IGOR SCALIANTE WIESE

UM MODELO DE INTEROPERABILIDADE PARA AMBIENTES DE DESENVOLVIMENTO DISTRIBUÍDOS DE SOFTWARE

MARINGÁ

2006

IGOR SCALIANTE WIESE

UM MODELO DE INTEROPERABILIDADE PARA AMBIENTES DE DESENVOLVIMENTO DISTRIBUÍDOS DE SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientadora: Profª. Drª. Elisa Hatsue Moriya Huzita

MARINGÁ

2006

Dados Internacionais de Catalogação-na-Publicação (CIP)

(Biblioteca Central - UEM, Maringá – PR., Brasil)

Wiese, Igor Scaliante

W651m Um modelo de interoperabilidade para ambientes de desenvolvimento distribuído de software / Igor Scaliante Wiese. -- Maringá : [s.n.], 2006.

90 f. : il., figs.

Orientadora : Profª. Drª. Elisa Hatsue Moriya Huzita.

Dissertação (mestrado) - Universidade Estadual de Maringá. Programa de Pós-graduação em Ciência da Computação, 2006.

1. Ambiente de desenvolvimento distribuído de software. 2. Interoperabilidade. 3. Metamodelos. I. Universidade Estadual de Maringá. Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 22.ed. 004.6

DEDICATÓRIA

Dedico este trabalho

Aos meus pais, ***Alceu Wiese*** e ***Cleusa Fátima Scaliante Wiese***, pelos pais amorosos, dedicados, compreensivos e incentivadores que sempre foram.

AGRADECIMENTOS

Agradeço primeiramente a **DEUS**, pela dádiva da vida, a saúde, a família, os amigos e o livre arbítrio.

Aos meus pais **Alceu Wiese** e **Cleusa Fátima Scaliante Wiese**, por todo o amor, paciência, carinho e principalmente por todo o incentivo. Sem a força e o apoio irrestrito dos meus amados pais, eu não encontraria forças para realizar esta difícil tarefa. Agradeço muito o presente que **DEUS** me deu, "os meus pais". Agradeço a ele por "ter escolhido" pais maravilhosos, que sempre me ensinaram princípios e valores fundamentais para minha vida. Vocês significam e representam o verdadeiro sentido da palavra "**AMOR**".

Também agradeço ao restante da **minha família** (tias, primos, padrinhos e madrinhas...) que também souberam entender os momentos de ausência e comprometimento com este trabalho, e por todo incentivo e carinho oferecido a mim ao longo de minha vida.

Aos meus **amigos**, agradeço o apoio, a compreensão e o carinho que sempre depositaram sobre mim. (desculpe não mencionar o nome de cada um de vocês, pois esta seria uma tarefa difícil!).

Agradeço à minha orientadora **Elisa Hatsue Moriya Huzita**, que depositou sua confiança em mim e em meu trabalho. Agradeço a ela por toda sua atenção, paciência, dedicação, incentivo, orientação, conselhos e conhecimento compartilhado comigo.

Agradeço a todos os professores que, de forma direta ou indireta, contribuíram com este trabalho, dentre eles: **Itana Maria de Souza Gimenes, Tânia Fátima Calvi Tait, Maria Madalena Dias, Antonio Mendes**, **Cristina A. D. Ciferri** e **Ricardo R. Ciferri**.

Agradeço aos amigos integrantes do grupo de pesquisa GESSD. **Lúcia Enami, Éderson Amorim, César Alberto da Silva, Flavio Schiavoni, Igor Steinmacher, Fabiana Lima, Rafael Gatto, Willian e Gustavo** pela amizade, respeito, incentivo e contribuições.

Agradeço a **todos os amigos** da turma 2004 do mestrado. O companheirismo e a cumplicidade nos momentos difíceis em que passamos com certeza nos fortaleceram.

Aos amigos **Rogério Santos Pozza** e **Edson Oliveira Alves**. Vocês foram os grandes incentivadores antes mesmo de ingressar como aluno regular no mestrado e hoje são exemplo de profissionais e pessoas com o qual me espelho. Obrigado!

Aos amigos de moradia e dia-a-dia **Luiz Arthur Feitosa, André Pozza, Ricardo Alexandre Lavarias, Kitio** e **Diego Pereira Franco.** Vocês são grandes amigos. Sempre me incentivaram e estiveram presentes nos momentos difíceis. Agradeço de forma especial todo esse carinho e atenção de vocês.

Também devo agradecer em especial, muito especial aos amigos que fiz no mestrado, amigos estes que formam também o time de FUTSAL do mestrado. **Francisco Thesko, Éderson Amorim, César Alberto da Silva, Rafael Liberato, André Schwerz, Rogério Santos Pozza, Fábio Zaupa.** Vocês foram muito mais que um "time de futsal", não ganhamos "apenas um campeonato" juntos. Vocês se tornaram amigos que onde quer que eu esteja

carregarei comigo as lembranças de um tempo ótimo. Vocês demonstraram o verdadeiro e real sentido da palavra "**AMIZADE**". Agradeço eternamente tudo que fizeram por mim, a atenção, o carinho, o apoio, as contribuições. Espero mesmo que o destino possa nos reservar muitos encontros ao longo de nossas vidas.

Agradeço aos funcionários do Departamento de Informática **Jayme F. Junior**, **Paulo C. Gonçalves**, **Marcelo R. Donato**, **Madalena** e, em especial à **Maria Inês Davanço**, pela amizade e paciência.

Agradeço especialmente ao **CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico**, pelo apoio financeiro despendido a este trabalho.

Agradeço aos bons compositores (dentre eles **Renato Russo, Almir Sater, Humberto Gessinger, Bono Vox,** e tantos outros...), e as boas bandas de MPB, rock e pop nacionais e internacionais. A Música sempre foi um incentivo e as letras e melodias uma inspiração.

Ao **Rotary International, Rotaract Club e Interact Club**, agradeço pelos ensinamentos éticos e profissionais, bem como por tudo que aprendi nesta brilhante organização não governamental. A Luta pela erradicação da Pólio e tantas outras lutas por um mundo melhor só me fazem orgulhoso e agradecido de ter tido a oportunidade de ter contribuído ao menos um pouquinho com esta causa tão nobre. Aos **amigos** que fiz neste "movimento", certamente tiveram grande participação na pessoa que me tornei hoje, agradeço do fundo do coração e ratifico a saudade dos momentos inesquecíveis que passei e que certamente passarei com vocês.

Por fim, não poderia de citar jamais, o time do meu coração, o **Sport Club Corinthians Paulista**. Ser corintiano é "especial". A torcida, a sua história, os momentos de glória e choro, enfim.. Agradeço ao meu querido pai **Alceu Wiese** e minha querida **avó Eurides e Cilly**, por me ensinarem o que é ser corintiano desde os primeiros meses de vida. "Corinthians Minha Vida, Corinthians Minha História, Corinthians Meu Amor" – como canta a fiel torcida!. "**VAIIII CORINTHIANSSSSS!!!!**"

“Aprender é a única coisa de que a mente nunca se cansa, nunca tem medo e nunca se arrepende." (Leonardo da Vinci).

RESUMO

Diversas ferramentas conhecidas como ferramentas CASE (*Computer-Aided Software Engineering*), têm sido utilizadas no desenvolvimento de software. Porém, estas ferramentas não têm trabalhado de forma integrada, mas sim de forma isolada, sem compartilhar informações ou serviços. Mesmo que a utilização individual destas ferramentas possa trazer benefícios para o desenvolvimento de software, o real benefício delas só poderá ser alcançado por meio da integração. O DiSEN (*Distributed Software Engineering ENvironment*) é um ambiente de desenvolvimento de software que considera as características de sistemas distribuídos e pode prover suporte à cooperação entre espaços de trabalho distintos, além de suportar um conjunto de ferramentas que auxiliam na tarefa do desenvolvimento de software. Esta dissertação apresenta um modelo que apóia a interoperabilidade entre as ferramentas instanciadas no DiSEN e a integração dos dados (artefatos) com o ambiente. O modelo de interoperabilidade proposto visa: (i) permitir a manipulação de um artefato em diferentes ferramentas; (ii) ser extensível, possibilitando a inclusão de novas ferramentas ao longo do ciclo de vida do projeto; (iii) permitir que o ambiente de desenvolvimento de software tenha seu próprio metamodelo; e (iv) possibilitar representar e compartilhar o conhecimento do ambiente em um formato padrão de dados. Este formato padrão representará aqueles que o ADDS será capaz de interoperar e integrar. A validação do modelo proposto será realizada utilizando técnicas da Engenharia de Software Experimental.

Palavras-Chave: Transformações de Modelos, Metamodelo, UML, MDA, Interoperabilidade. Ambiente de Desenvolvimento Distribuído de Software. Semântica de Dados. Metadados. Persistência.

ABSTRACT

A Lot of tools known as CASE (Computer-Aided Engineering Software), have been used in the software development. However, these tools have not worked in an integrated way, but in an isolated one, without sharing information or services. Even though the individual use of these tools can bring benefits for the software development, the real benefit of them could only be reached by means of integration. DiSEN (Distributed Software Engineering ENvironment) is a software development environment that considers the characteristics of distributed systems and can provide support for the cooperation among several locals, besides supporting a set of tools that assist the software development task. This work presents a model that support the interoperability between the instantiate tools in the DiSEN and the data integration (artifacts) with the environment. The proposed interoperability model aims to: (i) allow the artifacts manipulation in different tools; (ii) be extensible, making it possible the inclusion of new tools along the life cycle of a project; (iii) permit that the software development environment has own metamodel; and (IV) make it possible to represent and to share the knowledge of the environment in a standard format of data. This standard format is the metamodel refered in the item (iii) it will represent those that the ADDS will be able to interoperate and to integrate. The validation of proposed model will be carried through using techniques of the Experimental Software Engineering.

Key Words: Transformations Models, Metamodel, UML, MDA, Interoperability. Distributed software development environment. Semantics Data. Metadata. Persistence.

LISTA DE ILUSTRAÇÕES

Figura 1 – Arquitetura do DiSEN 20

Figura 2 – Funcionalidades da REQUISITE 22

Figura 3 – Exemplo de um documento XML estruturado 29

Figura 4 – Pontes existentes entre ferramentas CASE 30

Figura 5 – Domínios de aplicação do XMI 30

Figura 6 – Fragmento de um arquivo XMI 31

Figura 7 - Visão geral da MDA 33

Figura 8 - Arquitetura do Modelo de Interoperabilidade 54

Figura 9 – Arquitetura do Modelo de Interoperabilidade inserido no DiSEN 54

Figura 10 – Fragmento de um mapeamento 56

Figura 11 - Método de Transformação 58

Figura 12 – Fragmento do método Transformation() 58

Figura 13 – Fragmento do método de refinamento 60

Figura 14 – Fragmentos do diagrama de classes e da implementação do metamodelo em Java 61

Figura 15 – Fragmento de um XMI exportado por uma ferramenta CASE 63

Figura 16 – Fragmento de um mapeamento 63

Figura 17 – Fragmento da classe Model do metamodelo do DiSEN implementada em Java 65

Figura 18 – Processo de interação entre os componentes do modelo de interoperabilidade 69

Figura 19 – Interface da ferramenta de simulação, tela de importação 70

Figura 20 – Arquivo XMI de saída gerado a partir do metamodelo 72

Figura 21 - Fragmento do XMI da ferramenta EA, mostrando um esteriótipo <<extend> 74

Figura 22 – Fragmento do XMI da ferramenta Poseidon, mostrando um esteriótipo <<extend> 74

Figura 23 – Arquivo de Saída gerado pela ferramenta de simulação 75

Figura 24 – Diagrama de caso de uso realizado para teste do Metamodelo 76

Figura 25 – Diagrama de caso de uso correto da ferramenta EA 76

Figura 26 – Diagrama de caso de uso importado de maneira errada pela ferramenta EA 77

Figura 27 – Nova interface para a transformação dos artefatos 78

Figura 28 – Representação de um Diagrama de Caso de Uso na Ferramenta Requisite 80

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

ADS	Ambiente de Desenvolvimento de Software
ADDS	Ambiente de Desenvolvimento Distribuído de Software
CASE	*Computer Aided Software Engineering*
DiSEN	*Distributed Software Engineering ENvironment*
MDSODI	Metodologia para Desenvolvimento de Software Distribuído
MOOPP	Metodologia Orientada a Objetos para Desenvolvimento de Software para Processamento Paralelo
FIPA	*Foundation for Intelligent Physical Agents*
OSCAR	*Open-Source Component and Artefact Repository*
GENESIS	*GEneralized eNvironment for procESs management in cooperatIve Software engineering*
ODE	*Ontology-based software Development Environment*
MDA	*Model-Driven Architecture*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
MOF	*Meta Object Facility*
XML	*Extensible Markup Language*
XSLT	*Extensible Stylesheet Language Transformations*
SAIL	*Storage And Inference Layer*
SEI `	*Software Engineering Institute*
NISO	*National Information Standards Organization*
W3C	*World Wide Web Consortium*
SGML	*Standard Generalized Markup Language*
HTML	*HyperText Markup Languag*e
UML	*Unified Modeling Language*
OMG	*Object Management Group*
API	*Application Programming Interface*
OWL	*Web Ontology Language*
SQL	*Structured Query Language*
SGBD	Sistema Gerenciador de Banco de Dados

SUMÁRIO

1 INTRODUÇÃO 16

2 REFERENCIAL TEÓRICO 19

2.1 INTRODUÇÃO À ARQUITETURA DISEN 19
2.1.1 REQUISITE. 22
2.2 INTEROPERABILIDADE 25
2.3 METADADOS 27
2.4 XML 28
2.5 XMI 30
2.5.1 ESTRUTURA E ELEMENTOS DO XMI 32
2.6 MDA: MODEL-DRIVEN ARCHITECTURE 33
2.7 TRANSFORMAÇÕES DE MODELOS 35
2.7.1 CLASSIFICAÇÃO DAS TRANSFORMAÇÕES 40
2.8 CONSIDERAÇÕES FINAIS 41

3 TRABALHOS RELACIONADOS 42

3.1 OSCAR (*OPEN-SOURCE COMPONENT AND ARTEFACT REPOSITORY*) 42
3.2 SESAME 43
3.3 ODYSSEY SHARE 44
3.4 ARQUITETURA PARA COMPARTILHAMENTO DE CONHECIMENTO EM BIBLIOTECAS DIGITAIS 44
3.5 XMLDOC: FERRAMENTA DE APOIO À DOCUMENTAÇÃO DE ODE 46
3.6 UMA INFRA-ESTRUTURA PARA INTEGRAÇÃO DE FERRAMENTAS CASE 47
3.7 ODYSSEY-MDA 50
3.8 CONSIDERAÇÕES FINAIS 51

4 IMART: MODELO DE INTEROPERABILIDADE PARA AMBIENTES DE DESENVOLVIMENTO DISTRIBUÍDO DE SOFTWARE 52

4.1 FUNCIONALIDADES 52
4.2 COMPONENTES DO MODELO DE INTEROPERABILIDADE 54
4.3 CRIAÇÃO DOS MAPEAMENTOS E METAMODELO 63
4.4 EXTENSIBILIDADE DO MODELO DE INTEROPERABILIDADE 67
4.5 AVALIAÇÕES DO MODELO DE INTEROPERABILIDADE 69
4.5.1 FERRAMENTA DE SIMULAÇÃO 70
4.5.2 PRIMEIRO CENÁRIO: ATRIBUIR DADOS DO METAMODELO 72
4.5.3 SEGUNDO CENÁRIO: OBTER DADOS DO METAMODELO 73
4.5.4 TERCEIRO CENÁRIO: ADIÇÃO DE COMPLEXIDADE NA SEMÂNTICA DOS ARTEFATOS 74
4.5.5 QUARTO CENÁRIO: ADIÇÃO DE FUNCIONALIDADES NA FERRAMENTA REQUISITE. 80

5 CONCLUSÃO 82

5.1 CONTRIBUIÇÕES 83

5.2 LIMITAÇÕES DO MODELO DE INTEROPERABILIDADE 85
5.3 TRABALHOS FUTUROS 85

REFERÊNCIAS 87

1 INTRODUÇÃO

Qualidade e produtividade são fatores cruciais no desenvolvimento de produtos de *software*, e neste contexto torna-se essencial construir ferramentas para auxiliar o engenheiro de *software* em suas atividades. A utilização de ferramentas exerce influência direta no tempo de desenvolvimento do *software*, no seu custo e na qualidade do produto (HARRISON, OSSHER, TAAR, 2000).

Diversas ferramentas, conhecidas como ferramentas CASE (*Computer-Aided Software Engineering*), têm sido utilizadas no desenvolvimento de *softwares*; porém estas ferramentas não têm trabalhado de forma integrada, mas sim, de forma isolada, sem compartilhar informações ou serviços. Mesmo que a utilização individual destas ferramentas possa trazer benefícios para o desenvolvimento de *software*, o real poder delas só poderá ser alcançado por meio da integração (PRESSMAN, 2006).

A interoperabilidade é hoje um conceito-chave em várias áreas de pesquisa - como redes e sistemas distribuídos e bancos de dados - e entre as próprias aplicações, quando estas se tornam sistemas legados. Nestes contextos apresentados, a interoperabilidade pode ser tratada sob várias óticas. Por exemplo, em bancos de dados a interoperabilidade pode ser tratada em nível de base de dados heterogênea.

Segundo Mandviwalla e Grillo (1995), a interoperabilidade é um conceito antigo e tão importante como introduções sobre sistemas de código aberto, desenvolvimento de *software* e heterogeneidade. Para os autores, os dois objetivos principais da interoperabilidade são: fornecer acesso à informação que é armazenada em sistemas diferentes e facilitar a cooperação entre os programas (ferramentas) que manipulam esta informação.

A motivação para desenvolver esta dissertação surgiu a partir da necessidade de oferecer um modelo de interoperabilidade que permita a reutilização dos artefatos entre as ferramentas que façam parte de um ambiente distribuído de desenvolvimento de *software* e possibilite também que os artefatos sejam persistidos em um formato-padrão no ambiente. Estas características são importantes para apoiar o trabalho cooperativo de equipes distintas, utilizando diferentes ferramentas e fazendo-o em locais geograficamente separados.

Assim, esta dissertação tem por objetivo propor um modelo de interoperabilidade que permita a reutilização de artefatos gerados por diferentes ferramentas de um ambiente de desenvolvimento de software distribuído. Para tanto, foram tomados como base algumas propostas e modelos de mecanismos de interoperabilidade encontrados na literatura, que serão apresentados no capítulo 3. Em termos gerais, o modelo acrescentará aos ambientes de desenvolvimento distribuído de software (ADDS) as seguintes funcionalidades: (i) permitir que um artefato seja reutilizado por ferramentas diferentes; (ii) disponibilizar um modelo que seja extensível o suficiente para adicionar novas ferramentas ao longo do ciclo de vida do projeto; (iii) permitir que o ambiente de desenvolvimento de *software* tenha seu próprio padrão de armazenamento dos artefatos (iv) possibilitar que o conhecimento do ambiente seja representado e compartilhado. O conhecimento representará o(s) formato(s) do artefato com que o ADDS será capaz de interoperar.

Como estudo de caso será utilizado o DiSEN (*Distributed Software Engineering ENvironment*), que demanda um modelo de interoperabilidade que permita a reutilização dos artefatos gerados pelas ferramentas utilizadas neste ambiente.

Para o desenvolvimento desta dissertação foi adotado como metodologia de trabalho o estudo do referencial teórico que priorizou o aprendizado dos diferentes conceitos necessários para a sustentação e construção do modelo de interoperabilidade. A pesquisa por trabalhos correlatos serviu como inspiração para a definição dos componentes do modelo de

interoperabilidade. A identificação e descrição das funcionalidades dos componentes deste modelo foram realizadas com base nas funcionalidades requeridas por um ADDS; definição das tecnologias para prototipação dos componentes; e finalmente, a definição de quatro estudos de caso que permitiram avaliar as funcionalidades identificadas.

Esta dissertação está organizada da seguinte maneira: o Capítulo 2 apresentará os conceitos sobre a arquitetura DiSEN (contextualizando o modelo de interoperabilidade que será proposto) e os conceitos que apoiaram o desenvolvimento do modelo de interoperabilidade; o Capítulo 3 apresentará os trabalhos relacionados e a relação destes com o modelo de interoperabilidade; o Capítulo 4 descreverá a arquitetura do modelo de interoperabilidade e as avaliações realizadas; e por fim, o Capítulo 5 apresentará as conclusões e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo apresentará o DiSEN e os conceitos envolvidos na definição de sua arquitetura. O DiSEN é um ambiente de desenvolvimento de software distribuído e servirá como estudo de caso para validação do modelo de interoperabilidade, objeto desta dissertação. Esta seção também apresentará os conceitos sobre interoperabilidade, metadados, XML/XMI, MDA e transformações de modelos.

2.1 Introdução à arquitetura DiSEN

Para Tanenbaum e Steen (2002), sistemas distribuídos possuem propriedades, entre as quais se destaca a capacidade de adição de usuários e/ou recursos de forma que esta adição possa ser gerenciada sem maiores transtornos e o usuário possa utilizá-los sem perceber as diferenças entre a representação dos dados e o seu armazenamento.

Moura e Rocha (1992) definem ambiente de desenvolvimento de software (ADS) como um sistema computacional que provê suporte para o desenvolvimento, reparo e melhorias em softwares e para o gerenciamento e controle dessas atividades.

Um ambiente de desenvolvimento de software é definido como um ambiente para modelagem, execução, simulação e evolução de processos de desenvolvimento de software. (RABELLO et al., 2001).

Para que ambos os conceitos acima possam ser implementados torna-se necessária a utilização de ferramentas para a execução de atividades em um ADS. Desta forma, a interoperabilidade entre as ferramentas alocadas torna-se ponto vital para um ADS.

Encontra-se em desenvolvimento pelo Grupo de Estudos de Engenharia de Software para Sistemas Distribuídos, da Universidade Estadual de Maringá, o DiSEN, um ambiente de desenvolvimento de software distribuído, incorporando a tecnologia de agentes

segundo o padrão da FIPA (*Foundation for Intelligent Physical Agents*). Segundo Pascutti (2002), a arquitetura do DiSEN foi projetada para utilizar, dentre outras, a MDSODI (HUZITA, 1999), uma metodologia para desenvolvimento de software que leva em consideração algumas características identificadas em sistemas distribuídos, tais como concorrência, paralelismo, comunicação, sincronização e distribuição. A MDSODI também mantém as principais características do *Unified Process* (JACOBSON, BOOCH, RUMBAUGH, 1999), dirigida a use-case, centrada na arquitetura e de desenvolvimento iterativo e incremental.

Segundo Pascutti (2002), a arquitetura do DiSEN, conforme ilustra a figura 1, está organizada em três camadas, descritas a seguir:

- dinâmica - que é responsável pelo gerenciamento de configuração do sistema;
- aplicação - que conterá a MDSODI, o repositório para armazenamento dos dados necessários ao ambiente e os gerenciadores de *workspace*, objetos e agentes;
- infra-estrutura - que proverá suporte às tarefas de nomeação, persistência e concorrência e irá incorporar também o canal de comunicação.

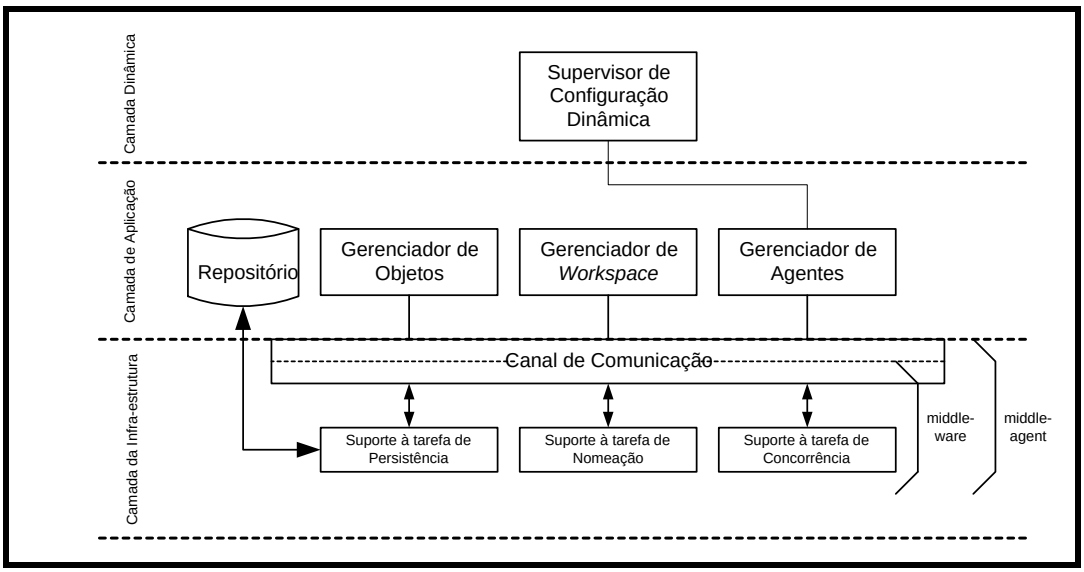

Figura 1 – Arquitetura do DiSEN.
Fonte: Pascutti (2002)

A seguir uma breve descrição dos elementos da arquitetura proposta por Pascutti (2002).

- Supervisor de configuração dinâmica: é responsável pelo controle e gerenciamento da configuração do ambiente, bem como dos serviços que podem ser acrescentados ao ambiente em tempo de execução.
- Gerenciador de objetos: é responsável pelo controle e gerenciamento do ciclo de vida dos artefatos. Um artefato pode ser um diagrama, modelo, manual, código-fonte ou código-objeto. O gerenciador de objetos é subdividido em vários outros gerenciadores, como os de acesso, de atividades, de recursos, de artefatos, de projetos, de processos e de versão e configuração. A especificação destes gerenciadores é encontrada em Pascutti (2002).
- Gerenciador de *workspace*: é responsável pelo controle e gerenciamento da edição cooperativa de documentos e itens de software. Para isso, provê suporte para um ou mais *workspaces* no uso dos dados mantidos no

repositório. A especificação deste gerenciador é encontrada mais detalhadamente em Pozza (2004).

- Gerenciador de agentes: é responsável pela criação, registro, localização, migração e destruição de agentes.
- Repositório: é responsável pelo armazenamento dos artefatos e dados das aplicações gerados no ADDS.
- Canal de comunicação: é responsável pela comunicação entre as camadas de infra-estrutura (*middleware* e *middle-agent*) e a camada de aplicação. Toda comunicação entre os elementos da arquitetura se dará por intermédio do canal de comunicação.

O objetivo do DiSEN é fornecer o suporte necessário para o desenvolvimento de software distribuído; a equipe poderá estar distribuída em locais geográficos distintos e trabalhar de forma cooperativa com uma metodologia para desenvolvimento de software distribuído.

Por se tratar de um ADDS, pode ocorrer que desenvolvedores utilizem diferentes ferramentas durante o desenvolvimento de um projeto de software. Eventualmente, pode-se ter a necessidade de adicionar ferramentas que não sejam nativas do ambiente (ferramentas externas) para complementar algumas atividades do processo de software. Assim, é importante que o ambiente ofereça apoio à interoperabilidade entre estas ferramentas, permitindo que os artefatos construídos por elas possam ser reutilizados e mantidos em um formato comum ao longo do desenvolvimento do software.

2.1.1 REQUISITE.

Segundo Batista (2003), a REQUISITE é uma ferramenta nativa do ADDS DiSEN que tem por objetivo auxiliar na modelagem de requisitos, prover um meio de

comunicação entre os *stakeholders*, prover apoio à rastreabilidade e à documentação de requisitos no DiSEN, oferecendo suporte à MDSODI.

Um dos pontos positivos da REQUISITE é a utilização de cenários para auxiliar na elicitação dos requisitos, negociação e validação.

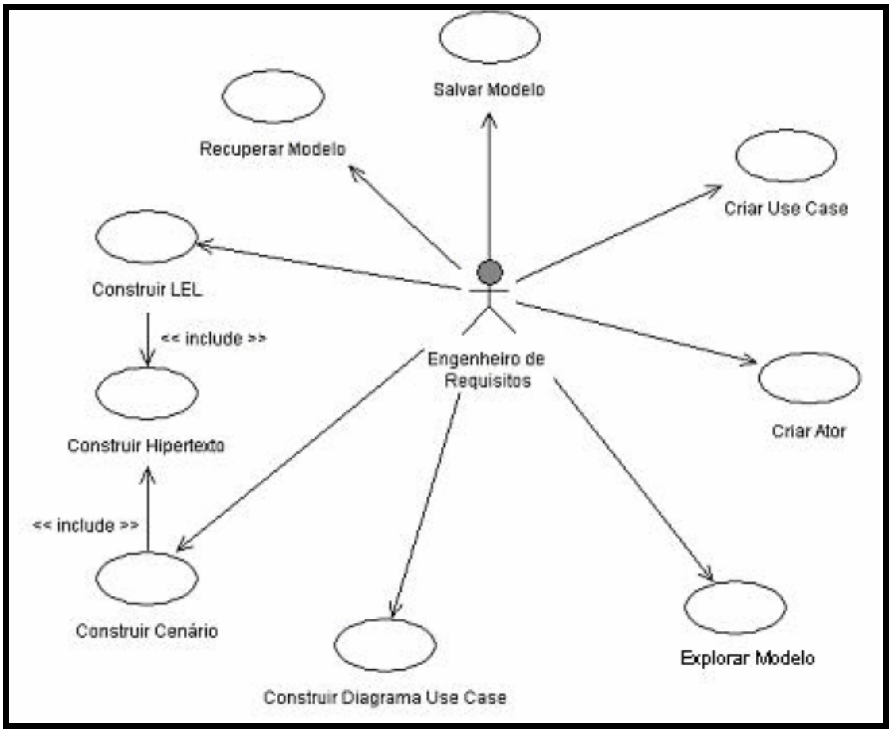

Figura 2 – Funcionalidades da REQUISITE.
Fonte: Batista (2003)

São a seguir relacionadas e descritas as principais funcionalidades da ferramenta REQUISITE (BATISTA, 2003), conforme apresentadas no diagrama de caso de uso da Figura 2.

- Recuperar modelo: permite ao engenheiro de requisitos recuperar um modelo localmente (*workspace*) ou no ambiente distribuído DiSEN.
- Salvar modelo: permite ao engenheiro de requisitos salvar um modelo localmente (*workspace*) ou no ambiente distribuído DiSEN.

- Explorar modelo: permite ao engenheiro de requisitos rastrear cenários, entradas do LEL, atores, casos de uso e diagramas de caso de uso no modelo.
- Construir cenário: permite ao engenheiro de requisitos criar, modificar, consultar ou remover um cenário do modelo. A primeira operação diz respeito à criação e à introdução de um novo cenário. De modo geral, é o resultado da fase de elicitação, e acontece com mais freqüência durante as etapas iniciais do processo.
- Construir LEL: permite ao engenheiro de requisito criar, modificar, remover e consultar uma entrada no LEL do modelo. A primeira operação diz respeito à criação de uma nova entrada no LEL, isto é, a identificação de um símbolo usado na linguagem do UdI. As operações de modificação e remoção de uma entrada no LEL dizem respeito ao processo de construção do LEL.
- Criar ator: permite ao engenheiro de requisitos adicionar, modificar, consultar e remover um ator ao modelo. Os atores previstos pela notação da MDSODI (GRAVENNA, 2000) são: exclusivo, paralelo, distribuído e paralelo e distribuído.
- Criar caso de uso – permite ao engenheiro de requisitos adicionar, modificar, consultar e remover um caso de uso ao modelo. Os caso de usos previstos pela notação da MDSODI (GRAVENNA, 2000) são: seqüencial, distribuído, paralelo e paralelo e distribuído.

As funcionalidades de criação do diagrama de caso de uso, criação de atores e casos de uso, bem como a recuperação e a persistência do modelo (salvar modelo), foram reimplementadas neste trabalho como parte da avaliação do modelo de interoperabilidade. Até

a construção do modelo de interoperabilidade apresentado nessa dissertação, os artefatos criados pela ferramenta REQUISITE não podiam ser reutilizados, nem poderiam ser compartilhados por outras ferramentas CASE.

2.2 Interoperabilidade

Segundo Mandviwalla e Grillo (1995), a interoperabilidade é um conceito antigo e tão importante como introduções sobre sistemas de código aberto, desenvolvimento de software e heterogeneidade. A interoperabilidade é um assunto-chave da computação atual e seus dois objetivos principais são: fornecer acesso à informação que é armazenada em sistemas diferentes e facilitar a cooperação entre os programas (ferramentas) que manipulam esta informação.

A interoperabilidade pode ser vista de diferentes maneiras, dependendo do contexto em que ela é aplicada. Wileden e Kaplan (1997) tratam a interoperabilidade como a habilidade de escrever múltiplos componentes de software em diferentes linguagens de programação, para interagir e comunicar-se com outros componentes. Produtos de software com base em múltiplas linguagens são desenvolvidos por várias razões - por exemplo, para reutilizar componentes de software ou dados existentes, como interface (ou para criar interfaces) para sistemas legados, ou até mesmo para compartilhar dados sobre um formato comum.

Segundo Bao e Horowitz (1996), o desenvolvimento de software em grande escala é um processo difícil e complexo que envolve o uso e a coordenação de muitas ferramentas diferentes. Produzir software de alta qualidade, no tempo e dentro dos custos previstos, somente é possível se as diferentes ferramentas trabalharem em conjunto e de forma eficaz para fornecer suporte às tarefas do desenvolvimento do software. Bao e Horowitz (1996) ainda sugerem que os usuários do ambiente de desenvolvimento de software,

normalmente, se questionem sobre três pontos importantes no momento de aplicar uma solução de interoperabilidade ao seu ADDS.

1) esta solução é aplicável ao nosso sistema?

2) podemos facilmente mudá-la quando o sistema evoluir e as exigências da integração mudarem?

3) quanto de trabalho nós teremos que refazer caso o sistema evolua?

Pode-se perceber a importância da interoperabilidade entre as ferramentas de um ADDS, a necessidade de um modelo comum de dados que permita a persistência dos artefatos e possibilite às ferramentas manipular diferentes tipos de dados e artefatos.

Coulouris, Dollimore e Kindberg (2001) ressaltam a importância da interoperabilidade em sistemas distribuídos quando constatam que o crescimento acelerado de aplicações para redes e sistemas distribuídos é resultado do uso de computadores no suporte para grupos de usuários que trabalham cooperativamente, e que estes dependem do compartilhamento de dados entre programas executados em diferentes estações de trabalho.

Para o SEI (*Software Engineering Institute*), os termos interoperabilidade e integração normalmente podem ser utilizados com o mesmo significado, uma vez que se referem à troca de dados entre dois sistemas ou ferramentas. Por fim, é importante reforçar que a interoperabilidade pode significar a habilidade de manipular os dados entre dois sistemas de acordo com a sua semântica, e isto vai além de uma simples troca de dados entre elas (SOFTWARE ENGINEERING INSTITUTE, 2004).

Nesta dissertação o termo interoperabilidade será utilizado conforme a definição dada pelo SEI.

Segundo os conceitos de interoperabilidade aqui apresentados e reforçados pela definição de Wegner (1996), interoperabilidade é a capacidade que podem ter dois ou mais componentes de software de cooperar entre si, apesar das possíveis diferenças que eles

possam apresentar. Tais diferenças referem-se às linguagens, interfaces e plataformas de execução. Assim, para permitir a cooperação e a manipulação dos dados, conforme as definições citadas nesta seção, torna-se essencial que os componentes consigam ler e entender dados representados em diferentes formatos e padrões. Isto mostra a importância da utilização de metadados para esta representação, pois a quantidade de informação que se quer compartilhar entre os componentes será limitada pela informação que pode ser entendida por ambos os componentes.

Desta forma, a interoperabilidade é um conceito fundamental em um ADDS, pelo qual se busca, a todo instante, a cooperação entre os desenvolvedores distribuídos nos diferentes locais de trabalho. Para habilitar essa cooperação é importante que as ferramentas utilizadas por um ADDS manipulem, compartilhem e reutilizem os artefatos gerados durante o processo de desenvolvimento do software.

Esta seção também procurou evidenciar os cuidados que deverão ser considerados para a construção do modelo de interoperabilidade, de forma que seja possível responder às três questões levantadas por Bao e Horowitz (1996).

2.3 Metadados

Segundo Rosetto (2003), os metadados podem auxiliar na interoperabilidade entre sistemas de informação, além de possibilitar que uma informação possa ser representada sintática e semanticamente, o que é necessário para permitir a extensibilidade do modelo de interoperabilidade por meio da descrição dos diferentes tipos de artefatos pertencentes a um ADDS.

O guia publicado pela *National Information Standards Organization* – NISO (2004), define metadados como uma informação estruturada que descreve, explica, localiza ou, de alguma outra forma, torna mais fácil a recuperação, utilização ou controle de uma

informação. Com a utilização dos metadados podem-se descrever recursos em todos os níveis de agregação.

Segundo NISO (2004), a descrição de um recurso utilizando metadados pode permitir que este seja compreendido por seres humanos e máquinas, de maneira que a interoperabilidade seja alcançada.

Segundo Vaz (2005), os metadados surgiram devido ao fato de as organizações necessitarem conhecer melhor os dados que elas mantêm. Metadados são dados que descrevem dados, provendo uma descrição concisa a respeito dos dados, que podem ser documentos, diagramas, tabelas, entre outros. Uma arquitetura de metadados deve ser extensível o suficiente para permitir a adição de novos metadados à medida que surjam novas necessidades.

É essencial que se compreenda qual o impacto que os metadados podem acrescentar ao desenvolvimento de sistemas eficazes, interoperáveis, escaláveis e de gerência da informação. Os metadados indicam um padrão, bem como a documentação e outros dados necessários para a identificação, a representação, a interoperabilidade, o desempenho e o uso dos dados contidos em um sistema de informação (GILLILAND-SWETLAND, 2005).

A utilização dos metadados é parte importante do modelo de interoperabilidade aqui apresentado. Eles serão responsáveis por permitir o mapeamento da semântica contida nos arquivos XML/XMI para um metamodelo descrito e implementado por meio de classes JAVA. Este metamodelo permitirá a extensibilidade de representação dos artefatos manipulados por um ADDS.

2.4 XML

Uma linguagem de marcação é um conjunto de códigos (símbolos) que podem ser inseridos no texto de um documento para demarcar e rotular as suas partes, possibilitando,

desta maneira, representar tanto o seu conteúdo (texto) quanto as instruções sobre sua estrutura (semântica).

Estimulado pela insatisfação com os formatos existentes (padronizados ou não), um grupo de empresas e organizações, que se autodenominou *World Wide Web Consortium* (W3C), começou a trabalhar no final do ano de 1997 e publicou a versão 1.0 da linguagem XML (*eXtensible Markup Language*), uma linguagem de marcação que combinasse a flexibilidade da SGML (*Standard Generalized Markup Language*) com a simplicidade da HTML (*HyperText Markup Language*). Atualmente encontra-se na versão 1.1, publicada em fevereiro de 2004.

A XML descreve uma classe de objetos de dados denominada documentos XML e descreve parcialmente o comportamento dos programas de computador que os processam. (http://www.w3.org/TR/2004/REC-xml11-20040204/, 2004).

Segundo a norma ISO8879 (1986), a XML também é um perfil, é um subconjunto da SGML, que é a linguagem de marcação mais generalizada, ou seja, a que serviu de base para a construção de outras, como por exemplo, a XML.

Na especificação da XML 1.1 (http://www.w3.org/TR/2004/REC-xml11-20040204/, 2004), a W3C apresenta os seus principais objetivos. O objetivo principal era criar uma linguagem que pudesse ser lida por software e integrada a outras linguagens. Para isso ela deveria:

- apoiar uma grande variedade de aplicações;
- ser compatível com SGML;
- permitir uma fácil criação de programas que consigam processar documentos XML;
- ser legível tanto por humanos quanto por máquinas;
- permitir a possibilidade de criação de *tags* sem limitação.

A XML, portanto, pode ser utilizada para permitir a interoperabilidade entre bases de dados heterogêneas e entre aplicações, justamente por ser uma linguagem com grande flexibilidade e poder semântico, além de possuir um conjunto de padrões para troca de informações de forma estruturada, independente de plataforma ou tecnologia.

Um exemplo de documento XML descrevendo uma receita de bolo pode ser visto na Figura 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<Receita nome="bolo" tempo_de_preparo="10 minutos" tempo_de_cozimento="2 horas">
  <título>Bolo simples</título>
  <ingrediente quantidade="3" unidade="xícaras">Farinha</ingrediente>
  <ingrediente quantidade="7" unidade="gramas">Fermento</ingrediente>
  <ingrediente quantidade="1.5" unidade="xícaras" estado="morna">Água</ingrediente>
  <ingrediente quantidade="1" unidade="colheres de chá">Sal</ingrediente>
  <Instruções>
   <passo>Misture todos os ingredientes, e dissolva bem.</passo>
   <passo>Cubra com um pano e deixe por uma hora em um local morno.</passo>
   <passo>Misture novamente, coloque numa bandeja e asse num forno.</passo>
  </Instruções>
</Receita>
```

Figura 3 – Exemplo de um documento XML estruturado.

2.5 XMI

A linguagem XML tem sido utilizada por várias aplicações, entre as quais em ferramentas que dão suporte ao desenvolvimento de software. Neste caso, o formato XMI (*XML Metadata Interchange*) está sendo utilizado para expressar o conteúdo de modelos orientados a objetos (OO) e outros documentos gerados por essa linguagem, em um formato que permita a troca de informações.

Segundo BRODSKY (1999), não existe uma ferramenta que suporte sozinha todo o processo de desenvolvimento do software, o que torna necessário utilizar-se de ferramentas diferentes para cada uma das fases do desenvolvimento do software. Fazer com que estas ferramentas troquem informações é um grande desafio. A Figura 4 mostra a interoperabilidade entre estas ferramentas. Para este trabalho, considerar-se-á que as

ferramentas estão inseridas em um ambiente de desenvolvimento distribuído de software, até mesmo para permitir que os artefatos gerados por elas possam ser reutilizados ao longo do ciclo de desenvolvimento do software.

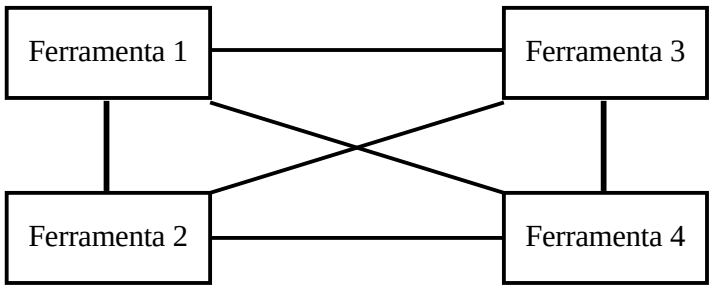

Figura 4 – Pontes existentes entre ferramentas CASE.
Fonte: Brodsky (1999**)**

O padrão XMI da OMG permite representar modelos UML em XML. Dessa forma é possível obter uma representação XML de um artefato criado por uma ferramenta que ofereça esse suporte (importação/exportação de XMI). O objetivo do XMI é permitir a interoperabilidade entre ferramentas CASE, repositórios e ferramentas de desenvolvimento por meio da troca de metadados em um arquivo (*stream*) de dados baseados no padrão XML (BRODSKY, 1999).

A representação por intermédio de metadados entre aplicações está ilustrada na Figura 5, que mostra a XMI sendo utilizada entre diferentes tipos de aplicações e com finalidades distintas.

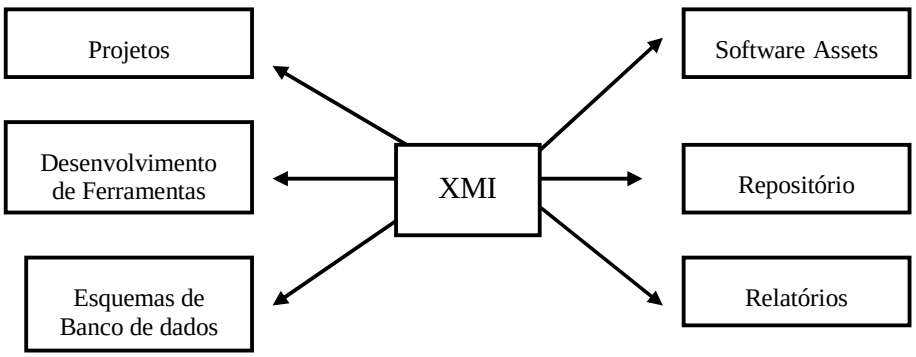

Figura 5 – Domínios de aplicação do XMI
Fonte: Brodsky (1999)

Apesar de todo o esforço de padronização pela OMG, ainda persistem os problemas com as ferramentas CASE atuais, pois a troca de metadados entre as ferramentas ainda é considerada uma tarefa difícil. Essa dificuldade advém do fato de que cada ferramenta armazena e codifica seus metadados de maneira distinta, comprometendo a interoperabilidade entre estas ferramentas e a reutilização dos artefatos gerados por elas. Estas diferenças ficarão evidentes no decorrer deste trabalho.

2.5.1 Estrutura e elementos do XMI

A Figura 6 mostra um fragmento de um arquivo XMI, que representa um diagrama de caso de uso gerado por uma ferramenta CASE.

```
<UML:Model name="Use Case Model" xmi.id="MX_EAID_E77A5D80_0EE2_4dfc_99C4_C8E98BA4992F" isSp
    <UML:Namespace.ownedElement>
        <UML:Actor name="Actor2" xmi.id="EAID_35E3E7B1_491B_41d2_8A7D_FD15F0D5A9BD" isSpe
        <UML:Actor name="Actor1" xmi.id="EAID_821FB01E_4C84_4f3c_8892_87D1A4956477" isSpe
        <UML:UseCase name="Use Case5" xmi.id="EAID_0B541C34_A031_48d9_B9B4_7A19141B4036"
            <UML:UseCase.extend>
                <UML:Extend xmi.idref="EAID_EB47E770_AEB1_48a6_908D_E4275D1E0001"/>
            </UML:UseCase.extend>
        </UML:UseCase>
        <UML:UseCase name="Use Case1" xmi.id="EAID_101E7E17_5FDC_4c76_826D_B4F37199A547"
            <UML:UseCase.extend>
                <UML:Extend xmi.idref="EAID_EB47E770_AEB1_48a6_908D_E4275D1E0003"/>
            </UML:UseCase.extend>
            <UML:UseCase.include>
                <UML:Include xmi.idref="EAID_669A181D_4BE0_4bf0_A711_805B08D50005"/>
            </UML:UseCase.include>
        </UML:UseCase>
```

Figura 6 – Fragmento de um arquivo XMI.

Comparando-se as figuras 3 e 6, pode-se observar que o XMI possui a mesma estrutura hierárquica do XML, ou seja, é organizado por meio de *tags*, atributos e elementos.

Dentre os elementos do XMI, destacam-se (OMG XMI, 2005):

- XMI: elemento raiz de um documento XMI;
- *XMI.header*: contém elementos que identificam o modelo, metamodelo e metametamodelo que formam o cabeçalho de um documento XMI;

- *XMI.model*: identifica o modelo ao qual pertencem os dados contidos no XMI;
- *XMI.content*: contém os metadados do modelo que foi construído na ferramenta CASE. Constitui o conteúdo do documento XMI.

Os elementos apresentados representam somente as partes principais de um documento XMI. Dentro de cada documento existem outras *tags* que formam outros elementos, podendo estas, inclusive, ser customizadas.

Neste trabalho é essencial o entendimento sobre a semântica do XMI produzido por diferentes ferramentas CASE apoiadas pelo ambiente de desenvolvimento de software distribuído, pois o modelo de interoperabilidade deverá ser capaz de entendê-lo e manipulá-lo para permitir o trabalho cooperativo e a reutilização dos artefatos produzidos durante o desenvolvimento do software.

2.6 MDA: Model-Driven Architecture

A MDA teve início com a idéia de separar a especificação de um sistema, dos detalhes de como aquele sistema utiliza as capacidades da sua plataforma. As três metas primárias da MDA são: portabilidade, interoperabilidade e reutilização, por meio da separação arquitetônica dos *concerns*. (OMG MDA, 2003).

Os objetivos da MDA provêem uma abordagem e habilitam ferramentas para a especificação de um sistema independentemente de plataforma, especificação de plataformas, escolha de uma plataforma específica e transformação da especificação de um sistema para uma plataforma específica. Pode-se ter uma visão geral da MDA na Figura 7. (OMG MDA, 2003).

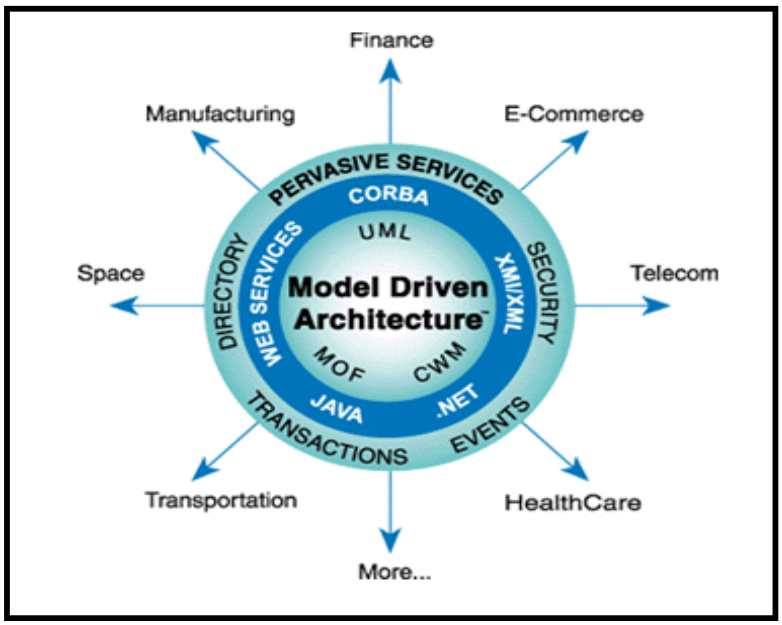

Figura 7 - Visão geral da MDA
Fonte: OMG MDA (2003).

Outros dois conceitos importantes para a MDA são: o modelo PIM (*platform independent model*) e PSM (*platform specific model*). Os dois modelos permitem a descrição de um modelo independente de plataforma que possibilita o mapeamento (transformação) para o modelo específico de plataforma, e a partir do PSM a transformação para uma linguagem de programação-alvo (OMG MDA, 2003).

A MDA provê quatro formas de transformação e mapeamento, e estas podem ser implementadas de modo automático, manual, por meio da utilização do *profile* da UML ou de *patterns* e marcações (OMG MDA, 2003).

O estudo das formas de transformações e mapeamentos da MDA pode acrescentar consistência e flexibilidade ao modelo de interoperabilidade aqui apresentado, já que suas principais metas podem ser aplicadas a qualquer ADDS. Para esta dissertação, elas poderiam auxiliar no trabalho cooperativo entre diferentes desenvolvedores, apoiando a interoperabilidade de diferentes ferramentas e permitindo a reutilização dos artefatos persistidos no ADDS.

2.7 Transformações de Modelos

Kleppe, Warmer e Wim (2003) definem a transformação de modelo como a geração automática a partir de um modelo-fonte (origem) em um modelo-alvo (destino), de acordo com uma definição da transformação constituída por um conjunto de regras que efetuam esta transformação.

Segundo Peltier, Ziserman e Bézivin (2000), a transformação de modelo consiste no processo de converter um modelo em outro modelo do mesmo sistema. Os desafios-chaves do paradigma dirigido a modelo consistem em definir, controlar e manter relacionamentos entre as diferentes visões de um modelo, incluindo o código do sistema.

Czarnecki e Helsen (2003) classificam as transformações de duas formas: (i) transformação modelo para modelo e (ii) modelo para código. No escopo deste trabalho, muito embora o modelo de interoperabilidade também esteja preocupado em oferecer mecanismos de transformação de modelo para o código, dar-se-á ênfase às transformações de modelo para modelo.

As transformações modelo para modelo se resumem à transformação de um modelo-origem, "*source*", para um modelo-destino, "*target*". Estes modelos podem ser construídos por metamodelos diferentes. Neste caso, é necessário existir um mapeamento, ou um metamodelo comum, que permita a realização das transformações.

Mens, Czarnecki e Gorp (2005) mostram a importância de refletir sobre algumas questões antes de decidir qual abordagem de transformação de modelos é a mais apropriada para resolver o problema em questão. Durante a reflexão algumas questões são analisadas com o intuito de investigar e sugerir critérios para auxiliar na escolha da abordagem de transformação.

As questões analisadas por Mens, Czarnecki e Gorp (2005) são:

a) O que necessita ser transformado em que?

Esta pergunta permite refletir sobre quais artefatos poderão ser manipulados e transformados. Isto significa identificar o tipo de transformação que será realizada, se é uma transformação modelo para modelo ou modelo para código. O tipo de transformação selecionado exigirá um conjunto de ferramentas, geradores e *parsers* para permitir as transformações, bem como uma linguagem para expressar a semântica e a sintaxe desse artefato. Daí a importância de se conhecer o artefato que se deseja transformar e possuir um metamodelo que consiga representar os dados deste artefato.

b) Quais são as características importantes de uma transformação do modelo?

Algumas características devem ser levadas em consideração no momento de adotar ou construir uma solução para realizar as transformações de modelos.

Uma característica importante é o nível de automatização utilizado para a realização das transformações. É provável que aconteçam casos onde a transformação manual seja mais indicada - por exemplo, quando for necessário resolver uma ambigüidade ou uma inconsistência nos requisitos que são expressos em linguagem natural.

A complexidade da transformação também deve ser avaliada com muito critério. Algumas transformações podem ser consideradas mais simples, como, por exemplo, *refactoring* entre modelos, enquanto outras, como os *parsers*, compiladores e geradores de código, podem ser consideradas mais complexas.

Outra característica importante para a transformação de modelos é a preservação. Embora haja uma grande quantidade de diferentes tipos de transformações que são úteis durante o desenvolvimento dirigido a modelos, é importante, em alguns casos, que aspectos do modelo de origem sejam preservados no modelo-destino. Por exemplo, em uma

transformação que exija um refinamento dos modelos é necessário que a sua semântica (o que ele representa) não seja distorcida de modo a se tornar inválida após uma transformação.

c) O que são critérios de sucesso (requisitos funcionais) de uma linguagem ou ferramenta de transformação?

Tão importantes quanto as características para os modelos e para a realização das transformações são os critérios de sucesso de uma linguagem ou ferramenta de transformação; afinal, é por meio deles que serão medidos os requisitos e necessidades sobre qual seja o melhor tipo de transformação. Algumas funcionalidades importantes para uma linguagem ou ferramenta de transformação são:

- criar, ler, apagar e atualizar as transformações: estas funcionalidades são importantes, pois podem determinar o quanto a ferramenta é extensível para atender a novos requisitos;
- sugerir quando aplicar transformações: para determinados cenários as ferramentas de transformação podem permitir a configuração ou sugerir ao usuário qual o melhor tipo de transformação para aquele contexto;
- customizar as transformações: em algumas situações pode ser necessária a customização das transformações com a finalidade de alcançar outro artefato-destino, ou até mesmo para corrigir algum provável erro na transformação;
- garantir que os modelos não sejam distorcidos após a transformação: é importante que as ferramentas não distorçam os artefatos transformados, pois isso implica em garantir não só a sintaxe dos artefatos, mas também as suas propriedades semânticas;

- tratar os modelos incompletos ou inconsistentes: as ferramentas de transformação devem permitir o tratamento das ambigüidades, dos novos requisitos e das inconsistências que podem atingir os artefatos durante o ciclo de vida do desenvolvimento do software;
- propiciar composição, decomposição e agrupamento das transformações: esta habilidade é útil para aumentar a legibilidade, a modularidade e a manutenibilidade de uma linguagem de transformação;
- testar, validar e verificar as transformações realizadas: é importante que as ferramentas sejam capazes de testar, validar e verificar para assegurar-se de que as transformações tenham sido realizadas de maneira correta;
- permitir transformações bidirecionais: as ferramentas devem efetuar transformações bidirecionais a fim de agilizar o processo de desenvolvimento dirigido a modelos, pois é importante que um modelo de origem possa ser transformado em um modelo de destino, e vice-versa;
- oferecer suporte para a propagação e habilidade de rastrear as alterações: é importante que as alterações no modelo de origem possam ser propagadas e rastreadas nos modelos gerados a partir dela.

Essas são características que devem fazer parte de qualquer solução de transformação de modelos. Uma maior quantidade de características implementadas pode significar um ganho no processo de transformação e manutenção dos modelos durante o ciclo de desenvolvimento do software.

d) Quais são os principais requisitos de qualidade (requisitos não funcionais) de uma linguagem ou ferramenta de transformação?

Um requisito importante é a usabilidade e utilidade da linguagem e/ou da ferramenta de transformação. Ambas devem ser úteis, o que significa serem utilizadas na prática para solucionar um problema. Em vez de utilizar-se de uma linguagem desenvolvida para transformação, pode-se escolher uma abordagem mais direta com relação à manipulação, utilizando-se de uma API para auxiliar nas transformações. A vantagem é que os desenvolvedores podem utilizar-se de uma linguagem da qual já detenham o conhecimento, o que não requer tempo extra de treinamento. A desvantagem é que a escolha errada de uma API pode restringir os tipos de transformação que podem ser executados.

A utilização do padrão XMI é importante, pois atualmente ele tem sido utilizado em larga escala pela indústria de software; portanto ele passa a ser imprescindível, uma vez que deve oferecer suporte à importação/exportação de artefatos nesta estrutura sintático-semântica.

Outro requisito importante é que as ferramentas devem permitir as transformações complexas. Assim podem atender a um maior escopo e ser mais úteis.

e) Que mecanismos podem ser usados para a transformação do modelo?

Na última questão levantada, Mens, Czarnecki e Gorp (2005) mostram preocupação com os tipos de mecanismo que podem ser utilizados para a realização das transformações. Os mecanismos são considerados como o conjunto de técnicas, linguagens e métodos. É importante para a aplicação e especificação de um mecanismo de transformação a utilização de algumas idéias dos principais paradigmas de programação, como, por exemplo, utilizar-se de uma solução procedural, orientada a objetos, funcional ou lógica, até mesmo híbrida, combinando alguns destes paradigmas.

A distinção principal entre mecanismos de transformação vem da escolha de uma abordagem declarativa ou operacional. A abordagem declarativa focaliza as necessidades da transformação, ou seja, "em que" um artefato será transformado. Já as abordagens operacionais focalizam “como" será realizada a transformação do artefato, isto é, elas focalizam as técnicas de transformação que serão utilizadas.

2.7.1 Classificação das transformações

Uma transformação é horizontal quando os modelos de origem e destino se encontram no mesmo nível de abstração. Um exemplo típico é o *refactoring*.

Transformação vertical é aquela na qual os modelos de origem e destino estão em níveis diferentes de abstração. Um exemplo típico desse tipo de transformação ocorre quando surge a necessidade de refinar os modelos gradualmente, adicionando detalhes a cada etapa da transformação.

Mens, Czarnecki e Gorp (2005) assim classificam as transformações:

a) Síntese: a síntese de alto nível é mais abstrata. Normalmente é uma especificação, como, por exemplo, um modelo de análise ou de projeto. A síntese de baixo nível é mais concreta, podendo-se citar como exemplo modelos que sejam traduzidos em código.

b) Engenharia reversa: é o inverso da síntese. Ela extrai uma especificação de alto nível de uma especificação de baixo nível.

c) Migração: é a transformação de um programa escrito em uma língua para outra, mantendo-se o mesmo nível de abstração.

d) Otimização: é a utilização de transformações para melhorar determinadas qualidades operacionais - por exemplo, o desempenho.

e) *Refactoring*: ocorre quando se deseja efetuar a mudança da estrutura interna do software para melhorar determinadas características dele, tais como: reusabilidade, modularidade e adaptabilidade.

f) Simplificação e normalização: são utilizadas para diminuir a complexidade sintática - por exemplo, traduzir a sintática em construções mais primitivas de uma linguagem.

2.8 Considerações finais

O DiSEN é um ambiente de desenvolvimento distribuído de software, que se preocupa em oferecer suporte para que equipes que se encontram em locais distintos possam trabalhar cooperativamente. Para isso é importante que os artefatos gerados durante todo o processo de desenvolvimento de software possam ser reutilizados e compartilhados por estes desenvolvedores.

Neste trabalho de dissertação propõe-se construir um modelo de interoperabilidade para permitir o trabalho cooperativo e possibilitar que os artefatos desenvolvidos durante o processo de desenvolvimento do software possam ser reutilizados, compartilhados e persistidos. Quando necessário, os artefatos sofrerão transformações para serem persistidos sobre um metamodelo que compõe o formato comum do ambiente, metamodelo este que representa o conhecimento sintático e semântico dos artefatos manipulados pelo ADDS. O DiSEN será utilizado como estudo de caso para aplicação e validação do modelo proposto.

Os conceitos apresentados neste capítulo oferecem o embasamento necessário para a contextualização, fundamentação e criação do modelo de interoperabilidade para o DiSEN, apresentado no capítulo 4.

3 TRABALHOS RELACIONADOS

A seguir serão apresentados alguns trabalhos relacionados com o tema, encontrados na literatura, que apresentam mecanismos capazes de propiciar formas distintas de manipulação de dados, em contextos bem-definidos e destinados a resolver problemas específicos. A análise destes trabalhos permitiu a identificação de características e funcionalidades importantes para um modelo de interoperabilidade, além de ter possibilitado traçar um paralelo entre os mecanismos de interoperabilidade encontrados e o modelo de interoperabilidade que é objeto deste trabalho.

3.1 OSCAR (*Open-Source Component and Artefact Repository*)

O projeto GENESIS (*GEneralized eNvironment for procESs management in cooperatIve Software engineering*) objetiva dar suporte a questões técnicas envolvidas no desenvolvimento de sistemas de software distribuído, oferecendo suporte ao gerenciamento de *workflow* e a processos para equipes distribuídas (AVERSANO et al., 2003).

OSCAR (BOLDYREFF, NUTTER, RANK, 2002a) é um repositório para armazenar e gerenciar artefatos distribuídos, e faz parte do projeto GENESIS. Os artefatos que podem ser manipulados são: modelos de processos, componentes de software, documentos de projetos ou qualquer outro tipo de entidade associada com o processo de engenharia de software. Cada artefato terá metadados associados a ele, que o descrevem, permitindo que usuários e repositórios manipulem e entendam este artefato (BOLDYREFF, NUTTER, RANK, 2002b).

OSCAR é um repositório implementado para o GENESIS, não sendo especificados pontos de extensão para que seja extensível a qualquer outro ADDS. O

principal objetivo do OSCAR é o armazenamento e recuperação dos artefatos, e não a interoperabilidade entre ferramentas de um ADDS particular.

É possível abstrair, a partir da análise da arquitetura do OSCAR (BOLDYREFF, NUTTER, RANK, 2002a), que para tornar os artefatos interoperáveis entre as ferramentas e para que este artefato possa ser armazenado e recuperado, é necessária a presença de um mecanismo que transforme o artefato salvo no repositório em um formato conhecido pelo usuário. Estes formatos de dados deverão ser mapeados de alguma forma, para que o artefato salvo no repositório do ADDS possa ser transformado no formato que a ferramenta seja capaz de manipular - no caso particular desta dissertação, um formato conhecido da ferramenta que manipula o artefato (arquivos XMI).

3.2 SESAME

O Sesame é uma arquitetura baseada na Web que permite armazenamento de dados RDF e *schemas*, o que possibilita a consulta online desta informação. O Sesame foi projetado para ser utilizado independentemente do banco de dados, pois é impossível saber qual é a melhor forma de armazenar os dados bem como o banco no qual estes devam ser armazenados. Para que esta funcionalidade fosse possível, foi adicionado à arquitetura um componente denominado SAIL (*Storage And Inference Layer*). O SAIL oferece métodos específicos a seus clientes e traduz estes métodos em chamadas para bancos de dados específicos (BROEKSTRA, KAMPMAN, HARMELEN, 2002).

O Sesame apresenta uma camada de abstração de dados antes do repositório, algo que é importante para um modelo de interoperabilidade. No DiSEN esta característica deve ser implementada para permitir que artefatos sejam transformados em um formato comum ao ADDS antes de serem persistidos e também no instante da sua reutilização. Esta

característica adicionará ao ambiente a capacidade de manipular diferentes tipos de dados e artefatos.

3.3 Odyssey Share

O projeto Odyssey (Odyssey, 2005) tem como principal objetivo prover mecanismos baseados em reutilização para o desenvolvimento de software, servindo como um arcabouço onde modelos conceituais, arquiteturas de software, e modelos implementacionais são especificados para domínios de aplicação previamente selecionados.

O Odyssey possui um domínio muito particular, que é o desenvolvimento baseado em componentes. Os artefatos produzidos pelas ferramentas que fazem parte do ambiente Odyssey podem ser importados ou exportados para outras ferramentas por meio do formato XML. Esta é uma forma interessante de compartilhar os artefatos do repositório.

Desta forma, o modelo de interoperabilidade, além de permitir que os artefatos sejam compartilhados por meio de um mecanismo de importação/exportação, deverá se preocupar com as diferenças de representação dos artefatos manipulados pelo DiSEN. Assim, torna-se necessário incorporar ao modelo uma forma de descrever os artefatos por meio da utilização dos metadados. Isto permitirá ao ambiente saber quais são os artefatos com que ele será capaz de interoperar. A descrição dos artefatos no Odyssey não foi relatada, deixando em aberto a forma como os artefatos são compartilhados no ambiente.

3.4 Arquitetura para compartilhamento de Conhecimento em Bibliotecas Digitais

A arquitetura objetiva integrar a base de conhecimento (informações) de duas bibliotecas: A Biblioteca Digital de Teses e Dissertações do IBICT e a Biblioteca Digital de Dissertação da SciELO.

Foi utilizada a linguagem OWL para representar o esquema comum, que se sobrepôs à heterogeneidade estrutural das fontes de dados. A linguagem OWL suporta várias ontologias, estejam elas relacionadas ou não (ALBUQUERQUE, KERN, 2004).

A arquitetura é constituída dos componentes a seguir relacionados e descritos (ALBUQUERQUE, KERN, 2004).

Interface usuário: tem a função de, automaticamente, transformar, converter e publicar os documentos XML nos dispositivos Web. É por meio desta interface que os usuários enviam a consulta sem ter conhecimento de como as informações estão de fato representadas.

Interface administrador: permite a criação e a manutenção do esquema conceitual e do catálogo de filtros e oferece suporte à tarefa de mapeamento.

Repositório de ontologias: armazena os esquemas conceituais que se encontram expressos diretamente em OWL. Além do esquema conceitual, também é armazenado um catálogo de filtros.

Interface mediador: recebe as consultas da interface usuário e as distribuí aos respectivos filtros.

Filtro SQL: é responsável por fazer a comunicação com os SGBDs relacionais por meio de instruções SQL.

Filtro XML: é responsável por fazer a comunicação com os SGBDs nativos XML.

Filtro XMLS: é responsável por fazer o mapeamento da estrutura de um modelo relacional para o padrão XML *Schema*.

A análise da arquitetura para compartilhamento de conhecimento em bibliotecas digitais permite a abstração de vários componentes que deverão fazer parte do modelo de interoperabilidade. A utilização de filtros permite a extensibilidade do modelo de

interoperabilidade. Para cada ferramenta pode-se construir um filtro que interprete o formato de dados para aquela ferramenta específica. Estes filtros seriam responsáveis por permitir a importação e exportação dos artefatos nos formatos que as ferramentas e o ambiente manipulam.

Um ADDS manipula vários tipos de artefato, como já foi mencionado. Esta característica adiciona certa complexidade ao modelo de interoperabilidade. No entanto, pretende-se descrever os artefatos utilizando-se os metadados, para permitir que estes artefatos sejam compartilhados. Estas descrições seriam armazenadas em uma base de conhecimento e permitiria ao ambiente, em tempo de execução, utilizar-se desta base para auxiliar na tarefa de importação e exportação dos artefatos.

3.5 XMLDoc: Ferramenta de Apoio à Documentação de ODE

A ferramenta XMLDoc é uma ferramenta que apóia a documentação integrada ao processo de software no contexto do ambiente ODE (*Ontology-based software Development Environment*). ODE é um ambiente de desenvolvimento de software (ADS) baseado em ontologias, isto é, sua infra-estrutura é fundamentada em ontologias de engenharia de software (FALBO, et al., 2004).

Usando a tecnologia XML, XMLDoc gera os documentos sempre com base nos dados do repositório central, favorecendo a atualização e a consistência do documento. A criação de documentos é dinâmica, ou seja, um documento só é efetivamente gerado no instante em que é solicitada sua exibição. Esta característica é importante, pois garante que o documento esteja sempre atualizado. Além disso, XMLDoc permite cadastrar modelos de documento. Estes modelos são a base para a padronização da documentação (FALBO, et al., 2004).

Alguns dos principais requisitos de XMLDoc são: (a) os desenvolvedores devem ter acesso a documentos atualizados e consistentes; (b) devem-se utilizar modelos para estabelecer como os documentos devem ser elaborados, de forma a auxiliar os desenvolvedores na realização dessa atividade; (c) documentos e diagramas devem ser consistentes entre si (FALBO, et al., 2004).

As características marcantes de XMLDoc são: (a) ser baseada em uma ontologia; (b) estar integrada a um ADDS e, por conseguinte, utilizar toda a funcionalidade de controle de processos de software disponível no ambiente (FALBO, et al., 2004).

Não obstante, é necessário avaliar se a ontologia se constitui na melhor forma de representar o conhecimento do ambiente (formatos dos artefatos), uma vez que a representação de conhecimento pode ser feita com outras abordagens - por exemplo, utilizando-se os princípios da MDA (OMG MDA, 2003) e do MOF (OMG MOF, 2003). Este conhecimento deverá ser representado de tal forma que seja possível a integração de artefatos gerados por diferentes ferramentas e o seu armazenamento obedecendo a um formato-padrão. XMLDoc também apresenta a característica de estar integrada a um ADDS; no entanto, não foram identificados pontos de extensão claros, que permitam sua extensão para outros ADDS, o que dificultaria a utilização do XMLDoc no ambiente do DiSEN.

3.6 Uma infra-estrutura para integração de ferramentas CASE

O objetivo da infra-estrutura é permitir aos usuários utilizar produtos de softwares que lhes sejam mais familiares na execução de suas atividades e compartilhar informações com outros usuários envolvidos em atividades semelhantes utilizando ferramentas diferentes. Isso foi possível com o mapeamento efetuado entre os artefatos das ferramentas que se desejava integrar e, a partir disto, um conversor foi gerado para permitir a transformação destes artefatos (SPINOLA, KALINOWSKI, TRAVASSOS, 2004).

A ferramenta Xmapper foi implementada para integrar ferramentas externas a uma infra-estrutura de apoio ao processo de inspeção de software. Os artefatos manipulados por ela devem pertencer ao domínio de inspeção de software e serem descritos em XML (SPINOLA, KALINOWSKI, TRAVASSOS, 2004).

Para possibilitar o intercâmbio de informações, a infra-estrutura foi composta de duas ferramentas principais: cliente e gerente de integração. A ferramenta cliente é responsável, mediante o componente de comunicação, por permitir ao usuário acesso, de forma transparente, às funcionalidades do gerente de integração (SPINOLA, KALINOWSKI, TRAVASSOS, 2004).

O gerente de integração é a parte principal da arquitetura. Ele permite que sejam tratadas questões referentes aos mapeamentos estruturais e semânticos dos artefatos para que seja possível a geração de conversores. O gerente de integração é quem mapeia os esquemas dos documentos a serem integrados. Esse documento é criado a partir de uma ontologia, descrevendo a semântica de um artefato, acrescida de informações estruturais provenientes dos esquemas dos artefatos (SPINOLA, KALINOWSKI, TRAVASSOS, 2004).

São a seguir apresentados os principais componentes da infra-estrutura (SPINOLA, KALINOWSKI, TRAVASSOS, 2004).

Gerente de integração: efetua transformações nos documentos XML, gerencia metadados (esquemas, conversores, ontologias e mapas de integração), estabelece mecanismos para mapeamento semi-automatizado entre informações estruturais e semânticas, cria de forma automatizada conversores (XSLTs), possibilita a definição semântica dos artefatos por meio de ontologias e torna transparente o acesso ao mecanismo de integração para seus usuários.

Gerente de metadado: é responsável por organizar a estrutura de armazenamento, salvar os vários tipos de documento manipulados pelo gerente de integração

(XML *schema*, uma ontologia (OWL), um mapa de integração ou um conversor (XSLT)), remover documentos e disponibilizar interfaces para interação com os demais componentes da infra-estrutura.

OWLEditor: é responsável por definir as ontologias em OWL. Na infra-estrutura as ontologias são utilizadas para auxiliar na formalização da semântica presente nos artefatos.

Assistente de publicação: nele são efetuados os mapeamentos estruturais e semânticos, e ele também permite a geração dos conversores entre os artefatos. Este mecanismo foi criado utilizando-se de XML *Schema*, OWL e XSLT, as duas primeiras voltadas para o mapeamento e a terceira, para transformações.

Componente transformação: é responsável por efetuar as transformações nos dados, utilizando para isso as regras de transformação criadas no assistente de publicação.

Componente de comunicação: ele torna transparente o acesso das ferramentas ao mecanismo de integração e disponibiliza interfaces que contêm as funcionalidades do gerente de integração relacionadas à transformação de documentos XML.

A grande dificuldade na adoção desta ferramenta para permitir a interoperabilidade dos artefatos e ferramentas do DiSEN é o fato de os artefatos serem diferentes e pertencerem a domínios diferentes.

Segundo Spinola, Kalinowski, Travassos (2004), os artefatos manipulados pela ferramenta Xmapper devem pertencer ao domínio de inspeção de software. Outro fator complicador para uma integração da Xmapper com o DiSEN é apresentado pelos mesmos autores quando eles afirmam que as ontologias podem auxiliar na tarefa de mapeamento dos artefatos, no entanto, elas nem sempre capturam toda a sua semântica.

Por isso deve ser considerada, para a implementação do modelo de interoperabilidade, a possibilidade de utilizar os conceitos presentes na MDA (OMG MDA,

2003) e no MOF (OMG MOF, 2003), pois estas podem se constituir em alternativas para a resolução do problema de representação semântica dos artefatos.

Outra necessidade e preocupação que o modelo de interoperabilidade do DiSEN deve possuir e que a infra-estrutura apresentada não contempla é a possibilidade de estes artefatos não serem somente compartilhados entre as ferramentas, mas sim, poderem ser armazenados e manipulados levando em consideração os aspectos de trabalho cooperativo.

XMapper também aponta que as abordagens para a integração de ferramentas CASE têm sido desenvolvidas, em particular, na área de desenvolvimento de software, porém não são apresentados pontos de extensão que permitam sua adição a um ADDS. Isto dificultaria uma possível adoção desta ferramenta no DiSEN.

3.7 ODYSSEY-MDA

A ferramenta ODYSSEY-MDA (MAIA, BLOIS, WERNER, 2005) é utilizada para realização de transformações sobre modelos UML. A Odyssey-MDA permite a definição de transformações a partir da sua infra-estrutura ou a partir de mecanismos desenvolvidos pelo usuário.

Suas principais características são: a independência de ambiente de desenvolvimento, pois foi feita em Java, utilizar-se de XMI como formato de transporte, executar transformações bidirecionais (PIM-PSM ou PSM-PIM) e a possibilidade de extensão por meio de *built-ins* ou *plug-ins*. Apesar de baseada no metamodelo MOF, ela só é capaz de manipular modelos UML. (MAIA, BLOIS, WERNER, 2005)

As transformações MDA se tornam fundamentais para a agilidade no desenvolvimento; no entanto, para o DiSEN, somente as transformações não são suficientes. É importante conhecer muito bem a semântica dos artefatos manipulados, além de se preocupar com a forma como eles serão persistidos. É preciso considerar que, além de

artefatos UML, outros poderão ser manipulados; por isso se devem considerar outras formas de transformação que não as utilizadas pelo ODYSSEY-MDA, que sugere sua utilização num contexto de desenvolvimento baseado em componentes.

3.8 Considerações Finais

Uma análise nos trabalhos supra-relacionados mostrou vários componentes que podem fazer parte do modelo de interoperabilidade aqui proposto (filtros, mecanismos de importação/exportação, base de conhecimento, transformadores e mapeadores, etc.), bem como as abordagens possíveis de ser consideradas para o desenvolvimento desta dissertação (MDA, metadados, MOF, ontologia e outras).

Os estudos realizados até o momento indicam que a utilização dos metadados é fundamental para descrição dos artefatos presentes em um ADDS. Os princípios e objetivos defendidos pelo MDA e o MOF são aplicados na manutenção da base de conhecimento (metamodelo) e construção dos mecanismos apresentados no capítulo 4.

4 IMART: MODELO DE INTEROPERABILIDADE PARA AMBIENTES DE DESENVOLVIMENTO DISTRIBUÍDO DE SOFTWARE

Este capítulo apresenta a especificação do IMART e para um ambiente de desenvolvimento distribuído de software. Inicialmente, são descritas as funcionalidades que o modelo de interoperabilidade pode adicionar a um ADDS. Posteriormente, é descrita a forma como os componentes do modelo foram adicionados à arquitetura do DiSEN, ambiente de desenvolvimento distribuído de software utilizado como estudo de caso. Também serão descritos como foram criados os componentes do modelo de interoperabilidade e a suas avaliações por meio de estudos de caso baseados em cenários.

4.1 Funcionalidades

A implementação do modelo de interoperabilidade deverá adicionar funcionalidades ao ADDS. Estas funcionalidades são importantes para apoiar o trabalho cooperativo e a reutilização dos artefatos, bem como a integração de ferramentas no ambiente de desenvolvimento. O modelo de interoperabilidade também permitirá ao ADDS manipular as informações contidas nos artefatos por meio de um metamodelo, possibilitando que estes artefatos sejam persistidos em diferentes formatos, ou manipulados de diferentes formas.

Para a construção do IMART foram considerados os questionamentos levantados por Bao e Horowitz (1996), segundo os quais o modelo de interoperabilidade deve ser aplicado a um domínio e ser extensível o suficiente para permitir que, à medida que sejam adicionados novos componentes na arquitetura, esta possa continuar válida sem a necessidade de grandes esforços de implementação, conforme descrito a seção 2.2.

Desta forma, o IMART deverá oferecer as funcionalidades a seguir relacionadas.

- **Permitir a reutilização de artefatos:** é importante para qualquer ambiente de desenvolvimento de software que os artefatos manuseados pelos membros da equipe de desenvolvimento possam ser reutilizados ao longo do desenvolvimento do projeto. Em um ambiente de desenvolvimento distribuído de software essa característica é ainda mais exigida, considerando-se a possibilidade de ocorrer que dois ou mais desenvolvedores estejam compartilhando artefatos e gerem versões deste mesmo artefato. O IMART permitirá que um artefato gerado por uma ferramenta instanciada no ADDS importe e exporte os artefatos para o ADDS habilitando o trabalho cooperativo, possibilitando desta forma a reutilização dos artefatos para acelerar o desenvolvimento do software. No entanto, não é tarefa do IMART resolver os conflitos relativos às possíveis versões geradas. Essa tarefa será do CVS por meio de outra interface, que deverá ser oferecida pelo ADDS e foge ao escopo deste trabalho.
- **Oferecer especificação de um formato-padrão de artefatos para o DiSEN:** será especificado um formato-padrão para representar os artefatos do ADDS. A representação da semântica dos artefatos, no formato-padrão, constituirá o conhecimento do ambiente. Este formato-padrão será chamado de metamodelo e será construído com base nos dados que se deseja manipular e reutilizar no ADDS. Esta característica é importante para o modelo de interoperabilidade, tendo-se em vista a unificação do formato de manipulação dos arquivos gerados pelas ferramentas instanciadas no ADDS.

- **Oferecer suporte à persistência em um formato padrão:** o modelo definirá mecanismos para que a persistência dos artefatos esteja baseada no formato-padrão de artefatos. No entanto, não será preocupação do IMART definir como e onde (CVS, Banco de Dados). Outro módulo deverá ser implementado para permitir o tratamento e aplicação das políticas de concorrência, merge, fragmentação e distribuição destes artefatos. Além da persistência, o metamodelo que representará o formato-padrão dos artefatos deverá permitir, por meio do próprio metamodelo, que interfaces sejam construídas para a manipulação das informações em outras formas além da serialização em arquivo.
- **Oferecer mecanismos de transformação dos artefatos:** cada ferramenta disponível (instanciada) no ADDS deverá conter a descrição dos artefatos que ela manipula no metamodelo do ambiente. Deverá também existir um mecanismo que efetue as transformações necessárias no metamodelo e consiga mapear e gerar o formato de saída para qualquer artefato persistido no ambiente.

As funcionalidades foram abstraídas com base nas necessidades do ambiente DiSEN, porém estas funcionalidades podem ser aplicadas a qualquer ADDS, tendo-se em vista a importância da reutilização dos artefatos, das transformações de modelos, do serviço de persistência para qualquer processo de desenvolvimento que preze qualidade e agilidade.

4.2 Componentes do Modelo de Interoperabilidade

Com vista a oferecer as funcionalidades apresentadas na seção 4.1, foram criados três componentes para constituir o modelo de interoperabilidade. A relação entre os componentes pode ser vista no diagrama de pacotes da Figura 8.

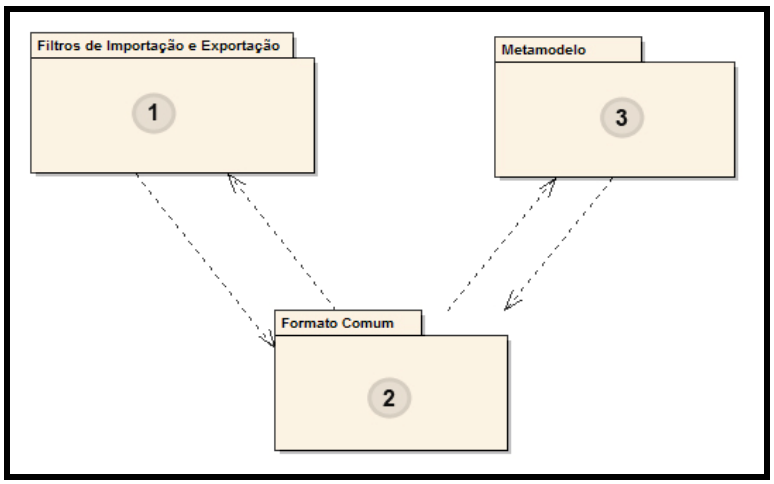

Figura 8 - Arquitetura do Modelo de Interoperabilidade

Como conseqüência da criação do modelo de interoperabilidade, é importante observar a intersecção destes componentes com a arquitetura do DiSEN apresentada na seção 2.1, Figura 1, uma vez que os componentes do modelo de interoperabilidade deverão adicionar funcionalidades a este ambiente. Além disso, estes componentes deverão interagir com os demais componentes já definidos na arquitetura do ADDS. A Figura 9 mostra os componentes do IMART inseridos na arquitetura do DiSEN. Mais detalhes sobre a extensibilidade dos componentes do IMART são discutidos na seção 4.4.

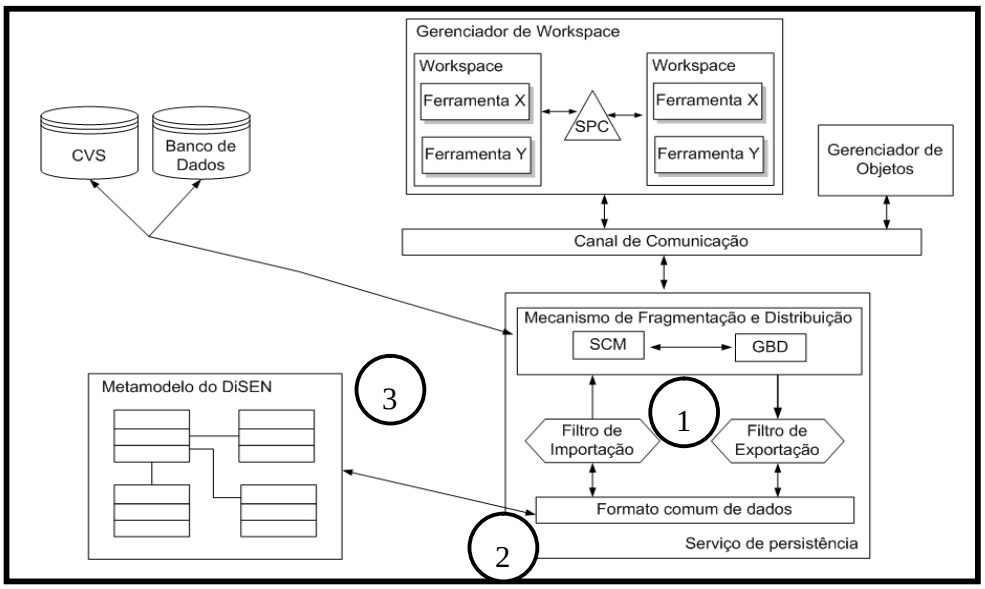

Figura 9 – Arquitetura do Modelo de Interoperabilidade inserido no DiSEN

Na Figura 9 podem-se destacar três componentes, que serão descritos a seguir, que não faziam parte até então da arquitetura inicial do DiSEN. Com a implementação e o estabelecimento da interação destes, será possível oferecer as funcionalidades já citadas na seção 4.1.

1) Filtros de importação e Exportação: constituem as *interfaces* que recebem ou disponibilizam os artefatos para serem persistidos. Eles também são responsáveis por atribuir os dados de um artefato ao metamodelo que os descreve. Os filtros constituem o transformador de modelos da arquitetura do IMART.

Para que o procedimento de transformação possa ser executado, é necessário que exista um mapeamento do artefato de entrada para as classes do metamodelo (caso o filtro escolhido seja o de importação), e um mapeamento do metamodelo para o artefato de saída (caso o filtro escolhido seja o de exportação).

Os filtros foram implementados utilizando-se o Castor (http://www.castor.org/, 2005), que fornece um *framework* de persistência capaz de fazer interagirem objetos Java, documentos XML e tabelas relacionais, além de permitir o *databinding* do XML. Ao contrário das duas principais API's XML, o DOM (*Document Object Model*) e o SAX (API simples para XML), que tratam da estrutura de um arquivo XML, o Castor permite tratar dados definidos em um arquivo XML por meio de um modelo de objeto que represente esses dados (metamodelo). Outras API's foram também avaliadas: (i) XStream (http://xstream.codehaus.org/,2006), que não é capaz de manipular atributos que fazem parte de uma *tag* do XML/XMI; e (ii) JAXB (https://jaxb.dev.java.net, 2005), que tem seu funcionamento baseado em *Schemas* e não oferece suporte adequado aos mapeamentos.

Mapeamentos: são parte importante dos filtros de importação e exportação, pois sem eles a tarefa de transformação não pode ser executada. É por meio deles que os métodos de *UnMarshall* (caso o filtro escolhido seja o de importação) e *Marshall* (caso o filtro escolhido seja o de exportação) podem interagir com o metamodelo para formar o artefato de saída (exportação), ou atribuir os valores de um artefato ao metamodelo (importação).

O Castor disponibiliza um método que permite o *marshal* de objetos Java e de arquivos XML, além de oferecer um conjunto de *ClassDescriptors* e de *FieldDescriptors* que formam o mapeamento e descrevem como um objeto deve ser *marshalled* e *unmarshalled*.

Os termos *marshal* e *unmarshal* significam o ato de converter um *stream* (seqüência dos bytes) de dados em um objeto e vice-versa. Assim, o ato de "*marshalling*" consiste em converter um objeto em um *stream*, e "*unmarshalling*", de converter um *stream* em um objeto. Os mapeamentos, portanto, explicitam as relações das classes que formam o metamodelo; desta forma eles se constituem nas regras de transformação, no instante em que o filtro de importação ou exportação é chamado (invocado). (http://www.castor.org/, 2005).

A Figura 10 mostra um fragmento do mapeamento do metamodelo que permite fazer interoperarem casos de uso. A *tag* <class> é o conjunto de *ClassDescriptors* e a *tag* <field> é o conjunto de *FieldDescriptors* necessários para o mapeamento.

```
<class name="enginedisen.Content">
 <field name="models" type="enginedisen.Model" collection="collection">
    <bind-xml name="Model" node="element" />
 </field>
</class>

<class name="enginedisen.Model">
  <field name="name" type="string">
    <bind-xml name="name" node="attribute" />
  </field>
  <field name="xmi_id" type="string">
    <bind-xml name="xmi.id" node="attribute" />
  </field>
   <field name="Specification" type="string">
    <bind-xml name="isSpecification" node="attribute" />
  </field>
    <field name="Root" type="string">
    <bind-xml name="isRoot" node="attribute" />
  </field>
    <field name="Leaf" type="string">
    <bind-xml name="isLeaf" node="attribute" />
```

1

2

Figura 10 – Fragmento de um mapeamento

O mapeamento acima indica que é necessário haver uma classe no metamodelo com o nome "*Content*", e esta contém uma relação para uma coleção (lista) de modelos que pode ser observada na marcação (1), da figura 10. A segunda classe se chama "*Model*". Neste mesmo fragmento percebem-se os atributos que compõem a classe modelo: Name, Xmi Id, *Specification*, *Root* e *Leaf*, que pode ser observado na marcação (2), da figura 10.

Uma classe no mapeamento indica uma *tag* no arquivo XML que está sendo manipulado. Os atributos no mapeamento indicam os valores de atributos de uma *tag* que serão lidos no arquivo XML e atribuídos (método set) ao metamodelo como atributos da classe, ou vice-versa.

O IMART deverá oferecer suporte à manipulação de vários mapeamentos. Cada mapeamento descreverá a relação do arquivo XML/XMI (artefato) a ser manipulado por um ADDS, com o metamodelo que permitirá a este artefato ser importado ou exportado.

Mencionar a manipulação dos artefatos do ADDS implica em dizer que não somente são atribuídos e recebidos dados do artefato por meio do metamodelo e dos mapeamentos, mas também que existe a possibilidade de efetuar um conjunto de transformações nesse artefato.

Transformações: o mecanismo de transformação é ponto importante do IMART. Os filtros e mapeamentos fornecem métodos que permitem a leitura e escrita de um artefato, com base no metamodelo criado; no entanto, eles não realizam as transformações necessárias das diferenças encontradas entre os elementos pertencentes aos artefatos que se deseja transformar. Assim, foram criados métodos específicos para realizar as transformações necessárias.

Os métodos de transformação utilizam-se do formato comum de dados, uma vez que antes das transformações é necessário fazer a formatação de entrada (leitura), e depois

da transformação, submeter o novo artefato a uma formatação de saída (escrita). A figura 11 mostra a seqüência de execução dos métodos de transformação.

```
public void transformaPoseidonEA() throws IOException, MappingException, MarshalException, ValidationException {
    lerArquivoXMLPoseidon();
    transformationHeaderEA();
    transformation();
    escreveArquivoXMLPoseidon();
}
```

Figura 11 - Método de transformação

O método transformation() é o principal método de transformação. É ele que atribui os novos valores ao metamodelo para efetuar a transformação de saída pelo filtro de exportação. A figura 12 mostra um fragmento de código do método que faz uma alteração no metamodelo.

```
UseCaseInclude useCaseInclude = new UseCaseInclude();
useCaseInclude.setInclude(inc);

IncludeBase includeBase = new IncludeBase();
IncludeAddition includeAddtion = new IncludeAddition();
if (associationEnd1.getNavigable().equals("true")) {
    includeBase.setUseCase(useCaseParticipante2);
    includeAddtion.setUseCase(useCaseParticipante1);
    useCase2.setUseCaseInclude(useCaseInclude);
} else {
    includeBase.setUseCase(useCaseParticipante1);
    includeAddtion.setUseCase(useCaseParticipante2);
    useCase1.setUseCaseInclude(useCaseInclude);
}

Include include = new Include();
include.setXmi_id(id);
include.setSpecification(specification);
include.setIncludeBase(includeBase);
include.setIncludeAddition(includeAddtion);

model.getOwnedElement().addInclude(include);
```

Figura 12 – Fragmento do método Transformation()

Desta forma é possível escrever métodos de transformação de artefatos, desde que a semântica esteja mapeada para os filtros de importação/exportação e estes representados no metamodelo por meio de classes.

A vantagem da transformação por meio de um *parser* (conjunto de métodos de transformação) que atribui os valores para a transformação em relação a outras técnicas, como, por exemplo, utilização do XSLT - *Extensible Stylesheet Language Transformations* (http://www.w3.org/TR/xslt, 2005), corresponde ao fato de os valores atribuídos no metamodelo poderem ser manipulados de outras formas, e não somente serem transformados em outro arquivo, um arquivo de saída, por exemplo.

Esta vantagem de manipulação permite, por exemplo, que ferramentas sejam construídas sobre este metamodelo, as quais podem ser desde ferramentas CASE até aquelas de geração de relatórios gerenciais e/ou informações e medições, sobre os artefatos importados para o ambiente.

Outra vantagem significativa do XSLT é a facilidade de manutenção e a não-necessidade de aplicação de regras para as transformações. Tudo é feito em nível semântico, e para realizar as transformações basta utilizar-se dos métodos básicos já descritos no metamodelo necessários para os mapeamentos. Já no XSLT é necessário aprender como aplicar as regras de transformação, e a saída será sempre a geração de um arquivo de saída. À medida que novos requisitos sejam adicionados ao metamodelo, será necessário adicionar também novas regras no XSLT, o que pode ser uma tarefa custosa e complexa.

2) Formato comum de dados: é o processo de refinamento pelo qual o artefato deverá passar antes e após a sua geração. Um método é utilizado para preparar um artefato para ser transformado (leitura), e após a transformação, um segundo método é executado para formatar a saída (escrita) do artefato transformado. Os métodos são

necessários, pois diferentes ferramentas podem conter "*namespaces*" distintos antes das *tag's* do arquivo XML/XMI de entrada. Os métodos de formatação e de escrita são capazes de identificar as *tag's* e aplicar os *namespaces* necessários para a leitura dos arquivos XMI gerados por ferramentas externas ao ambiente. Outra tarefa realizada pelo método de formatação de escrita é a identação do arquivo de saída, uma vez que o filtro de exportação gera um arquivo de saída em uma única linha. A Figura 13 mostra um trecho do método de refinamento utilizado pelo formato comum para identar um arquivo XMI exportado pelo IMART.

```
StringBuffer novoArquivo = new StringBuffer();
int cont = 0;
String espaco = "";
String[] vetor = arquivo.toString().split("\n");
for (String linha: vetor) {
    if (linha.indexOf("<?") == -1) {
        if (linha.indexOf("</") == 0) {
            espaco = espaco(cont);
            cont--;
        } else if (linha.indexOf("<") == 0) {
            cont++;
            espaco = espaco(cont);
            if ((linha.indexOf("</") > -1) || (linha.indexOf("/>") > -1)){
                cont--;
            }
        }
        linha = espaco + linha;
```

Figura 13 – Fragmento do método de refinamento

3) Metamodelo: O metamodelo é parte importante da arquitetura do modelo de interoperabilidade e representa o conhecimento que o ambiente possui sobre os artefatos que ele é capaz de manipular. Desta forma, eles representam a semântica destes dados e artefatos. Como visto no Capítulo 2, os metamodelos e metadados permitem descrever os dados que serão manipulados por meio de estruturas bem-definidas.

O metamodelo foi criado para permitir que os valores de um arquivo XMI de entrada, correspondente ao arquivo XMI do artefato gerado por uma ferramenta externa ao

ADDS, pudessem ser manipulados. O metamodelo é um conjunto de classes e relacionamentos, estruturados de acordo com a semântica do artefato, que precisa ser manipulado pelo ambiente. No caso particular desta dissertação, os testes foram executados considerando-se os elementos de diagramas de caso de uso: atores, casos de uso, relações, relações com estereótipo <<include>> e <<extend>>.

A marcação (1) da Figura 14 mostra um fragmento do diagrama de classe que representa o metamodelo construído para manipulação dos elementos de um diagrama de caso de uso. A marcação (2) mostra um fragmento do pacote de classes Java que representam a implementação do metamodelo.

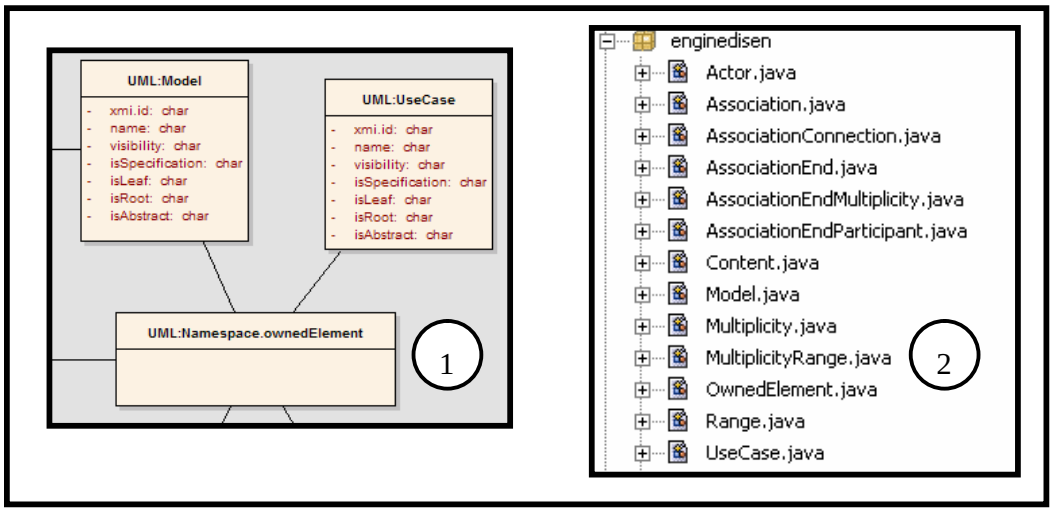

Figura 14 – Fragmentos do diagrama de classes e da implementação do metamodelo em Java

Uma vantagem da utilização do metamodelo para representação dos artefatos consiste na possibilidade de implementação de diferentes parsers e métodos que possibilitem a persistência do artefato de diferentes formas. É possível, por exemplo, serializar um arquivo XMI de saída para uma ferramenta-alvo escolhida, gerar um arquivo XMI e persisti-lo em um CVS, ou até mesmo construir métodos que persistam suas informações em diferentes bancos

de dados. Isso é possível, pois o metamodelo permite a extensibilidade de representação dos artefatos e a flexibilidade na forma de manipular os dados contidos nele.

4.3 Criação dos mapeamentos e metamodelo

Como foi apresentada na Seção 4.2, a arquitetura do modelo de interoperabilidade contém três componentes principais. Os mapeamentos e o metamodelo são construídos ou modificados sempre que um novo tipo de artefato necessitar ser manipulado pelo ambiente de desenvolvimento de software distribuído. Isso permite que a arquitetura seja flexível e possa ser estendida de acordo com os artefatos que ela necessite manipular.

A tarefa de criar o mapeamento e o metamodelo não é trivial, pois requer conhecimento sobre o artefato que se deseje manipular no ambiente - no caso particular do DiSEN, sobre a semântica de diagramas de caso de uso e como essa semântica é representada em arquivos XMI, arquivos estes que servem como entrada para o modelo de interoperabilidade.

A fim de facilitar esta tarefa criou-se uma técnica para orientar a criação dos mapeamentos e do metamodelo. Essa técnica consiste de alguns passos, descritos a seguir:

a) Descrição dos mapeamentos: para a criação do mapeamento é necessário exportar um arquivo XMI do artefato gerado por uma ferramenta externa a ser integrada ao ADDS. Uma vez exportado o artefato, é necessário estudar a sua semântica. Cada *tag* no arquivo XMI exportado representará uma classe no mapeamento, e os atributos contidos na *tag* deverão ser descritos no mapeamento. A Figura 15 mostra um fragmento de um arquivo XMI que representa um diagrama de caso de uso exportado por uma ferramenta CASE.

```
<UML:Model xmi.id = 'I64888lcdml0637e69daemm7f55' name = 'modelo' isSpecification = 'false'
  isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
  <UML:Namespace.ownedElement>
   <UML:UseCase xmi.id = 'I64888lcdml0637e69daemm7eb0' name = 'teste segunda relacao'
      visibility = 'public' isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
      isAbstract = 'false'/>
```

Figura 15 – Fragmento de um XMI exportado por uma ferramenta CASE

A figura 16 mostra um fragmento de um mapeamento que corresponde a este artefato.

```
<class name="enginedisen.Model">
  <field name="name" type="string">
    <bind-xml name="name" node="attribute" />
  </field>
  <field name="xmi_id" type="string">
    <bind-xml name="xmi.id" node="attribute" />
  </field>
   <field name="Specification" type="string">
    <bind-xml name="isSpecification" node="attribute" />
  </field>
    <field name="Root" type="string">
    <bind-xml name="isRoot" node="attribute" />
  </field>
    <field name="Leaf" type="string">
    <bind-xml name="isLeaf" node="attribute" />
  </field>
    <field name="Abstract" type="string">
    <bind-xml name="isAbstract" node="attribute" />
  </field>
  <field name="ownedElement" type="enginedisen.OwnedElement">
    <bind-xml name="Namespace.ownedElement" node="element" />
 </field>
</class>
```

1

```
<class name="enginedisen.OwnedElement">
 <field name="actors" type="enginedisen.Actor" collection="collection">
    <bind-xml name="Actor" node="element" />
 </field>
 <field name="usecases" type="enginedisen.UseCase" collection="collection">
    <bind-xml name="UseCase" node="element" />
 </field>
 <field name="associations" type="enginedisen.Association" collection="collection">
    <bind-xml name="Association" node="element" />
 </field>
</class>
```

2

Figura 16 – Fragmento de um mapeamento

Para este mapeamento pode-se perceber que foi criada uma classe para cada *tag* do arquivo XMI. Cada classe criada é indicada no *ClassDescriptor* do arquivo de

mapeamento. Quando uma *tag* é criada antes do fechamento da *tag* pai, é necessário indicar no mapeamento que existe um relacionamento entre estas classes. Um exemplo desta situação é mostrado nas figuras 15 e 16, onde a *tag* <UML:Namespace.ownedElement> é criada antes do fechamento da *tag* <Uml:Model>. O mesmo acontece com a *tag* <UML:UseCase>, que é criada antes do fechamento da *tag* <UML:Namespace.ownedElement>. Quando acontece esta situação o mapeamento deve conter um *FieldDescriptor* específico para indicar a relação.

Cada atributo do arquivo XMI também é indicado no mapeamento. A figura 16 a marcação (1) mostra como a classe *Model* e seus atributos são mapeados. Os atributos da classe *Model* são: xmi.id, name, *specification*, *root*, *leaf*, *abstract*; e a relação com a classe *ownedElement* é indicada por meio da *tag* <field>.

A *tag* <field> contém um atributo "*name*" e um "*type*", onde o atributo "*type*" representa a semântica do dado que está sendo mapeado. A *tag* <bind-xml> também contém um atributo "*name*" e um "*node*". O atributo "*node*" indica se ele é um atributo ou um elemento do XML/XMI de entrada. Sempre que for mapeado um atributo de uma *tag* do arquivo de entrada, o valor de *node* receberá "*attribute*". Para mapear uma relação entre as classes, como é o caso da Relação *Model-OwnedElement*, o atributo *node* receberá "*element*".

Para cada campo "*name*" da *tag* <field> devem existir um método set e um método *get* com o mesmo valor atribuído para "*name*". Por exemplo, no mapeamento existe a *tag* <field name="*Specification*" type="*string*"> da classe *Model*; no entanto, a classe *Model* do metamodelo implementado em Java deve conter um atributo do tipo string com o nome *Specification* e com os métodos *set* e *get*, conforme mostra a figura 17.

```
public String getSpecification() {
  return specification;
}

 public void setSpecification(String isSpecification) {
   this.specification = isSpecification;
}
```

Figura 17 – Fragmento da classe Model do metamodelo do DiSEN implementada em Java

O campo "*name*" da *tag* <bind-xml name="*isSpecification*" type="*attribute*"> é retirado do arquivo XMI de entrada e representa o nome do atributo contido na *tag* <UