeling Language), and the PSMs (Platform Specific Model) is done according to Web Services Platforms. To implement this system, we devised meta model s for the Web Services, JWSDP (Java Web Services Developement Pack) and WSOracle. Thus, we provide definitions of transformation of UML for the Web Services, JWSDP and WSOracle. The development of the system puts in evidence the process of transformation of PIM for PSM used in MDA.

Keywords: Medical Information System, Web Services, Model Driven Architecture (MDA), Model Transformation, Mapping.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ADL	Architecture Description Language
API	Applications Programming Interface
ATL	Atlas Transformation Language
AXIS	Apache Extensible Interaction System
AMM	ATLAS Model Management Architecture
BPEL4WS	Business Process Execution Language for Web Services
B2C	Business to Consumer
CIM	Modelo Independente de Computação
COM	Component Object Model
CM	Centros Médicos
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
DCOM	Distributed Component Object Model
DFM	Departamento de Física e Matemática
EJB	Enterprise Java Beans
EMF	Eclipse Modeling Framework
HTML	Hyper Text Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
J2EE	Java 2 Enterprise Edition
JAXB	Java Architecture for XML Binding
JAX-RPC	Java API for XML – Remote Procedure Call
JDK	Java Development Kit
JESS	Java Expert System Shell
JMI	Java Metadata Interface
JSP	Java Server Pages
JSTL	JavaServer Pages Standard Tag Library
JWSDP	Java Web Services Developer Pack
LQN	Layered Queueing Networks
MDA	Model Driven Architecture
MOF	Meta-Object Facility
MDR	MetaData Repository

NUTES	Núcleos de Telesaúde de Pernambuco
OCL	Object Constraint Language
OMG	Object Management Group
PC	Personal Computers
PIM	Modelo Independente de Plataforma
PSM	Modelo Específico de Plataforma
RPC	Remote Procedure Call
SBI	Sociedade Brasileira de Informática em Saúde
SI	Sistemas de Informação
SOA	Service-Oriented Architecture
SOA	Simple Object Access Protocol
SET-WS	Sistema Especialista para Telediagnóstico no ambiente de Web Services
SMTP	Simple Mail Transfer Protocol
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
URL	Universal Resource Locator
USP	Universidade de São Paulo
XMI	XML Metadata Interchange
XSLT	Extensible Stylesheet Language Transformation
W3C	World Wide Web Consortium
WSDL	Web Services Description Language

Lista de Códigos

Sumário

1 INTRODUÇÃO

A cada dia, as diversas áreas do conhecimento têm exigido mais esforços dos profissionais de informática para resolver problemas específicos. Para atender esta demanda, novas tecnologias e metodologias estão sendo criadas em um ritmo cada vez maior.

Por isso, uma das áreas, em que o uso da informática tem crescido e tem se tornado essencial, é a área médica, pois os profissionais da área de saúde, a cada dia necessitam de ajuda para resolver problemas ou tomar decisões, assim como para obter informações básicas em um determinado tópico, e ainda para manter o conhecimento atualizado sobre um determinado assunto (SIGULEM, 1997).

Em virtude disso, observamos que o objetivo fundamental da Informática Médica é o de colocar à disposição do médico a informação, onde e quando ela for necessária. Assim como os bancos e as companhias aéreas não podem funcionar sem o apoio da informática, torna-se cada vez mais difícil a prática da “boa medicina”, sem o auxílio das tecnologias da informação.

Assim, o uso da informática aliado aos conhecimentos médicos promovem uma série de aplicações que vão desde sistemas de telemedicina e sistemas de compartilhamento de informação médica até sistemas inteligentes de apoio ao diagnóstico médico.

No entanto, devido ao crescente uso da informática para ajudar os médicos nas suas atividades, tem crescido também o número de softwares para dar suporte às informações médicas. Sendo assim, os desenvolvedores de sistemas, preocupados em atender esse público, estudam e fornecem meios que facilitem o desenvolvimento desses softwares, reconhecendo a importância de promover uma maior portabilidade e interoperabilidade entre essas informações. E, com isso, temos observado a relevância da MDA (*Model Driven Architecture*) e dos *Web Services* em prover essas características.

MDA é uma arquitetura baseada em modelos. Ela apresenta uma abordagem neutra e independente de fabricante, utiliza modelos e metamodelos para descrever aspectos estruturais e questões relacionadas com interoperabilidade, ao invés de dar ênfase, às interfaces e seqüências de interações específicas dentro de uma plataforma previamente determinada. A OMG (*Object Management Group*) criou o padrão MDA com o objetivo de aumentar o nível de abstração para descrever sistemas de softwares. A idéia principal é usar linguagens de modelagem em todo o ciclo de vida de um software ao invés de utilizá-las como simples linguagens de projeto e especificação.

Neste contexto, os desenvolvedores constroem modelos independentes de plataforma que são processados por compiladores de modelos para realizarem transformações nos mesmos para algum contexto ou para gerar código final em uma determinada linguagem. Aliado a isso, temos o uso de plataformas *Web Services* para prover interoperabilidade em tempo de execução das aplicações de software da área médica.

1.1 Descrição do Problema

Temos observado que a informática na saúde está em constante crescimento, fazendo uso de novas ferramentas e tecnologias da computação, como é o caso da Inteligência Artificial, Realidade Virtual, Multimídia e Internet. Assim, a informática auxilia os médicos em suas consultas, na manipulação de informações, como dados do paciente, na troca de informações entre instituições médicas, no auxílio a uma segunda opinião, entre outros.

Dessa forma, no que diz respeito à tecnologia da informação disponível hoje para a área de saúde, de acordo com AUDY et al. (2002), é possível dar um salto na qualidade com a construção de ambientes capazes de prover sistemas de informação médica com dados selecionados, lapidados e modelados de forma a apoiar o processo decisório médico.

Assim, os sistemas de informação médica provêem muitas vantagens quando usados para facilitar o acesso a colaboração e o compartilhamento de dados entre Centros Médicos, pacientes e Centros de Pesquisas.

Entretanto, o problema é que os sistemas de informação médica possuem diversas informações, que geralmente não são compartilhadas entre as instituições médicas, ou seja, esses sistemas não são interoperáveis. Este problema decorre porque a grande maioria dos sistemas é baseada em diferentes plataformas de hardware e software, na qual a integração das informações é representada de maneira completamente diferente nas diversas bases de dados envolvidas.

Além disso, ainda não existem modelos de dados, vocabulários e conjunto de cenários comuns entre instituições médicas. E deve-se considerar ainda, que as informações em saúde são altamente complexas e pouco estruturadas.

Por isso, com o objetivo de tentar resolver a questão da interoperabilidade entre as aplicações de instituições médicas, nós propomos um sistema de informação médica baseado em *Web Services* da W3C (2006) que é um padrão aberto para o desenvolvimento de sistemas que utilizam entidades distribuídas em plataformas diferentes na Internet.

A modelagem do nosso sistema de informação médica será feita com o uso do padrão MDA da OMG (2006). Esse padrão foi escolhido visto que será criado um modelo independente de plataforma para o sistema de informação médica, e uma vez criado, o mesmo poderá ser aplicado a diferentes plataformas, tendo assim uma independência de plataforma.

Uma outra vantagem pela qual o padrão MDA foi escolhido é que, por esse padrão trabalhar a nível de modelos, criaremos modelos independentes de plataforma e modelos específicos para plataformas de *Web Services*. Com isso, escolhemos para nosso trabalho as plataformas de *Web Services* da Oracle (ORACLE, 2005) e o JWSDP (*Java Web Services Developer Pack*) da SUN (2005). Mostraremos com esse trabalho, que usando a tecnologia de MDA, na modelagem do nosso sistema de informação médica, criaremos um modelo independente de plataforma (PIM), que poderá ser aplicado a várias plataformas específicas.

Assim, observaremos que o uso de MDA para a modelagem de software facilita o trabalho do desenvolvedor, já que o foco está no desenvolvimento do PIM.Uma vez o PIM criado, este pode ser transformado em vários PSMs de diferentes plataformas. E a criação dos PSMs e as definições de transformações são criadas uma única vez e aplicadas no desenvolvimento de diversos sistemas. Além disso, podem ocorrer também mudanças no sistema, sendo assim, basta fazer as modificações no PIM e através das ferramentas utilizadas na transformação de modelos, gera-se novamente o PSM e o código.

1.2 Objetivos Gerais e Específicos da Dissertação

Esta dissertação tem como objetivo geral o desenvolvimento de um sistema de informação médica de auxílio no diagnóstico médico. Este desenvolvimento é baseado em uma abordagem orientada a modelos. Assim, será desenvolvido um modelo independente de plataforma (PIM) para o sistema e transformaremos este PIM em algumas plataformas específicas (PSM) de *Web Services.*

O desenvolvimento deste sistema, coloca em evidência o processo de transformação de PIM para PSM utilizado em MDA. E a escolha do sistema de informação médica ocorreu devido ser uma área em constante estudo em nossa região.

Esta dissertação tem como objetivos específicos, os seguintes itens:

- Desenvolver um Sistema de Informação Médica, utilizando a plataforma dos *Web Services* no contexto da abordagem MDA;

- Usar uma metodologia, onde a transformação de modelos é dividida em duas etapas: especificação de correspondências e na definição de transformação;
- Desenvolver um Modelo Independente de Plataforma para o sistema de informação médica proposto;
- Desenvolver Metamodelos para as plataformas dos *Web Services*: da Oracle, JWSDP versão 1.5 e Java 1.5;
- Criar regras de transformação entre o PIM (UML) e PSM (Java, ou JWSDP, ou WSDL , ou WSOracle);
- Usar a transformação do PIM do sistema de informação médica para os PSMs para gerar parte do código.

1.3 Organização da Dissertação

Esta dissertação está organizada em seis capítulos. O Capítulo 1 apresenta uma introdução ao tema, o qual descreve o problema a ser estudado, os objetivos pretendidos e a organização do trabalho.

O Capítulo 2 ilustra o estado da arte, onde conceitos de Informática aplicada à área de Saúde e de sistemas de informação médica são apresentados. Uma visão geral das tecnologias de MDA e de *Web Services* também é apresentada, visto que estas são as tecnologias de base para o desenvolvimento deste trabalho.

No Capítulo 3, propomos um sistema de informação médica, utilizando um PIM, uma metodologia e uma modelagem das plataformas de *Web Services*. E utilizamos metamodelos na plataforma WSDL, JWSDP, *Web Services* da Oracle e Java.

Aborda-se, no Capítulo 4, a criação das regras de transformações de UML para WSDL, de UML para JWSDP, de UML para WSOracle e de UML para Java.

No Capítulo 5, a implementação e os testes realizados com o sistema de informação médica proposto são apresentados.

E por fim, no sexto Capítulo, apresentamos as considerações finais da dissertação, ressaltando as contribuições da pesquisa realizada e também sugestões para trabalhos futuros.

2 ESTADO DA ARTE

Neste Capítulo, os conceitos fundamentais das tecnologias que servem de base para o desenvolvimento da nossa proposta são apresentados. Tais conceitos têm papel relevante no entendimento individual de cada tecnologia e na obtenção do conhecimento para melhor integrá-las. Abordamos neste Capítulo, a aplicação da Informática na área de saúde, uma visão geral da abordagem MDA, conceitos relacionados à tecnologia de *Web Services* e alguns projetos relacionados.

2.1 Aplicação da Informática na Área da Saúde

As expressões "informática médica" e "informática em saúde" têm sido usadas como sinônimos. Dado que as atividades relativas à saúde abrangem não só a medicina, mas também a enfermagem, a nutrição, a veterinária e a odontologia. Assim, a SBIS - Sociedade Brasileira de Informática em Saúde (SBIS, 2005) resolveu utilizar o termo mais amplo "saúde" ao invés de "médica", ao contrário do que se faz na Europa, Ásia e nos Estados Unidos. A expressão "Informática Médica" tem sua origem entre os anos de 1968 e 1970, na Rússia, França e países de língua inglesa, ao se referir, inicialmente, a uma interface entre disciplinas, como ciência da computação aplicada a medicina ou ciência da informação médica (BEMMEL, 1999).

De um modo simplificado, podemos dizer que a "Informática em Saúde" é o estudo e uso de computadores e sistemas de comunicação e informação na assistência médica, ensino e pesquisa na área da saúde (BEMMEL, 1999).

2.1.1 Sistemas e Tecnologias de Informação na Área de Saúde

Na Figura 2.1 mostramos a interação de dados, informação e conhecimento de sistemas e tecnologias de informação na área de saúde.

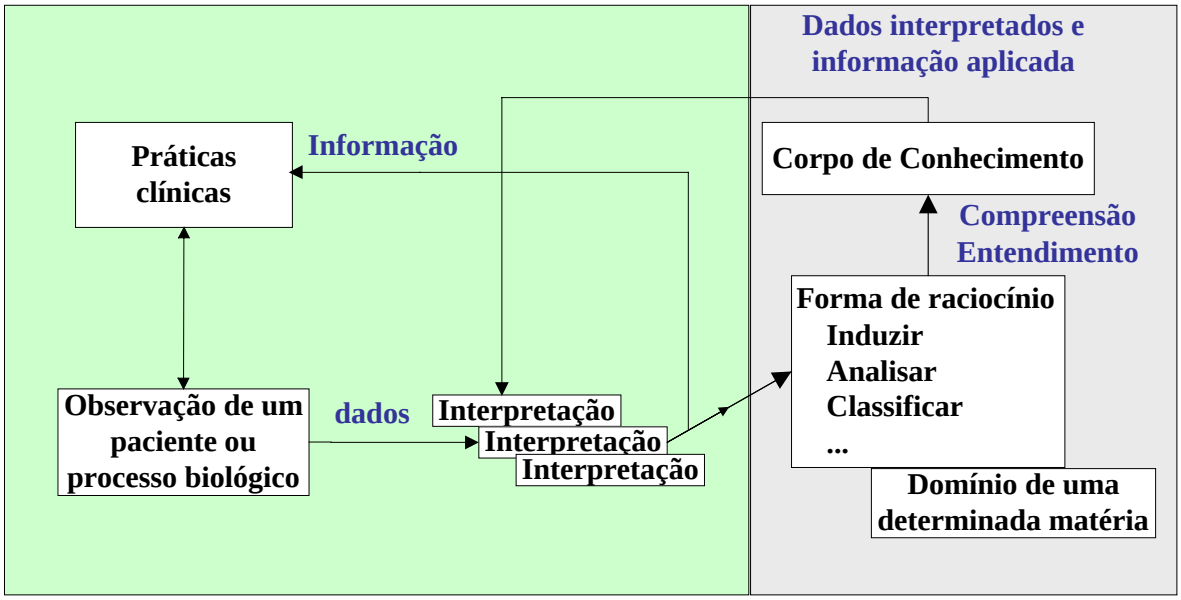

Figura 2.1: Dados, Informação e Conhecimento (VASCONCELOS, 2003).

Assim, vemos a seguir o conceito de dados, informação e conhecimento:

- Dados: Conjunto de fatos, observações ou conceitos para processar e interpretar por seres humanos ou máquinas eletrônico-digitais (computadores). Exemplo: dados provenientes da observação de um paciente ou de um determinado processo biológico;
- Informação: Conjunto de dados interpretados. Conjunto de fatos relevantes e com significado num determinado contexto. Dados processados e interpretados. Exemplo: diagnóstico médico de um paciente, ou processo terapêutico a desenvolver;
- Conhecimento: Informação aplicada na resolução de um problema ou no processo de tomada de uma decisão. Exemplo: aplicação de “informática médica” e conseqüente raciocínio e enquadramento no processo de cura de uma determinada doença.

Esses termos são a base para o entendimento e para a relação de informação entre médicos e centros de saúde.

Diversos investigadores e profissionais no mundo da área de Ciências Médicas e de Ciência da Computação se reúnem em pesquisas com intuito de solucionar problemas na área de informática médica, como mostra a Figura 2.2.

Figura 2.2: Profissões na área médica e de computação (VASCONCELOS, 2003).

2.1.2 Impacto dos Sistemas e Tecnologias da Informação

Dentre os impactos dos sistemas e tecnologias da informação, podemos citar (VASCONCELOS, 2003):

- O papel dos pacientes na gestão dos seus problemas de saúde tem vindo a sofrer alterações com a integração dos sistemas e tecnologias de informação na área da saúde e respectivas ciências médicas;
- A maior parte da informação médica disponível na Internet tem um caráter associativo e, por vezes, enganador. A informação médica necessita de dados concretos da sua proveniência;
- Neste contexto, é necessário desenvolver mecanismos para verificar e avaliar a qualidade das fontes de informação médica.

2.1.3 Exemplos de algumas aplicações de informática na saúde

Como exemplo de aplicações de informática na saúde, destacamos as seguintes (VASCONCELOS, 2003):

- Gestão de bases de dados clínicos, no qual temos: Sistemas de Informação (SI) focado no paciente, SI para gerir informação médica especializada (tais como Cuidados

Primários, Pediatria, Obstetrícia, Cirurgia, Oncologia e Odontologia) e SI para gerir Informação hospitalar e de outros serviços de saúde;

- Sistemas de apoio à decisão: com os sistemas de automação do ciclo de diagnóstico e terapia e sua representação;
- Processamento de imagem: em que Cardiologia, Radiologia, Neurologia, Odontologia são as áreas médicas de maior interesse.

2.1.4 Aplicações e tendências

A seguir, mostramos algumas das aplicações e tendências da informática na área médica:

- Transição e integração gradual dos serviços hospitalares e clínicos com serviço de saúde individualizados através de postos de saúde especializados;
- Serviço de saúde focado no paciente;
- Serviço de saúde focado no paciente envolve ativamente o paciente no seu processo de saúde:
    - Gestão de registros do paciente;
    - Partilha de registros do paciente (médico-paciente);
    - Registro e acesso a dados clínicos do paciente.

2.2 MDA

Nesta seção, apresentamos a MDA (*Model Driven Architecture*) ou arquitetura orientada a modelos, que é uma abordagem criada pela OMG (*Object Management Group*) para promover o uso de modelos no desenvolvimento de softwares com o intuito de prover soluções para o problema do desenvolvimento, manutenção, evolução de sistemas e favorecer a interoperabilidade e portabilidade de sistemas (OMG, 2006). MDA promove uma abordagem na qual a especificação do sistema é feita de forma independente de plataforma e, para cada uma das plataformas específicas, tal especificação pode ser automaticamente transformada em uma implementação correspondente (OMG, 2006).

De acordo com KLEPPE, a MDA apresenta alguns benefícios como (KLEPPE *et al*., 2003):

- Produtividade:

O foco do desenvolvedor passa a ser o desenvolvimento do PIM. A transformação do PIM para o PSM precisa ser definida uma única vez e pode ser aplicada no desenvolvimento de diversos sistemas. Devido a este fato, tem-se uma redução no tempo de desenvolvimento.

Com as transformações, detalhes técnicos são adicionados automaticamente do PIM para o PSM aumentando a produtividade no desenvolvimento do sistema. Entretanto, tais ganhos somente serão obtidos se a geração do PSM a partir do PIM for automatizada por ferramentas.

- Portabilidade

  A portabilidade é alcançada pelo foco no desenvolvimento do PIM, que é por definição, independente da plataforma. Um mesmo PIM pode ser automaticamente transformado em vários PSMs de diferentes plataformas, através de mapeamentos. Tudo o que for especificado no PIM é completamente portável.

- Interoperabilidade

  Diferentes PSMs gerados a partir de um mesmo PIM podem ter relacionamentos entre si, os quais são chamados de pontes (*bridges*) dentro do vocabulário MDA.

- Manutenção e Documentação

  Os desenvolvedores focam-se no PIM, que é usado para gerar o PSM, que por sua vez é usado para gerar o código-fonte. O modelo é uma representação abstrata do código, portanto o PIM preenche a funcionalidade de documentação de alto nível. Como o PIM não é descartado no final do desenvolvimento, mudanças que eventualmente sejam feitas no sistema podem ser realizadas alterando-se o PIM e gerando-se novamente o PSM e o código-fonte.

MDA baseia-se em alguns padrões da OMG como UML (*Unified Modeling Language*), MOF (*Meta-Object Facility*) e XMI (*XML Metadata Interchange*). UML é uma notação padrão na indústria para representação gráfica de modelos de software orientado a objetos (OMG, 2005a). MOF é um framework para gerenciamento de metadados usado na definição de vários metamodelos propostos pela OMG (2002). O XMI é o formato para representação, intercâmbio e compartilhamento de objetos, utilizando XML, que é um padrão que permite a troca facilitada de metadados, entre as ferramentas de modelagem (baseadas na UML da OMG) e os repositórios (OMG-MOF) (OMG, 2005b).

2.2.1 MOF

O MOF (*Meta-Object Facility*) é um padrão da OMG que especifica uma linguagem abstrata para descrever outras linguagens (OMG, 2002). Neste contexto, linguagem significa uma sintaxe abstrata de uma linguagem, isto é, o MOF não é usado para descrever uma gramática, ele é usado para descrever a estrutura dos objetos que podem ser representados em uma dada linguagem. O MOF é também freqüentemente referenciado como um metametamodelo e as sintaxes abstratas descritas por ele são chamados de metamodelos.

Arquitetura MOF

A arquitetura MOF baseia-se no paradigma de orientação a objetos e no modelo de arquitetura de meta-modelagem de vários níveis de meta-informações. Isto significa que sua arquitetura é organizada em níveis onde as entidades em um dado nível são definidas como instâncias das entidades no nível imediatamente superior. Para maior clareza na apresentação dos conceitos, o padrão mostra o modelo com quatro níveis de abstração de informação: M0 (objetos), M1 (modelos), M2 (metamodelos) e M3 (metametamodelo).

Na Figura 2.3, um exemplo de arquitetura a quatro níveis é apresentado com alguns possíveis elementos e pacotes de elementos para cada um dos níveis definidos. Com exceção do nível M3, que contém o modelo MOF, todos os demais podem ser alterados e estendidos. A razão desta restrição está associada à forma de limitar o número de níveis de uma arquitetura onde o nível de cima descreve o nível imediatamente inferior e é descrito pelo nível imediatamente superior. Uma das formas de parar este processo é definir o último nível em termos de si mesmo. O efeito colateral é a sua imutabilidade e este é o caso do nível M3.

O nível M3 contém o modelo MOF (metametamodelo), que é o unificador da arquitetura como um todo, pois fornece a linguagem abstrata, que é usada para representar os metamodelos definidos na arquitetura em quatro camadas.

Já o nível M2 contém os metamodelos descritos através do modelo MOF. Na Figura 2.3, o metamodelo UML é mostrado, através de elementos (artefatos) da linguagem do nível de cima, que é o metametamodelo do (M3). É importante deixar claro que a linguagem é suficientemente geral para representar outros pacotes definidos pelos usuários ou não. O fato de apenas o metamodelo UML estar presente é puramente ilustrativo.

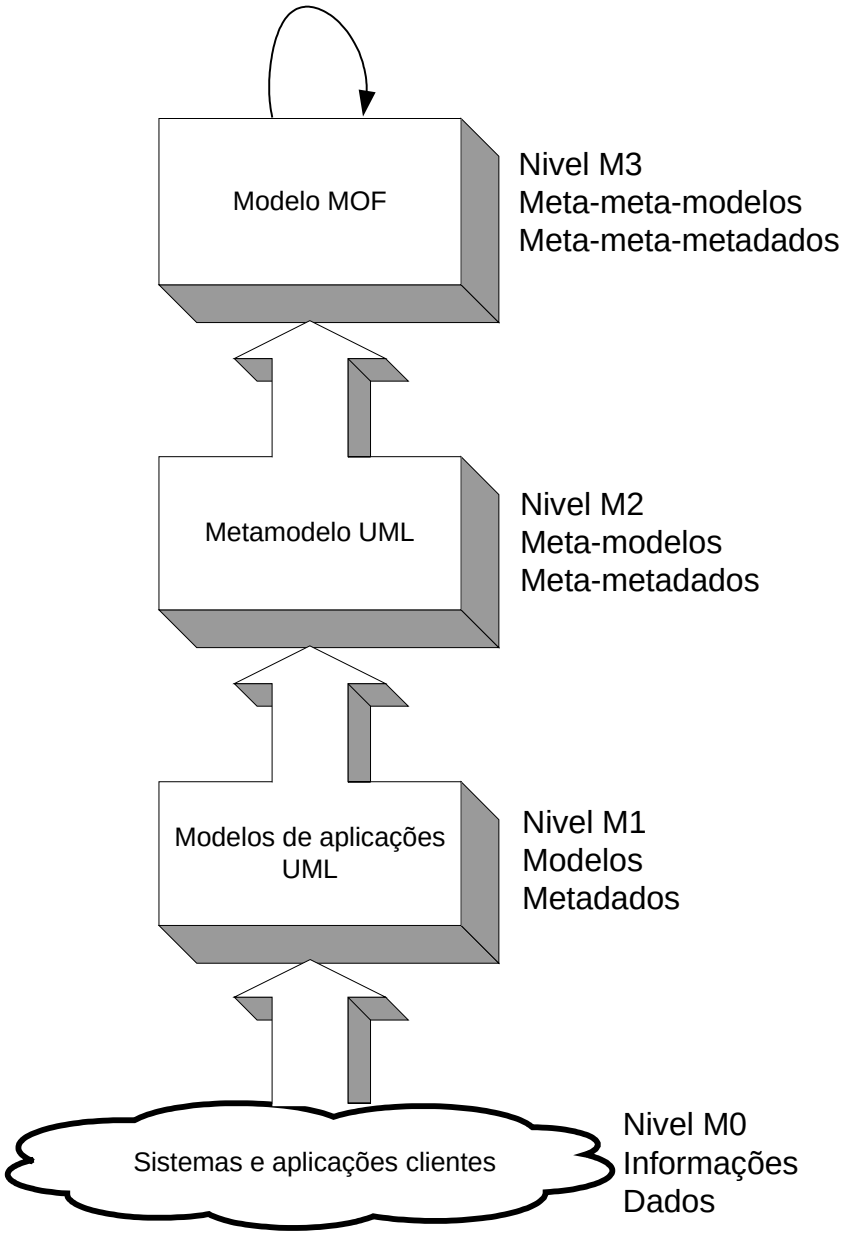

Figura 2.3: Exemplo de uso da arquitetura em quatro camadas.

O nível M1, onde as meta-informações reais estão localizadas, consiste de modelos representando aplicações ou sistemas específicos. Cada modelo é definido de acordo com um simples metamodelo M2, que fornece as informações para instanciar e interpretar os elementos do modelo.

Finalmente, no nível M0, temos as entidades que são o objetivo principal da modelagem, tais como os objetos de dados de um sistema.

Modelo MOF

O Modelo MOF consiste de um conjunto de elementos de modelagem que permitem a descrição de metamodelos na arquitetura a quatro níveis. Estes elementos são, na verdade, os tipos de meta-metamodelo definido no nível M3, os quais são usados para instanciar elementos que formam os metamodelos no nível M2. Os principais elementos do modelo MOF são mostrados na Figura 2.4 e descritos na Tabela 2.1.

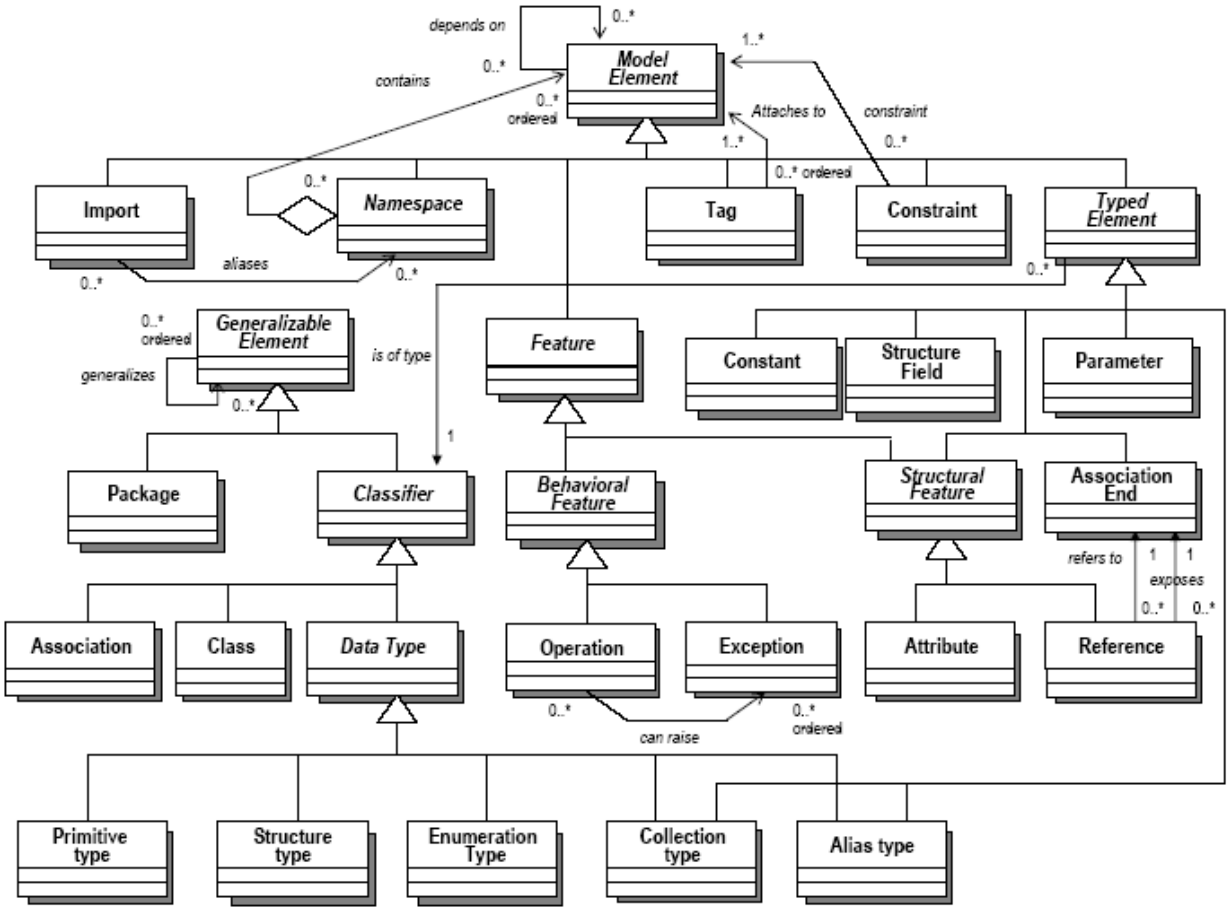

Figura 2.4: Fragmento do Modelo MOF (metametamodelo) (OMG, 2002).

MOF é o mecanismo básico do MDA para definição de linguagens de metamodelos. Ele provê um conjunto de conceitos, em particular: Diagramas de Classe para definir a sintaxe abstrata e OCL (*Object Constraint Language*) para definir semânticas de uma linguagem de modelagem (OMG, 2005a). Como exemplo, temos o metamodelo UML, onde a semântica é definida usando uma mistura de OCL e texto informal.

Tabela 2.1: Principais Elementos do Modelo MOF

Elemento	Descrição
Class	O elemento *Class* é o principal elemento do modelo MOF. Ele pode ter três tipos de elementos internos: *Attributes*, *References* e *Operations*.
Package	O elemento *Package* é usado para encapsular os demais elementos do modelo MOF e viabilizar a modularização na criação de meta-modelos. Ele é essencialmente um mecanismo de escopo para os elementos definidos em seu contexto.
Data Type	*Um Data Type* é usado para representar um tipo anônimo que não corresponde a um elemento identificado de um meta-modelo. Normalmente, ele é usado como parte da especificação de outro elemento do metamodelo, tal como para dar o tipo de um atributo de uma *Class* ou parâmetro de uma *Operation*.
Association	O elemento *Association* descreve o relacionamento entre dois elementos *Class*. Ele basicamente define um conjunto de dois pontos terminais que são elementos *Class*, e é um dos pontos fracos da especificação, por não permitir relacionamentos com mais de dois elementos *Class*.
Constraint	Este elemento permite que uma definição feita dentro de um metamodelo seja expandida com relação a sua semântica, na forma de regras que descrevem a consistência de modelos derivados de um meta-modelo. Em princípio, *Constraints* podem ser expressos em qualquer linguagem, incluindo uma linguagem natural. No entanto, a especificação MOF recomenda o uso de uma linguagem formal, o OCL (*Object Constraint Language*) que é definida como parte da UML.
Operation	O elemento *Operation* também é usado internamente no elemento *Class*. Ele fornece uma maneira de se associar operações a um elemento do metamodelo que está em definição.
Constant	*Constant* representa uma constante, uma ligação entre um nome e um valor.
Exception	*Exceptions* representam exceções que podem acontecer na execução normal de uma operação definida pelo elemento *Operation*.
Reference	Uma *Reference* é usada internamente no elemento *Class* para explicitar o papel que suas instâncias desempenham quando fazem parte de uma associação representada pelo elemento *Association*.
Tag	É o elemento que viabiliza uma alteração na forma de instâncias de elementos do modelo MOF, no momento de mapeamentos tecnológicos previstos na especificação.

Temos diversos instrumentos que permitem definir e manipular metamodelos baseados em MOF no qual citamos, por exemplo, UML2MOF (MATULA, 2004). Este é um instrumento que permite converter um modelo UML num metamodelo MOF.

2.2.2 UML

A especificação UML (*Unified Modeling Language*) foi adotada pela OMG em 1997, como uma linguagem gráfica para visualizar, especificar, construir e documentar modelos de software (OMG, 2005a). Apesar de contribuir sensivelmente para o processo de desenvolvimento de software, esta especificação não define nenhuma metodologia ou processo para desenvolver software. Em sua versão 1.4 (OMG, 2005a), a UML apresenta em sua estrutura: a semântica da linguagem UML, um guia da notação (sintaxe da linguagem UML), alguns exemplos de Perfis UML (mecanismo de extensão), Intercâmbio de Modelos (via XMI) e uma Linguagem de restrições de Objetos (OCL – *Object Constraint Language*). Embora já exista a versão UML 2.0, nós preferimos usar UML versão 1.4 no nosso trabalho, pois a mesma mostrava-se a versão mais estável e utilizada no início de nossas pesquisas. Contudo, o princípio apresentado nesta dissertação pode ser facilmente aplicado a UML 2.0.

A Figura 2.5 apresenta um fragmento do metamodelo da UML versão 1.4. A descrição dos aspectos semânticos da linguagem está intimamente relacionada com a descrição da arquitetura de metamodelagem em que o metamodelo UML está inserido e as formalizações necessárias para descrevê-lo.

O elemento *Classifier* da linguagem UML é especializado em *Class*, o elemento *BehavioralFeature* é especializado em *Operation*, o elemento *Structurefeature* é especializado em *Attribute* e o elemento *Feature* é especializado em *Structurefeature* e *BehavioralFeature*.

A primeira forma de extensão, através de perfil, é mais facilmente implementável. Um perfil UML é um pacote de elementos relacionados com extensibilidade que capturam padrões de variação e uso para um domínio específico. Como exemplo de perfil UML temos o perfil UML EDOC (OMG, 2004). A linguagem OCL é uma linguagem textual para descrever restrições que é usada na especificação UML. Trata-se de uma linguagem baseada em expressões que estão limitadas ao escopo do modelo ao qual fazem referência. O padrão XMI é um padrão de troca de modelos e metadados utilizado entre várias ferramentas, repositórios e *middlewares* que necessitam de um alto grau de interoperabilidade. O XMI é o caminho padrão para mapeamento de objetos em XML.

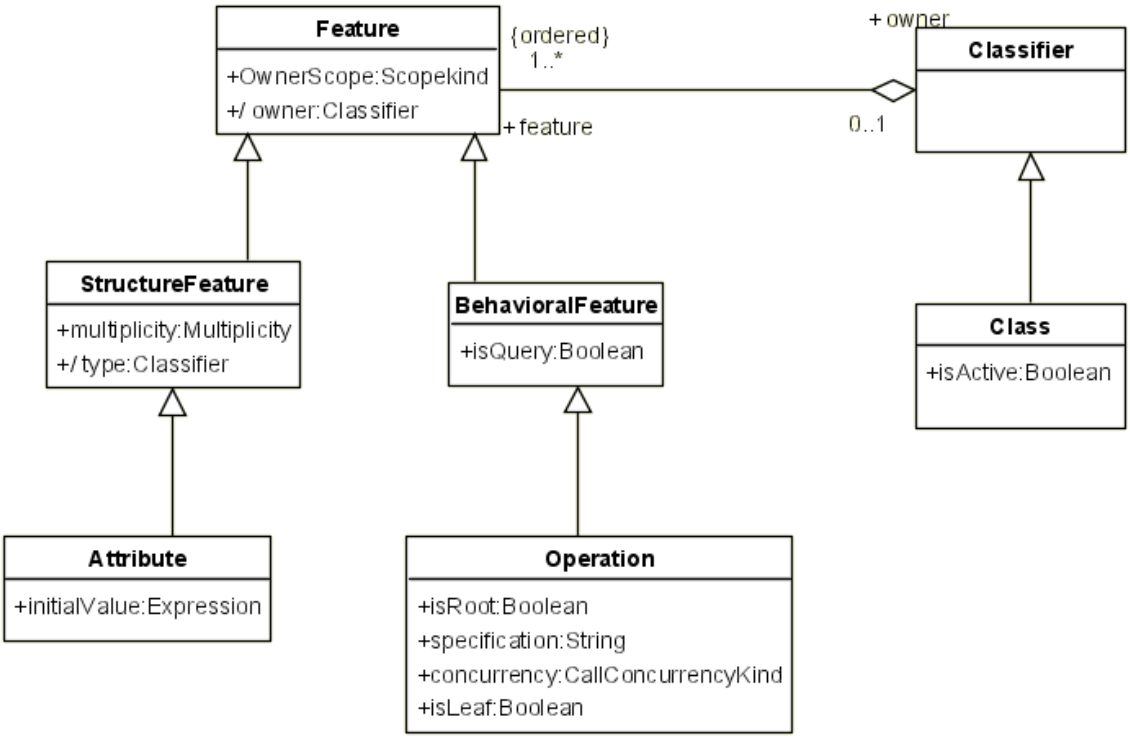

Figura 2.5: Fragmento do metamodelo UML

A arquitetura de metamodelagem é a mesma apresentada na seção 2.2.1 sobre MOF e o nível em que o metamodelo UML está inserido é o M2.

Como exemplo de uso conjunto do metamodelo UML e do metametamodelo MOF, a Figura 2.6 mostra como ambos podem se relacionar para a criação de um elemento de um modelo UML.

Na Figura 2.6, podemos observar que os elementos do metamodelo UML são representados, dentro de um repositório, por artefatos do metametamodelo MOF, provavelmente em tempo de compilação. E uma vez disponibilizada em uma ferramenta, os usuários podem fazer uso de ícones (de uma interface gráfica) representando os artefatos de modelagem para criar instâncias dos componentes do metamodelo UML dentro de um novo modelo definido pelo usuário projetista. Dentro da instância de artefato *Account*, novas instâncias de outros elementos são criadas e associadas a *Account* para representar uma classe com um atributo (*name*) e dois métodos (*balance()* e *getName()*). Quando este modelo é transformado em um programa para fazer parte do nível de informações, ele é representado por um objeto com valores inicializados internamente.

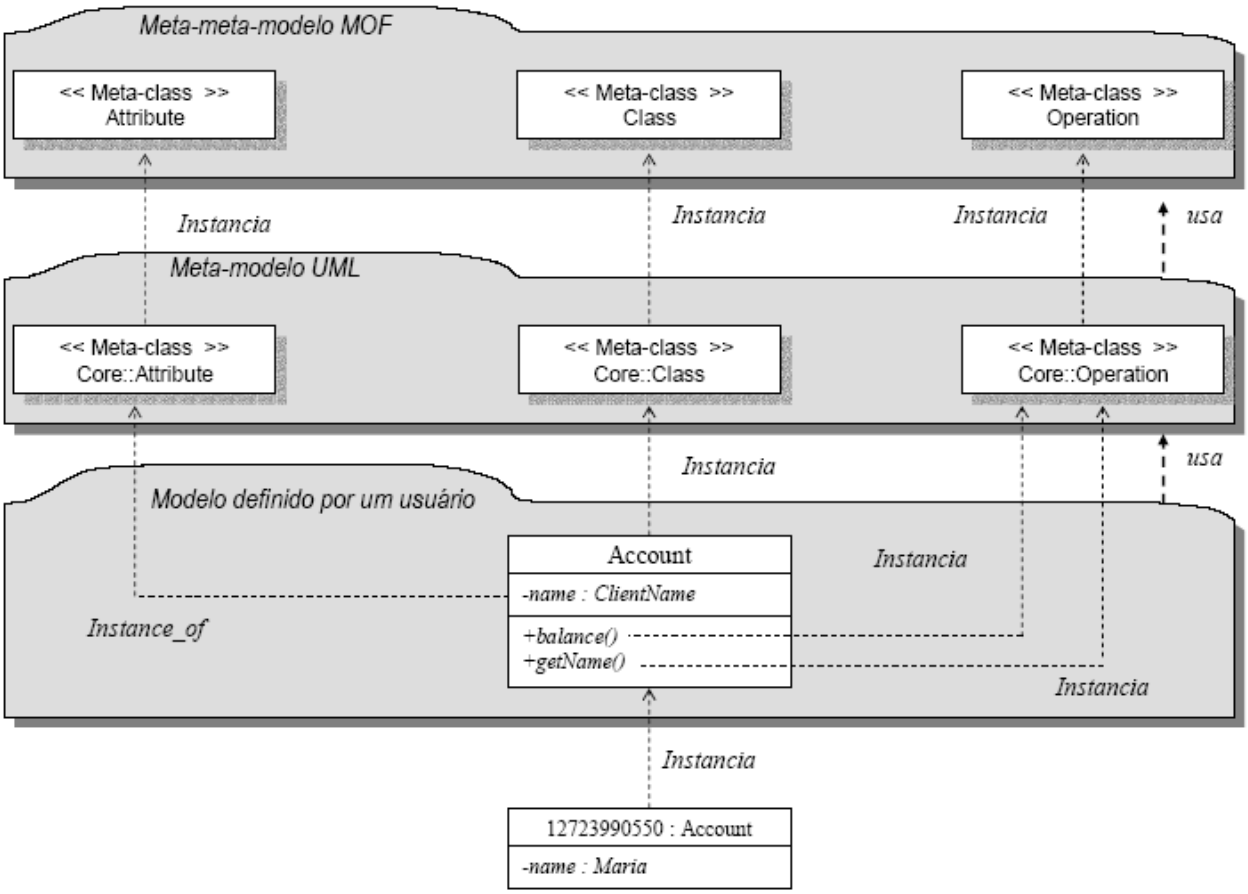

Figura 2.6: Metamodelo UML na arquitetura MOF

Uma importante diferença existente entre as abordagens UML e MOF está relacionada ao fato de que a partir do MOF é que o metamodelo UML é definido. A UML é a linguagem utilizada pelo usuário MDA para a representação de modelos (PIM e PSM). Entretanto, através da utilização de MOF, pode-se propor novos metamodelos mais adaptados a um domínio ou plataforma.

2.2.3 Modelos

Os modelos ocupam o papel central no desenvolvimento baseado em MDA. É principalmente através da manipulação dos modelos que o sistema sendo desenvolvido pode ser compreendido, projetado, construído, implantado e modificado. MDA define três principais modelos:

- Modelo Independente de Computação (CIM);
- Modelo Independente de Plataforma (PIM);
- Modelo Específico da Plataforma (PSM).

A próxima seção apresenta o conceito de CIM, PIM e PSM no contexto em que está inserido.

Modelos Independentes de Computação (CIM)

Um CIM é um modelo que captura os requisitos de negócio de um sistema em uma linguagem própria do negócio e voltada para os usuários do sistema. Isto fornece uma base para o projeto funcional e um meio importante para as pessoas do negócio validarem um projeto (HENDRYX & ASSOCIATES, 2005).

Modelos Independentes de Plataforma (PIM)

O PIM descreve o sistema, porém não apresenta os detalhes da tecnologia que será usada na implementação. O PIM detalha características estruturais e comportamentais do sistema que são independentes da sua implementação em uma plataforma específica. O modelo independente de plataforma é a visão de um sistema desatrelado das características de uma plataforma específica.

Modelos Específicos da Plataforma (PSM)

Modelo específico de plataforma é a visão de um sistema voltado ou desenvolvido para uma plataforma específica, tal como Java, CORBA, C++, etc. O PSM combina a especificação do modelo PIM com detalhes específicos de uma determinada plataforma.

2.2.4 Mapeamentos e Transformação de modelos

Um dos pontos chaves na abordagem MDA é a noção de mapeamento. Através dos mapeamentos são estabelecidas as correspondências entre um elemento de um modelo e um outro elemento correspondente de um outro modelo. Este é um importante aspecto no que diz respeito a equivalências entre os modelos. Mapeamento é um conjunto de regras e técnicas usadas para modificar um modelo obtendo-se outro (COSTA *et al.*, 2003) . Os mapeamentos são utilizados pelas transformações entre PIM e PSM (OMG, 2006).

A transformação de modelos em MDA é o processo pelo qual a partir de especificações com um alto nível de abstração e independência de plataforma seja possível gerar, de forma automática, sistemas completos para múltiplas plataformas.

A transformação de modelos consiste no processo de converter um modelo em outro modelo do mesmo sistema (OMG, 2006). A idéia principal é construir modelos no seu mais alto nível de abstração e transformá-los em modelos com um menor nível de abstração, de forma automática ou semi-automática, facilitando e tornando o processo de desenvolvimento mais rápido. A Figura 2.7 mostra um exemplo de transformação de modelos.

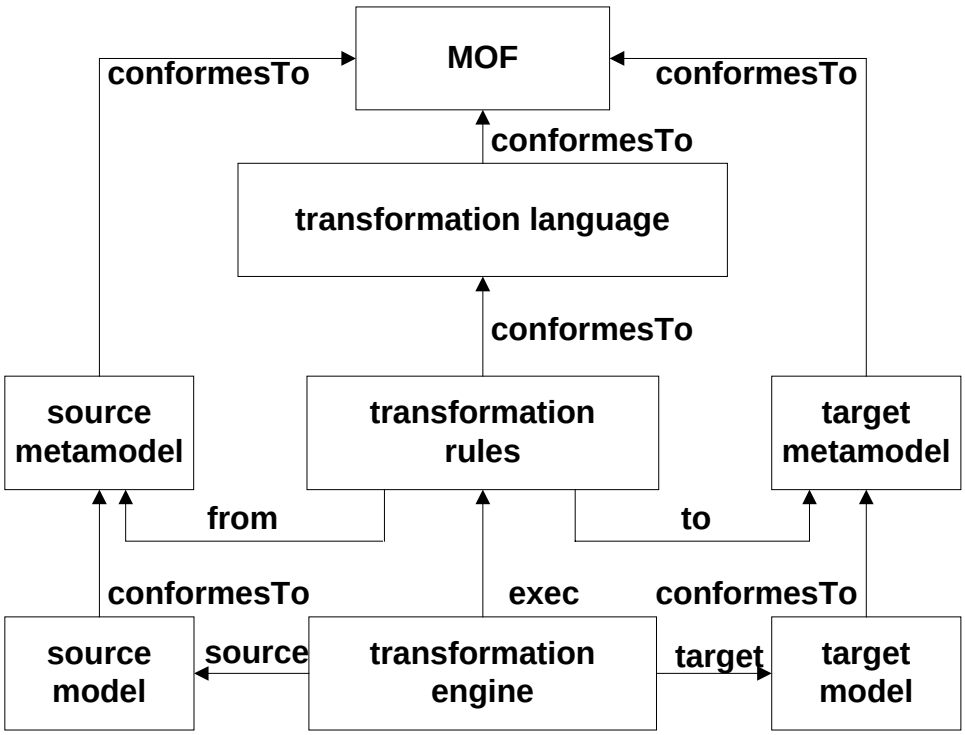

Figura 2.7: A transformação de modelos em MDA (BEZIVIN et al., 2004a)

Cada elemento mostrado na Figura 2.7 tem uma função importante em MDA para compreender os elementos de uma transformação e as relações entre eles. O MOF é o metametamodelo usado para construir metamodelos. O PIM (*source model*) reflete as funcionalidades, a estrutura e o comportamento do sistema. O PSM (*target model*) é voltado para implementação e corresponde a primeira fase de ligação do PIM para a plataforma de execução. O PSM não é a implementação final, mas tem informações suficientes para gerar arquivos de interface, um código de linguagem de programação, uma linguagem de definição de interfaces, arquivos de configuração e outros detalhes da implementação (BEZIVIN *et al.*, 2004a).

Dois ou mais elementos de diferentes metamodelos são equivalentes se eles são compatíveis e se eles não podem contradizer um ao outro (BEZIVIN *et al*., 2004a). A Transformação de modelos é realizada por um modelo de transformação que executa regras de transformação. E as regras de transformação especificam como gerar um modelo alvo (PSM) de um modelo fonte (PIM) (BEZIVIN *et al.*, 2004a).

É importante notar, que não é sempre possível conseguir o automatismo das transformações. Para que isso seja possível, é necessário que haja uma firme conexão entre os elementos nos modelos, com regras claras e não ambíguas.

2.2.5 Ferramentas de Suporte ao MDA

Aqui, iremos mostrar algumas ferramentas muito importantes no suporte ao desenvolvimento em MDA.

ATL

ATL (*Atlas Transformation Language*) é uma linguagem de transformação desenvolvida como parte da plataforma AMMA (*ATLAS Model Management Architecture*) para a realização de transformações de modelos no contexto de MDA (AMMA, 2006) (BÉZIVIN *et al.*, 2003) (BÉZIVIN *et al.*, 2005). Esta linguagem é baseada em OCL e usa um repositório MDR ou EMF para armazenar e manipular os meta-modelos fonte (PIM) e alvo (PSM) para executar a transformação seguindo os mapeamentos definidos, usando regras de transformação. A ATL é uma linguagem simples e permite manipulações com elementos de metamodelos e modelos.

ATL é aplicada no contexto do padrão de transformação como mostrado na Figura 2.8. Neste padrão, uma modelo fonte **Ma** é transformado em um modelo alvo **Mb** de acordo com uma definição de transformação **mma2mmb.atl** escrito na linguagem ATL. Estes três elementos são modelos respectivamente conforme ao MMa, MMb e metamodelos da ATL. Todos o metamodelos são conforme ao metametamodelo MOF no contexto de padrões da OMG. Os modelos e meta modelos fonte e alvo podem ser expressos em XMI (JOUAULT *et al.*, 2006).

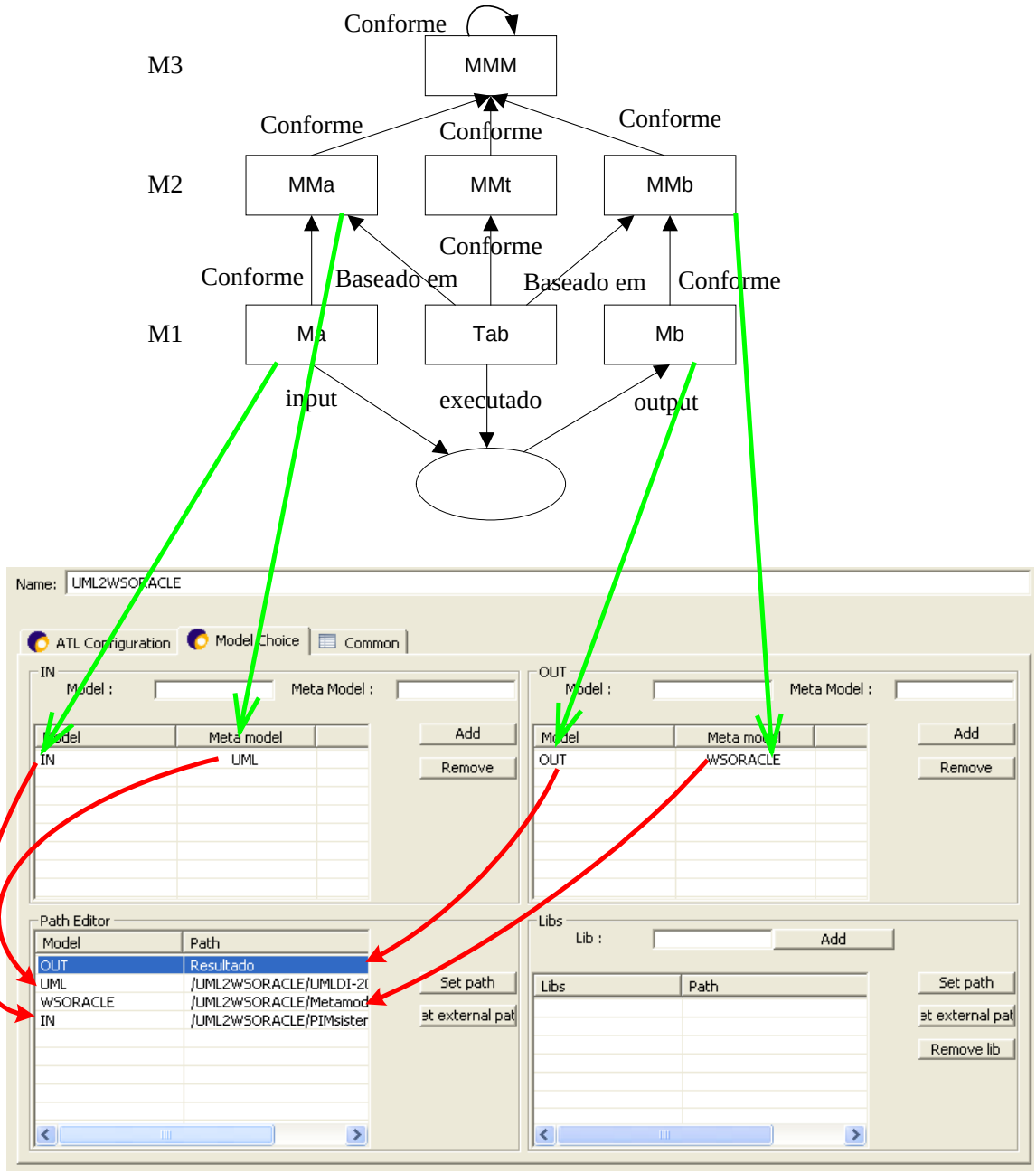

Figura 2.8: A transformação de modelos usando ATL (JOUAULT *et al.*, 2006).

MDR

O MDR (*Metadata Repository*) (MATULA, 2003) implementa um repositório de metadados baseado no padrão MOF da OMG e está sendo desenvolvido como parte do projeto NetBeans. Este inclui a implementação de um repositório MOF com um mecanismo de armazenamento persistente para o armazenamento de metadados. A interface do repositório MOF é baseada no JMI. O MDR também define características adicionais que ajudam a incorporá-lo em um IDE. Assim,o MDR, está habilitado a carregar qualquer metamodelo MOF (descrição de metadados) e armazenar instâncias deste metamodelo

(metadados conforme o metamodelo). Metamodelos e metadados podem ser importados ou exportados de/para o MDR usando o padrão XMI. Os metadados no repositório podem ser gerenciados através de rotinas usando uma API JMI.

2.3 *Web Services*

Atualmente, a necessidade de compartilhamento de informações tem se tornado um fator crítico, devido a globalização, a popularização de PCs (*Personal Computers*) e avanços tecnológicos.

A popularização da Internet e a evolução da infra-estrutura de redes, que permitem que dados trafeguem em altas velocidades, possibilita que aplicações distribuídas sejam utilizadas de forma cada vez mais natural.

Diversas tecnologias estão sendo propostas e aplicadas nessa área de pesquisa, hoje, uma das mais mencionadas é a tecnologia de *Web Services*. Os *Web Services* surgiram como uma solução para tentar melhorar a comunicação entre sistemas distribuídos.

Os *Web services* são serviços oferecidos por uma aplicação para outras aplicações via Web (W3C, 2006). Os clientes desses serviços podem agregá-los para formar um software final, que possibilita transações comerciais ou criar um novo *Web Services* (SUN MICROSYSTEMS, 2006). Uma empresa pode ser fornecedora de *Web Services* e também consumidora de outros serviços.

Um *Web Service*, portanto, é um componente de *software* ou uma unidade lógica de aplicação que se comunica através de tecnologias padrões de Internet (IWEB, 2003). Esses componentes provêem dados e serviços para outras aplicações. As aplicações acessam os *Web Services* através de protocolos e formatos de dados padrões, como HTTP, XML, SOAP e UDDI.

As vantagens de se utilizar a abordagem de *Web Services* são:

- Simplicidade: é mais simples de se implementar que as soluções tradicionais que utilizam CORBA ou DCOM;
- Padrões abertos: utilizam padrões abertos como HTTP, SOAP, UDDI, ao invés de tecnologias proprietárias;
- Flexibilidade: alterações nos componentes são muito mais simples para o sistema como um todo do que alterações nos adaptadores tradicionais;
- Escopo: cada sistema pode ser tratado de maneira individual, já que para transformá-lo em um componente basta implementar uma camada que o encapsule. Na abordagem

tradicional, todos os sistemas devem ser tratados ao mesmo tempo, já que farão parte da mesma solução monolítica de integração.

2.3.1 Tecnologias de Apoio aos *Web Services*

As principais tecnologias dos *Web Services* são o XML (*eXtensible Markup Language*) (W3C, 2003), SOAP (*Simple Object Access Protocol*) (GUDGIN, et al., 2003), WSDL (*Web Services Description Language*) (CHRISTENSEN et al., 2003) e UDDI (*Universal Description, Discovery and Integration*) (OASIS-UDDI, 2005). Segue uma breve descrição sobre cada uma destas tecnologias.

XML

XML é uma linguagem muito parecida com HTML (*Hyper Text Markup Language*), mas com a possibilidade de definir *tags*, que são marcações de dados (*meta–markup language*) (W3C, 2003). Essas tags fornecem um formato para descrever dados semi-estruturados. O XML provê um sistema para criar dados diferentes do HTML que define apenas formatação de textos e caracteres.

A principal utilização de XML está relacionada à representação de dados, documentos e demais entidades cuja essência fundamenta-se na capacidade de agregar informações.

XML é essencial para *Web Services* porque permite solucionar um problema crucial de interoperabilidade existente em outras linguagens, pois a XML permite que cada *host* contenha sua própria linguagem, mas que a comunicação seja efetuada de forma universal, podendo ser compreendida por qualquer sistema. A Figura 2.9 apresenta um trecho de um documento XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <Laudo>
        <Paciente matrícula=”123”>
              <Nome> João da Silva </Nome>
        </Paciente>
        <Conclusões>
              <LesãoArterial>
                    <Grau> 40% </Grau>
                    <Localização> 1/3 proximal </Localização>
                    <Artéria> 2a. Marginal </Artéria>
              </LesãoArterial>
        </Conclusões>
</Laudo>
```

Figura 2.9: Exemplo de um documento XML.

WSDL

WSDL é uma linguagem baseada em XML que é utilizada para descrever um *Web Service* (CHRISTENSEN *et al.*, 2001).

Um *Web Service* deve, portanto, definir todas as suas interfaces, operações, esquemas de codificação, entre outros neste documento. Um documento WSDL define um *XML Schema* para descrever um *Web Service*.

WSDL é basicamente composto de:

- Types - descreve tipos de dados abstratos.
- Message descreve a estrutura da mensagem.
- PortType - apresenta a Interface do Web Service.
- Binding - indica como o serviço é acessado.
- Service - descreve quem provê o serviço.

A Figura 2.10 apresenta a estrutura de um arquivo WSDL.

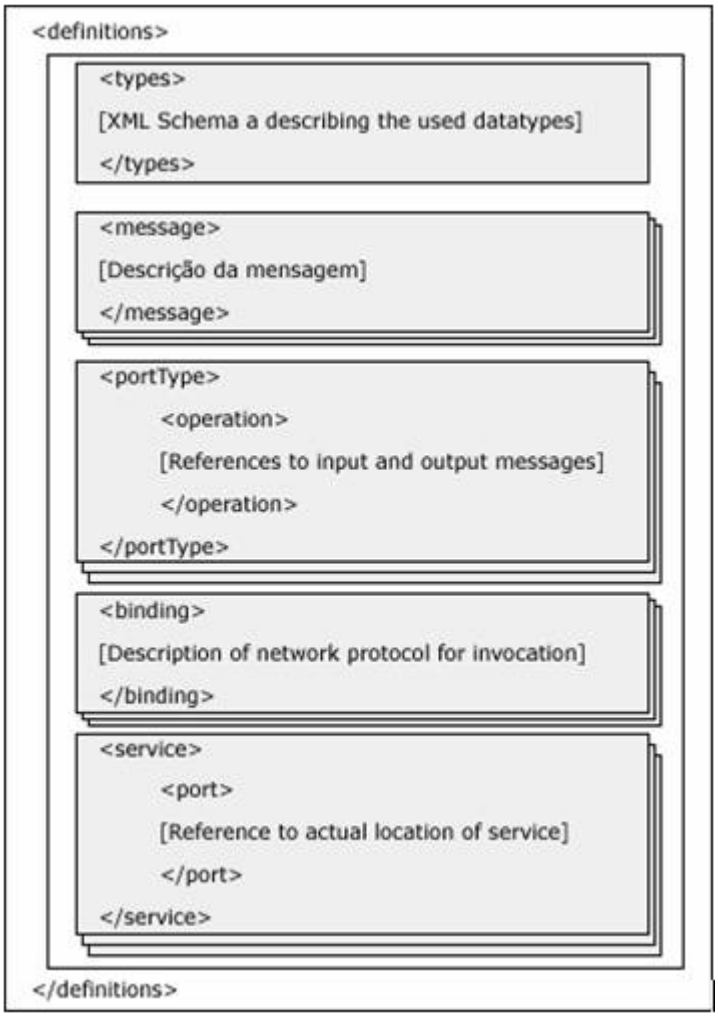

Figura 2.10: Estrutura do Arquivo WSDL

SOAP

SOAP é um protocolo que *define* uma estrutura padrão de interoperabilidade, um mecanismo de RPC *(Remote Procedure Call)*, um *framework* para empacotar e desempacotar mensagens XML entre o Servidor e o Cliente (GUDGIN, *et al.*, 2003).

Apesar de ter sido inicialmente concebido como a tecnologia para transpor a lacuna entre plataformas baseadas em comunicação RPC, SOAP se tornou em um dos mais conhecidos formatos de mensagens e protocolo utilizado por Web Services baseados em XML. Por este motivo, o acrônimo SOAP é referido freqüentemente como *Service-Oriented Architecture Protocol* (protocolo de arquitetura orientada a serviços) ao invés de *Simple Object Access Protocol*.

Normalmente, os envelopes, nome dado às mensagens SOAP, são transmitidos via HTTP (W3C, 2002) ou SMTP (POSTEL, 2006) devido ao seu fácil acesso, mas é possível transmitir SOAP sob praticamente qualquer protocolo.

Uma mensagem SOAP, ou um envelope, representado na Figura 2.11, consiste basicamente em:

- Um cabeçalho composto por zero ou mais entradas;
- Um corpo composto por zero ou mais entradas;
- Zero ou mais elementos adicionais.

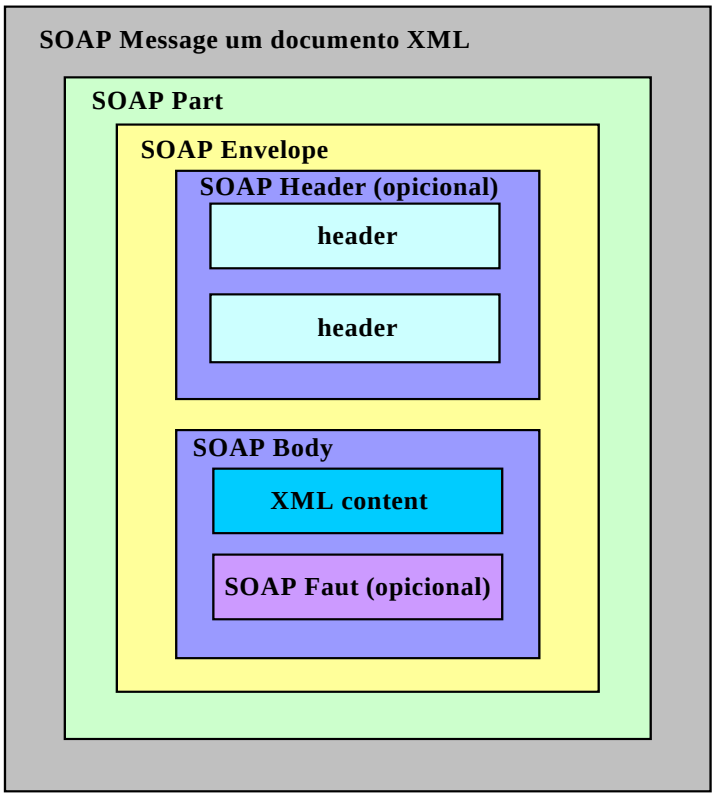

Figura 2.11: Mensagens SOAP

A especificação SOAP estabelece um formato padrão de mensagem, que consiste em um documento XML, capaz de hospedar dados RPC e centrados em documentos. Isto facilita o intercâmbio de dados de modelos síncronos e assíncronos.

UDDI

O UDDI (*Universal Description, Discovery, and Integration)* é uma especificação técnica que tem como objetivo descrever, descobrir e integrar *Web Services*. É um elemento central do grupo de padrões que compõe a pilha de componentes dos *Web Services*. UDDI corresponde a um *Web Service registry* que provê um mecanismo para busca e publicação de *Web Services* (OASIS-UDDI, 2005).

UDDI age como um diretório de telefone na Internet para registrar e encontrar *Web Services*.

Figura 2.12: Estrutura de UDDI (OASIS-UDDI, 2005).

O UDDI administra a informação sobre os *Web Services* na Internet da seguinte maneira, como mostra a Figura 2.12:

- As companhias que fornecem serviços registram a informação sobre esses serviços no registro de negócio de UDDI;
- Os usuários de serviço procuraram pela informação de serviço na UDDI que eles requerem, então fazem uso do serviço desejado.

A idéia é que no momento que se publica um Web Service, o mesmo já deve estar disponível para que qualquer cliente possa acessá-lo através da Internet. Geralmente, a maneira mais utilizada é fazer com que a aplicação cliente conheça a URL do serviço.

2.3.2 SOA

SOA (*Service-Oriented Architecture*) ou arquitetura orientada a serviços tem como seu componente fundamental o conceito de serviços. Estes Serviços podem ser compostos em Processos de Negócio, permitindo agilizar os mesmos e lidar com a sua dinâmica (APACHE TOMCAT, 2005).

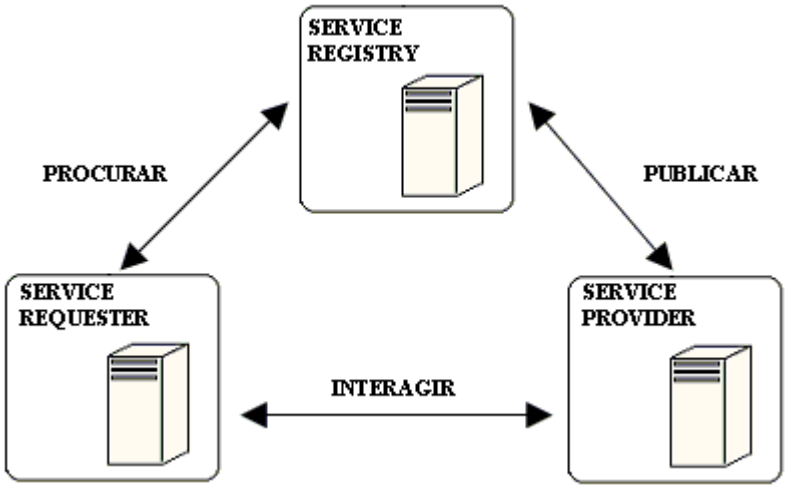

Figura 2.13: Arquitetura básico de um *Web Services* (APACHE TOMCAT, 2005)

Basicamente uma SOA é baseada em um modelo simples envolvendo três entidades como mostra a Figura 2.13:

O *Service Requester* é a aplicação que solicita um serviço. A aplicação recorre através dos *Service Registries* as informações referentes à localização do serviço. O *Service Registry* é uma aplicação que retorna as informações para uso de um serviço.

O *Service Provider* funciona de forma semelhante a um sistema de páginas amarelas, oferecendo os serviços para que possam ser encontrados de forma fácil.

Um bom exemplo para este cenário é um *E-commerce* de uma Livraria. A requisição é realizada através de um cliente. O cliente que pode ser um *browser* qualquer, solicita o preço de um determinado livro. O *Web Services* recebe e processa este pedido e retorna uma resposta contendo o preço do livro.

Web Services e consumidores de *Web Services* são geralmente associados a negócios do tipo B2B. Isto acontece quando a empresa que provê o serviço também é cliente de outro serviço. Os Web Services são projetados para suportar interação e interoperabilidade podendo ser, ou não, de arquiteturas diferentes.

Existem cinco itens básicos nesta arquitetura:

- HTTP: (*Hipertext Transport Protocol*) usado para transportar as informações pela web;
- XML: (*Extensible Markup Language*) é utilizada na definição e semântica dos dados.
- SOAP: (*Simple Object Access Protocol*) é o formato de mensagens para chamada de métodos remotos.
- WSDL: (*Web Services Description Language*) descreve as características oferecidas pelo serviço.
- UDDI: (*Universal Discovery, Description and Integration*) descreve um tipo especial de registro que lista os *Web Services* na Internet.

Em linhas gerais, a SOA faz com que toda e qualquer aplicação só saia do papel se estiver devidamente atrelada a um processo de negócio já existente, o que traz agilidade para solucionar as demandas, redução de custos com desenvolvimento, simplicidade, padronização, reaproveitamento e compartilhamento de software.

SOA é uma revolução organizacional e uma evolução técnica. Podemos perceber isto no Tabela 2.2 que apresenta comparativo entre o desenvolvimento baseado em componentes e SOA.

Tabela 2.2 Comparativo entre o desenvolvimento baseado em componentes e SOA (MARQUES, 2005).

Baseada em Componentes	Baseada em Serviços
Modelo procedimental (orientado à invocação de funções)	Modelo "colaborativo"/*event-driven* (orientado às chamadas para fornecimento/consumo de serviços)
Produto mais rígido (build to last)	Produto mais flexível (build to change)
Ciclos de desenvolvimento longos	Desenvolvimento (e deployment) incremental
Favorece o desenvolvimento de aplicações isoladas e independentes ("nichos" aplicacionais)	Favorece o desenvolvimento de soluções integradas
"Partes" fortemente ligadas (tightly coupled)	"Partes" fracamente ligadas (loosely coupled)
Modelo de comunicação orientada ao objeto	Modelo de comunicação orientado a "mensagens"
Foco na implementação ("como faz")	Abstração ("o que faz")

Desta forma, SOA é um novo paradigma de desenvolvimento de aplicações, que permite expor a interface da Empresa para o mundo exterior. É uma nova abordagem de desenvolvimento de aplicativos relacionada à modelagem de processos de negócio.

2.3.3 Benefícios dos *Web Services*

Os *Web Services* trazem diversos benefícios quando utilizados de maneira correta e bem sucedida. Os serviços são baseados em um conjunto de padrões da Internet definidos pelo W3C (CHRISTENSEN *et al*., 2003), não requerem configurações especiais nos *firewalls*, pois o protocolo http, o qual é o mais utilizado, atua como transporte na comunicação entre cliente e Web Service.

Além disso, possuem independência de plataforma, devido a ser baseado em XML, o qual pode gerar documentos complexos. Outra característica que deve ser destacada é o re-uso dos componentes pertencentes aos sistemas integrados, onde cada componente pode representar um serviço distinto, podendo participar de múltiplos sistemas provendo maiores benefícios imediatos e aumento da agilidade do negócio.

2.3.4 Plataformas para desenvolvimento de *Web Services*

A disseminação no uso de *Web Services* nos últimos anos incentivou o mercado a oferecer uma grande variedade de ferramentas e aplicações para prover suporte a essa tecnologia. Atualmente, as principais plataformas para *Web Services* são: JWSDP, WebSphere, AXIS e ".NET". A seguir, encontra-se um resumo dos principais fornecedores citados e seus produtos.

. NET

A Microsoft disponibilizou ferramentas para a tecnologia de W*eb Services* no seu pacote de produtos do framework ".NET". Segundo a Microsoft, ".NET" é uma plataforma de *software* que conecta sistemas e dispositivos através de várias tecnologias, permitindo o acesso às informações e possibilitando que o usuário interaja com estes dispositivos inteligentes através da *Web*.

Assim, ".Net" é basicamente um conjunto de XML *Web Services* que possibilita que sistemas e aplicativos, novos ou já existentes, conectem seus dados e transações independente

do sistema operacional, do tipo de computador, do dispositivo móvel ou de qual linguagem de programação tenha sido utilizada na sua criação (MICROSOFT, 2006).

AXIS

Apache AXIS é um projeto *open source* para um servidor e cliente SOAP. A Apache (APACHE, 2005) define o AXIS (*Apache Extensible Interaction System*) como um mecanismo de comunicação SOAP. Ele pode ser usado como uma biblioteca do cliente para invocar os serviços SOAP disponíveis em outra máquina ou como uma ferramenta no lado do servidor para executar serviços acessíveis pelo SOAP.

Dessa forma, no lado do cliente, ele fornece uma API para invocar serviços do SOAP RPC emitindo e recebendo mensagens através do protocolo SOAP. E no lado do servidor, tem-se um mecanismo para escrever e acessar serviços RPC ou serviços de mensagens, que devem ser hospedados por um *servlet container* (tal como no *Apache Tomcat*).

O AXIS permite a separação entre os problemas de infra-estrutura e o processamento do negócio devido ao fato da execução das tarefas de autenticação e autorização serem executadas em seqüência antes de chamar os serviços de negócio. E propicia o processamento específico de uma mensagem SOAP em uma seqüência de operações através da aplicação de um modelo modular que aceita as evoluções das especificações sem causar grandes impactos na arquitetura da plataforma.

O *Apache* AXIS em conjunto com o *Apache Tomcat* e o *Java Development Kit* (JDK) fornecem um conjunto suficiente de utilitários destinados a realizar todas as etapas de desenvolvimento de um projeto sem precisar de bibliotecas adicionais.

Sun ONE

De acordo com a Sun, Sun ONE (*Sun Open Net Environment*) é uma plataforma baseada em padrões abertos que fornece um conjunto de práticas para construir e disponibilizar *Web Services*. E tem como recursos chaves as tecnologias XMl e Java para prover interoperabilidade entre as aplicações. A XML fornece estruturas de dados padrão e neutras com relação à plataforma para representar os dados contextuais, e o Java fornece um conjunto de interfaces neutras em relação à plataforma para acessar e usar essas informações (SUN, 2006).

A sua arquitetura baseia-se em padrões abertos como: UDDI, SOAP, XML e Java. A sua plataforma é composta pelo ambiente operacional Solaris; do *software iPlanet* formado por servidores de aplicações, localização, Web, comércio e comunicações; e do ambiente de desenvolvimento integrado. Além de oferece uma plataforma para Internet integrável, escalável e de custo acessível, foi desenhada para integração simples com hardwares e softwares existentes, independente de plataforma ou fabricante.

JWSDP

O JWSDP (*Java Web Services Developer Pack*) é um conjunto de ferramentas integradas e gratuitas que permite desenvolvedores Java implementarem e testarem aplicações *Web* com XML e *Web Services*. O JWSDP engloba as APIs de Java para XML, JAXB (*Java Architecture for XML Binding*), *JavaServer Faces, Web Services Interoperability Sample Application, Web Services Security,* JSTL (*JavaServer Pages Standard Tag Library*)e *Java WSDP Registry Server*. As ferramentas *Ant Build Tool* e *Apache Tomcat Container* também fazem parte deste pacote distribuído pela Sun.

Recentemente, JWSDP foi transferido da Sun para ser gerenciado pelo projeto *GlassFish*.

WebSphere

A IBM é um fornecedor altamente diversificado de tecnologias e ferramentas para web services e servidores de aplicação. O IBM WSDK (*WebSphere Software Developer Kit*) para *Web Services* (IBM, 2005) é um conjunto de ferramentas para a criação, busca, invocação e testes para *Web Services*. O WSDK versão 5.0.1 encontra-se em conformidade com as últimas versões das especificações para *Web Services* incluindo WS-Security, SOAP, WSDL e UDDI. Entre as principais ferramentas de desenvolvimento, podem ser citadas *WebSphere Studio Application Developer* (IDE), *Web Services Gateway, Web Services Invocation Framework* e o *UDDI Explorer*.

JDeveloper

O *Oracle JDeveloper 10g* é um ambiente de desenvolvimento integrado com suporte end-to-end para modelagem, desenvolvimento, depuração, otimização e implantação de aplicações Java e *Web Services* [29]. Oracle JDeveloper 10*g* é uma nova abordagem para o

desenvolvimento J2EE com funcionalidades que possibilitam o desenvolvimento visual e declarativo. O inovador Oracle Application Development Framework simplifica o desenvolvimento em J2EE. Oracle JDeveloper oferece uma escolha de abordagem de desenvolvimento, escopo de tecnologia e plataforma de desenvolvimento.

2.4 Trabalhos Relacionados

Nesta seção, nós mostramos alguns trabalhos relacionados a nossa pesquisa. Primeiramente, os trabalhos correlacionados da área médica são apresentados e em seguida os trabalhos correlacionados a MDA.

2.4.1Trabalhos Relacionados à Área Médica

A seguir, uma análise de alguns trabalhos relacionados a área médica é feita. Sendo que suas principais características são destacadas. Os trabalhos apresentados são: **MIDster** que é um sistema de compartilhamento de imagens médicas baseado em modelos P2P e *Web Services* (PISA *et al.,* 2004), **HealthNet** é um sistema de apoio a segunda opinião médica, encontra-se em desenvolvimento uma segunda versão baseada em *Web Services* (BARBOSA, 2001), (BARBOSA *et al.*, 2003) e **SET-WS** é um sistema especialista para telediagnóstico no ambiente de *Web Services* (FILHO *et al.*, 2004).

MIDster

O projeto MIDster (PISA *et al.*, 2004) acrônimo para *P2P Web Service Medical Image Distributed System*, propõe um sistema de compartilhamento de imagens médicas baseado em modelos P2P e *Web Services*. A arquitetura MIDster oferece um relacionamento entre usuários e seus recursos em redes Intranet/Internet que definem uma plataforma de conhecimento de coleções de imagens médicas.

A Figura 2.14 mostra a arquitetura do sistema MEDster. Essa arquitetura é centrada na disponibilidade de um *Web Services* pelo qual um cliente autorizado pode interagir diretamente com os demais clientes conectados simultaneamente. Entre as funcionalidades descritas na arquitetura MIDster incluem-se:

- O Mecanismo de busca de imagens médicas padrão DICOM;

- Comunicação síncrona e assíncrona entre clientes conectados;
- Validação dos clientes que participam da rede de compartilhamento;
- Controle de versão dos programas utilizados na camada do servidor.

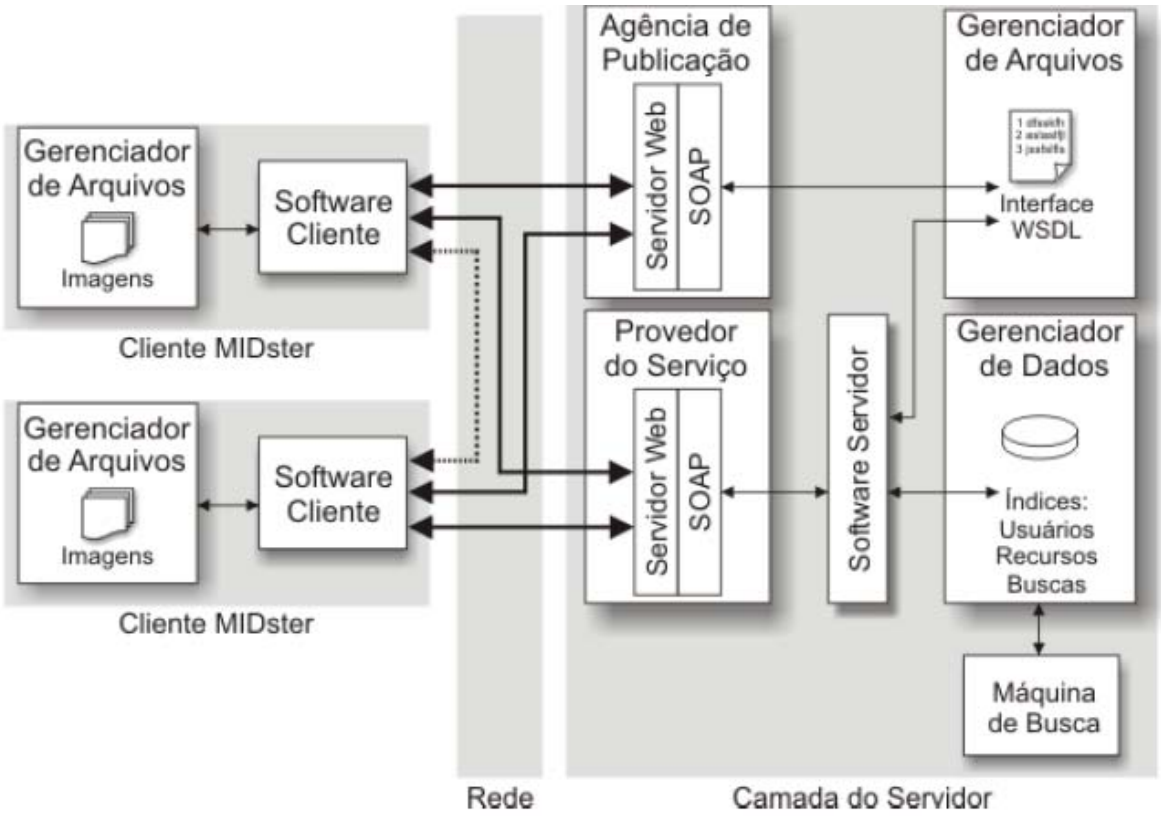

Figura 2.14: Arquitetura do sistema MIDster

Assim, MIDster suporta produção modular de software; encoraja a reutilização de código; permite a integração de diferentes linhas de programadores, sistemas operacionais e hardwares; e possibilita longevidade na manutenção.

Esse projeto MIDster foi desenvolvido no laboratório ImagCom (http://imagcom.org) pertencente ao Departamento de Física e Matemática (DFM), Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto (FFCLRP), Universidade de São Paulo (USP). Como outros projetos do grupo ImagCom, este também reflete o objetivo do grupo que é estudar padrões, tecnologias e metodologias que sejam adequadas para gerar soluções ótimas para a área de processamento de imagens médicas, incluindo computação, representação, arquivamento, transmissão e recuperação dessas imagens.

HealthNet

O HealthNet (BARBOSA, 2001), (BARBOSA *et al.*, 2003) é um sistema de apoio à segunda opinião em saúde. Ele é atualmente um dos instrumentos de apoio ao Programa de Saúde da Família (PSF), em Pernambuco, a partir da Rede de NUTES (Núcleos de Telesaúde de Pernambuco). A primeira versão do sistema disponibiliza o serviço de telediagnóstico.

Este serviço possibilita que agentes de saúde, que se encontram em unidades de saúde, solicitem a profissionais de saúde especialistas de centros de referência, apoio no diagnóstico e conduta terapêutica de seus pacientes em um ambiente *store-and-foward* na Internet, utilizando uma base de dados centralizada.

Já na segunda versão do HealthNet, que se encontra em desenvolvimento, o conceito de segunda opinião é estendido, englobará o telediagnóstico e a cooperação em saúde através de serviços *on-line* e em tempo real. O objetivo da segunda opinião em saúde pode variar da busca em conjunto de um parecer final sobre o caso ou a simples discussão deste para fins educacionais.

Ainda na segunda versão do HealthNet, a abordagem Web continuará e fará uso dos serviços providos pelo Projeto Infravida. Este projeto coordenado pelo Centro de Informática da UFPE em colaboração com outros estados brasileiros, e financiado pelo CNPq, propõe a construção de uma infra-estrutura de vídeo digital para aplicações de telesaúde. A arquitetura proposta é baseada em Web Services onde os serviços serão acessados via interface descrita em WSDL (*Web Service Description Language*), que é baseada em XML.

Os serviços propostos pelo Infravida, e que serão integrados ao HealthNet, incluem: Controle de Acesso, Suporte ao Trabalho em Grupo, Vídeo sob Demanda, Videoconferência, Anotações em Vídeo e Integração de Dados. A nova arquitetura do HealthNet propõe, também, uma base de dados distribuída. Assim, cada parceiro de médio a grande porte poderá ter sua própria base dados que será gerenciada por um ou vários servidores centrais. Estas bases de dados poderão ser integradas com Prontuários Eletrônicos do Paciente (PEP) de instituições participantes. O projeto encontra-se em desenvolvimento e muito ainda deverá ser testado para sua completa validação. Principalmente em se tratando de questões com eficiência, confiabilidade e segurança serão avaliadas.

SET-WS

O SET-WS (Sistema Especialista para Telediagnóstico no ambiente de Web Services) (FILHO *et al.,* 2004) é um projeto baseado em várias bases de conhecimento usando JESS (*Java Expert System Shell*) elaboradas a partir de fatos reportados por especialistas. A aplicação em Java (Java Script / JSP / Servlet) faz a comunicação entre os usuários por meio de páginas utilizando scripts XML para passagem de mensagens aos servidores, onde tais aplicações encontram-se hospedadas e o ambiente de *Web Services* viabiliza a busca através de um protocolo semelhante a "páginas amarelas" pela rede Internet nas bases de regras que compõem o resultado para que este possa ser disponibilizado.

O sistema traz uma solução de diagnóstico que age colaborativamente, via sistemas especialistas em um ambiente de *Web Services*. O sistema tem as seguintes características:

- proposta de um ambiente interativo homem-máquina a partir de bases de conhecimentos;
- envolvimento das mais avançadas tecnologias para alcance de maior desempeno e confiabilidade;
- manutenção de um acesso amigável do cliente ao processo de telediagnóstico.

A maior contribuição do Sistema Especialista para Telediagnóstico usando a Tecnologia de *Web Services* é manter o foco nos resultados provenientes das bases de conhecimentos.

2.4.2. Trabalhos Relacionados a MDA

A seguir é feita também uma análise de alguns trabalhos relacionados ao mapeamento entre modelos, definição de transformação e transformação de modelos usando a abordagem MDA. Os trabalhos apresentados são: Um Framework de Transformação de Modelos para a Construção Automatizada de Modelos UML para Modelos de Desempenho (AMBROGIO, 2005), Aplicando a abordagem MDA para Aplicações de B2B (BÉZIVIN *et al.*, 2004b) Transformação de Modelo Independente de Plataforma baseado em Triple (BILLIG *et al.*, 2004).

Um Framework de Transformação de Modelos para a Construção Automatizada de Modelos UML para Modelos de Desempenho.

AMBROGIO (2005) introduz um *framework* de transformação de modelos para a modelagem de transformações baseada em padrões de modelos de software para modelos de desempenho. Especificamente, ela focaliza a transformação de modelos de UML para modelos de LQN. A transformação é especificada em nível de metamodelos e pode ser efetivamente usada para aumentar o grau de interoperabilidade entre ferramentas de desenvolvimento de software e ferramentas de análise de desempenho.

A Figura 2.15 ilustra o *framework* da transformação de modelos proposto, que é dividido nas seguintes camadas:

- A camada de metamodelagem na qual está dividida em três sub-camadas (M1, M2 e M3);
- A camada de implementação de modelos;
- A camada de ferramenta.

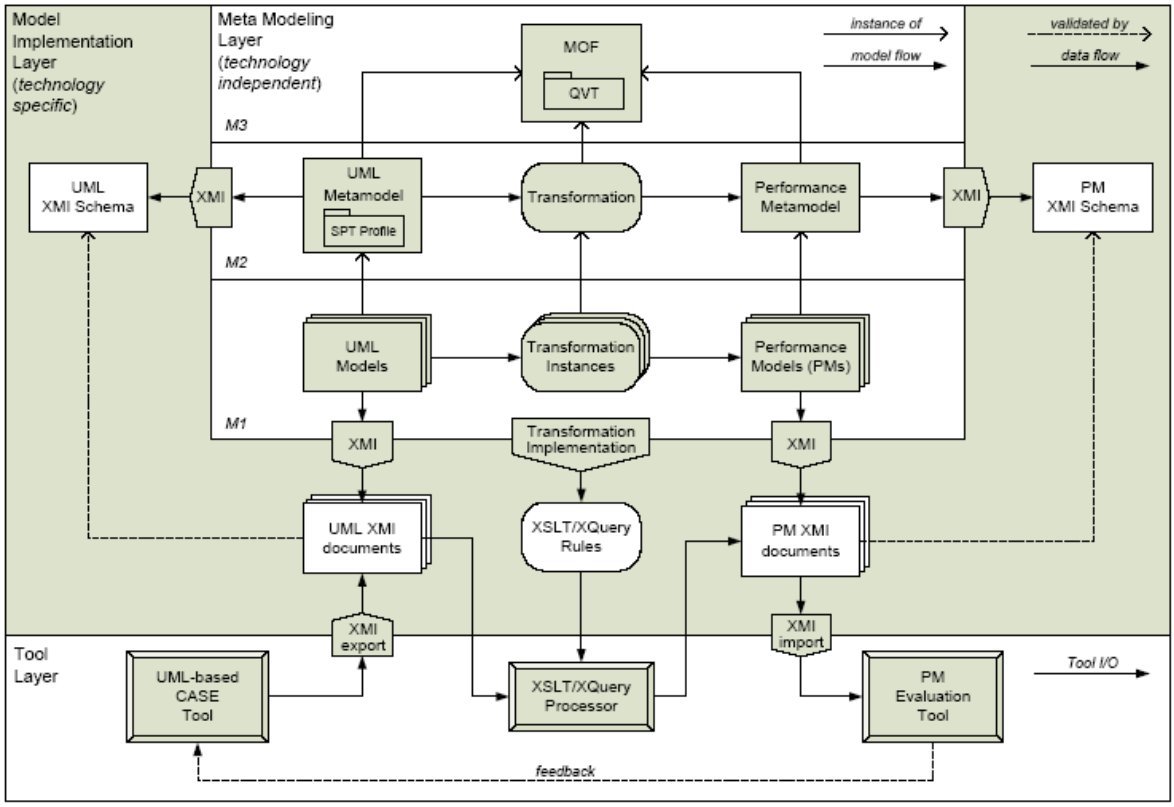

Figura 2.15: O *framework* para transformar modelos de UML para modelos de desempenho

As três sub-camadas da camada de metamodelagem são extensões das camadas da arquitetura de metadados MOF correspondente. A sub-camada M3 é a camada do modelo MOF que apóia a especificação QVT. A sub-camada M2 é uma extensão da camada de metamodelo UML que apóia a metamodelagem da transformação de um metamodelo UML

para um metamodelo de desempenho. Nesta sub-camada, ambos os metamodelos são usados para definir como são transformados modelos em M1. Finalmente, a sub-camada M1 é uma extensão da camada do modelo UML que apóia a representação de exemplos de transformação definidos na sub-camada M2.

Os metamodelos e modelos da camada de metamodelagem são traduzido em tecnologia-específica de documentos e schemas XML, respectivamente, na camada de implementação de modelo (UML XMI Schema, PM XMI Schema, UML documentos de XMI e PM XMI documenta na Figura 2.15). A tradução é executada de acordo com as regras para o schema e documento definida pela especificação de XMI. Semelhante, a transformação QVT baseada em padrões pode ser traduzida em regras expressadas pelo uso de linguagens de transformação XML baseada em padrões como XSLT ou XQuery.

Na camada de ferramenta, os modelos UML construídos nos projetos de software usam as ferramentas CASE baseadas em UML com capacidades de importar/exportar XMI (por exemplo, ArgoUML, Rational Rose, Eclipse /UML, etc.). O documento XMI da UML exportado pela ferramenta CASE é validado pelo uso do schema XMI correspondente e feito exame na entrada por um processador padrão XSLT/XQuery que aplique regras XSLT/XQuery para obter a saída do modelo de desempenho no formato de XMI.

O modelo, assim, obtido é validado pelo uso do schema XMI correspondente e feito exame eventualmente na entrada por uma ferramenta de avaliação do modelo de desempenho que renda os índices de desempenho de interesse (por exemplo, tempo de resposta, throughput, end-to-end delay, etc.). Tais índices fornecem a avaliação necessária para predizer o desempenho do sistema e o caso de melhor modelo fonte UML.

Assim, depois de criado um framework de transformação de modelos é criado um metamodelo LQN (*layered queueing networks*) conforme o MOF. O framework foi então aplicado para a construção automatizada de modelos UML 2.0 em LQN que foram especificados em um nível de metamodelos pelo uso de uma abordagem relacional baseado em elementos de transformação de metamodelos.

Aplicando a abordagem MDA para Aplicações de B2B: Mapeamentos

Jean Bézivin, Slimane Hammoudi, Denivaldo Lopes e Frédéric Jouault em (BÉZIVIN et al., 2004b) apresentam e discutem alguns mapeamentos necessários para criar uma aplicação B2B no contexto de MDA. Com essa finalidade foram propostos alguns metamodelos para: BPEL4WS, *Web Services* e Java. Em seguida alguns mapeamentos foram

criados de UML para estas plataformas. Assim, mapeamentos de UML para BPEL4WS depois mapeamentos de UML para *Web Services* e finalmente, mapeamentos de UML para a plataforma Java foram mostrados.

Na Figura 2.16, mapeamentos do metamodelo UML (fragmento) para o metamodelo BPEL4WS (fragmento) é apresentado. Estes mapeamentos são encontrados mostrando equivalências entre os elementos do meta-modelo fonte UML, e elementos do meta-modelo alvo BPEL4WS. Nesta figura, uma notação gráfica é usada para ilustrar os mapeamento de UML para BPEL4WS.

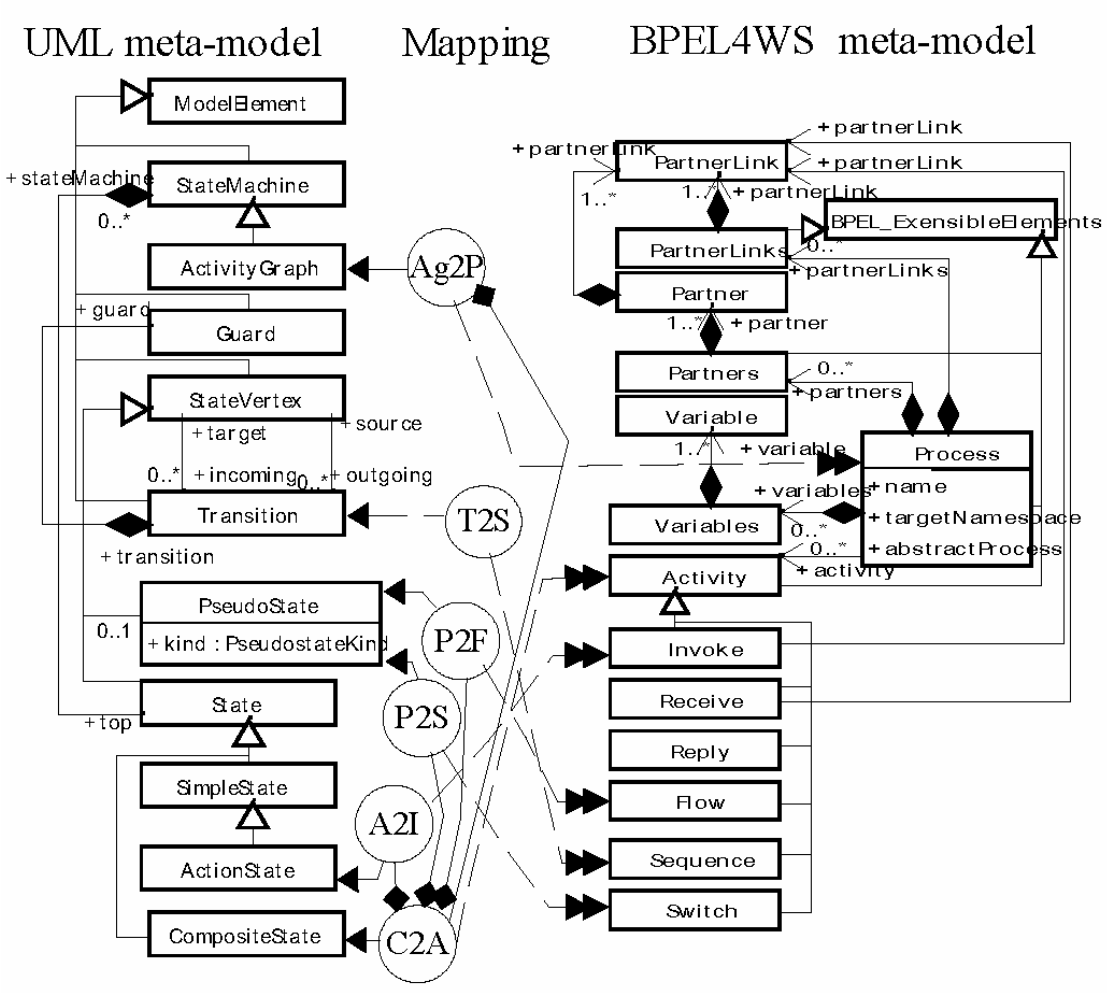

Figura 2.16: Mapeamento de UML para BPEL4WS

A linguagem de transformação ATL foi usada para definir as regras de transformação. De acordo com a Figura 2.16, os seguintes mapeamentos são encontrados (Ag2P, T2S, P2F, P2S, A2I e C2A). Os mesmos passos realizados no exemplo anterior para BPEL4WS foram feitos para *Web Services* e para a plataforma Java.

Transformação de Modelo Independente de Plataforma baseado em Triple

Andréas Billig, Susanne Busse, Andréas Leicher e Jorn Guy em (BILLIG et al., 2004) propõem um *framework* baseado em ontologias que provêm mapeamentos de PIM pra PSM. O *framework* está baseado em uma especificação de componentes independente de plataforma que é construída de acordo com a definição de componente usada na Linguagem de Descrição da Arquitetura (ADL). O *framework* é capaz de manipular mapeamentos apontados para várias plataformas como: Enterprise JavaBeans, CORBA, .Net, COM etc. Um usuário pode adicionar mapeamentos para cada plataforma de interesse. Cada mapeamento define uma relação entre o modelo de componente independente de plataforma e um modelo de componente específico da plataforma.

Além disso, o *framework* permite definir vários mapeamentos de PIM para PSM para cada plataforma específica. Cada mapeamento pode ser associado com um conceito particular ou características que descrevem uma situação quando o mapeamento deve ser aplicado.

Assim, um mapeamento particular é selecionado conforme as exigências de uma situação específica. Por exemplo, eles começam com uma especificação de componente independente de plataforma e geram dois diferentes modelos EJB de acordo com as exigências do usuário. Um desenvolvedor pode especificar algumas propriedades como a qualidade de atributos de serviço que devem ser levados em conta em uma situação particular. O framework escolhe o mapeamento apropriado baseado nas propriedades especificadas e gera Componentes EJB específicos das plataforma aperfeiçoada.

O framework está baseado em modelos de característica como também em Triple. Modelos de característica descrevem propriedades e alternativas para transformações de modelos. Eles são usados para especificar exigências de mapeamento. Triple é uma linguagem de programação dedutiva, semelhante a F-lógica. É usada para selecionar o mapeamento apropriado e executar a transformação de modelo baseado neste mapeamento.

A Figura 2.17 mostra a arquitetura do framework.

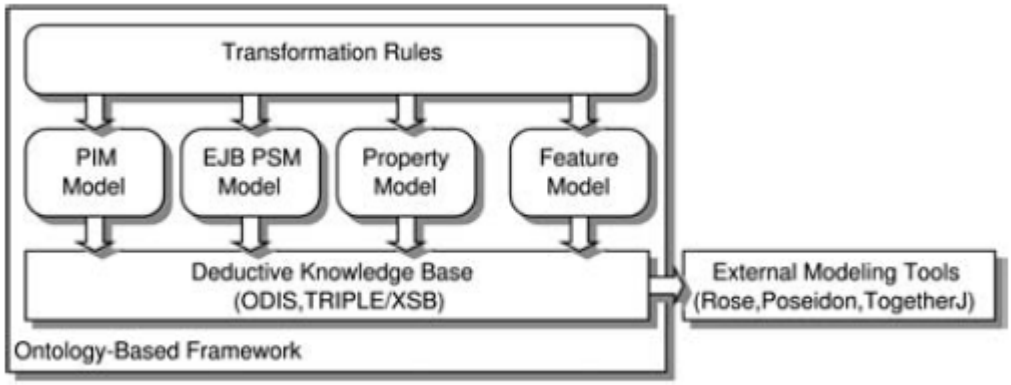

Figura 2.17: Arquitetura do Framework.

A Figura 2.17 é baseada em uma base de conhecimento que provê capacidades de raciocínio e transformação. Isto consiste principalmente da descrição de modelos independente de plataforma e componente específicos de plataforma como também de regras de transformação. Estas regras transformam um modelo PIM em um modelo PSM baseado em uma característica de instancias do modelo que descreve requisições do usuário. Um modelo de propriedade permite marcar elementos do modelo com característica de valores. Ambos mapeamentos são parametrizados nessas marcas e na característica dos modelos.

2.5 Conclusão

Neste capítulo, alguns conceitos relacionados aos sistemas de informática em saúde, MDA e *Web Services* foram apresentados.

Mostramos basicamente uma visão geral a respeito das tecnologias e aplicações da informática na área médica, e alguns aspectos referentes ao impacto deste uso cada vez mais presente, assim como algumas de suas principais tendências.

Além disso, apresentou-se a ferramenta MDA e suas principais características, ainda abordou-se *Web Services,* com suas tecnologias, benefícios, dentre outros.

E, por fim, apresentamos alguns trabalhos relacionados a nossa pesquisa, tanto da área médica quanto MDA.

3 SISTEMA DE INFORMAÇÃO MÉDICA PROPOSTO (SIMP)

Os Sistemas de Informação Médica provêem muitas vantagens quando usados para facilitar o acesso, a colaboração e o compartilhamento de dados entre centros médicos, pacientes e centros de pesquisas.

O uso de sistemas distribuídos associado à Internet, quando aplicado ao desenvolvimento de sistema de informação, permite aos usuários locais realizarem atividades além da capacidade do sistema local. Permite, também, fornecer a esses usuários um diagnóstico médico mais preciso através de um processo de compartilhamento de informações.

Neste trabalho, propomos um sistema de informação médica - SIMP que torna possível a obtenção de um diagnóstico médico mais preciso, através do auxílio de outros centros médicos, já que a interação possibilita uma grande troca de informações. Com isso, o médico pode contar com um grande número de informações em diferentes bases de dados.

Para desenvolver esse sistema, faz-se uso da tecnologia de *Web Services* para permitir que os médicos, independente da sua localização, possam trocar informações através da Internet.

A modelagem do sistema proposto foi feita usando a tecnologia de MDA para prover um sistema independente de plataforma e reutilizável.

3.1 Diagnóstico Médico

Existe uma diferença entre diagnóstico e diagnóstico médico. Segundo Miranda Sá (PORTAL MÉDICO, 2002), diagnóstico significa reconhecimento e pode ser empregado para designar qualquer reconhecimento de qualquer coisa ou de qualquer situação, enquanto diagnóstico médico consiste no reconhecimento de uma enfermidade em uma pessoa com base na análise de dados.

Assim, um médico só pode prover um provável diagnóstico de uma doença de um paciente, depois da coleta de uma boa quantidade de dados. Porém, um médico não tem meios de compartilhar estes dados e o diagnóstico (informação) do paciente com outros médicos fisicamente distantes.

Hoje, o conhecimento médico é desenvolvido principalmente nos principais CM (Centros Médicos)[1], que possuem um excelente conhecimento e melhores recursos financeiros e de infra-estrutura. Assim, pacientes de localidades distantes dos grandes centros urbanos precisam se deslocar para estes CM à procura de especialistas e de melhores serviços (BASIC, 2001).

Geralmente, a comunicação entre os CM e os locais que requerem informações mais especializadas é feita por cartas, fax, telefones ou e-mails. A comunicação não é eficaz no que diz respeito à rapidez, qualidade de dados transportados e interação com os CM. Então, muitas vezes é necessário levar os pacientes aos principais CM, ou levar o especialista para os lugares menos desenvolvidos para obter um diagnóstico mais preciso (BASIC, 2001).

Entretanto, o transporte do paciente para os CM pode levar a alguns riscos, de acordo com seriedade da doença a ser tratada, como, custos extras, problemas com exceder o número de pacientes nos principais CM, falta de estrutura para acomodar o paciente e das pessoas que acompanham esses pacientes.

Dessa forma, para contornar situações relativas a custos, qualidade de diagnóstico e desenvolvimento de doenças conhecidas, uma estrutura de comunicação adequada deve ser provida, onde os dados podem ser acessados por especialistas situados em localidades geograficamente distantes.

Por isso, o sistema proposto neste trabalho provê a possibilidade do médico ter uma segunda opinião com uma base de informações mais interativa. Na qual, o profissional pode usar um sistema distribuído com informações médicas, as quais estarão compartilhadas para fornecer um diagnóstico mais preciso aos pacientes.

3.2 Cenários do SIMP

Os possíveis cenários que os médicos interagem com a arquitetura do SIMP proposto são mostrados na Figura 3.1. Os cenários são os seguintes (MELO *et al.*, 2006):

Cenário 1 – O médico tem acesso a um sistema médico local, que possui um banco de dados (pacientes, doenças, medicamento, exames e sintomas);

[1] Centro Médico é um local onde se encontram médicos de várias especialidades, com equipamentos de ponta e com alta tecnologia.

Cenário 2 e 4 - O médico tem acesso a um sistema médico integrado[2], que também traz um banco de dados (pacientes, doenças, medicamentos, exames, sintomas);

Cenário 3 - O médico não tem um sistema médico local, nem integrado, assim, falta uma base de dados nesse cenário. Sob esta circunstância e através da Internet, o médico pode acessar o sistema, por exemplo, ele acessa os dados do paciente pelo seu browser para desenvolver o diagnóstico e para facilitar seu trabalho.

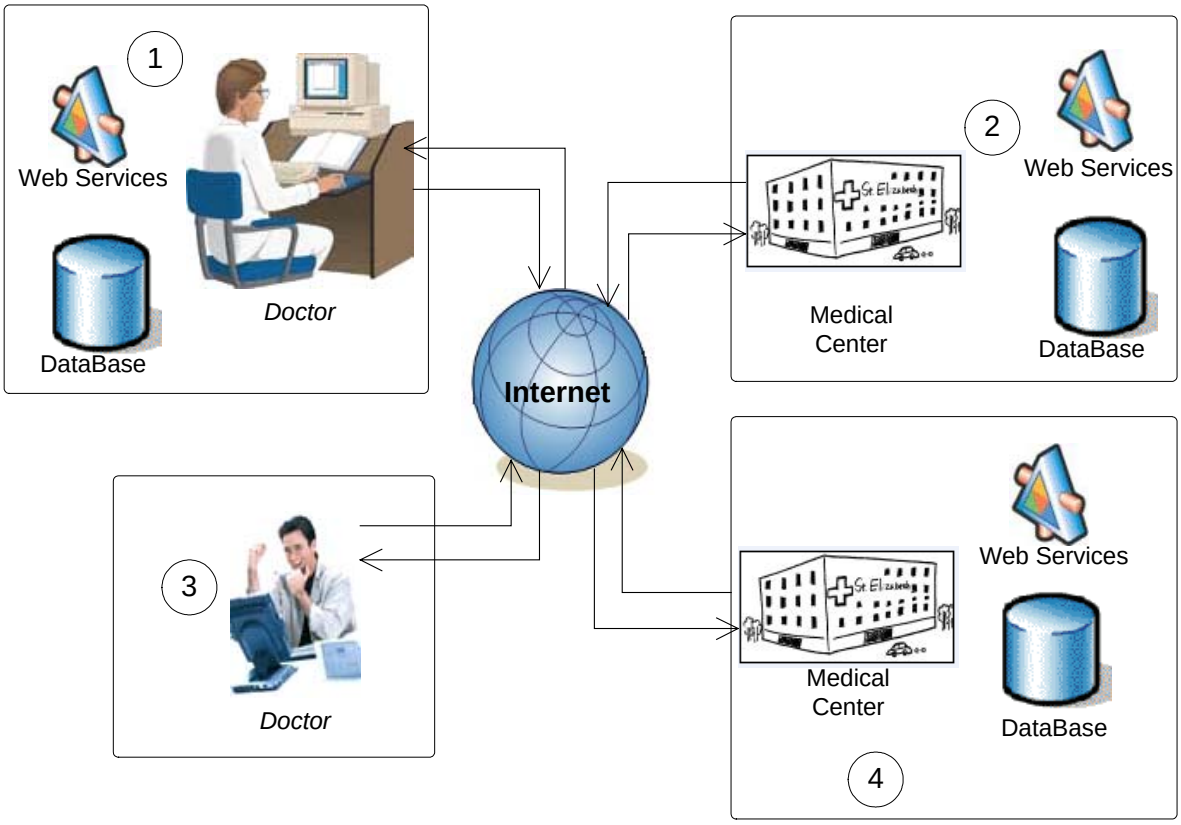

Figura 3.1 - Cenários do sistema de informação médica proposto – SIMP (MELO *et al.*, 2006)

A inter-relação entre os cenários 1, 2, 3 e 4, pode ser mostrada como segue:

- Inter-relação do cenário 1 com o cenário 2 ou com o cenário 4 - o médico faz uso do sistema médico local (1), para fornecer um diagnóstico com maior precisão, ele pode pedir mais informação de um CM (2 ou 4). Neste tipo de inter-relacionamento, o sistema médico local (1) usa um consumidor de serviços de informação médica do CM remoto (2 ou 4) e estes se transformam num fornecedor destes serviços solicitados por (1).

[2] Sistema médico integrado - um sistema que armazena dados de muitas clínicas (pediátricas, ortopédicas, ginecológicas, clínico geral...) pertencendo a um centro médico

- Inter-relação do cenário 2 com o cenário 4 ou com o cenário 1 - o médico faz uso de um sistema médico integrado de um determinado CM (2), e este solicita informação de um outro CM (4). Aqui, além do sistema médico integrado (2), o especialista pode pedir a informação para um outro centro médico (4) para fornecer um diagnóstico mais claro e mais preciso. Neste tipo de inter-relação, o sistema médico integrado (2) utiliza um consumidor de serviços de informações médicas e no centro médico remoto (4) em um fornecedor destes serviços. Uma outra abordagem, provável de acontecer dentro desta mesma inter-relação, é o centro médico (2 ou 4) se tornar um receptor de serviços de informação médica e o doutor, com um sistema médico local disponível (1), transforma-se em um consumidor destes serviços.
- Inter-relação do cenário 3 com o cenário 1, 2 ou 4 - o doutor tem acesso as informações médicas pela Internet. A fim de fornecer para seus pacientes um diagnóstico mais preciso, o doutor (3) se torna um consumidor de informação médica (1, 2 ou 4) e esses por sua vez, se tornam um provedor desses serviços.

3.3 Arquitetura do SIMP

A arquitetura do SIMP se baseia no uso de quatro elementos:

- Base de Conhecimento com informações médicas;
- Serviços de Acesso a Base via Web Services;
- Interface de Acesso via Internet para acessos externos ou local para acesso interno;
- Usuário especializado.

A Figura 3.2 apresenta a forma como estes quatro elementos interagem entre si para prover o funcionamento do sistema de informação médica. Nesta figura, temos os elementos que interagem para criar o ambiente de utilização do SIMP, no qual há dois usuários especializados. Um usuário com acesso a Internet e outro usuário com acesso local. Estes usuários fazem uso de uma interface de aplicação, que pode acessar tanto o servidor Web, quanto o servidor de aplicativos.

A interface de aplicação acessa os servidores a procura pelo responsável por executar as solicitações do usuário especializado. O responsável por executar as solicitações dos usuários especializados são os *Web Services*, que processam as solicitações e realizam consultas a base de conhecimento de informações médicas. O resultado desta execução é enviado de volta ao usuário especializado através da interface de aplicação.

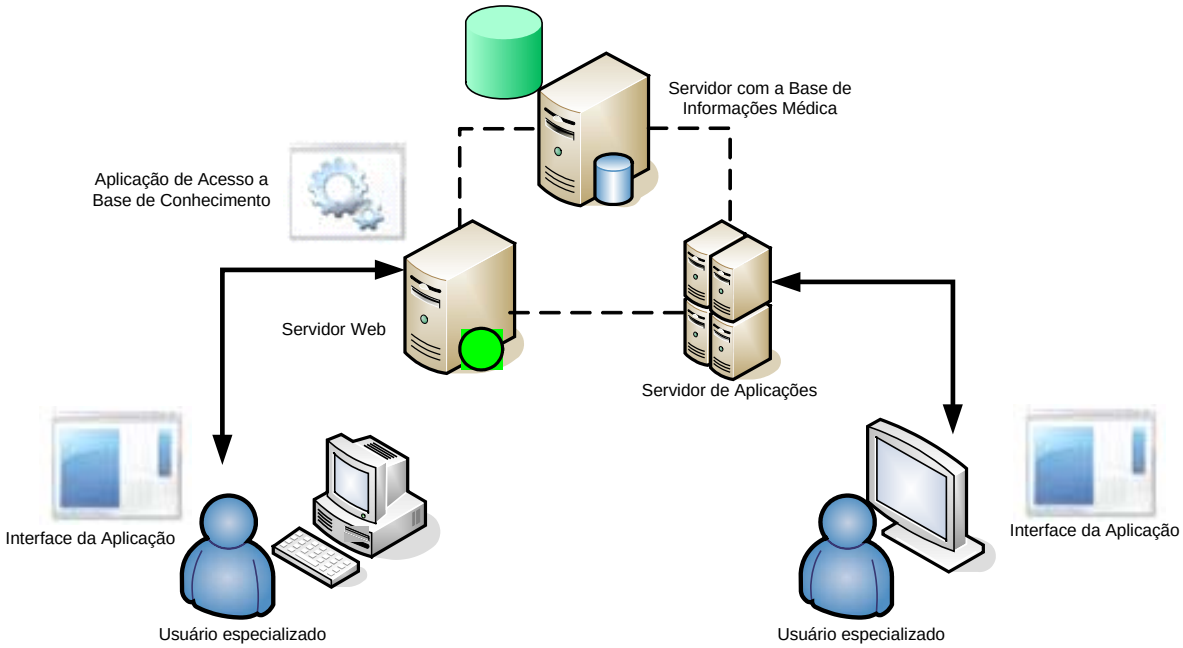

Figura 3.2: Arquitetura do SIMP

3.4 Modelagem do SIMP

Em um processo de diagnóstico médico, um médico requer que um paciente lhe forneça informação sobre o que está sentindo para, posteriormente, prover um diagnóstico.

O SIMP permite a um médico estabelecer relações de negócio com centros médicos, para ter acesso a dados médicos compartilhados. Assim, este médico pode tirar vantagens dos dados médicos compartilhados para prover um diagnóstico preciso.

3.4.1 Casos de Uso do SIMP

Apresentamos a seguir os principais casos de uso do sistema proposto. Em conjunto, apresentamos também uma breve descrição destes casos de uso.

A Figura 3.3 apresenta os casos de uso do sistema de informação médica proposto.

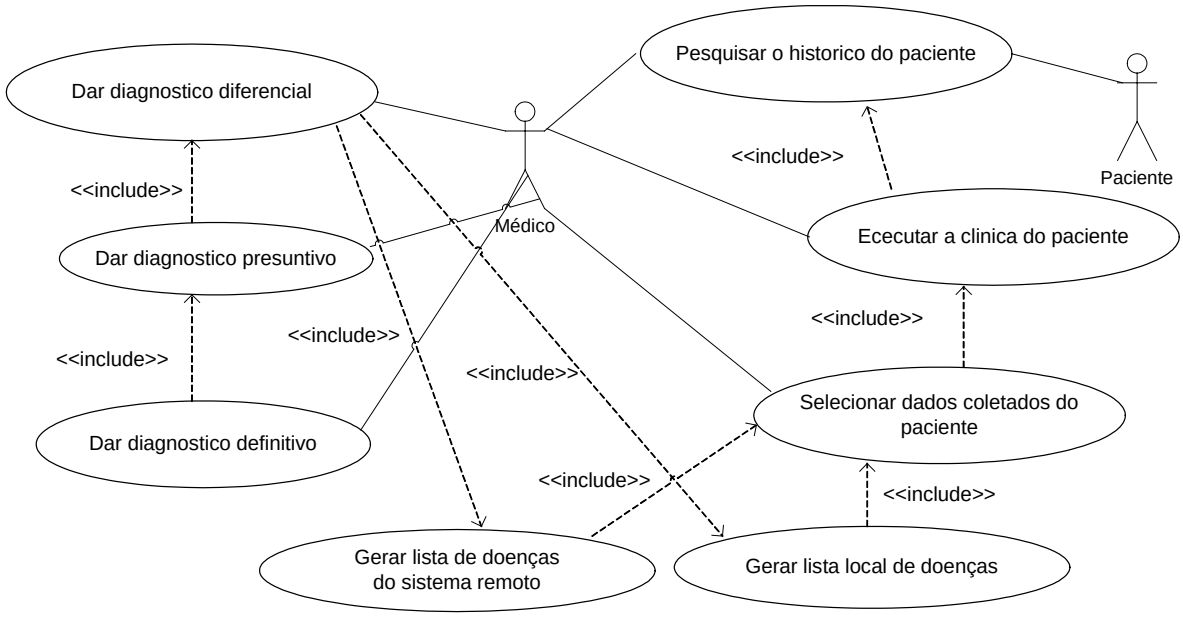

Figura 3.3 – Diagrama de caso de uso do sistema proposto.

A Tabela 3.1 mostra a descrição do caso de uso: “Pesquisar histórico do paciente”

Tabela 3.1 – Caso de Uso “Pesquisar histórico do Paciente”.

Caso de Uso 1	Pesquisar histórico do paciente
Descrição: Este caso de uso descreve o levantamento de dados do paciente	
Pré-condições: A atendente realizar um cadastro do paciente.	
Atores: Médico e paciente	
Cenários:	
1. O Médico busca o paciente no sistema	
2. O Médico pergunta quais doenças que o paciente já teve.	
3. O paciente informa as doenças e o médico as insere no sistema.	
4. O Médico pergunta quais cirurgias que o paciente já fez.	
5. O paciente informa as cirurgias e o médico as insere no sistema.	
6. O Médico pergunta quais medicamentos que o paciente já tomou.	
7. O paciente informa os medicamentos e o médico os insere no sistema.	
8. O Médico pergunta quais exames que o paciente já fez.	
9. O paciente informa os exames e o médico os insere no sistema.	
10. O Médico pergunta quais sintomas que o paciente está sentindo.	
11. O paciente informa os sintomas e o médico os insere no sistema.	
Cenários alternativos (3): inclusão de doença no sistema	
3.1 Se a doença já está cadastrada no sistema, o médico associa a doença ao paciente.	
3.2 Se a doença não está cadastrada no sistema, o sistema informa ao médico que a doença não está cadastrada. O médico escolhe no sistema a opção cadastrar doença, realizando então o cadastro.	
3.3 No momento do cadastro o sistema faz a validação dos dados, caso os dados não estejam de acordo com a validação, o cadastro não é concluído, gerando uma mensagem de erro.	
Cenários alternativos (5): inclusão de cirurgias no sistema	
5.1 Se a cirurgia já está cadastrada no sistema, o médico associa a cirurgia ao paciente.	
5.2 Se a cirurgia não está cadastrada no sistema, o sistema informa ao médico que a cirurgia não está cadastrada. O médico escolhe no sistema a opção cadastrar cirurgia, realizando	

então o cadastro.
5.3 No momento do cadastro o sistema faz a validação dos dados, caso os dados não estejam de acordo com a validação, o cadastro não é concluído, gerando uma mensagem de erro.
Cenários alternativos(7): inclusão de medicamentos no sistema
7.1 Se o medicamento já está cadastrado no sistema, o médico associa o medicamento ao paciente.
7.2 Se o medicamento não está cadastrado no sistema, o sistema informa ao médico que o medicamento não está cadastrado. O médico escolhe no sistema a opção cadastrar medicamento, realizando então o cadastro.
7.3 No momento do cadastro o sistema faz a validação dos dados, caso os dados não estejam de acordo com a validação, o cadastro não é concluído, gerando uma mensagem de erro.
Cenários alternativos(9): inclusão de exames no sistema
9.1 Se o exame já está cadastrado no sistema, o médico associa o exame ao paciente.
9.2 Se o exame não está cadastrado no sistema, o sistema informa ao médico que o exame não está cadastrado. O médico escolhe no sistema a opção cadastrar exame, realizando então o cadastro.
9.3 No momento do cadastro o sistema faz a validação dos dados, caso os dados não estejam de acordo com a validação, o cadastro não é concluído, gerando uma mensagem de erro.
Cenários alternativos (11): inclusão de sintomas no sistema
11.1 Se o sintoma já está cadastrado no sistema, o médico associa o sintoma ao paciente.
11.2 Se o sintoma não está cadastrado no sistema, o sistema informa ao médico que o sintoma não está cadastrado. O médico escolhe no sistema a opção cadastrar sintoma, realizando então o cadastro.
11.3 No momento do cadastro o sistema faz a validação dos dados, caso os dados não estejam de acordo com a validação, o cadastro não é concluído, gerando uma mensagem de erro.

A Tabela 3.2 mostra a descrição do caso de uso: “Fazer a clinica do paciente”

Tabela 3.2 – Caso de Uso “Fazer a clinica do paciente”

Caso de Uso 2	Fazer a clinica do paciente
Descrição: Neste caso de uso o médico com base nas informações coletadas, examina o paciente fisicamente a procura de sinais no seu corpo.	
Pré-condições: Ter feito o histórico do paciente	
Atores: Médico	
Cenários:	
1. O Médico faz a clinica do paciente, ou seja, com base nas informações coletadas no levantamento histórico do paciente, examina fisicamente a mesmo a procura por sinais no seu corpo.	
2. Caso o médico sinta necessidade poderá realizar o cadastro de novos sintomas.	
Cenários alternativos (2): inclusão de sintomas no sistema	
2.1 Se o sintoma já está cadastrado no sistema, o médico associa o sintoma ao paciente.	
2.2 Se o sintoma não está cadastrado no sistema, o sistema informa ao médico que o sintoma não está cadastrado. O médico escolhe no sistema a opção cadastrar sintoma, realizando então o cadastro.	
2.3 No momento do cadastro o sistema faz a validação dos dados, caso os dados não estejam de acordo com a validação, o cadastro não é concluído, gerando uma mensagem de erro.	

A Tabela 3.3 mostra a descrição do caso de uso: “Selecionar os dados coletados do paciente”.

Tabela 3.3 – Caso de Uso “Selecionar os dados coletados do paciente”

Caso de Uso 3	Selecionar os dados coletados do paciente (doenças, cirurgias, medicamentos, exames e sintomas).
Descrição: Neste caso de uso o médico seleciona os dados coletados que serão utilizados para a geração da lista de doenças refinada.	
Pré-condições: Ter os dados cadastrados inseridos no sistema.	
Atores: Médico	
Cenários:	
1. O Médico escolhe no sistema a opção “Montar a ficha do paciente”, ou seja, o sistema exibe uma lista contendo: todas as doenças, cirurgias, medicamentos, exames e sintomas informados pelo paciente.	
2. Selecionar da ficha do paciente a(s) doença(s) que o médico considera fundamentais para o diagnóstico.	
3. Selecionar da ficha do paciente a(s) cirurgia(s) que o médico considera fundamentais para o diagnóstico.	
4. Selecionar da ficha do paciente o(s) medicamento(s) que o médico considera fundamentais para o diagnóstico.	
5. Selecionar da ficha do paciente o(s) exames(s) que o médico considera fundamentais para o diagnóstico.	
6. Selecionar da ficha do paciente o(s) sintomas(s) que o médico considera fundamentais para o diagnóstico.	
2. Com base nos itens selecionados na ficha do paciente, o médico escolhe no sistema a opção “Gerar lista refinada”.	
Cenários alternativos (2): selecionar doenças da ficha do paciente.	
2.1 Se o sistema retornar uma lista de doenças, o médico pode escolher uma, nenhuma ou todas doenças exibidas.	
2.2 Se a lista de doenças estiver vazia, o sistema não permitirá a seleção de doenças.	
Cenários alternativos (3): selecionar cirurgias da ficha do paciente.	
3.1 Se o sistema retornar uma lista de cirurgias, o médico pode escolher uma, nenhuma ou todas cirurgias exibidas.	
3.2 Se a lista de cirurgias estiver vazia, o sistema não permitirá a seleção de cirurgias.	
Cenários alternativos (4): selecionar medicamento da ficha do paciente.	
4.1 Se o sistema retornar uma lista de medicamentos, o médico pode escolher um, nenhum ou todos medicamentos exibidos.	
4.2 Se a lista de medicamentos estiver vazia, o sistema não permitirá a seleção de medicamentos.	
Cenários alternativos (5): selecionar exames da ficha do paciente.	
5.1 Se o sistema retornar uma lista de exames, o médico pode escolher um, nenhum ou todos exames exibidos.	
5.2 Se a lista de exames estiver vazia, o sistema não permitirá a seleção de exames.	
Cenários alternativos (6): selecionar sintomas da ficha do paciente.	
6.1 Se o sistema retornar uma lista de sintomas, o médico pode escolher um, nenhum ou todos sintomas exibidos.	
Cenários Alternativo	

1. Se a lista de sintomas estiver vazia, o sistema não permitirá a seleção de sintomas.

A Tabela 3.4 mostra a descrição do caso de uso: “Gerar Lista de doenças do sistema local”

Tabela 3.4 – Caso de Uso “Gerar Lista de doenças do sistema local”

Caso de Uso 4	Gerar Lista de doenças do sistema local.
Descrição: Neste caso de uso o médico associa os dados coletados do caso de uso 3 à base local de informações médicas (doenças x sintomas) para a geração da lista local de doenças.	
Pré-condições: Ter a ficha do paciente com dados relevantes ao diagnóstico, gerada.	
Atores: Médico	
Cenários:	
1. O Médico escolhe no sistema a opção “Gerar Lista local de doenças”.	
2. O Sistema associa os dados coletados do caso de uso 3 à base local de informações médicas (doenças x sintomas).	
3. O Sistema exibe a lista de doenças gerada localmente.	
Cenários alternativos (3): exibir lista de doenças geradas local.	
2.1 Se a base local de informações médicas contiver dados relacionados à ficha do paciente com dados relevantes ao diagnóstico então o sistema retorna ao médico uma lista de doenças local.	
2.2 Se a base local de informações médicas não contiver dados relacionados à ficha do paciente com dados relevantes ao diagnóstico ou estiver vazia, então o sistema retornará uma lista vazia.	

A Tabela 3.5 mostra a descrição do caso de uso: “Gerar Lista de doenças a partir do sistema remoto”. O Caso de uso 5 é um dos principais casos de uso do sistema.

Tabela 3.5 – Caso de Uso “Gerar Lista de doenças a partir do sistema remoto”

Caso de Uso 5	Gerar Lista de doenças a partir do sistema remoto.
Descrição: Neste caso de uso o médico associa os dados coletados do caso de uso 3 à base remota de informações médicas (doenças x sintomas) para a geração da lista remota de doenças.	
Pré-condições: Ter a ficha do paciente com dados relevantes ao diagnóstico, gerada.	
Atores: Médico	
Cenários:	
1. O Médico escolhe no sistema a opção “Gerar lista remota de doenças”.	
2. O Sistema contacta centro(s) médico(s) remoto(s) através do Web Services.	
3. O Web Services cujo o serviço é o compartilhamento de informações médicas associa os dados coletados do caso de uso 3 à base remota de informações médicas (doenças x sintomas).	
4. O Web Services envie ao consumidor de serviço a lista de doenças gerada remotamente.	
Cenários alternativos (4): enviar lista remota de doenças.	
4.1 Se a base remota de informações médicas contiver dados relacionados à ficha do paciente com dados relevantes ao diagnóstico então o Web Services retorna ao médico uma lista de doenças.	
4.2 Se a base remota de informações médicas não contiver dados relacionados à ficha	

do paciente com dados relevantes ao diagnóstico ou estiver vazia, então o Web Services retornará uma lista vazia.

A Tabela 3.6 mostra a descrição do caso de uso: “Dar o Diagnóstico Diferencial”.

Tabela 3.6 – Caso de Uso “Dar o Diagnóstico Diferencial”

Caso de Uso 6	Dar o Diagnóstico Diferencial.
Descrição: Neste caso de uso o médico associa os dados coletados da lista gerada localmente e da lista gerada remotamente, retornando assim uma lista de doenças compatíveis, para dar o diagnóstico diferencial.	
Pré-condições: Ter a lista local e remota, gerada.	
Atores: Médico	
Cenários:	
1. O Sistema obtém a lista de doenças geradas localmente.	
2. O Sistema obtém a lista de doenças geradas remotamente.	
3. O Sistema gera uma lista de doenças compatíveis à partir das listas geradas localmente e remotamente.	
4. O Sistema exibe para o médico a lista de doenças compatíveis, dando o diagnóstico diferencial.	
Cenários alternativos (1): obter uma lista de doenças gerada localmente.	
1.1 Se o médico fizer uso de um sistema local, o mesmo gerará uma lista local de doenças.	
1.2 Se o médico não fizer uso de um sistema local, o mesmo não poderá gerar uma lista local de doenças.	
Cenários alternativos (2): obter uma lista de doenças gerada remotamente.	
2.1 Se o Web Services do centro médico remoto retornar uma resposta à solicitação do consumidor de serviços, então a lista de doenças será gerada remotamente.	
2.2 Se o Web Services do centro médico remoto não retornar resposta à solicitação do consumidor de serviços, então a lista de doenças gerada remotamente não poderá ser obtida.	

A Tabela 3.7 mostra a descrição do caso de uso: “Dar o Diagnóstico Presuntivo”.

Tabela 3.7 – Caso de Uso “Dar o Diagnóstico Presuntivo”

Caso de Uso 7	Dar o Diagnóstico Presuntivo.
Descrição: Neste caso de uso o médico com base no diagnóstico diferencial, poderá diminuir o numero de patologias hipotéticas e desta forma estabelecer o diagnóstico presuntivo.	
Pré-condições: Ter o diagnóstico diferencial.	
Atores: Médico	
Cenários:	
1. O Sistema exibe o diagnóstico diferencial.	
2. O Medico a partir do diagnóstico diferencial, da o diagnóstico presuntivo.	
Cenários alternativos (2): Dar o diagnóstico presuntivo.	
2.1 Se os dados do diagnóstico diferencial forem suficientes, o médico da o diagnóstico presuntivo.	

2.2 Se os dados do diagnóstico diferencial não forem suficientes para dar o diagnóstico presuntivo, o médico levanta mais informações para refinar a lista de patologias.

A Tabela 3.8 mostra a descrição do caso de uso: “Dar o Diagnóstico Definitivo”.

Tabela 3.8 – Caso de Uso “Dar o Diagnóstico Definitivo”

Caso de Uso 8	Dar o Diagnóstico Definitivo.
Descrição: Neste caso de uso o médico com base no diagnóstico presuntivo da o diagnóstico definitivo, ele diz ao paciente qual a doença que ele tem definitivamente.	
Pré-condições: Ter o diagnóstico presuntivo.	
Atores: Médico	
Cenários:	
1. O Sistema exibe o diagnóstico presuntivo.	
2. O Medico a partir do diagnóstico presuntivo, da o diagnóstico definitivo.	
Cenários alternativos (2): Dar o diagnóstico definitivo.	
2.1 Se os dados do diagnóstico presuntivo forem suficientes, o médico da o diagnóstico definitivo.	
2.2 Se os dados do diagnóstico presuntivo não forem suficientes para dar o diagnóstico definitivo, o médico solicita exames complementares.	
2.3 Analisa os exames complementares.	
2.4 Associa os exames complementares.	
2.5 Levanta mais informações para poder dar o diagnóstico definitivo.	

3.4.2 Interação do SIMP

Na Figura 3.4, apresentamos a forma com que os elementos do sistema interagem entre si. Neste diagrama, mostramos como o médico interage para armazenar no sistema as informações obtidas do paciente na consulta, tais como sintomas apresentados, doenças contraídas, cirurgias realizadas, medicamentos ingeridos e exames realizados. Com isso, o sistema gera uma lista geral de informações do paciente, que poderá ser filtrada para uma lista resumida. E, com base nessas informações, o centro remoto e o local criam a lista de prováveis doenças, que auxiliem o médico na tomada de decisão.

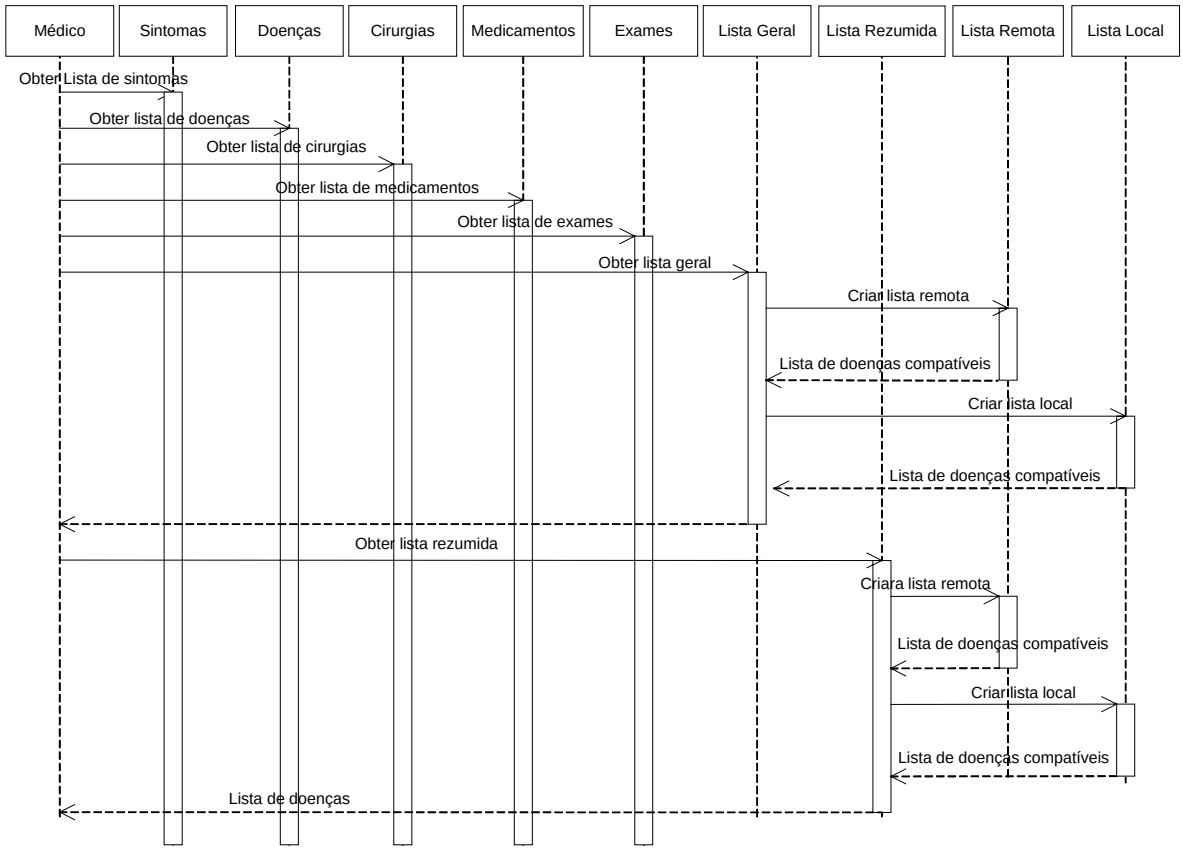

Figura 3.4 – Diagrama de seqüência do sistema proposto.

3.4.3 PIM em UML

A Figura 3.5 mostra o PIM em UML para o sistema de informação médica proposto. Neste trabalho, parte do PIM é apresentado sem a modelagem da base de dados para simplificar a apresentação do nosso trabalho. Na Figura 3.5, algumas classes executam características do sistema proposto: *Médico, sintomas, doenças, medicamento, cirurgias, exames, pacientes, lista geral, lista resumida, lista local, lista remota, diagnostico diferencial, diagnostico presuntivo* e *diagnostico definitivo.* As descrições de algumas classes definidas no modelo são detalhadas como segue:

A classe *ListaGeral* define alguns métodos que representam a lista geral, como *Obterlistageral*. Esta classe é associada diretamente à classe *Médico* para obter uma lista com as informações coletadas no consultório do paciente.

A classe *ListaRezumida* define alguns métodos que representam a lista selecionada, como *MostrarListaRezumida*. A classe *ListaRezumida* herda as características da classe *ListaGeral*, mas mostra que contém as informações essenciais selecionadas da classe *ListaGeral* para o diagnóstico do paciente.

A classe *ListaLocal* define alguns métodos que representam a lista local, como *CriarListaLocal*. Com a lista local é retornada uma lista de doenças, encontrada através da *ListaRezumida* que é por fim encaminhada para o centro médico local.

A classe *ListaRemota* define alguns métodos que representam a lista remota, como *EncontrarBasesInformaçãoCompartilhadas*, para criar a lista remota. Na classe *ListaRemota*, uma lista das doenças é retornada dos centros médicos remotos.

A classe *DiagnosticoDiferencial* define alguns métodos que representam o diagnóstico diferencial, como *ObterListaRemota, ObterListaLocal, ObterListaDoençasCompatíveis*. O diagnóstico diferencial é gerado com base na junção da lista retornada das doenças do centro médico local, e da lista do centro médico remoto. O diagnóstico diferencial é obtido de uma lista de doenças compatíveis com a última informação para o centro médico local e os centros médicos remotos.

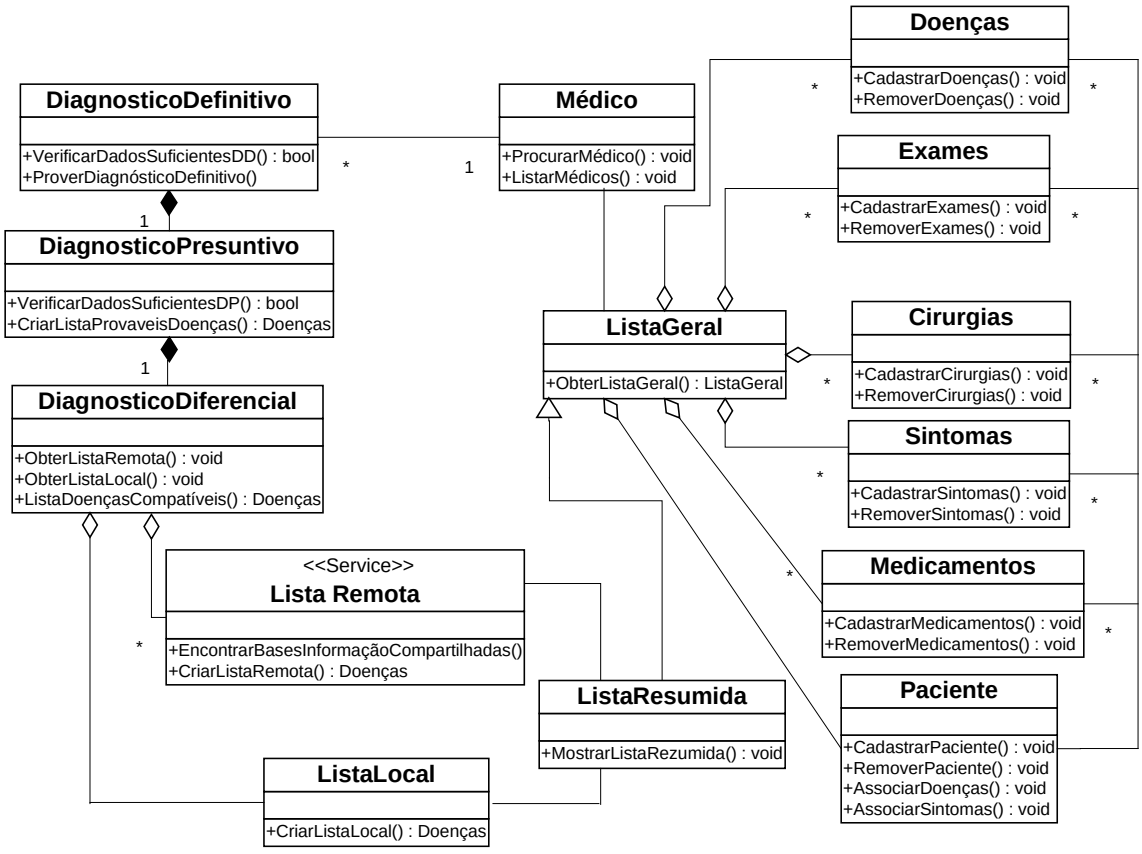

Figura 3.5 – PIM para o sistema de informação médica.

A classe *DiagnosticoPresuntivo* define alguns métodos que representam o diagnóstico presuntivo, como *VerificarDadosSuficientesDP* e *CriarListaProváveisDoenças*. No diagnóstico presuntivo o número de doenças hipotéticas diminui com relação ao número de doenças listadas no diagnóstico diferencial.

A classe *DiagnósticoDefinitivo* define alguns métodos que representam o diagnóstico definitivo, como *VerificarDadosSuficientesFornecerDiagnosticoDefinitivo.* No diagnóstico definitivo, o médico fornece o diagnóstico definitivamente ao paciente.

Os CM fornecem os serviços para encontrar a base de dados médica compartilhada, e mostram a lista das doenças desta lista de modo que o médico possa contar com uma segunda opinião e com uma sustentação para o guiar no processo formar uma decisão.

3.5 Metamodelos

O MDA é baseado em transformações entre modelos. Os modelos podem ser fonte e alvo, e estão em conformidade com seus respectivos metamodelos. O metamodelo fonte UML é escolhido, em conformidade ao metamodelo MOF. Para implementar o sistema de informações médicas, nós escolhemos os *Web Services*, o JWSDP e o WSoracle. Assim, um metamodelo para *Web Services* é necessário para construir os modelos específicos da plataforma (PSM) de nosso sistema de informações médicas.

3.5.1 Metamodelo UML

A Figura 3.6 apresenta um fragmento do metamodelo da UML.

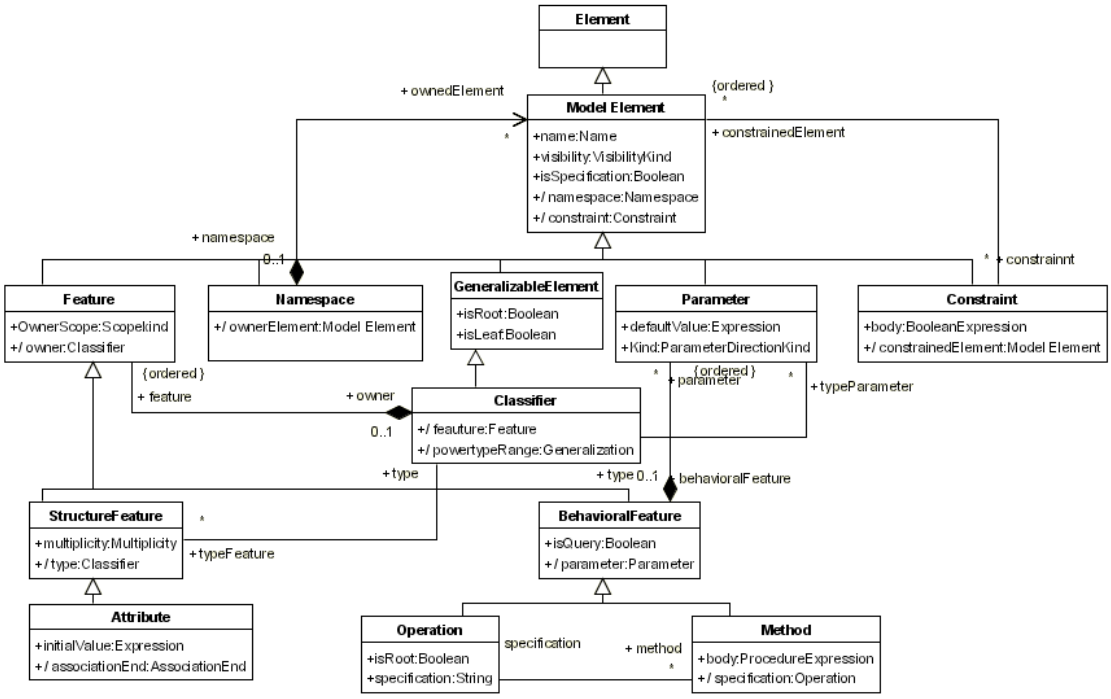

Figura 3.6 : O Metamodelo da UML (fragmento) (OMG, 2005a).

O elemento *ModelElement* da linguagem UML é especializado em *Feature, Namespace, GeneralizableElement, Parameter e Constraint* o elemento *Feature* é especializado em *StructuralFeature* e *BehavioralFeature*, e o elemento *GeneralizableElement* é especializado em *Classifier*.

3.5.2 Metamodelo do WSDL

O metamodelo de WSDL é composto dos elementos seguintes:

- *Definition***:** principal elemento do metamodelo WSDL contendo os elementos *Import, Type, Message, PortType, Binding e Service*;
- *Import:* permite a associação de um espaço de nomes *namespace* à localização de um documento XML;
- *Type:* utilizado para definir um tipo de dado abstrato simples ou complexo em conformidade com um esquema XML;
- *Message:* descreve um formato abstrato de uma mensagem específica que o *Web Service* envia ou recebe. Contém partes, por exemplo, *Part* que descrevem cada parte de uma mensagem;
- *PortType:* define a conversão de um serviço. Contém um conjunto de operações que um serviço envia e/ou recebe. Estas operações são caracterizadas pelo elemento *Operation* que descreve os tipos de chamadas de maneira abstrata. Os tipos de chamada são caracterizados pelos elementos *OneWayOperation*, *RequestResponseOperation, SolicitResponseOperation* e *NotificationOperation*. O elemento *ParamType* identifica quais são as mensagens de *input*, *output* e *fault*;
- *Binding:* descreve uma rota concreta dos componentes de uma conversão com o protocolo de comunicação utilizado, ou seja, a maneira como uma chamada pode ser terminada de maneira concreta (por exemplo, utilizando SOAP com HTTP). O elemento *BindingOperation* contém as propriedades *input, output*, e *fault* descritos de acordo com a realização concreta da chamada;
- *Service:* a descrição de um serviço, identifica a sua conversão *PortType* e as suas localizações *endpoints*.

Figura 3.7 ilustra um metamodelo de WSDL (BEZIVIN et al., 2004a) que é usado em nossa pesquisa.

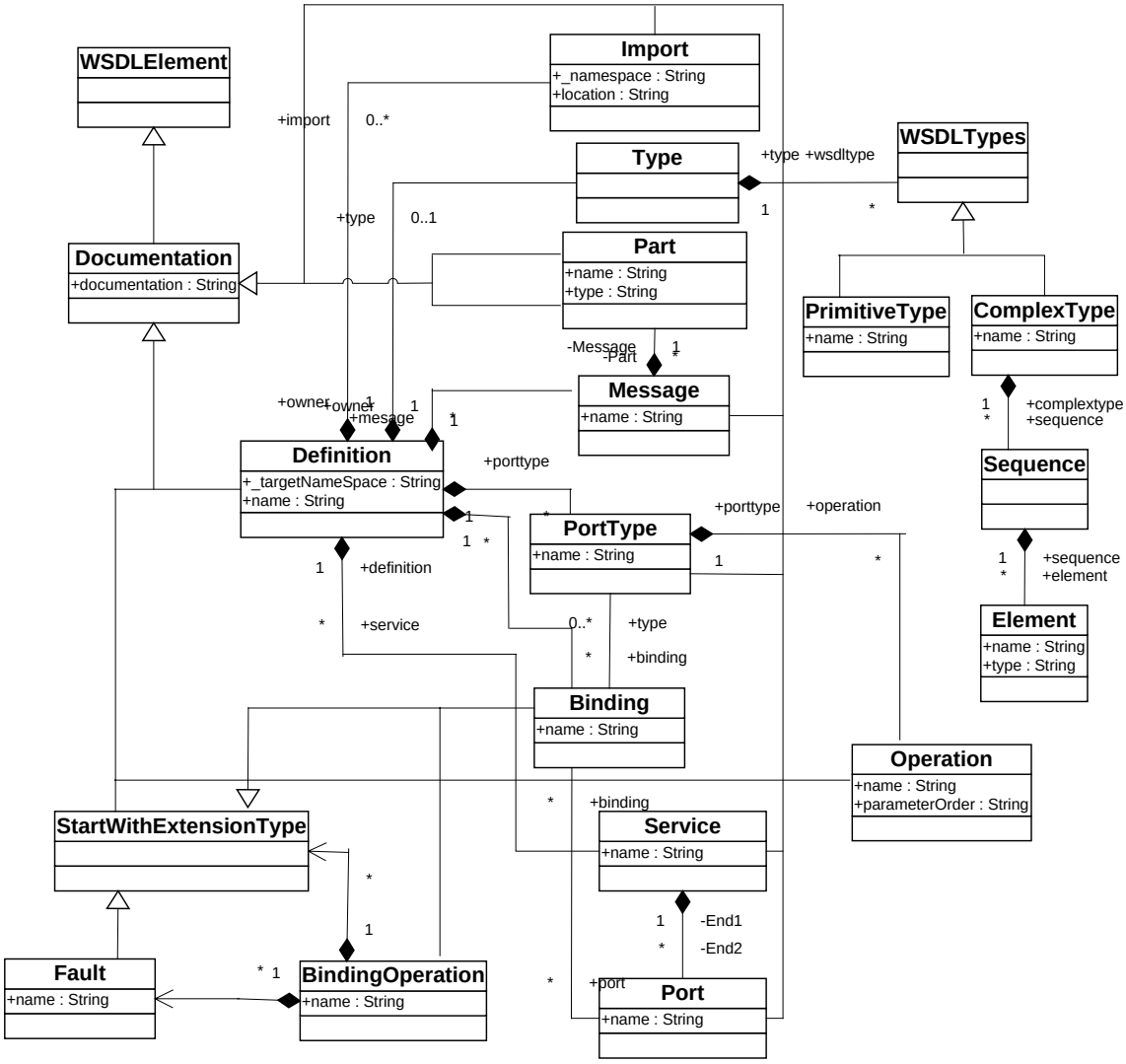

Figura 3.7 – Fragmento do Metamodelo WSDL (BEZIVIN et al., 2004a).

3.5.3 Metamodelo Java

O metamodelo Java foi estendido de uma versão 1.3 para uma versão 1.5. Este metamodelo é constituído dos elementos listados na Tabela 3.9, onde destaca-se como uma novidade o conceito de Annotation que também é considerado no metamodelo que pode ser visualizado na Figura 3.8.

Tabela 3.9 – Elementos do metamodelo Java

Elementos	Conceito
JElement	elemento raiz do metamodelo java;
JPackage	contém JClass, JInterface, etc;
JClassifier	contém JMember (JField e JMethod);
JClass	especialização de JClassifier, implementa um JInterface.
JInterface	outra especialização de JClassifier, contendo apenas os protótipos de métodos (sem o corpo) e atributos de tipo final static (constante).
JField	contém apenas um JPrimitiveType ou JClass ou JInterface.
JMethod	contém as operações (i.e. comportamento) de uma classe ou conversão no Java. Tem um parâmetro de regresso e um ou vários parâmetros de entrada
JParameter	especifique os parâmetros de um método no Java.
Annotations	são meta-informações que "anotam" (marcação) dados no código.

O Esquema de criação de Annotations foi implementado na versão 5 do Java, mas este conceito já era presente em versões anteriores de uma forma um pouco diferente. Como é o caso da marcação Javadoc @Deprecated que marca um pedaço de código como depreciado, informando ao programador que evite o uso da API marcada (MAGALHÃES, 2006).

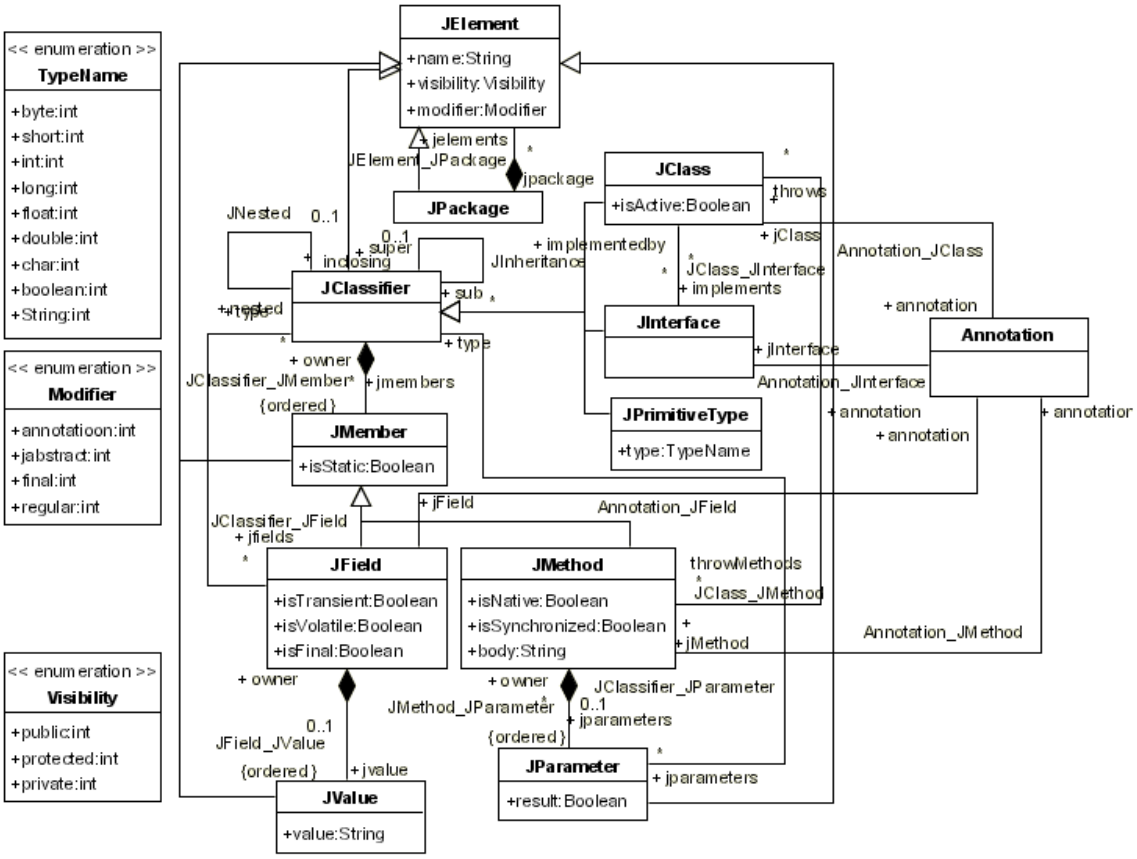

Figura 3.8 – Fragmento do Metamodelo Java.

3.5.4 Metamodelo JWSDP

A Figura 3.9 apresenta um metamodelo para a versão 1.5 de JWSDP. Este metamodelo foi criado através dos arquivos de configuração e desenvolvimento. Este metamodelo tem quatro pacotes: *webservices*, *configinterface*, *configwsdl*, *web e javawsdlmapping*. O pacote *webservices* possui as classes: “WebServices”, “WSDescription”, “PortComponet”, “WsdlPort” e “ServiceImplBean”. O pacote c*onfiginterface* possui as classes: “_Configuration”, “Service” e “Interface”, o pacote *configwsdl* possui as classes: “*Configuration”* e “*_WSDL”*, o pacote *web* possui as classes: “*WebApp”*, “*Servlet”* e “*ServletMapping” e o pacote javawsdlmapping* possui as classes: “JavaWsdlMapping”, “JavaXmlTypeMapping”, “RootTypeQname”, “VariableMapping”, “PackageMapping”, “ServiceInterfacemapping”, “WsdlServicemapping”, “PortMapping”, “ServiceEndpointMethodMapping”, “WsdlReturnValueMapping”, “WsdlMessage”, “MethodParamPartMapping” e “WsdlMessageMapping”.

As descrições de alguns pacotes definidos no metamodelo de JWSDP são detalhadas a seguir:

- O pacote *Configwsdl* é usado para gerar o Configwsdl.xml que especifica o lugar do arquivo WSDL. O atributo packageName especifica o pacote Java para gerar os stubs. O atributo location do arquivo WSDL é especificado com uma URL;
- O pacote *ConfigInterface* é usado para gerar o ConfigInterface.xml que especifica a informação na interface. O atributo name é o nome do serviço. Os atributos targetNamespace e typeNamespac são definidos como URLs. O atributo packagename especifica o pacote dentro das classes dos serviços serão executados;
- O pacote *Web* é usado para gerar o arquivo web.xml que é usado pelo container Web para identificar o serviço e para definir algumas propriedades como: O elemento de servletmapping que define uma mapeamento entre um servlet e uma URL padrão. O atributo servletname é o nome do servlet para o qual nós estamos mapeando uma URL padrão. O atributo url-pattern descreve um padrão usado para resolver URLs. O elemento servlet contém os dados declarativos de um servlet. O servletname define o nome oficial do servlet, usado como referência de definição de outro servlet no descriptor de desenvolvimento.

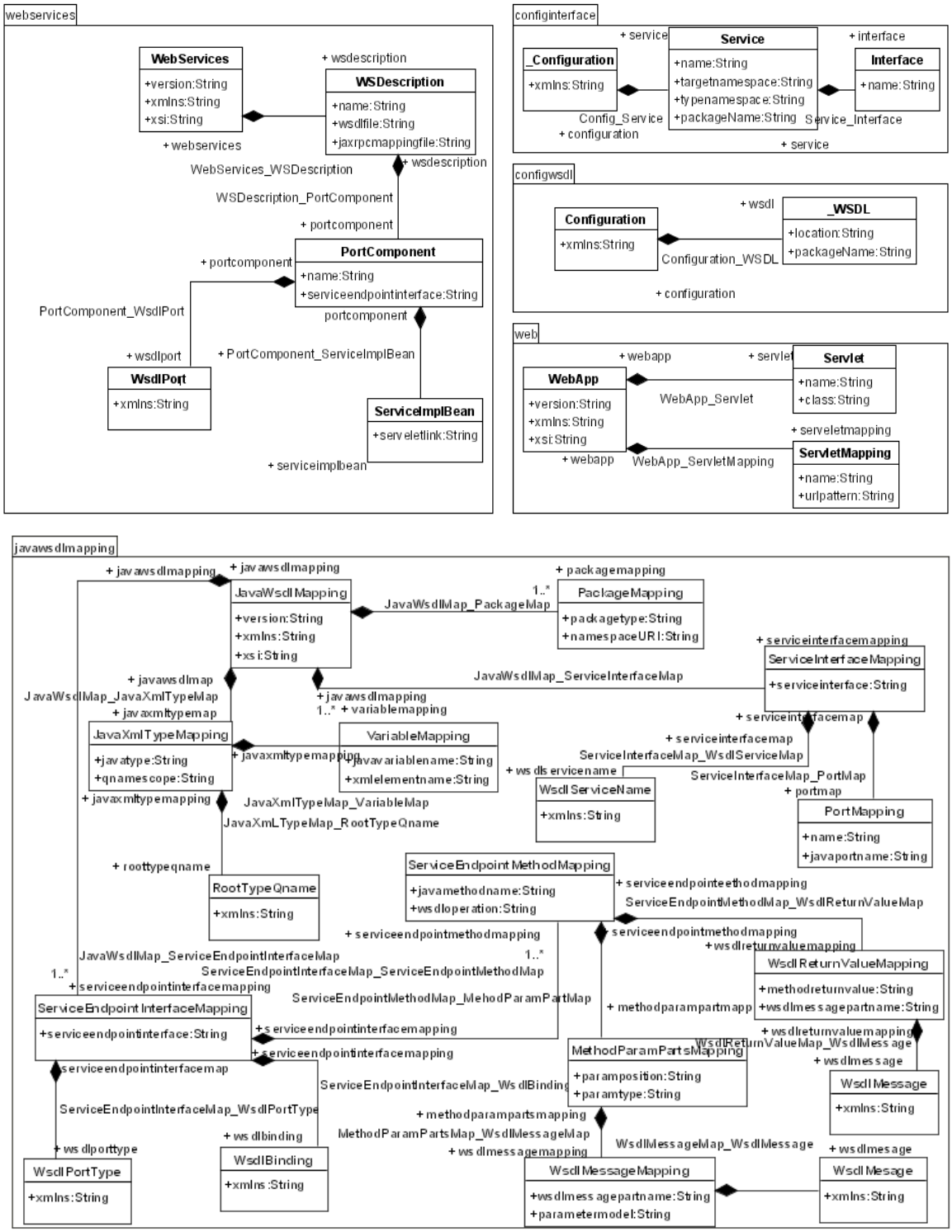

Figura 3.9 – Metamodelo JWSDP

Os *Web Services* criados na plataforma JWSDP obedecem o *Tamplete* apresentado na Figura 3.10.

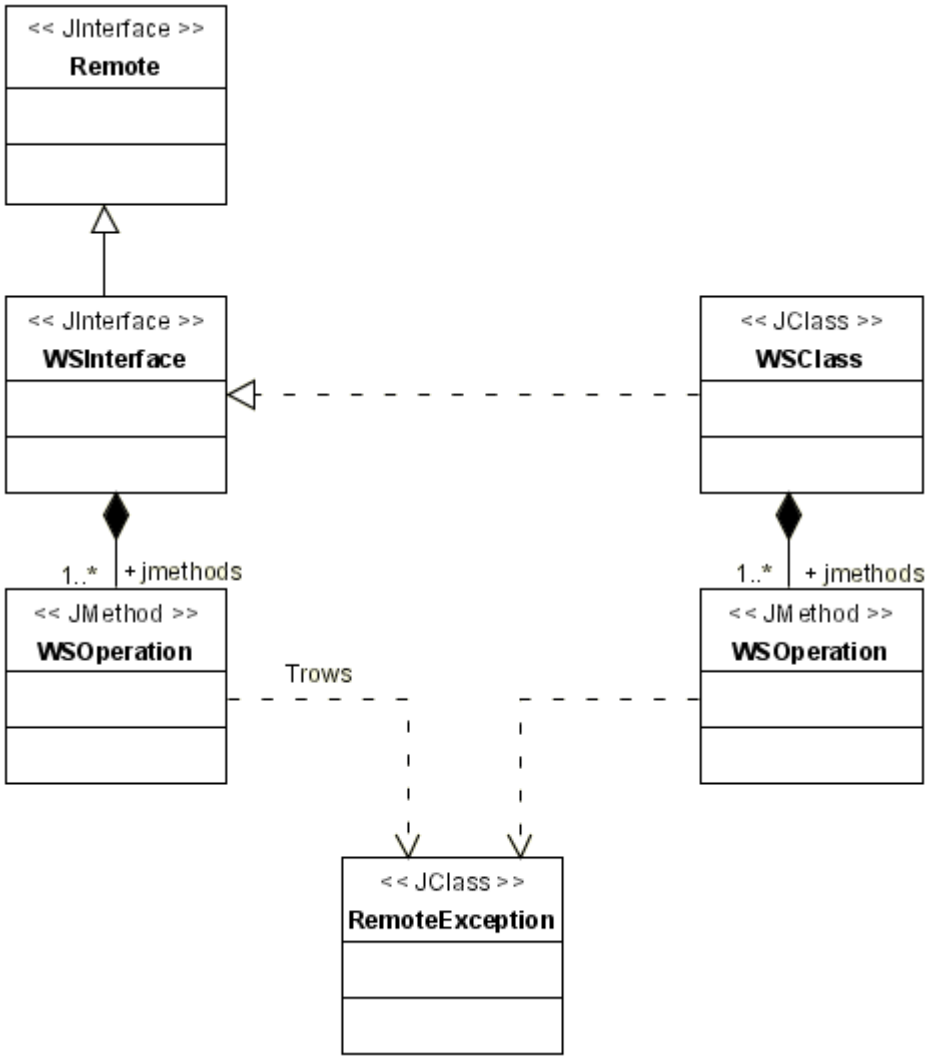

Figura 3.10 – *Template* JWSDP (LOPES, 2005)

3.5.5 Metamodelo do WSOracle

A Figura 3.11 mostra um metamodelo para os *Web Services* do WSOracle. Este metamodelo foi criado através dos arquivos de configuração e desenvolvimento do *Web Services* da Oracle. Este metamodelo é composto por sete pacotes: *oraclewebservices, web, web services, orionapplication, application, e datasources.*

O pacote *oraclewebservices* possui as classes "OracleWebServices", "WSDescription", "PortComponent" e "Operations". O pacote *webservices* possui as classes: "WebServices", "WSDescription", "WsdlPort", "ServiceImplBean" e "PortComponent". O pacote *web* possui as classes: "WebApp", "Servlet" e "ServletMapping". O pacote *application* possui as classes: "Application", "DisplayName", "Web" e "Module", o pacote *orionapplication* possui as classes: "OrionApplication" e "DataSources". O pacote

datasources possui a classe: “DataSources”. finalmente o pacote *javawsdlmapping* possui as classes: “JavaWsdlMapping”, “PackageMapping”, “JavaXmlTypeMapping”, “VariableMapping”, “ServiceInterfaceMapping”, “WsdlServicename”, “WsdlportType”, “WsdlReturnValueMapping”, “PortMapping”, “ServiceEndpointInterfaceMapping”, “WsdlMapping”, “ServiceEndpointMethodMapping” e “WsdlMessage”.

As descrições de alguns pacotes definidos no metamodelo de WSOracle são detalhadas a seguir:

- O pacote *oraclewebservices* é usado para gerar o arquivo oraclewebservices.xml para o *Web Service* específico da Oracle. O pacote é um descritor de desenvolvimento que define propriedades específicas de desenvolvimento para uma aplicação *Web Service* rodando em cima do *Web Services* Oracle. O descriptor de desenvolvimento oracle-webservices é usado junto com o pacote *Web Services* padrão. Ele contém informação de desenvolvimento e de run-time que são específico para o *Web Services* da Oracle. O atributo name em WSDescription mapeia o elemento nome para o pacote webservices. O atributo name em PortComponent mapeia o elemento nome para o pacote webservices.
- O pacote *Web* é usado para gerar o arquivo web.xml que é usado pelo container Web para identificar o serviço e para definir algumas propriedades como: O elemento de *servletmapping* que define uma mapeamento entre um servlet e uma URL padrão. O atributo servletname é o nome do servlet para o qual nós estamos mapeando uma URL padrão. O atributo *url-pattern* descreve um padrão usado para resolver URLs. O elemento servlet contém os dados declarativos de um servlet. O servletname define o nome oficial do servlet, usado como referência de definição de outro servlet no descriptor de desenvolvimento.
- O pacote de *Javawsdlmapping* é usado para gerar o arquivo javawsdlmapping.xml que mapeia os parâmetros, métodos e interface de Java para WSDL.
- O pacote *Webservices* é usado para gerar o arquivo webservices.xml que define o *Web Service* endpoint e arquivos de configuração associados. O atributo name em *port-component* no pacote webservices provê o mesmo valor que o atributo name em *port-component* no pacote oracle-webservices. O atributo name em *webservice-description* no pacote webservices provê o mesmo valor que o atributo name em webservice-description no pacote oracle-webservices.

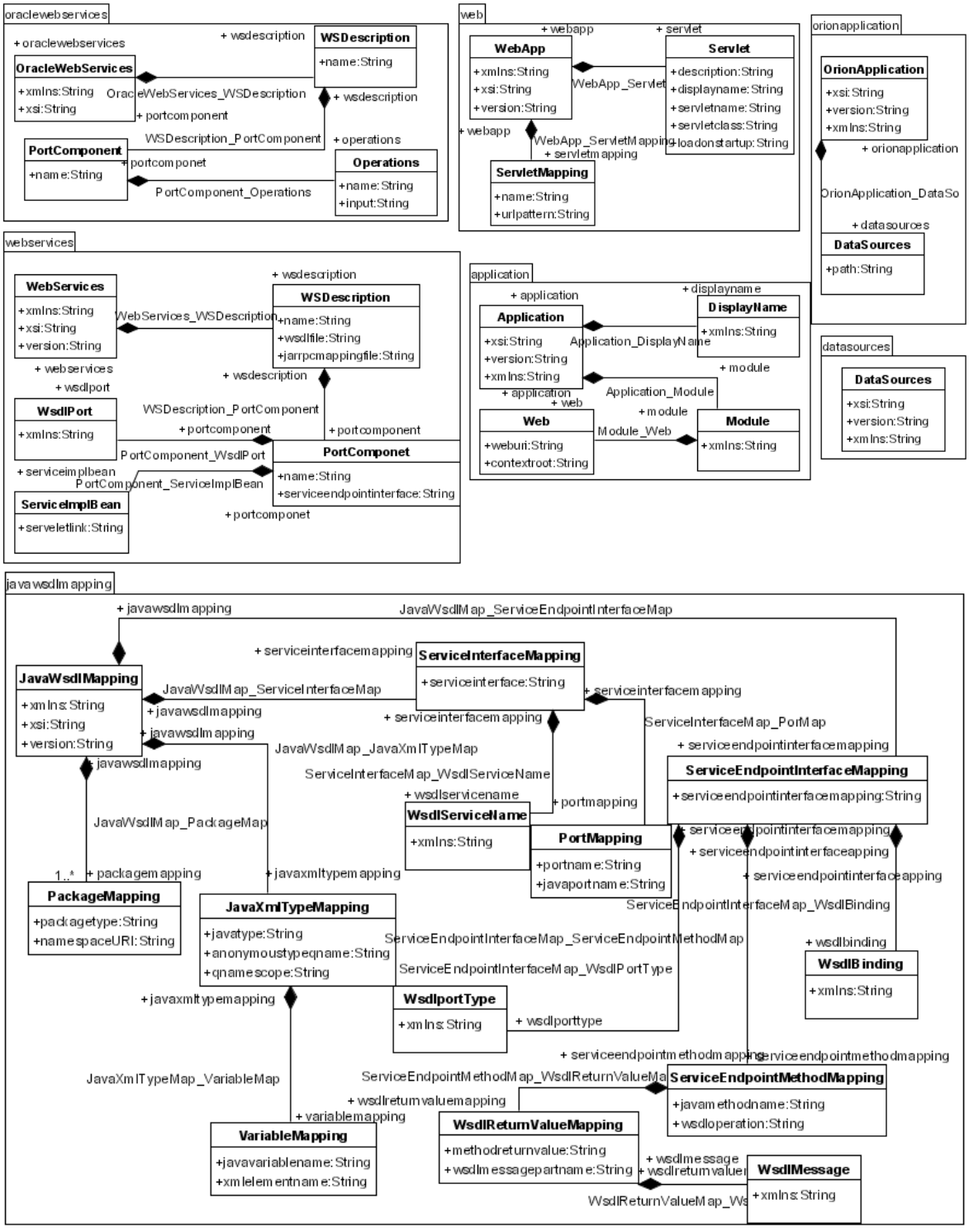

Figura 3.11– Metamodelo do WSOracle.

Os *Web Services* criados na plataforma Oracle obedecem o *Tamplete* apresentado na Figura 3.12.

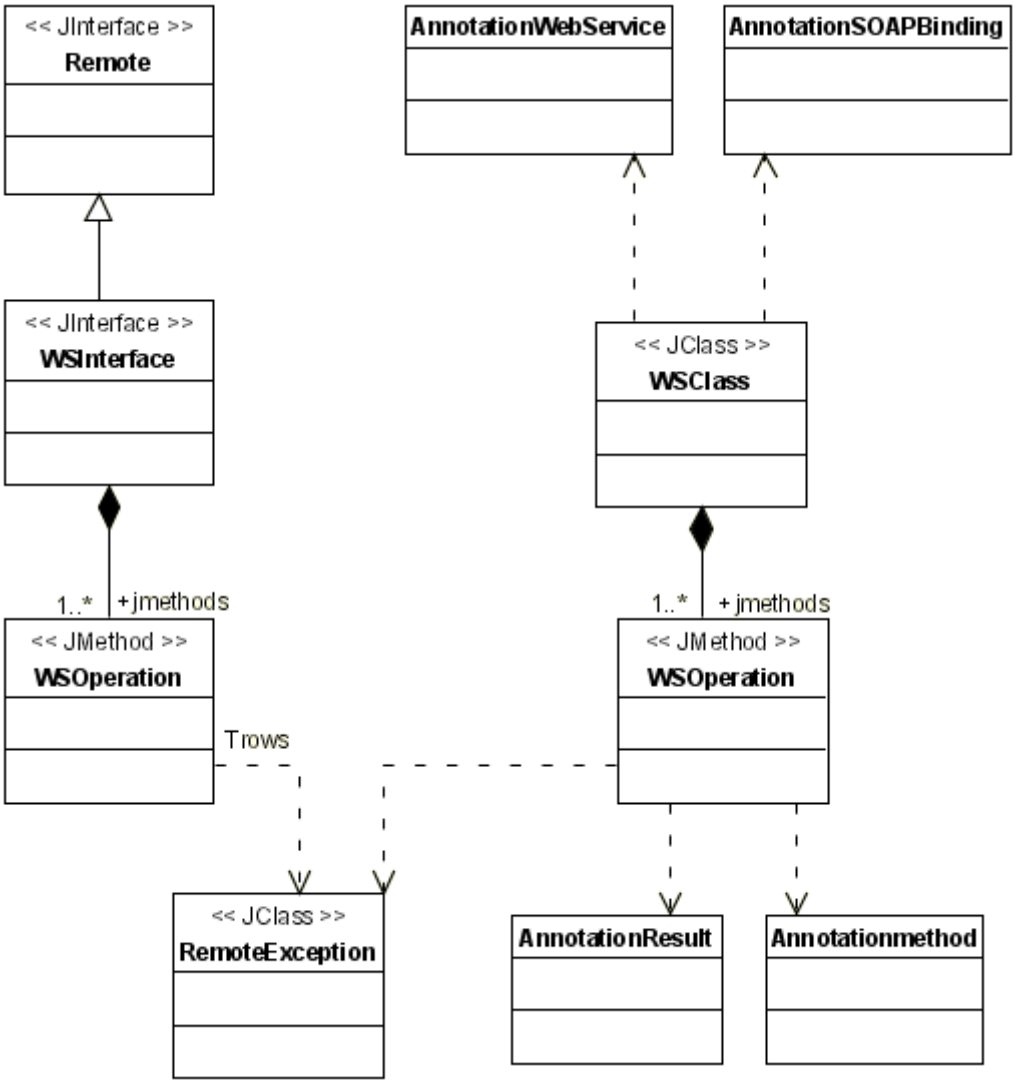

Figura 3.12–*Template* WSOracle.

3.6 Conclusão

No Capítulo 3, apresentamos a criação de um sistema de informação médica, que interage com diferentes cenários e cuja arquitetura é baseada nos seguintes elementos: Base de Conhecimento com Informações Médicas, Serviço de Acesso a Base via *Web Services*, Interface de Acesso via Internet e Usuário Especializado.

Posteriormente, mostramos a modelagem do sistema, com seus respectivos casos de uso e a criação do PIM e dos metamodelos, que será a base para fazer as especificações de correspondências e definições de transformação entre os metamodelos: UML e WSDL; UML e WSOracle; e UML e JWSDP.

4 TRANSFORMAÇÕES DE MODELOS

Em um processo de Transformação de Modelos, os mapeamentos ou especificações de correspondências determinam as correspondências entre os elementos de dois metamodelos. Uma regra de transformação descreve em detalhes, as etapas da transformação de um modelo em um outro modelo respeitando a especificação de correspondências. A execução de uma regra de transformação gera elementos de um modelo alvo a partir de um modelo fonte.

Neste capítulo, nós apresentamos as especificações de correspondências e definições de transformação entre os metamodelos:

- UML e WSDL;
- UML e WSOracle (incluindo as especificações de correspondências e transformações de UML para Java considerando a API da Oracle);
- UML e JWSDP (incluindo as especificações de correspondências e transformações de UML para Java considerando a API da JWSDP).

A Figura 4.1 apresenta as transformações que serão demonstradas neste Capítulo.

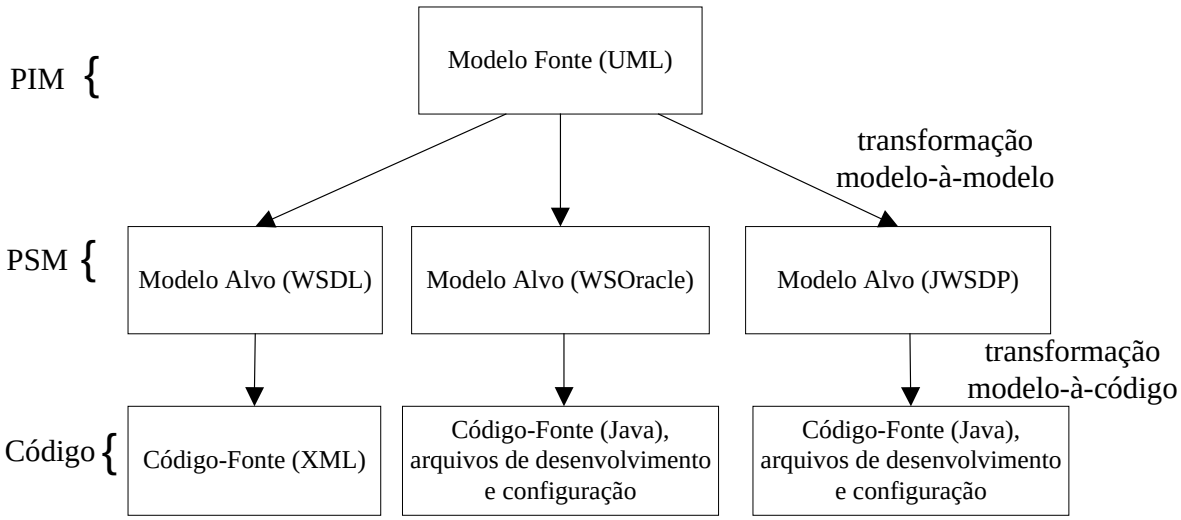

Figura 4.1 - As transformações: modelo-à-modelo e modelo-à-código

Utilizamos a linguagem de transformação ATL (*Altas Transformation Language*)(AMMA, 2006) (ECLIPSE, 2004) para criar as regras de transformação. Uma regra de transformação em ATL é criada manualmente a partir da especificação das correspondências entre os modelos. O uso da ATL deveu-se por esta ferramenta estar de acordo com MDA. A ATL faz uso de um repositório MDR (*Meta Data Repository*) (NETBEANS, 2006) ou EMF (*Eclipse Modeling Framework*) (ECLIPSE, 2006) para

armazenar e manipular os metamodelos fonte e alvo. A ATL executa a transformação de acordo com as regras definidas em sua codificação. Em seguida, estas regras de transformações são aplicadas ao PIM do Sistema de Informação Médica para gerar parte do código nas plataformas de *Web Services*, WSOracle e JWSDP.

As transformações foram geradas obedecendo-se os seguintes passos:

- Mapeamento Manual: é a identificação inicial das equivalências entre os elementos do modelo fonte para o modelo alvo. Por exemplo, a partir da definição de um método em UML podemos fazer a sua equivalência a uma *operation* em WSDL;
- Escrita da Regra: utilizando-se uma linguagem de transformação escreve-se a regra baseada no mapeamento manual. Para esta finalidade, utilizamos a ATL associado ao Eclipse (ECLIPSE, 2005), como linguagem de transformação;
- Geração do PSM: Após a escrita e execução da regra em ATL, é gerado o PSM a partir do PIM;
- Geração do Código-Fonte: A partir do PSM e do metamodelo da plataforma alvo é gerado o código-fonte, através de uma transformação de modelo-à-código.

4.1 Transformações de UML para WSDL

Um documento WSDL descreve os serviços oferecidos por um *Web Service*. Desta forma, um documento WSDL fornece uma documentação dos serviços disponibilizados para que possíveis clientes possam utilizá-lo de forma automatizada. Um documento WSDL é um documento XML composto por elementos com funcionalidades bem definidas. A Figura 2.10 do Capítulo 2 apresenta os elementos de um arquivo WSDL.

Por isso, o objetivo principal da transformação de um modelo UML para um modelo WSDL é a geração de um documento WSDL para um serviço definido dentro de um PIM em UML. A criação das regras de transformações em ATL para a geração de um arquivo WSDL a partir de um modelo UML deve ser baseada nas correspondências entre os elementos pertencentes ao metamodelo fonte (UML) e o metamodelo alvo (WSDL).

4.1.1 Mapeamentos

Os metamodelos apresentados no Capítulo 3, metamodelo UML e metamodelo WSDL, foram analisados com o objetivo de se encontrar elementos de um metamodelo fonte

(UML) que tivessem correspondência com elementos do metamodelo alvo (WSDL). Através deste levantamento, foram encontradas algumas equivalências. Na Figura 4.2, nós apresentamos um fragmento de equivalência encontrada durante este estudo.

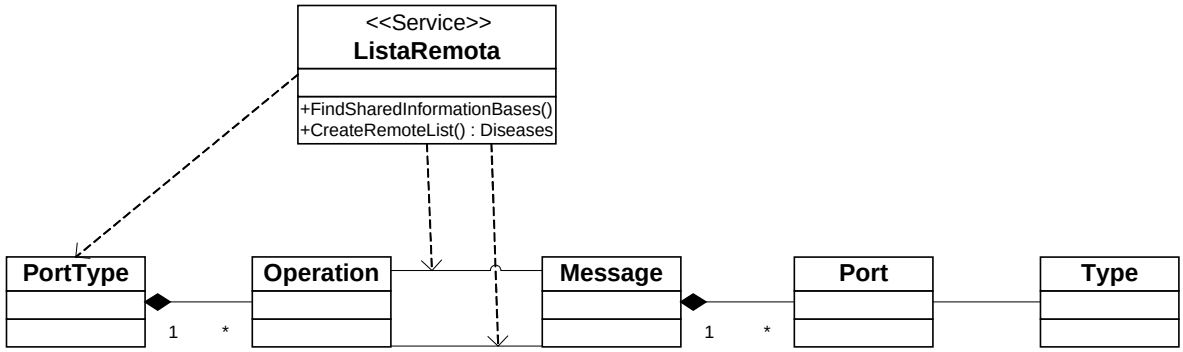

Figura 4.2 - Equivalência entre Elementos UML e WSDL

Na Figura 4.2 apresentamos um elemento *Class* com o estereótipo *Service* em UML. Este elemento *Class* correspondente ao *PortType* em WSDL. Da mesma forma, um método de UML é equivalente a uma operação em WSDL. Um parâmetro *input* ou *output* em UML corresponde a uma mensagem *input* ou *output* em WSDL. Se os parâmetros *input* ou *output* que estão em um método de UML não são baseados em tipos primitivos, por exemplo, baseados em outras classes da UML, então estes tipos correspondem a *Type* em WSDL (BEZIVIN et al., 2004a).

As equivalências encontradas entre o metamodelo fonte e metamodelo alvo são apresentadas na Tabela 4.1.

Tabela 4.1 – Mapeamentos entre o metamodelo UML e metamodelo WSDL.

Mapeamentos		
Nome	**De UML**	**Para WSDL**
P2D	Package	Definition
C2S	Class	Service, Port, Binding e PortType.
O2O	Operation	Operation e Message
P2Part	Parameter	Part
Dt2T	DataType	Type
M2Bo	Method	BindOperation
A2E	Attribute	Element
A2T	Attribute	Type
D2P	DataType	PrimitiveType

Depois de identificarmos alguns elementos equivalentes entre o metamodelo UML e o metamodelo WSDL, as regras de transformações baseadas nestas equivalências (mapeamentos) foram criadas.

4.1.2 Definições de transformação em ATL

A definição de transformação em ATL, para realizar a transformação de um modelo UML em modelo WSDL está baseada na especificação de correspondência apresentadas na Tabela 4.1. Apresentamos um fragmento de código-fonte em ATL da regra D2P, baseado na especificação D2P.

```
1   rule D2P {
2   from d : UML!DataType
3   To t : WSDL!PrimitiveType(
4   name <- d.name
5   )
6   }
```

4.1 Fragmento do Código-Fonte em ATL da Regra D2P

Através da regra D2P, um elemento *DataType* em UML é mapeado para um elemento *PrimitiveType* em WSDL.

```
1   rule C2C {
2   from c: UML!Class (c.stereotype ->
3   exists(e|e.name='Service'))
4   To t: WSDL!ComplexType( name <- c.name,
5   Sequence <- seq
6   ),
7   seq: WSDL!Sequence(
8   Element <- c.feature->select
9   (e|e.oclIsTypeOf(UML!Attribute))
10  )
12  } rule2E {
13  A from at : UML!Attribute
14  To t : WSDL!Element(
15  name <- at.name,
```

```
16    type <- at.type.name
17    )
18    }
```

4.2 Fragmento do Código-Fonte em ATL da Regra C2S

A regra apresentada no fragmento de Código-fonte 4.2 cria uma instância de *WSDL!ComplexType*. O atributo *name* de um *WSDL!ComplexType* é definido de acordo com o valor de *c.name* de um elemento *Class* em UML. A *Sequence* é definida de acordo com o valor de *seq*, isto é, criando um *WSDL!Sequence*. *Seq* cria uma instância de *WSDL!Sequence*. O *element* é definido de acordo com a transformação de *UML!Attribute* para *WSDL!Element* realizado pela regra A2E. Na regra A2E, *name* é definido com o valor de *at.name* e o *type* é definido com o valor de *at.type.name*.

4.1.3 Resultado

Através da execução das Regras definidas na seção 4.1.2, obtivemos o PSM em formato XML. Neste processo de execução o PIM do Sistema de Informações Médica, foi utilizado e fundamental, pois o PSM obtido foi baseado nos elementos pertencentes a este modelo. Todos os elementos pertencentes ao PIM que continham o estereótipo “Service” foram utilizados no processo de transformação.

A execução das regras D2P, C2C e A2E resultam em um PSM em formato XMI que é apresentado no trecho de código 4.1. O elemento do PIM “Lista Remota” possui o estereótipo “Service”, sendo resultante no PSM obtido.

```
1     <?xml version = '1.0' encoding = 'windows-1252' ?>
2     <XMI xmi.version = '1.2' timestamp = 'Tue Mar 07 13:20:26 GMT-03:00 2006'>
3     <XMI.header>
4               <XMI.documentation>
5                  <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
6                  <XMI.exporterVersion>1.0</XMI.exporterVersion>
7           </XMI.documentation
8     </XMI.header>
9     <XMI.content>
10       <WSDL.PrimitiveType xmi.id = 'a1' name = 'int'/>
12       <WSDL.PrimitiveType xmi.id = 'a2' name = 'void'/>
```

```
13      <WSDL.PrimitiveType xmi.id = 'a3' name = 'char[]'/>
14          <WSDL.ComplexType xmi.id = 'a4' name = 'Lista Remota'>
15        <WSDL.ComplexType.sequence>
16          <WSDL.Sequence xmi.id = 'a5'>
17                      <WSDL.Sequence.element>
18              <WSDL.Element xmi.id = 'a6' name = 'nome' type = 'String'/>
19              <WSDL.Element xmi.id = 'a7' name = 'lista' type = 'Lista Refinada'/>
20            </WSDL.Sequence.element>
21                  </WSDL.Sequence>
22        </WSDL.ComplexType.sequence>
23      </WSDL.ComplexType>
24      <WSDL.Element xmi.id = 'a8' name = 'lista' type = 'Lista Refinada'/>
25        <WSDL.Element xmi.id = 'a9' name = 'nome'/>
26  </XMI.content>
27  </XMI>
```

4.3 Fragmento do PSM na execução das regras D2P, C2C e A2E

A transformação do PIM (modelo UML) para PSM (modelo WSDL) é uma transformação do tipo modelo para modelo. Desta forma, para obter o código final (documento WSDL) uma transformação do tipo modelo para código é necessária. Em outras palavras, outra transformação definida em ATL usa o PSM (WSDL) para gerar o documento WSDL, isto é, um documento XML.

4.2 Transformações de UML para WSOracle

As regras de transformações de UML para WSOracle tem como objetivo principal gerar um *Web Service* funcional na plataforma *Oracle* a partir dos metamodelos UML e WSOracle. Para a criação destas regras o estudo da ferramenta *Oracle JDeveloper 10g* (ORACLE, 2005) foi fundamental. O *Oracle JDeveloper 10g* é um software que disponibiliza a seus usuários um ambiente de desenvolvimento integrado com suporte a modelagem, desenvolvimento, debug e *deploying* de aplicações Java e *Web Services*. Através do estudo desta ferramenta identificamos que um *Web Service* criado por ela é criado por dois grupos de arquivos:

- Arquivos de Configuração: arquivos em formato XML que dão suporte a execução do *Web Service* criado pela ferramenta *Oracle JDeveloper 10g*;

- Arquivos de Serviços: arquivos “*.java”* que implementam a interface e o serviço propriamente dito.

Desta forma, verificou-se que para alcançar o objetivo de criar um *Web Service* na plataforma *Oracle* a partir das regras de transformação entre o metamodelo UML e WSOracle, seria necessário criar as regras de transformação em duas etapas:

- Etapa 1: Gerar regras de transformação para os Arquivos de Configuração.
- Etapa 2: Gerar regras de transformação para os Arquivos de Serviços.

Assim, obedecendo aos passos da geração das transformações definidas no item 4, as seguintes atividades foram realizadas para cada etapa:

1. Mapeamento Manual: identificamos as equivalências entre os elementos do metamodelo UML para o metamodelo alvo WSOracle para a etapa de geração das regras de transformação dos Arquivos de Configuração. E também, identificamos as equivalências entre os elementos do metamodelo UML para o metamodelo alvo em Java versão1.5 para a etapa de geração das regras de transformação dos arquivos de Serviços;
2. Escrita da Regra: utilizando-se a ferramenta Eclipse acrescida do *plug-in* da ATL criou-se as regras de transformação baseado no mapeamento manual;
3. Geração do PSM: Após a escrita e execução da regra em ATL, o PSM é gerado a partir do PIM;
4. Geração do Código-Fonte: A partir do PSM e do metamodelo da plataforma alvo é gerado o código-fonte.

4.2.1 Mapeamentos

Mapeamentos para os Arquivos de Configuração

Para esta atividade foram identificadas as equivalências entre o metamodelo fonte (UML) e o metamodelo alvo (WSOracle). O metamodelo WSOracle foi apresentado no Capítulo 3 e foi criado de acordo com os arquivos de configuração identificados no processo de estudo da ferramenta *Oracle JDeveloper 10g*. O resultado destas equivalências é apresentado na Tabela 4.2.

A Tabela 4.2 apresenta todos os mapeamentos encontrados entre o metamodelo UML e o metamodelo WSOracle. Cada mapeamento identificado é uma linha da Tabela 4.2. Assim,

podemos observar que o mapeamento C2Web representa a informação de que um elemento Class do metamodelo UML corresponde a um elemento WebApp no metamodelo WSOracle.

Tabela 4.2 – Mapeamentos entre o metamodelo UML e metamodelo WSOracle.

Mapeamentos		
Nome	**De UML**	**Para WSOracle**
C2Web	Class	WebApp
C2OracleWebServices	Class	OracleWebServices
C2WebServices	Class	WebServices
C2Application	Class	Application
C2OrionApplication	Class	OrionApplication
C2DataSources	Class	DataSources
C2JavaWSDLMapping	Class	JavaWSDLMapping

Mapeamentos para os Arquivos de Serviços

Para esta atividade foram identificadas as equivalências entre o metamodelo fonte (UML) e o metamodelo alvo (Java1.5). O metamodelo Java1.5 foi apresentado no Capítulo 3 e foi criado de acordo com o Java versão 1.5. Uma novidade importante nesta versão foi a utilização intensiva do elemento *Annotations*. *Annotations* são meta-informações que "anotam" (marcação) dados no código (SUN, 2004). O conceito *Annotations* já era presente em versões anteriores a java 1.5 de uma forma um pouco diferente. Como é o caso da marcação *Javadoc @Deprecated* que marca um pedaço de código como depreciado, informando ao programador que evite o uso da API marcada.

O resultado das equivalências (mapeamentos) entre o metamodelo UML e o metamodelo Java1.5 é apresentado na Tabela 4.3. Cada mapeamento identificado é uma linha da Tabela 4.2. Assim, podemos observar, o mapeamento C2JC que representa a informação de que um elemento *Class* do metamodelo UML corresponde a um elemento *JClass* no metamodelo Java1.5.

Tabela 4.3 – Mapeamentos entre o metamodelo UML e metamodelo Java1.5.

Mapeamentos		
Nome	**De UML**	**Para JAVA1.5**
P2JP	Package	JPackage
C2JC	Class	JClass
I2I	Interface	JInterface
M2M	Method + Operation	Jmethod
Pinout2JP	Parameter	JParameter
A2F	Attribute	Jfield
Ae2F	AssociationEnd	Jfield
DT2JPT	DataType	JprimitiveType

4.2.2 Definições de transformação em ATL

Regras de Transformação para os Arquivos de Configuração

As Regras de transformação para a geração dos Arquivos de Configuração estão baseadas na especificação das correspondências apresentadas na Tabela 4.2. Apresentamos, o código-fonte em ATL das regras de transformação, que tem como resultado de execução, os Arquivos de Configuração de um *Web Service* para a plataforma Oracle. Desta forma, as regras C2Web, C2OracleWebServices, C2WebServices, C2Application, C2OrionApplication, C2DataSources e C2JavaWSDLMapping recuperam as informações necessárias para gerar os arquivos de configuração "web.xml", "oraclewebservices.xml", “webservices.xml”, “application.xml”, “orionapplication.xml”, “datasources.xml” e "javawsdlmapping.xml” respectivamente.

Na regra C2Web, temos a especificação de que um elemento *Class* do metamodelo UML é mapeada para um elemento *WebApp* do metamodelo WSOracle. O elemento *WebApp* é a entidade principal do arquivo de configuração “web.xml” do *framework* da Oracle.
Assim, através da execução da Regra C2Web, são obtidas todas as informações pertencentes ao elemento *WebApp*.

No fragmento de código 4.4, apresentamos o código-fonte em ATL da regra C2Web baseado na especificação de correspondência C2Web.

```
1   -- == File : UML2WSORACLE_Deploy.atl
2   Module UML2WSORACLE;
3   Create OUT : WSORACLE  from IN : UML;
4   Rule C2Web{
5       from c: UML!Class (c.stereotype ->exists(e|e.name='Service'))
6       to webapp : WSORACLE!WebApp(
7           xmlns <- 'http://java.sun.com/xml/ns/j2ee'
8           xsi <- 'http://www.w3.org/2001/XMLSchema-instance',
9           version <- '2.4',
10          servlet <- serv,
11          Servletmapping <- smpp
12      ),
13      serv: WSORACLE!Servlet(
14          description <- 'Web Service ' + c.namespace.name +          'SoapHttpPort',
```

```
15          displayname <- 'Web Service ' + c.namespace.name + 'SoapHttpPort',
16          servletname <- c.name + 'SoapHttpPort',
17          Servletclass <- c.namespace.name + '.' + c.name,
18          Loadonstartup <- '1'
19       ),
20       smpp : WSORACLE!ServletMapping(
21          name <- c.namespace.name + 'SoapHttpPort',
22          urlpattern <-c.namespace.name + 'SoapHttpPort'
23       )
24    }
```

4.4 Fragmento do Código-Fonte em ATL da Regra C2Web.

É importante observar na linha 5, do fragmento de código 4.4, que somente as classes que possuírem o estereótipo de nome *Service* terão seus dados mapeados para um elemento *WebApp*.

Na regra C2OracleWebServices, temos a especificação de que um elemento *Class* do metamodelo UML é mapeada para um elemento OracleWebServices do metamodelo WSOracle. O elemento OracleWebServices é a entidade principal do arquivo de configuração "OracleWebServices.xml" do framework da Oracle. Assim, através da execução da Regra C2OracleWebServices, são obtidas todas as informações pertencentes ao elemento OracleWebServices.

No fragmento de código 4.5, apresentamos o código-fonte em ATL da regra C2OracleWebServices, baseado na especificação C2OracleWebServices.

```
1    -- == File : UML2WSORACLE_Deploy.atl
2    module UML2WSORACLE;
3    create OUT : WSORACLE  from IN : UML;
4     Rule C2OracleWebServices{
5     from c : UML!Class (c.stereotype -> exists(e|e.name='Servico'))
6     to oraclews : WSORACLE!OracleWebServices(
7        xmlns <- 'http://www.w3.org/2001/XMLSchema-instance',
8        xsi <- 'http://xmlns.oracle.com/oracleas/schema/oracle-webservices-10_0.xsd',
9         wsdescription <- wsdescr
10        ),
11        wsdescr: WSORACLE!WSDescription(
12        ),
13        name <- c.namespace.name,
```

```
14          portcomponent <- portcomp
15          ),
16                Portcomp: WSORACLE!PortComponent(
17    name <- c.namespace.name + 'SoapHttpPort',
18          operations <- oper
19          ),
20          oper: WSORACLE!Operations(
21          name <- c.feature ->select (e|e.oclIsTypeOf(UML!Operation))->first().name
22          )
23    }
```

4.5 Fragmento do Código-Fonte em ATL da Regra C2OracleWebServices.

É importante observar na linha 5 do fragmento de código 4.5, que somente as classes que possuírem o estereótipo de nome *Service* terão seus dados mapeados para um elemento OracleWebServices.

Regras de Transformação para os Arquivos de Serviço

As Regras de transformação para a geração dos Arquivos de Serviços estão baseadas na especificação das correspondências apresentadas na Tabela 4.3. Apresentamos aqui, o código-fonte em ATL das regras de transformação, que apresentam como resultado de execução, os Arquivos de Serviços de um *Web Service* para a plataforma Oracle. Estes arquivos são arquivos “.java” que implementam a interface e o serviço propriamente dito. Assim como nas regras de transformação dos Arquivos de Configuração, as regras de transformação para os Arquivos de Serviço fazem uso de estereótipo para marcar as classes pertencentes ao PIM que representarão um serviço. ou seja, na geração destas regras, somente os elementos que possuírem o estereótipo *Service* serão transformados em serviços e, consequentemente, terão Arquivos de Serviços gerados. Desta forma, as regras P2JP,C2JC,I2I,M2M,Pinout2JP,A2F,Ae2F,DT2JPT recuperam as informações necessárias para gerar os arquivos de serviços baseadas nas classes com o estereótipo *Service*.

Na regra C2JC, temos a especificação de que um elemento *Class* do metamodelo UML é mapeada para um elemento *JClass* do metamodelo Java1.5. Na regra I2I, temos a especificação de que um elemento *Interface* do metamodelo UML é mapeada para um elemento *JInterface* do metamodelo Java1.5. Na regra M2M, temos a especificação de que um elemento *Method* do metamodelo UML é mapeada para um elemento *JMethod* do

metamodelo Java1.5. O elemento *JClass* é a entidade principal do arquivo de serviço responsável por fornecer os serviços disponíveis como *Web Services*. Estas três regras representam a base para definição do arquivo ".java" que irá representar o *Web Service*.

No fragmento de código 4.6 apresentamos o código-fonte em ATL da regra P2JP baseado na especificação de correspondências P2JP.

```
1   Rule P2JP{
2   From pck : UML!Package
3   to jp : JAVAM!JPackage(
4   Name <-pck.name,
5   jelements <- pck.ownedElement
6   )
7   }
```

4.6 Fragmento do Código-Fonte em ATL da Regra P2JP.

Através da regra P2JP um elemento *Package* em UML é mapeado para um elemento *JPackage* em Java.

No fragmento de código 4.7, apresentamos o código-fonte em ATL da regra C2JC baseado na especificação de correspondência C2JC.

```
1   rule C2JClass{
2       from c : UML!Class (c.stereotype -> exists(e|e.name='Service'))
3       to conf : JAVAM!JClass(
4         name <- c.name,
5         visibility <- if c.visibility = #vk_public then
6                   #public
7                   else if c.visibility = #vk_private then
8                   #private
9                   Else
10                  #protected
11                  endif  endif,
12        modifier <- if c.isAbstract then
13                  #jabstract
14                  else if c.isLeaf then
15                  #final
16                  Else
17                  #regular
18                  endif        endif,
```

```
19        isActive <- c.isActive,
20        jpackage <- c.namespace,
21        super <- if c.generalization -> select(e|e.parent <> c)->size() > 0
22                  Then
23                     c.generalization -> select(e|e.parent <> c)-> first().parent
24                  Else
25                     JAVAM!Jclass
26                  endif,
27        implements <- c.clientDependency -> select(e|e.stereotype->exists(e|e.name='realize'))-
    >collect(e|e.supplier)
28        annotation <- annot
29        ),
30        annot:JAVAM!Annotation(
31                  value <- '@WebService(name = ' + c.name + ',serviceName = ' + c.name +
    ',targetNamespace=http://' + c.name + '.ws'
32        )
33  }
```

4.7 Fragmento do Código-Fonte em ATL da Regra C2JC.

É importante observar na linha 5, do fragmento de código 4.7, que somente as classes que possuírem o estereótipo de nome *Service* terão seus dados mapeados para um elemento *JClass*.

4.2.3 Resultados

Neste segmento, os resultados obtidos das execuções das regras apresentadas na seção 4.2.2 serão apresentados. Estes resultados são obtidos obedecendo-se os seguintes passos:

1. **Execução das Regras escritas em ATL para os Arquivos de Configuração:** Para a execução destas regras são necessários os metamodelos UML e WSOracle. Também, a utilização do modelo PIM é necessária. Estes arquivos devem estar em formato XMI. A utilização destes arquivos dar-se conforme apresentado na Figura 4.3. Na entrada IN, o metamodelo fonte UML é apresentado. Na entrada OUT, o metamodelo WSOracle é apresentado. Para o processo de transformação, o modelo PIM também é necessário. Visto que é a partir deste modelo que será gerado o documento resultado desta execução, um arquivo PSM. Em nossa execução denominamos este arquivo de “Resultado.xmi”;

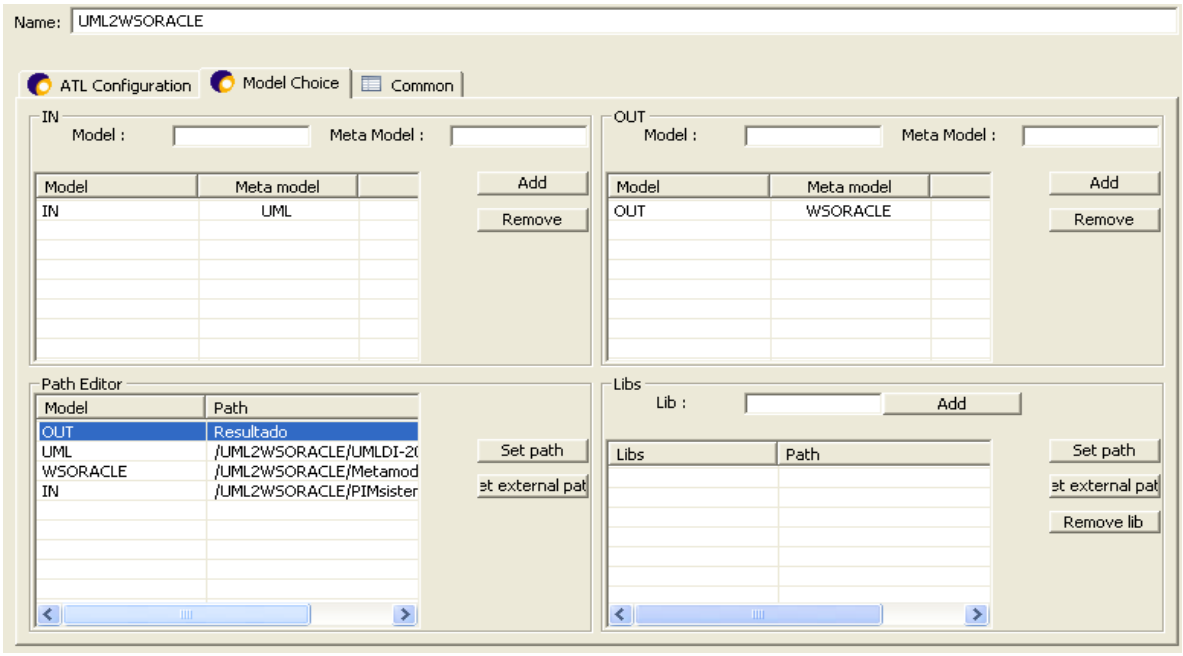

Figura 4.3 Processo de transformação de UML para WSOracle em ATL para os arquivos de configuração.

Após a obtenção do arquivo “Resultado.xmi”, novas regras são executadas com a finalidade de criar os Arquivos de Configuração. Conforme a Figura 4.4, temos, na entrada IN, o metamodelo WSORACLE, o arquivo Resultado.xmi e as bibliotecas para a criação dos Arquivos de Configuração ("web.xml", "oraclewebservices.xml", “webservices.xml”, “application.xml”, “orionapplication.xml”, “datasources.xml” e "javawsdlmapping.xml”), ou seja, estas regras geram de fato os Arquivos de Configuração na máquina.

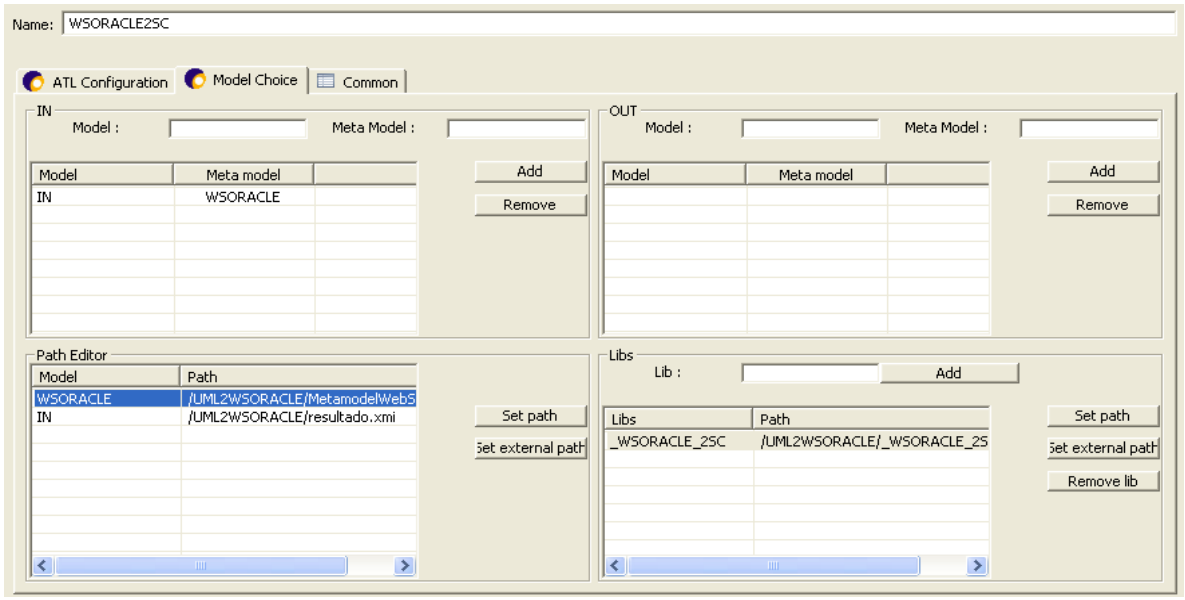

Figura 4.4 Processo de transformação para criação dos arquivos de configuração WSOracle.

2. **Execução das Regras escritas em ATL para os Arquivos de Serviços:** Para a execução destas regras são necessários os metamodelos UML e Java. Também, é necessária a utilização do modelo PIM. Estes arquivos devem estar em formato XMI. A utilização destes arquivos dar-se conforme apresentado na Figura 4.5. Na entrada IN, o metamodelo fonte UML é apresentado. Na entrada OUT, o metamodelo Java é apresentado. Para o processo de transformação, o modelo PIM também é necessário. Visto que é a partir deste modelo que será gerado o documento resultado desta execução, um arquivo PSM. Em nossa execução denominamos este arquivo de "result_javasc.xmi";

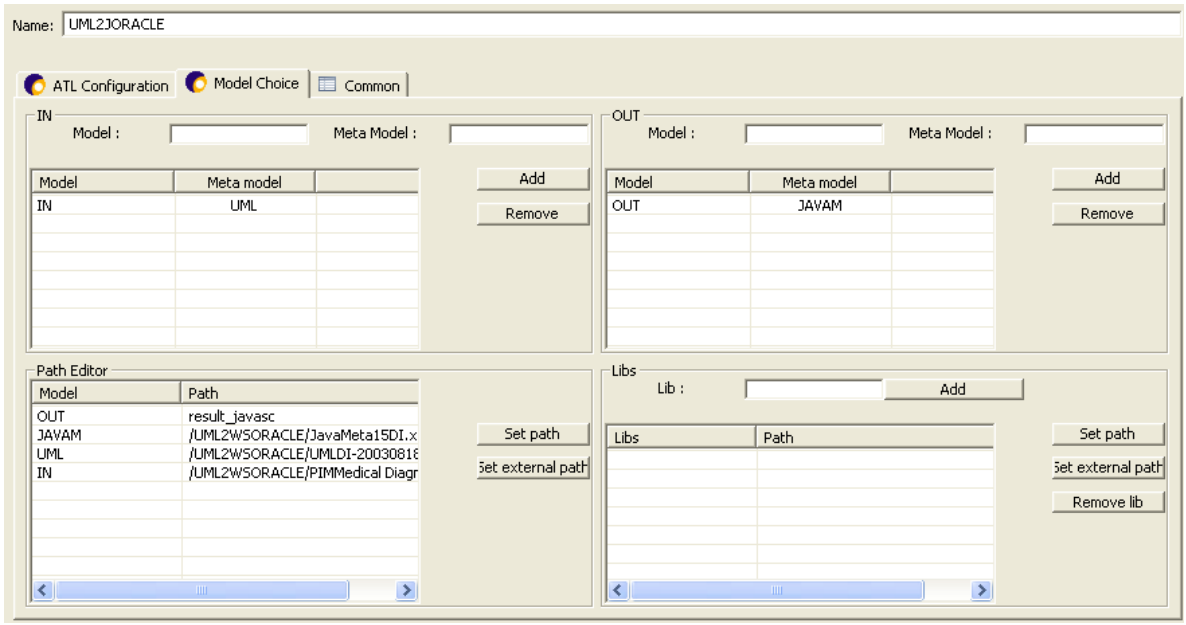

Figura 4.5 Processo de transformação de UML para WSOracle em ATL para os arquivos de serviços.

Após a obtenção do arquivo "result_javasc.xmi", novas regras são executadas com a finalidade de criar os Arquivos ".java" do serviço. Conforme a Figura 4.6, temos o metamodelo Java, o arquivo "result_javasc.xmi" e as bibliotecas "Strings" e "Java2SorceCode" para a criação dos Arquivos ".java", ou seja, estas regras geram de fato os Arquivos ".java" na máquina.

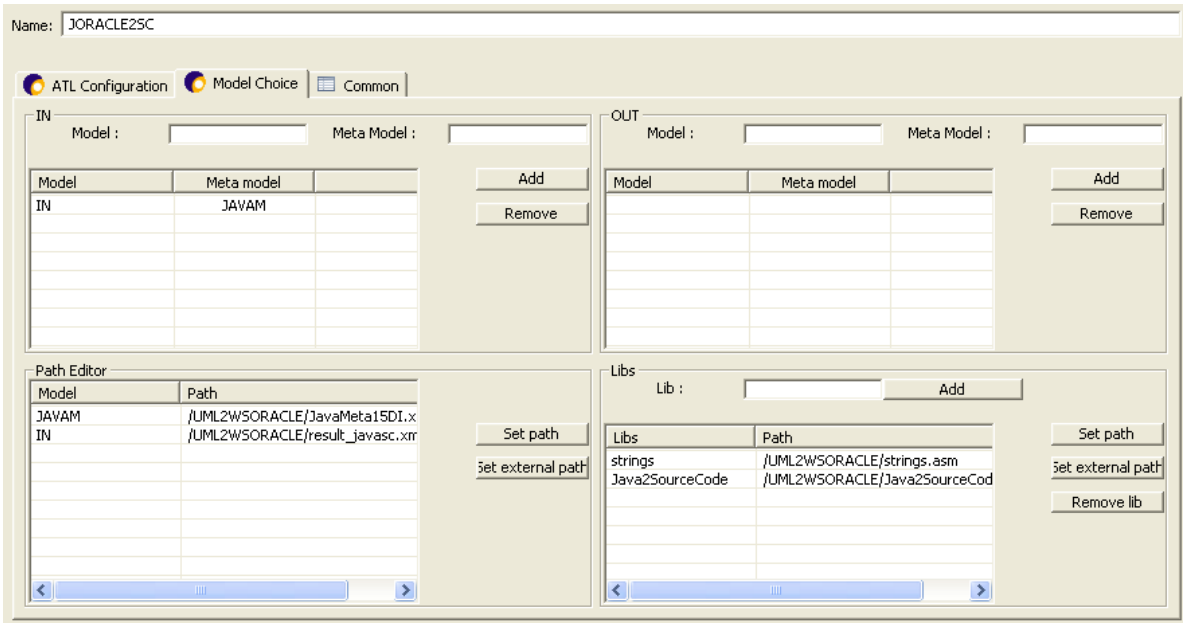

Figura 4.6 Processo de transformação para criação dos arquivos de serviço.

Apresentaremos a seguir, os resultados obtidos a partir destes passos.

Resultados obtidos para os Arquivos de Configuração

A geração dos Arquivos de Configuração para o metamodelo WSOracle é obtida através da regra WSORACLE_Deploy2SC_query. Esta regra em conjunto com um arquivo de bibliotecas denominado _WSORACLE_2SC captura as informações do arquivo "Resultado.xmi" e cria os arquivos de configuração em formato de um documento XML.

O fragmento de código 4.8 apresenta um trecho do código-fonte da regra WSORACLE_Deploy2SC_query.

```
1   query _WSORACLE_Deploy2SC_query =
2   WSORACLE!WebApp.allInstances()->
3   Select(e | e.oclIsTypeOf(WSORACLE!WebApp))->
4   collect(x | x.toString().writeTo('C:/SourceCode/ORACLE/J2EE/' + 'MedicalSystem' + '/' +'web.xml'))->size() +
5
6   WSORACLE!OracleWebServices.allInstances()->
7   Select(e | e.oclIsTypeOf(WSORACLE!OracleWebServices))->
8   collect(x | x.toString().writeTo('C:/SourceCode/ORACLE/J2EE/' + 'MedicalSystem' + '/'
    +'oraclewebservices.xml'))->size() +
9   . . .
10  Uses _WSORACLE_2SC;
```

4.8 Fragmento de Código da Regra WSORACLE_Deploy2SC_query

A linha 3, do fragmento de código 4.8, indica que se o elemento encontrado no arquivo ***Resultado.xmi*** for do tipo *WebApp*, será selecionado então todas as suas propriedades. A linha 4, do mesmo fragmento informa que para todo elemento do tipo *WebApp* será chamado uma função *toString()*. Esta função faz parte do arquivo de bibliotecas _WSORACLE_2SC. O fragmento de código 4.9 apresenta esta função.

```
1  helper context WSORACLE!WebApp def: toString() : String =
2      '<?xml version="1.0" encoding="UTF-8" ?>\n' +
3      '\t<web-app xmlns="'+ self.xmlns+'"\n'+
4      '\t\txsi.schemaLocation="'+ self.xsi +'"\n' +
5      '\t\tversion="'+ self.xsi +'">\n' +
6      self.servlet.toString() +
7      self.servletmapping.toString() +
8      '</web-app>';
```

4.9. Fragmento de Código da biblioteca WSORACLE_2SC_query

O arquivo “web.xml” é criado através da execução da função toString() do elemento WebApp e também através do uso dos valores das suas propriedades conforme o fragmento de código 4.9. Na linha 4 do fragmento de código 4.8, indicamos que o arquivo “*web.xml”*, será criado no diretório “C:\SourceCode\Oracle\J2EE\MedicalSystem\”.

O fragmento de código 4.10 apresenta o arquivo de configuração “*web.xml”* obtido a partir da execução da regra “WSORACLE_Deploy2SC_query”.

```
1   <?xml version = '1.0' encoding = 'windows-1252' ?>
2   <XMI xmi.version = '1.2' timestamp = 'Tue Feb 07 15:57:59 GMT-03:00 2006'>
3   <XMI.header>
4   <XMI.documentation>
5       <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
6       <XMI.exporterVersion>1.0</XMI.exporterVersion>
7   </XMI.documentation>
8   </XMI.header>
9   <XMI.content>
10  <WSORACLE.web.WebApp xmi.id = 'a1'
11      xmlns ='http://java.sun.com/xml/ns/j2ee'
12      xsi = 'http://www.w3.org/2001/XMLSchema-instance' version = '2.4'>
13      <WSORACLE.web.WebApp.servletmapping>
```

```
14          <WSORACLE.web.ServletMapping xmi.id = 'a2'
15            name = 'SistemaInformacoesMedicoSoapHttpPort'
16            urlpattern = 'SistemaInformacoesMedicoSoapHttpPort'/>
17        </WSORACLE.web.WebApp.servletmapping>
18        <WSORACLE.web.WebApp.servlet>
19          <WSORACLE.web.Servlet xmi.id = 'a3'
20            description = 'Web Service SistemaInformacoesMedicoSoapHttpPort'
21            displayname = 'Web Service SistemaInformacoesMedicoSoapHttpPort'
22            servletname = 'ListaRemotaSoapHttpPort'
23            servletclass = 'SistemaInformacoesMedico.ListaRemota'
24            loadonstartup = '1'/>
25        </WSORACLE.web.WebApp.servlet>
26    </WSORACLE.web.WebApp>
27    </XMI.content>
28    </XMI>
```

4.10 Fragmento de Código do arquivo web.xml

Os demais arquivos de configuração são obtidos da mesma forma.

Resultados obtidos para os Arquivos de Serviços

A geração dos Arquivos de Serviços para o metamodelo Java é obtida através da regra “Java2SourceCode_query”. Esta regra em conjunto com um arquivo de bibliotecas denominado “_Java_2SC” captura as informações do arquivo “Result.xmi” e cria os arquivos de serviços em formato Java na máquina.

O fragmento de código 4.11 apresenta um trecho do código-fonte da regra “Java_Deploy2SC_query”.

```
1   query Java2SourceCode_query =
2   JAVA!JClassifier.allInstances()->
3   select(e | e.oclIsTypeOf(JAVA!JClass) or
4   e.oclIsTypeOf(JAVAM!JInterface))->
5   collect(x | x.toString().writeTo('C:/SourceCode/ORACLE/J2EE/' + x.jpackage.name.replaceAll('.', '/') + '/' + x.name +
    '.java'));
6   . . .
7   uses Java2SourceCode;
```

4.11. Fragmento de Código da Regra Java_Deploy2SC_query

O fragmento de código 4.12 apresenta um trecho do arquivo de serviço “ListaRemota.java”, gerado a partir da execução das regras de transformação. Neste arquivo temos a presença do elemento *Annotattion* utilizado tanto para a classe quanto para os métodos que esta classe possui.

```
1   package sistemainformacoesmedico;
2   @WebService (name = "ListaRemota" , serviceName="ListaRemota", targetNamespace =
    "http://SistemaInformacoesMedico.ws")
3   public Class ListaRemota {
4       public ListaRemota() {
5       }
6   @WebResult (name="result", targetNamespace="http://SistemaInformacoesMedico.ws")
7   @WebMethod (operationName = "EncontrarBasesInformacoesCompartilhadas", action="")
8   public int EncontrarBasesInformacoesCompartilhadas () {
9   }
10  @WebResult (name="result", targetNamespace="http://SistemaInformacoesMedico.ws")
11  @WebMethod (operationName = "CriarListaRemota", action="")
12  public int CriarListaRemota () {
13  }
14  }
```

4.12. Fragmento de Código da classe ListaRemota.java

Na linha 2 do fragmento de código 4.12, demonstramos o uso do *Annotation* usado para auxiliar na descrição dos serviços disponibilizados através da Classe Lista Remota. Da mesma forma, temos na linha 6 e 7 o uso do *Annotation*, desta vez auxiliando na descrição dos Métodos da Classe. Este arquivo e os demais arquivos “.java” se encontram no diretório “C:\SourceCode\ ORACLE\J2EE\”.

4.3 Transformações de UML para JWSDP

As regras de transformações de UML para JWSDP tem como objetivo principal gerar a partir dos metamodelos UML e JWSDP um *Web Service* funcional na plataforma JWSDP. Para a criação destas regras foi fundamental o estudo da ferramenta *Java Web Services Developer Pack* versão 1.5 (JWDP) (SUN, 2005).

O Java Web Services Development Pack é a principal ferramenta Java da Sun para construção de aplicações distribuídas que comunicam-se através de Web Services. O JWSDP

é um pacote da Sun específico para a criação de *WebServices* com Java. Este pacote contém as *libs* (bibliotecas) necessárias para criar e executar um *Web Service*, vindo também com um servidor de aplicação próprio, o *Sun Application Server*. Desta forma, depois de um estudo da ferramenta identificamos que um *Web Service* criado no JWSDP é composto por dois grupos de arquivos:

- Arquivos de Configuração: arquivos em formato XML que dão suporte a execução do *Web Service* criado pela ferramenta JWSDP;
- Arquivos de Serviços: arquivos ".java" que implementam a interface e o serviço propriamente dito.

Assim, para atingir o objetivo de criar um *Web Service* na plataforma JWSDP 1.5 com MDA será necessário a criação de regras de transformação entre o metamodelo UML e JWSDP. Para a criação dessas regras de transformação são necessárias duas etapas:

- Etapa 1: Gerar regras de transformação para os Arquivos de Configuração;
- Etapa 2: Gerar regras de transformação para os Arquivos de Serviços.

Com isso, obedecendo aos passos da geração das transformações definidas na seção 4, as seguintes atividades foram realizadas para cada etapa:

1. Mapeamento Manual: identificamos as equivalências entre os elementos do metamodelo fonte UML para o metamodelo alvo JWSDP para a etapa de geração das regras de transformação dos Arquivos de Configuração. Também, identificamos as equivalências entre os elementos do metamodelo UML para o metamodelo alvo Java 1.5 para a etapa de geração das regras de transformação dos Arquivos de Serviços;
2. Escrita da Regra: utilizando-se a ferramenta Eclipse acrescida do *plug-in* de ATL criou-se as regras de transformação baseado no mapeamento manual definido anteriormente;
3. Geração do PSM: Após a escrita e execução da regra em ATL, o PSM é gerado a partir do PIM;
4. Geração do Código-Fonte: A partir do PSM e do metamodelo da plataforma alvo é gerado o código-fonte.

4.3.1 Mapeamentos

Mapeamentos para os Arquivos de Configuração

Identificamos as equivalências entre o metamodelo fonte (UML) e o metamodelo alvo (JWSDP). Nós nos baseamos no metamodelo de JWSDP proposto por Denivaldo Lopes (2005) para JWSDP versão 1.3 para propormos um novo metamodelo para a versão 1.5. O resultado destas equivalências é apresentado na Tabela 4.4.

Tabela 4.4 – Mapeamentos entre o metamodelo UML e metamodelo JWSDP.

Mapeamentos		
Nome	**De UML**	**Para WSOracle**
C2ConfigInterface	Class	ConfigInterface
C2ConfigWsdl	Class	ConfigWsdl
C2WebServices	Class	WebServices
C2Web	Class	Web
C2JavaWSDLMapping	Class	JavaWSDLMapping

A Tabela 4.4 apresenta todos os mapeamentos encontrados entre o metamodelo UML e o metamodelo JWSDP. Cada mapeamento identificado é uma linha da Tabela 4.4. Assim, como exemplo podemos observar o mapeamento C2ConfigInterface que representa a informação de um elemento *Class* do metamodelo UML corresponde a um elemento *ConfigInterface* no metamodelo JWSDP.

Mapeamentos para os Arquivos de Serviços

Identificamos as equivalências entre o metamodelo fonte (UML) e o metamodelo alvo (Java) como mostrado na Tabela 4.3 do item 4.2.1. Cada mapeamento identificado é uma linha da Tabela 4.3.

4.3.2 Definições de transformação em ATL

Regras de Transformação para os Arquivos de Configuração

As Regras de transformação para a geração dos Arquivos de Configuração estão baseadas na especificação das correspondências apresentadas na Tabela 4.4. Nesta seção,

apresentamos, o código-fonte em ATL das regras de transformação que apresentam como resultado de execução, os Arquivos de Configuração de um *Web Service* para a plataforma JWSDP. Desta forma, as regras C2ConfigInt, C2ConfigWsdl, C2Web, C2WebServices, e C2JavaWSDLMapping recuperam as informações necessárias para gerar os arquivos de configuração "configinterface.xml", "configwsdl.xml", “web.xml”, “webservices.xml” e "javawsdlmapping.xml”, respectivamente.

Na regra C2ConfigInt, temos a especificação de que um elemento *Class* do metamodelo UML é mapeada para um elemento *_Configuration* do metamodelo JWSDP. O elemento *_Configuration* é a entidade principal do arquivo de configuração “ConfigInterface.xml” do metamodelo *JWSDP*. Assim, através da execução da Regra C2ConfigInt são obtidas todas as informações pertencentes ao elemento _Configuration.

No fragmento de código 4.13, apresentamos o código-fonte em ATL da regra C2ConfigInt, baseado na especificação C2ConfigInterface.

```
1   -- == File : UML2JWSDP_Deploy.atl
2   Module UML2_JWSDP_Deploy;
3   Create OUT : JWSDP  from IN : UML;
4   Rule C2ConfigInt{
5       from c : UML!Class (c.stereotype -> exists(e|e.name='Service'))
6       to conf : JWSDP!_Configuration(
7           xmlns <- 'http://java.sun.com/xml/ns/jax-rpc/ri/config',
8           service <- serv
9           ),
10          serv: JWSDP!Service(
11          name <- c.name,
12          targetnamespace <- 'http://' + c.namespace.name + '.org/wsdl',
13          typenamespace <- 'http://' + c.namespace.name + '.org/wsdl',
14          packageName <- c.namespace.name,
15          interface <- interf
16              ),
17          interf : JWSDP!Interface(
18          name <- c.feature.name
19      )
24  }
```

4.13 Fragmento do Código-Fonte em ATL da Regra C2ConfigInt.

É importante observar na linha 5, do fragmento de código 4.13, que somente as classes que possuírem o estereótipo de nome *Service* terão seus dados mapeados para um elemento "*_Configuration"*.

Na regra C2ConfigWsdl, temos a especificação de que, um elemento *Class* do metamodelo UML é mapeada para um elemento "*Configuration"* do metamodelo JWSDP. O elemento "*Configuration"* é a entidade principal do arquivo de configuração "ConfigInterface.xml" do metamodelo *JWSDP*. Assim, através da execução da Regra C2ConfigWsdl, são obtidas todas as informações pertencentes ao elemento "*Configuration*".

No fragmento de código 4.14, apresentamos o código-fonte em ATL da regra "C2ConfigWsdl", baseado na especificação "C2ConfigWsdl".

```
1   -- == File : UML2JWSDP_Deploy.atl
2   Module UML2_JWSDP_Deploy;
3   Create OUT : JWSDP  from IN : UML;
4   Rule  C2ConfigWsdl{
5       from c : UML!Class(c.stereotype -> exists(e|e.name='Servico'))
6       to web : JWSDP!Configuration(
7           xmlns <- 'http://java.sun.com/xml/ns/jax-rpc/ri/config',
8           wsdl <- wsdl
9           ),
10          wsdl: JWSDP!_WSDL(
11          location <- 'http://' + c.namespace.name + '.org:8099/jaxrpc-' + c.namespace.name + '/' + c.name +
    '?wsdl',
12          packageName <- c.namespace.name + '.service'
19      )
24  }
```

4.14. Fragmento do Código-Fonte em ATL da Regra C2ConfigWsdl.

É importante observar na linha 5, do fragmento de código 4.14, que somente as classes que possuírem o estereótipo de nome *Service* terão seus dados mapeados para um elemento *Configuration*.

Regras de Transformação para os Arquivos de Serviços

As regras de transformação para a geração dos arquivos de serviços estão baseadas na especificação das correspondências apresentadas na Tabela 4.3 na Seção 4.2.2 "Regras de

transformação para arquivos de serviços". Apresentamos o código-fonte em ATL das regras de transformação que apresentam como resultado de execução, os Arquivos de Serviços de um *Web Service* para a plataforma JWSDP. Estes arquivos são arquivos ".java" que implementam a interface e o serviço propriamente dito.

Assim como nas regras de transformação dos Arquivos de Configuração, as regras de transformação para os Arquivos de Serviço fazem uso de estereótipo para marcar as classes pertencentes ao PIM que representarão um serviço. Ou seja, na geração destas regras, somente os elementos que possuírem o estereótipo "*Service*", serão transformados em serviços e consequentemente, terão Arquivos de Serviços gerados. Desta forma, as regras P2JP, C2JC, I2I, M2M, Pinout2JP, A2F, Ae2F, DT2JPT recuperam as informações necessárias para gerar os arquivos de serviços baseadas nas classes com o estereótipo "*Service*". Na Seção 4.2.2 são apresentadas algumas regras de um metamodelo UML para um metamodelo Java.

No fragmento de código 4.15, apresentamos o código-fonte em ATL da regra P2JP, baseado na especificação P2JP.

```
1   Rule P2JP{
2   From pck : UML!Package
3   to jp : JAVAM!JPackage(
4   Name <-pck.name,
5   jelements <- pck.ownedElement
6   )
7   }
```

4.15 Fragmento do Código-Fonte em ATL da Regra P2JP.

Através da regra P2JP um elemento *Package* em UML é mapeado para um elemento *JPackage* em JAVA.

No fragmento de código 4.16, apresentamos o código-fonte em ATL da regra C2JC baseado na especificação de correspondência C2JC.

```
1   rule C2JC{
2       from c : UML!Class (c.stereotype -> exists(e|e.name='Service'))
3       to conf : JAVAM!JClass(
4         name <- c.name,
5         visibility <- if c.visibility = #vk_public then
6                   #public
7                   else if c.visibility = #vk_private then
```

```
8                   #private
9                   Else
10                  #protected
11                  endif  endif,
12          modifier <- if c.isAbstract then
13                  #jabstract
14                  else if c.isLeaf then
15                  #final
16                  Else
17                  #regular
18                  endif          endif,
19          isActive <- c.isActive,
20          jpackage <- c.namespace,
21          super <- if c.generalization -> select(e|e.parent <> c)->size() > 0
22                  Then
23                     c.generalization -> select(e|e.parent <> c)-> first().parent
24                  Else
25                     JAVAM!Jclass
26                  endif,
27          implements <- c.clientDependency -> select(e|e.stereotype->exists(e|e.name='realize'))-
    >collect(e|e.supplier)
28          )
29  }
```

4.16 Fragmento do Código-Fonte em ATL da Regra C2JC.

É importante observar na linha 5, do fragmento de código 4.16, que somente as classes que possuírem o estereótipo de nome *Service* terão seus dados mapeados para um elemento *JClass*.

4.3.3 Resultado

Neste segmento serão apresentados os resultados obtidos das execuções das regras apresentadas na Seção 4.2.2. Estes resultados são obtidos obedecendo-se os seguintes passos:

1. **Execução das Regras escritas em ATL para os Arquivos de Configuração**: para a execução destas regras são necessários os metamodelos UML e JWSDP. Também, a utilização do modelo PIM é necessária. Estes arquivos devem estar em formato XMI. A utilização destes arquivos deu-se conforme apresentado na Figura 4.7. Na entrada IN, o metamodelo fonte UML é apresentado. Na entrada OUT, o metamodelo JWSDP

é apresentado. O modelo PIM também é necessário. A partir deste modelo, será gerado o documento resultado desta execução, um arquivo PSM. Em nossa execução denominamos este arquivo de “Result.xmi”;

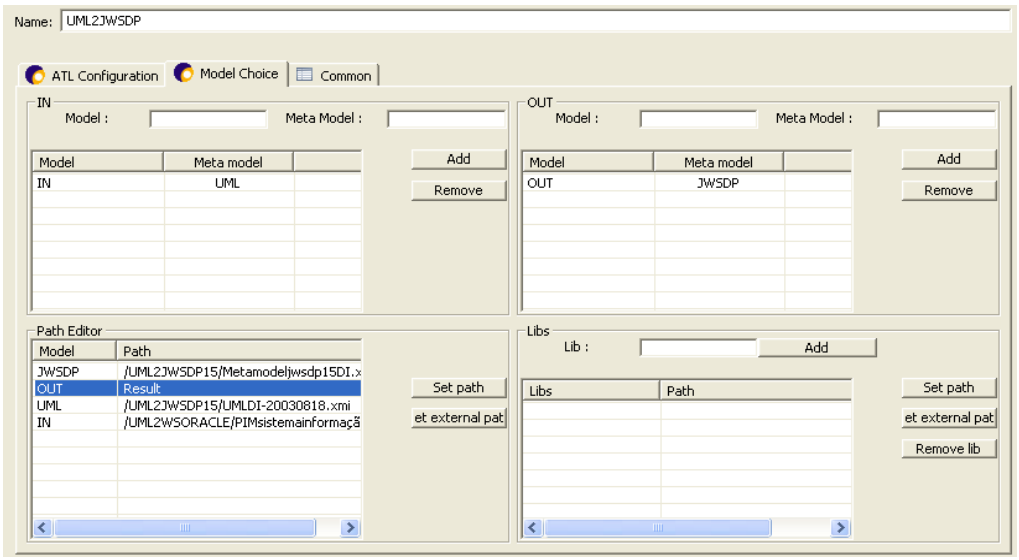

Figura 4.7. Processo de transformação de UML para JWSDP em ATL para os arquivos de configuração.

Após a obtenção do arquivo “Result.xmi”, novas regras são executadas com a finalidade de criar os Arquivos de Configuração. Conforme a Figura 4.8, temos, na entrada IN, o metamodelo JWSDP, o arquivo Result.xmi e as bibliotecas para a criação dos Arquivos de Configuração ("configinterface.xml", "configwsdl.xml", “webservices.xml”, “web.xml”, e "javawsdlmapping.xml”), ou seja, estas regras geram de fato os Arquivos de Configuração na máquina.

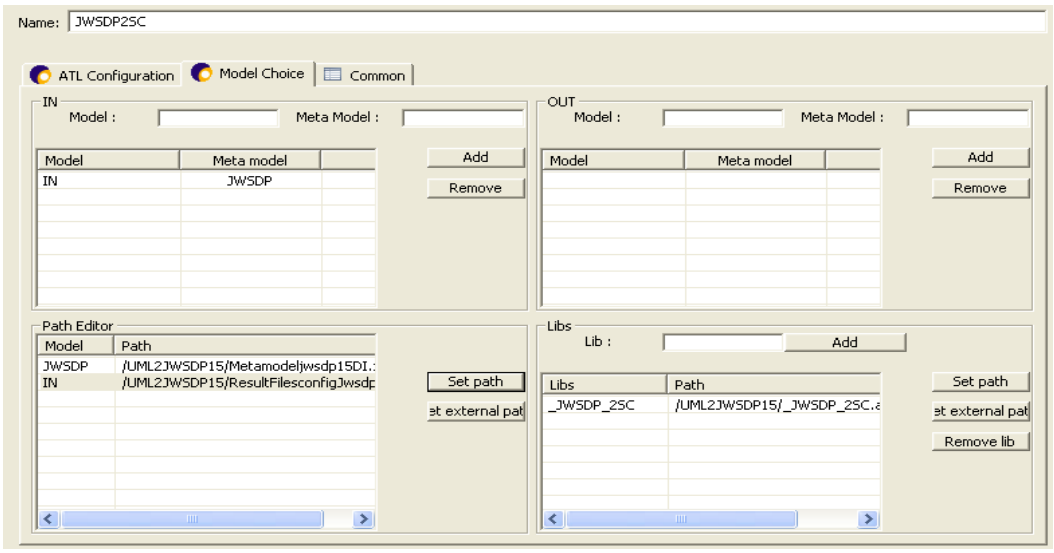

Figura 4.8 Processo de transformação para criação dos arquivos de configuração JWSDP.

2. **Execução das Regras escritas em ATL para os Arquivos de Serviços:** Para a execução destas regras são necessários os metamodelos UML e Java. Também, é necessária a utilização do modelo PIM. Estes arquivos devem estar em formato XMI. A utilização destes arquivos deu-se conforme apresentado na Figura 4.9. Na entrada IN, o metamodelo fonte UML é apresentado. Na entrada OUT, o metamodelo Java é apresentado. Para o processo de transformação, o modelo PIM também é necessário. Visto que é a partir deste modelo, que será gerado o objeto resultado desta execução, um arquivo PSM. Em nossa execução denominamos este arquivo de "result_javasc.xmi";

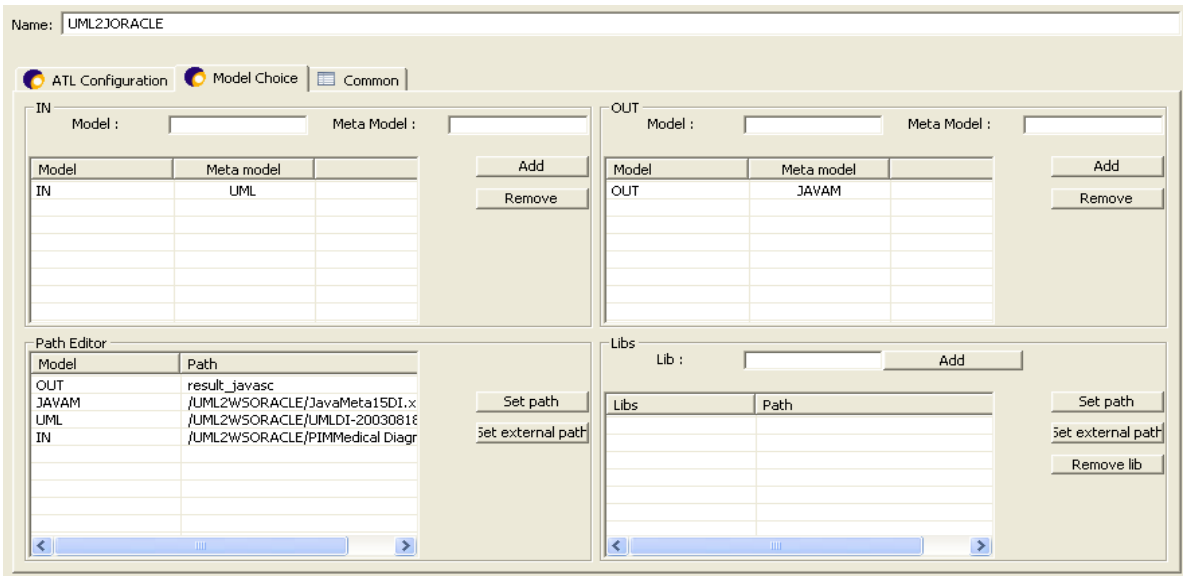

Figura 4.9 Processo de transformação de UML para Java em ATL.

Após a obtenção do arquivo "result_javasc.xmi", novas regras são executadas com a finalidade de criar os Arquivos .java do serviço. Conforme a Figura 4.10, temos, o metamodelo Java, o arquivo "result_javasc.xmi" e as bibliotecas "Strings" e "Java2SorceCode" para a criação dos Arquivos ".java", ou seja, estas regras geram de fato os Arquivos ".java" na máquina.

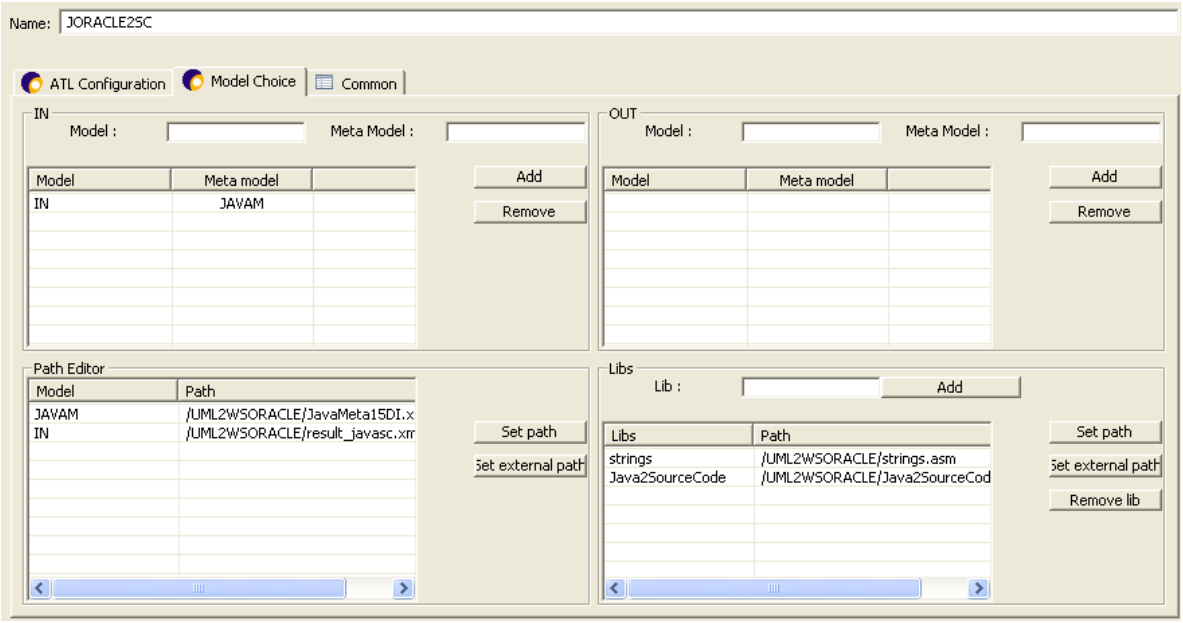

Figura 4.10 Processo de transformação para criação dos arquivos de serviço em Java.

Apresentaremos a seguir, os resultados obtidos a partir destes passos.

Resultados obtidos para os Arquivos de Configuração

A geração dos Arquivos de Configuração para o metamodelo JWSDP é obtida através da regra "JWSDP_Deploy2SC_query". Esta regra em conjunto com um arquivo de bibliotecas denominado "_JWSDP_2SC" captura as informações do arquivo "Result.xmi" e cria os arquivos de configuração em formato XML.

O fragmento de código 4.17 apresenta um trecho do código-fonte da regra "JWSDP_Deploy2SC_query".

```
1   query _JWSDP_Deploy2SC_query =
2   JWSDP!_Configuration.allInstances()->
3   select(e | e.oclIsTypeOf(JWSDP!_Configuration))->
4   collect(x | x.toString().writeTo('C:/SourceCode/JWSDP/J2EE/' +' 'SistemaInformacaoMedica' + '/' +
    'configinterface.xml'))->size() +
5
6   JWSDP!Configuration.allInstances()->
7   select(e | e.oclIsTypeOf(JWSDP!Configuration))->
8   collect(x | x.toString().writeTo('C:/SourceCode/JWSDP/J2EE/' + 'SistemaInformacaoMedica' + '/' +
    'configwsdl.xml'))->size() +
9   . . .
10  Uses _WSORACLE_2SC;
```

4.17. Fragmento de Código da Regra JWSDP_Deploy2SC_query

A linha 3 do fragmento de código 4.17, indica que se o elemento encontrado no arquivo "*Result.xmi*" for do tipo "*_Configuration"*, será selecionado então todas as suas propriedades. A linha 4 do mesmo fragmento, informa que para todo elemento do tipo "*_Configuration"*, será chamado uma função *toString()*. Esta função faz parte do arquivo de bibliotecas "_JWSDP_2SC". O fragmento de código 4.18 apresenta esta função.

```
1   helper context JWSDP!_Configuration def: toString() : String =
2       '<?xml version="1.0" encoding="UTF-8"?>\n' +
3       '<configuration xmlns="'+self.xmlns+'">\n'+
4       self.service.toString()+
```

4.18. Fragmento de Código da biblioteca JWSDP_2SC_query

O arquivo "configinterface.xml" é criado através da execução da função toString() do elemento "*_Configuration"* e também através do uso dos valores das suas propriedades, conforme o fragmento de código 4.18. Na linha 4 do fragmento de código 4.17, indicamos que o arquivo "configinterface.xml", será criado no diretório "C:\SourceCode\JWSDP\J2EE\SistemaInformacaoMedica\".

O fragmento de código 4.19 apresenta o arquivo de configuração "configinterface.xml" obtido a partir da execução da regra "JWSDP_Deploy2SC_query".

```
1   <?xml version = '1.0' encoding = 'windows-1252' ?>
2   <XMI xmi.version = '1.2' timestamp = 'Tue Aug 09 17:53:59 GMT-03:00 2005'>
3     <XMI.header>
4       <XMI.documentation>
5         <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
6         <XMI.exporterVersion>1.0</XMI.exporterVersion>
7       </XMI.documentation>
8     </XMI.header>
9   <XMI.content>
10      <JWSDP.configinterface.Service xmi.id = 'a1' name = 'Lista Remota'
11          targetnamespace = 'http://modelo 2.org/wsdl'
12          typenamespace = 'http://modelo 2.org/wsdl"
13          packageName = 'SistemadeInformacaoMedica'/>
14      < JWSDP.configinterface._Configuration xmi.id = 'a2' xmlns = 'http://java.sun.com/xml/ns/jax-
    rpc/ri/config'/>
15      <JWSDP.configinterface.Interface xmi.id = 'a3'/>
16  </XMI.content>  </XMI>
```

4.19. Fragmento de Código do arquivo configinterface.xml

Os demais arquivos de configuração são obtidos da mesma forma.

Resultados obtidos para os Arquivos de Serviços

A geração dos Arquivos de Serviços para o metamodelo Java é obtida através da regra "Java2SourceCode_query". Esta regra em conjunto com um arquivo de bibliotecas denominado "_Java_2SC" captura as informações do arquivo "Result.xmi". e cria os arquivos de serviços em formato Java na máquina.

O fragmento de código 4.20 apresenta um trecho do código-fonte da regra "Java_Deploy2SC_query".

```
1  query Java2SourceCode_query =
2  JAVA!JClassifier.allInstances()->
3  select(e | e.oclIsTypeOf(JAVA!JClass) or
4  e.oclIsTypeOf(JAVAM!JInterface))->
5  collect(x | x.toString().writeTo('C:/SourceCode/J2EE/' + x.jpackage.name.replaceAll('.', '/') + '/' + x.name +
   '.java'));
6  . . .
7  uses Java2SourceCode;
```

4.20. Fragmento de Código da Regra Java_Deploy2SC_query

O fragmento de código 4.21 apresenta uma biblioteca em ATL usada para transformar um modelo Java em um "Java_2SC".

```
1  helper context JAVA!JClass def: visibility() : String =
2      if self.visibility=#public then
3       'public '
4      else if self.visibility=#private then
5      'private '
6      else
7      'protected'
8      endif endif;
```

4.21. Fragmento de Código da biblioteca Java _2SC_query

Na linha 5 do fragmento de código 4.20, indicamos que os arquivos ".java", serão criados no diretório "C:\SourceCode\ JWSDP\J2EE\".

O fragmento de código 4.22 apresenta um trecho do arquivo de serviço "ListaRemota.java", gerado a partir da execução das regras de transformação.

```
1   package sistemainformacoesmedico;
2   public Class ListaRemota {
3       public ListaRemota() {
4       }
5   public int EncontrarBasesInformacoesCompartilhadas () {
6   }
7   public int CriarListaRemota () {
8   }
9   }
```

4.22. Fragmento de Código da classe ListaRemota.java

Este arquivo e os demais arquivos ".java" se encontram no diretório "C:\SourceCode\ JWSDP\J2EE\".

4.4 Conclusão

Neste capítulo, apresentamos as especificações de correspondências e definições de transformação entre os metamodelos: UML e WSDL; UML e WSOracle; UML e JWSDP.

Verificamos, que as regras de transformação em ATL são criadas manualmente a partir da especificação das correspondências entre os metamodelos.

Usamos a ferramenta ATL para criar e executar as regras de transformação, devido esta ferramenta estar de acordo com MDA. Em seguida, observamos que, usando regras de transformação do metamodelo UML para WSDL, conseguimos um documento WSDL como resultado. O resultado das regras de transformação de UML para WSOracle e de UML para JWSDP cria arquivos de configuração e arquivos de serviços. Os arquivos de configuração são criados no formato ".xml" da plataforma específica (WSOracle e JWSDP) e os arquivos de serviços são criados no formato ".java" da plataforma Java. Mostramos também que as regras de transformações são aplicadas ao PIM do Sistema de Informação Médica, pois é a partir do PIM que geramos os PSMs das plataformas.

Esta abordagem MDA permite a geração de documentos WSDL, arquivos de configuração e de desenvolvimento. Porém, permite somente a geração dos esqueletos do código de Java para os Web Services em JWSDP e para o WSOracle. Depois, completamos manualmente os esqueletos do código em Java. Algumas pesquisa (BEZIVIN et al., 2004a) apontam a ação semântica para suportar a geração completa do código baseada na linguagem de programação como Java.

Por fim, conseguimos observar que a diferença da transformação entre UML para WSOracle e UML para JWSDP está nos arquivos de configuração e na forma como as API’s são usadas.

5 PROTOTIPAGEM DO SISTEMA DE INFORMAÇÃO MÉDICA

5.1 Introdução

Com o propósito de realizar testes e mensurar resultados sobre o modelo proposto no Capítulo 3, os esqueletos de código gerado por nossa abordagem MDA apresentada no Capítulo 4 foram completados para que um protótipo funcional fosse obtido. Neste Capítulo, alguns detalhes da implementação são apresentados.

O protótipo desenvolvido demonstra a aplicabilidade do sistema de informação médica proposto utilizando *Web Service*. Este protótipo foi desenvolvido também com base nos resultados obtidos no Capítulo 4, tais resultados facilitam o desenvolvimento de *software* com intuito de prover soluções de interoperabilidade entre sistemas.

O protótipo foi desenvolvido para o ambiente Web, onde foi utilizada a tecnologia JSP (*Java Server Pages*) que é baseada na linguagem de desenvolvimento Java. O servidor de aplicações Web utilizado foi o Tomcat da Apache. O *Web Services* foi implementado usando a plataforma da Sun conhecida como JWSDP.

Todo o processo de implementação foi baseado nas classes e arquivos de configuração obtidos no Capítulo 4. As classes resultantes do processo de transformação em MDA do metamodelo UML para o metamodelo JWSDP, nortearam o desenvolvimento do protótipo usando a ferramenta JWSDP.

O processo aplicado para o desenvolvimento, bem como as características de implementação e ferramentas utilizadas são descritas nas seções subseqüentes.

5.2 Implementação do Protótipo

Nós iniciamos a explicação do desenvolvimento do protótipo fazendo alguns comentários sobre as ferramentas utilizadas.

As principais ferramentas utilizadas para o desenvolvimento do protótipo são:

- Eclipse (ECLIPSE, 2005);
- JWSDP (SUN, 2005);
- Tomcat (APACHE TOMCAT, 2005).

A ferramenta Eclipse foi utilizada para o desenvolvimento das classes Java, já o JWSDP é utilizado para o desenvolvimento do *Web Services* e o Tomcat para hospedar as aplicações Web. Nas seções seguintes, estas ferramentas são apresentadas com mais detalhes.

5.3 Implementação do Protótipo

Para uma melhor estruturação do protótipo proposto a sua arquitetura foi dividida no uso dos seguintes elementos: base de conhecimento, serviço de acesso a base, interface gráfica com o usuário.

5.3.1 Base de conhecimento

A base de conhecimento foi construída utilizando o SGBD MySQL 5.0 (MySQL, 2005). O MySQL é um dos SGBD's relacionais livres mais conhecidos no mundo. Nas versões anteriores a 5.0, o mesmo deixava a desejar, pois ainda não proporcionava todas as funcionalidades que um SGBD deve fornecer para gerenciar as informações de uma empresa como, por exemplo, transações e regras de integridade referencial. Mas na versão 5.0, este SGBD já possui praticamente as mesmas funcionalidades que muitos SGBD's comerciais (MySQL, 2005).

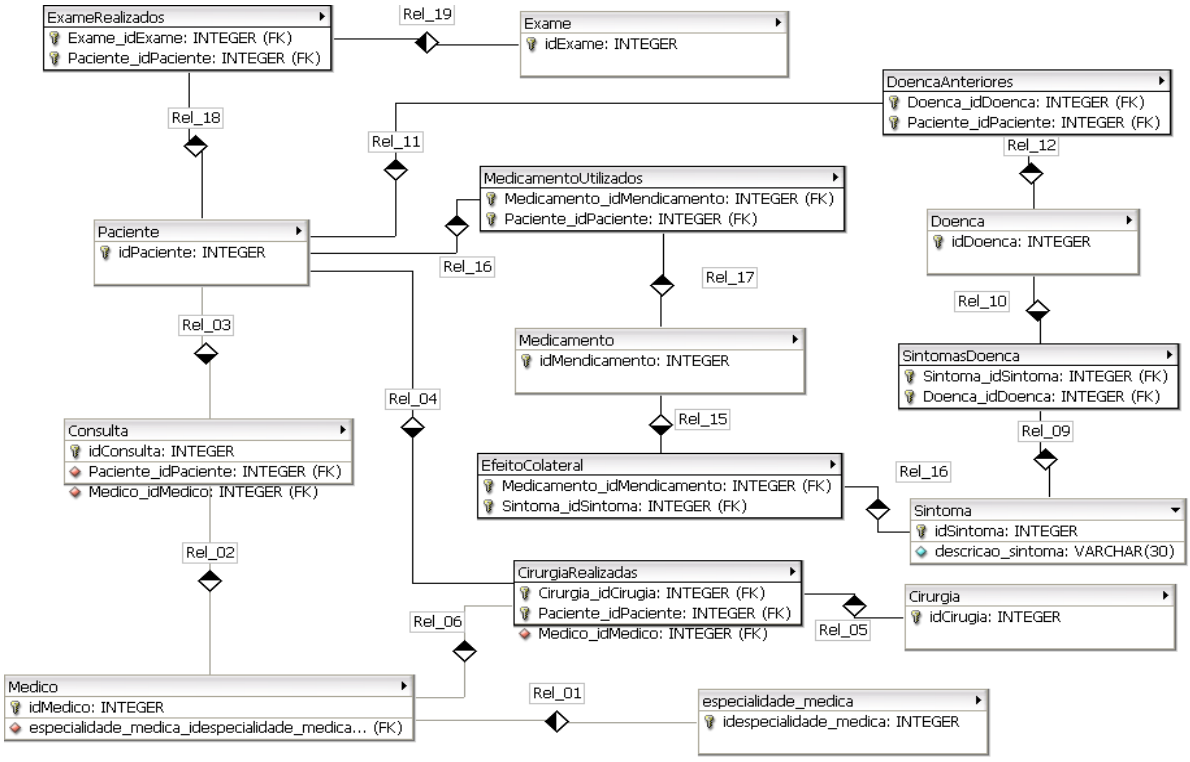

Figura 5.1 – Modelo Relacional da Base de Conhecimento médica proposta.

Em nosso protótipo, o modelo relacional considera várias entidades que armazenam informações pertinentes a assuntos médicos como: exames, doenças, pacientes, consultas, medicamentos, médicos, etc. A Figura 5.1 exibe como estas entidades estão relacionadas entre si.

Dentre as entidades listadas pode-se destacar o relacionamento entre as entidades “doencas” e “sintomas” que é chamado de “sintomasdoenca”. Este relacionamento visa integrar as informações da primeira entidade com a segunda, onde a primeira é responsável por armazenar as informações sobre as doenças que um paciente pode apresentar, já a segunda indica as possíveis queixas de um paciente. Assim, a entidade “sintomasdoenca” contém sintomas que uma doença pode apresentar.

5.3.2 Serviço de Acesso a Base com *Web Service*

O protótipo foi implementado com o intuito de acessar e manipular bases de dados com informações médicas através da utilização de um *Web Service*, ou seja, o serviço implementado neste protótipo visa disponibilizar funcionalidades que permitam o usuário gerenciar informações cadastradas em um banco de dados.

Para disponibilizar o serviço alguns requisitos são necessários:

1. Possuir um servidor de aplicações Web

No caso deste protótipo foi utilizado o servidor de aplicações Web conhecido como Apache Tomcat.

2. Implementar o serviço que será disponibilizado

Para implementar o serviço a plataforma JWSDP da Sun foi utilizada. Ela é uma implementação Java para *Web Service* que pode funcionar sobre o servidor de aplicações Web chamado de Tomcat. A Figura 5.2 mostra a página principal do JWSDP configurada dentro do Tomcat.

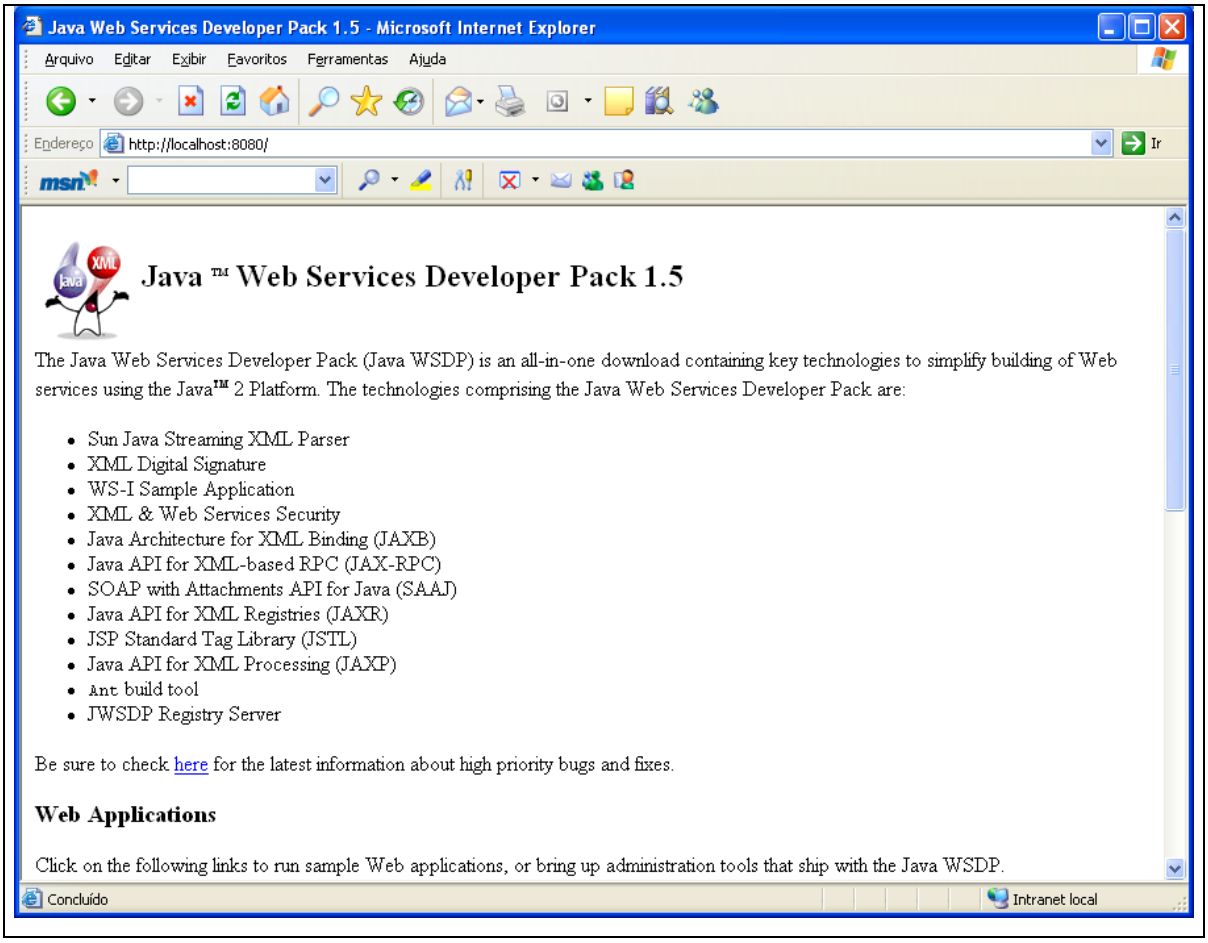

Figura 5.2 – Página principal do serviço JWSDP.

3. Disponibilizar o serviço

Nesta etapa a interface do serviço foi criada e esta é disponibilizada via *Web Service*. A Tabela 5.1 apresenta alguns dos métodos que são disponibilizados por este serviço.

Tabela 5.1 - Métodos do Serviço Web Diagnóstico Médico

Nome dos Métodos do Serviço
CadastrarCirurgias
CadastrarDoencas
CadastrarExames
CadastrarMedicamentos
CadastrarSintomas
CadastrarPaciente
RemoverExames
AssociarDoencas
AssociarSintomas
RemoverMedicamentos
EncontrarBasesInformacaoCompartilhadas
CriarListaRemota

Todos os métodos disponibilizados nesse serviço tem como objetivo manipular informações da base que contém as informações médicas. Por exemplo, o método “CriarListaRemota” que pode ser visto no Fragmento de Código 5.1 tem como objetivo retornar uma lista de todas as doenças associadas as queixas que um paciente pode apresentar.

```
public Object CriarListaRemota( String[] sintomas ){
    try{
        String sql =  "SELECT D.DESCRICAO, SUM(SD.PESO) AS PESO, COUNT(*) AS TOTAL "
                    + "FROM DOENCA D, SINTOMASDOENCA SD";
                    + "WHERE D.IDDOENCA = SD.DOENCA_IDDOENCA AND SINTOMA_IDSINTOMA IN (";
        for( int i = 0; i < sintomas.length; i++ ){
            sql += sintomas[i];
            if( i <= sintomas.length - 2 )
               sql += ",";
        }
        sql += ") GROUP BY D.IDDOENCA ORDER BY SD.PESO";

        ResultSet result = this.Consultar(sql);
        if( result != null && sintomas.length > 0 ){
            Vector doencas = new Vector();

            while( result.next() ){
                Doenca doenca = new Doenca();
                doenca.setDescricao(result.getString("descricao"));
                doenca.setPeso(result.getString("peso");

                 doencas.add(doenca);
            }

            return (Object)doencas;
        }

    }catch(Exception e){
        e.printStackTrace();
    }
}
```

Fragmento de Código 5.1 – Trecho do método CriarListaRemota.

5.3.3 Interface gráfica com o usuário

O protótipo foi implementado com o intuito de acessar bases de dados com informações médicas através da utilização de um *Web Service*, mas para que esse acesso aconteça de forma transparente, ou seja, sem exigir conhecimentos técnicos do usuário, é necessário disponibilizar uma interface gráfica que auxilie este usuário.

A interface do usuário que será apresentada foi desenvolvida para o ambiente Web. A mesma apresenta diversas funções que são disponibilizadas por meio de um *Web Service*.

Dentre as interfaces do usuário escolhemos a interface consulta para mostrarmos uma das funcionalidades do protótipo desenvolvido. Esta interface pode ser vista na Figura 5.3. A mesma tem como função auxiliar o médico no seu diagnóstico fornecendo uma listagem de possíveis doenças que se enquadram na descrição detalhada durante a consulta. Por ser apenas um protótipo, na interface consulta o médico considera: os sintomas que o paciente pode estar sentindo e as doenças que o paciente já apresentou. Sendo assim, o diagnóstico gerado pelo

sistema é feito através da solicitação que é realizada após clicar no botão “Gerar Diagnóstico”.

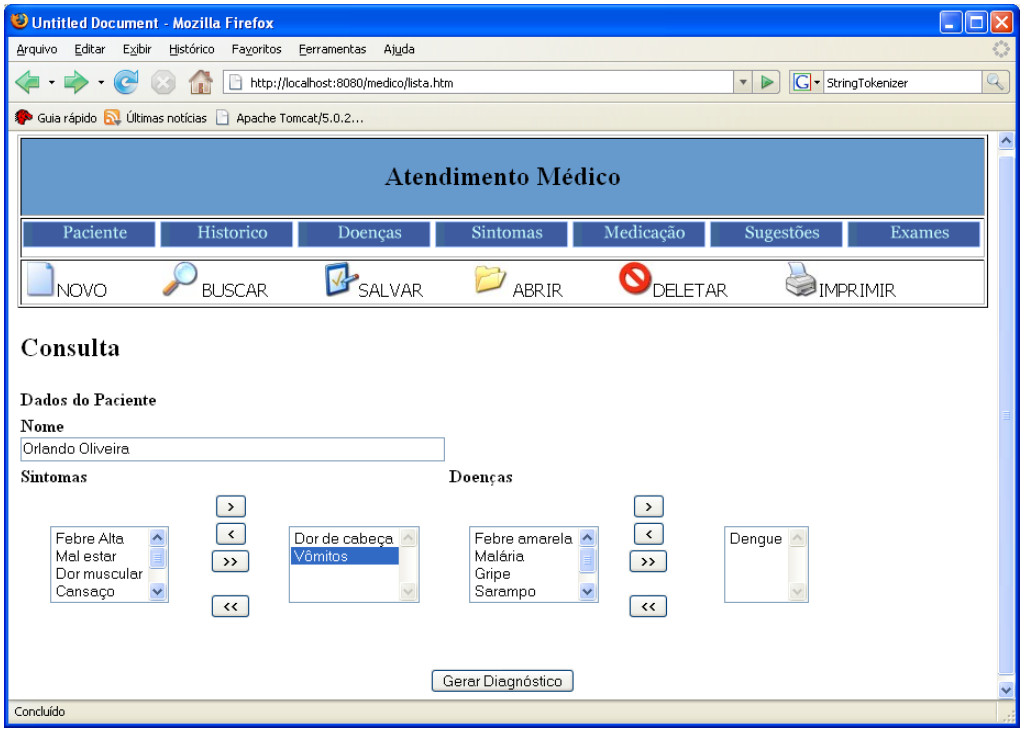

Figura 5.3 – Interface de consulta

No momento em que este botão é pressionado uma requisição, contendo os sintomas e as doenças consideradas é enviada para um método no *Web Service* que gera a lista de prováveis doenças, como pode ser visto na Figura 5.4.

Figura 5.4 – Listagem de prováveis doenças.

Esta lista de prováveis doenças é gerada com base na consulta do paciente para auxiliar o médico na sua tomada de decisão.

5.4 Conclusão

Neste Capítulo, o protótipo do sistema de informação médica foi apresentado, assim como, algumas das ferramentas utilizadas no seu desenvolvimento.

Mostramos basicamente uma visão geral de como o protótipo está estruturado. Considerando três elementos: Base de conhecimento, onde foi utilizado o MySQL, serviço de acesso a base, onde foi utilizado o *Web Service* JWSDP e uma interface gráfica com o usuário, desenvolvida para o ambiente Web, utilizando JSP para auxiliar o usuário na solicitação dos serviços.

Verificou-se que com a utilização do *Web Service*, o sistema de informação médica pôde acessar e manipular bases de conhecimento com informações médicas de forma bem sucedida. Este fato pode auxiliar o médico a tomar as suas decisões devido à possibilidade de poder trocar informações com outras bases.

E, por fim, vale a pena destacar que o desenvolvimento deste protótipo visa apenas validar nossa proposta de um ambiente de informação médica que foi modelado em UML e que parte do código foi gerado através da abordagem MDA. Com base no processo de transformação MDA, obteve-se um esqueleto conforme o modelo PIM. Uma vez que as assinaturas das classes e métodos estavam disponíveis, estas puderam ser completadas para se obter um sistema funcional.

6 CONCLUSÃO

Este trabalho teve como objetivo desenvolver um sistema de informação médica de auxílio no diagnóstico médico, que foi baseado em uma abordagem orientada a modelos, no qual o sistema permite o compartilhamento de informações entre especialistas fisicamente distantes. O desenvolvimento deste sistema colocou em evidência o processo de transformação de PIM para PSM, utilizado em MDA.

Por MDA ser uma arquitetura baseada em modelos, consegue-se mostrar que é uma abordagem neutra, independente de plataforma e de fabricante, e que tem o objetivo de aumentar o nível de abstração na forma de desenvolver softwares, utilizando modelos e metamodelos.

Sendo assim, construímos um modelo PIM para o sistema médico proposto e construímos também metamodelos para as plataformas de *Web Services,* JWSDP e WSOracle.

Com base no PIM e nos metamodelos criados, identificamos as especificações de correspondências e as definições de transformação de UML e WSDL; UML e WSOracle; UML e JWSDP.

Usamos a ferramenta ATL para criar e executar as regras de transformação, e em seguida, observamos que, usando as regras de transformação do metamodelo UML para WSDL, conseguimos um documento WSDL como resultado. Mostramos que as regras de transformação são aplicadas ao PIM do nosso Sistema de Informação Médica, e que o resultado das regras de transformação de UML para WSOracle ou para JWSDP criou arquivos de configuração e arquivos de serviços. Os arquivos de configuração são criados no formato “.xml” específicos para a plataforma WSOracle e para a plataforma JWSDP, e que os arquivos de serviços são criados no formato “.java” da plataforma Java que são semelhantes para as duas plataformas.

A abordagem MDA nos permitiu a geração de documentos WSDL, arquivos de configuração e de desenvolvimento para os *Web Services,* JWSDP e WSOracle. Porém, permitiu somente a geração dos esqueletos do código de Java que já vem com parte do PIM para esses *Web Services*. Depois, completamos manualmente os esqueletos do código em Java.

Conseguimos observar que a diferença da transformação entre UML para WSOracle e UML para JWSDP está nos arquivos de configuração e na forma como as API’s são usadas.

Por fim, mostramos um protótipo do sistema médico proposto destacando os elementos da arquitetura do sistema de informação médica como: Base de conhecimento, onde foi utilizado o

MySQL, serviço de acesso à base, onde foi utilizado o *Web Service* JWSDP e uma interface gráfica com o usuário, desenvolvida para o ambiente Web, utilizando JSP para auxiliar o usuário na solicitação dos serviços.

Verificou-se que, com a utilização do *Web Service*, o sistema de informação médica pode acessar e manipular bases de conhecimento com informações médicas de forma bem sucedida. Este sistema pode auxiliar o médico a tomar as suas decisões devido à possibilidade de poder trocar informações com outras bases.

E, por fim, vale a pena destacar que o desenvolvimento deste protótipo visa apenas validar a utilização de *Web Service* como uma ferramenta para troca de informações médicas.

6.1 Trabalhos Futuros

O uso das tecnologias de *Web services* e MDA para o desenvolvimento do Sistema de Informação Médica permitiram a construção de um modelo independente de plataforma que mostrou-se bem aplicado para as plataformas de *Web Services*.

A aplicação do modelo permitiu definir a metodologia proposta neste trabalho como aplicável para o sistema proposto em diferentes plataformas.

A partir dos resultados alcançados, outros estudos podem ser realizados, e novas técnicas e métodos podem ser incorporados. Como propostas para trabalhos futuros, nós podemos citar:

- Introduzir novas funcionalidades ao sistema de informação médica;
- Usar o Modelo PIM para outras plataformas como a da IBM e dotNet;
- Criar uma padronização baseada em uma abordagem dirigida por modelos para as bases de conhecimento de sistemas de informação médica;
- Acrescentar padrões de segurança no uso dos *Web Services*;
- Pesquisar metodologias e algoritmos para geração semi-automática de definições de transformação a partir de especificações de correspondências;
- Adicionar ação semântica nos modelos para que as regras de transformação possam gerar os códigos de forma automatizada;
- Adicionar o controle de versões entre o desenvolvimento dos modelos e a criação das regras;

- Propor uma metodologia e ferramentas para gerenciar e suportar a evolução de platformas.

6.2 Contribuições

- Desenvolvimento da arquitetura do sistema de informação médica proposto;
- Desenvolvimento do modelo independente de plataforma para o sistema de informação médica;
- Desenvolvimento do metamodelo WSOracle;
- Extensão do metamodelo JWSDP da versão 1.3 para a versão 1.5;
- Extensão do metamodelo Java da versão 1.3 para a versão 1.5;
- Criação das especificações de correspondências e das regras de transformação entre o PIM (UML) e os PSM, WSOracle e JWSDP.

Referências

AMBROGIO, Andréa**. A Model Transformation Framework for the Automated Building of Performance Models from UML Models.** WOSP'05, July 2005;

AMMA. **ATL: Atlas Transformation Language.** 2006. Disponível em http://www.sciences.univ-nantes.fr/lina/atl/. Acesso em: 15/12/2005;

APACHE TOMCAT. **The Apache Software Foundation**. 2005. Disponível em: http://tomcat.apache.org/. Acesso em: 23/07/2006;

APACHE. **AXIS**. 2005. Disponível em: http://ws.apache.org/axis/. Acesso em: 20/06/2005;

AUDY, Rejane B.; ENDRES, Ana Cristina T.; MALVEZZI, Maria Luiza F. **Informações Gerenciais em Hospital de Referência**. In: Anais do VIII Congresso Brasileiro de Informática em Saúde, Natal, 2002;

BARBOSA, A.K.P. **HealthNet: Um Sistema Integrado de apoio ao Telediagnóstico e à Segunda Opinião Médica.** Dissertação de Mestrado em Ciência da Computação, Centro de Informática, UFPE, Recife-PE, Brazil, 2001;

BARBOSA, A.K.P.; NOVAES, M.A.; VASCONCELOS, A.M.L. **A Web Application to Support Telemedicine Services in Brazil**. AMIA´03 Annual Symposium, Washington-USA, 8-12 de Novembro, 2003;

BASIC, Alice Shimada. **Um Ambiente Colaborativo de Apoio ao Diagnóstico Médico Assistido por Computadores de Alto Desempenho.** Dissertação de Mestrado, Universidade de São Paulo, Brasil, 2001;

BEMMEL, J. H. and MUSEN, M. A **Handbook of Medical Informatics**. 1999. Disponível em: http://www.mieur.nl/mihandbook/r_3_3/handbook/home.htm. Acesso em: 18/03/2006;

BÉZIVIN, Jean; JOUAULT, Frédéric; ROSENTHAL, Peter; VALDURIEZ, Patrick; ASSMANN, Uwe; AKSIT, Mehmet; AREND, Rensink. **Modeling in the Large and Modeling in the Small. Proceedings of the European MDA Workshops: Foundations and Applications**, MDAFA 2003 and MDAFA 2004, LNCS 3599, Springer-Verlag GmbH (2005) 33.46;

BÉZIVIN, Jean; DUPÉ, Grégoire; JOUAULT, Frédéric; PITETTE, Gilles and ROUGUI, Jamal Eddine. **First Experiments with the ATL Model Transformation Language: Transforming XSLT into XQuery**. 2nd OOPSLA Workhop on Generative Techniques in the context of Model Driven Architecture, Outubro de 2003;

BEZIVIN, Jean; HAMMOUDI Slimane; LOPES, Denivaldo; JOUAULT, Jouault. **Applying MDA Approach for Web Service Platform.** Enterprise Distributed Object Computing Conference, Eighth IEEE International (EDOC'04), 2004a;

BÉZIVIN, Jean; HAMMOUDI, Slimane; LOPES, Denivaldo; JOUAULT, Frédéric. **Applying MDA Approach to B2B Applications: A Road Map, Workshop on Model Driven Development** (WMDD 2004), Tuesday, June 15, 2004b – at ECOOP 2004, Oslo, Norway (June 14–18, 2004);

BILLIG, Andréas; BUSSE, Susanne; LEICHER, Andréas; Suß, Jorn Guy**. Platform Independent Model Transformation Based on TRIPLE. IFIP International Federation for Information Processing,** pp. 493–511, 2004;

CHRISTENSEN, Erik; CURBERA, Francisco; MEREDITH, Greg; WEERAWARANA, Sanjiva. **Web Services Description Language (WSDL)** 1.1 W3C Note 15 March 2001. Disponível em: http://www.w3.org/TR/wsdl>. Acesso em: 08/09/2006;

COSTA, Marco and DA SILVA, Alberto R. **Arquitetura de Sistemas de Informação e a Iniciativa MDA**. 2003. Disponível em: http://berlin.inesc.pt/alb/static/papers/2003/ UML_MDA_UNI .pdf Acesso em: 23/04/2006;

ECLIPSE. **ATL Home Page**. 2004 Disponível em: http://www.eclipse.org/gmt/atl/. Acesso em: 20/03/2005;

ECLIPSE. **Eclipse - An Open Development Platform**. 2005. Disponível em: http://www.eclipse.org/. Acesso em: 05/11/2006;

ECLIPSE. **Eclipse Modeling Framework** (EMF). Disponível em: http://www.eclipse.org/ modeling/emf/?project=emf. 2006. Acesso em: 20/08/2006;

FILHO, Antonio Luis; LABIDI, Sofiane; ADELOUHAD, Zair. **SET-WS: Sistema Especialista para Telediagnóstico no ambiente de Web Services**. Publicação: [SBQS] III: 2004, 31 maio a 4 junho: Brasília;

GUDGIN, Martin; HADLEY, Marc; MENDELSOHN, Noah; MOREAU, Jean- Jacques; NIELSEN, Henrik Frystyk. SOAP Version 1.2 Part 1: **Messaging Framework W3C Recommendation** 24 June 2003. Disponível em: http://www.w3.org/TR/soap12-part1/. Acesso em: 08/09/2006;

HENDRYX & ASSOCIATES. **Business Rules with MDA**. http://www.lcc.uma.es/~av/RM-ODP/Presentations/BusinessRulesWithMDA(andODP)-02-10-05.pdf. Acesso em: 13/01/2006;

IBM. **WebSphere**. 2005. Disponível em http://www.ibm.com.br/products/software/websphere/. Acesso em: 08/05/2005;

IMASTER. **Entendendo os Web Services**. Disponível em: http://www.imasters.com.br/ artigo/4245/webservices/entendendo_os_webservices. 2006. Acesso em: 21/12/2006;

IWEB. **Web Services**. Disponível em: http://www.iweb.com.br/iweb/pdfs/20031008-webservices-01.pdf. 2003. Acesso em: 21/12/2006;

JOUAULT, Frédéric; ALLILAIRE, Freddy; BÉZIVIN, Jean; KURTEV, Ivan and ALDURIEZ, Patrick. **ATL: a QVT-like transformation language**. Conference on Object Oriented Programming Systems Languages and Applications, 2006;

KLEPPE, A; WARMER J; BAST W. **MDA Explained: The Model Driven Architecture Pratice and Promise**. 2ª Ed. EUA. Addison-Wesley – Object Technology Series. 169p, 2003;

LOPES, Denivaldo. **Estudo e aplicações de uma abordagem MDA para as plataformas de Services We**b. Tese de Doutorado, Universidade de Nantes, Nantes, 2005. Disponível em: http://www.dee.ufma.br/~dlopes/dlopes_thesis.pdf. Acesso em 12/12/2006;

MAGALHÃES, Eder**. Annotations em uso**. 2006. Disponível em: http://www.plugmasters. com.br/ sys/materias/541/1/Annotations-em-uso Acesso em: 05/11/2006;

MARQUES, José Alves. **Visão que incorpora na arquitetura tecnológica o suporte aos conceitos SOA**. 2005. Disponível em: https://dspace.ist.utl.pt/bitstream/2295/ 55252/1/7_SOA_.pdf. Acesso em: 05/10/2006;

MATULA, M. **NetBeans Metadata Repository** - WhitePaper, 2003. Disponível em: http://mdr.netbeans.org/MDR-whitepaper.pdf. Acesso em: 08/08/2006;

MATULA, Martin. **UML2MOF Tool NetBeans / Sun Microsystems Netbeans**. 2004. Disponível em: http://mdr.netbeans.org/uml2mof/. Acesso em: 05/09/2006;

MELO, Simone A. B de; LOPES, Denivaldo; ABDELOUAHAB, Zair: **Developing Medical Information System with MDA and Web Services**. Software Engineering Research and Practice 2006: 562-568

MICROSOFT. **.NET Framework**. 2006. Disponível em: http://www.microsoft.com/brasil/ msdn/framework/Default.mspx. Acesso em: 20/06/2005;

MIRANDA, Ruy. **Timidez e Fobia Social**. Disponível em: http://www.timidez-ansiedade.com/. 2006. Acesso em: 04/08/2006;

MySQL. **MySQL 5.0.** 2005. Disponível em: http://www.mysql.com/. Acesso em 05/01/2006;

NETBEANS. **MetaData Repository (MDR)**. 2006. Disponível em: http://mdr.netbeans.org. Acesso em: 20/03/2005;

OASIS-UDDI. **OASIS UDDI Advancing Web Services Discovery Standard**. 2005. Disponível em: http://www.uddi.org/. Acesso em: 08/09/2006;

OMG. **Meta-Object Facility (MOF) specification, OMG document ad/03-02-08**. 2002. Disponível em: http://www.omg.org/technology/documents/formal/mof.htm. Acesso em: 08/09/2006;

OMG. **Model Driven Architecture (MDA)**. 2006. Disponível em: http://www.omg.org/mda. Acesso em: 08/09/2006;

OMG. **Unified Modeling Language (UML) specification**. 2005a. Disponível em: http://www.omg.org/ technology/documents/formal/uml.htm. Acesso em: 08/09/2006;

OMG. **MOF 2.0 / XMI Mapping Specification**, v2.1. 2005b. Disponível em: http://www.omg.org/ technology/documents/formal/xmi.htm. Acesso em: 26/09/2006;

OMG. **UML Profile for enterprise distributed Object Computing (EDOC)**. 2004. Disponível em: http://www.omg.org/technology/documents/formal/edoc.htm. Acesso em: 18/11/2006;

ORACLE. **Oracle JDeveloper**. 2005. Disponível em: http://www.oracle.com/technology/ products/jdev/index.html. Acesso em: 05/04/2006;

PISA, Ivan Torres; LOPES, Paulo Roberto de Lima; HOLANDA, Adriano Jesus de; PIRES, Daniel Facciolo; RUIZ, Evandro Eduardo Seron. **MIDster: sistema distribuído de imagens médicas baseado em modelos peer-to-peer (P2P) e serviços web.** Em: IX Congresso Brasileiro de Informática em Saúde - CBIS2004, 2004, Ribeirão Preto, 2004;

PORTAL MÉDICO. **Ato Médico**. 2002. Disponível em: http://www.portalmedico.org.br/jornal/ jornais2002/novembro/pag_5.htm. Acesso em: 06/05/2006;

POSTEL, Jonathan B. **SMTP - SIMPLE MAIL TRANSFER PROTOCOL** – RFC 821. 1982. Disponível em: http://www.ietf.org/rfc/rfc0821.txt. Acesso em: 08/09/2006;

SBIS - Sociedade Brasileira de Informática em Saúde. **O que é Informática em Saúde?** 2005. Disponível em: http://www.sbis.org.br/. Acesso em: 08/09/2006;

SIGULEM, Daniel. **Um Novo Paradigma de Aprendizado na Prática Médica da UNIFESP/EPM**. São Paulo, 1997. Disponível em: http://www.virtual.epm.br/material/tis/curr-med/infosaude. Acesso em: 25/10/2006;

STREIBEL, Martin. **Implementando Web Services**. Trabalho de Especialização, Universidade Federal do Rio Grande do Sul, Porto Alegre, Dezembro de 2005. Disponível em: http://palazzo.pro.br/artigos/martin.htm. Acesso em 10/06/2006;

SUN MICROSYSTEMS, INC. **The Java Web Services Tutorial 1.3**. 2006. Disponível em http://java.sun.com/webservices/downloads/webservicestutorial.html. Acesso em: 08/09/2006;

SUN. **Java J2SE 1.5.** 2004. Disponível em http://java.sun.com/j2se/1.5.0/docs/index.html Acesso em: 12/04/2006;

SUN. **Java Web Services Developer Pack (JWSDP) 1.5**. 2005. Disponível em http://java.sun.com/webservices/down-loads/1.5/index.html. Acesso em: 10/01/2006;

SUN. **ONE**. 2006. Disponível em: http://br.sun.com/produtos-solucoes/software/amb_aberto.html. Acesso em: 10/04/2005;

VASCONCELOS, José Braga de. **Fundamentos Sistemas de Informação focados no paciente**. Portugal, 2003. Disponível em: http://www2.ufp.pt/~jvasco/InfoMedica-1Mod.pdf. Acesso em: 20/07/2006;

WAINER, Jacques; CAMPOS, Carlos José Reis de; SIGULEM, Daniel. **O que é pesquisa em informática em saúde?** 2004. Disponível em: http://www.sbis.org.br/cbis9/arquivos/545.pdf. Acesso em: 17/04/2006;

W3C. **Extensible Markup Language (XML)**. 2003. Disponível em: http://www.w3.org/XML/. Acesso em: 08/09/2006;

W3C. **HTTP Specifications and Drafts**. 2002. Disponível em: http://www.w3.org/Protocols/Specs.html. Acesso em: 08/09/2006;

W3C. **Web Services Activity**. 2006. Disponível em: http://www.w3.org/2002/ws/. Acesso em: 30/10/2006.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo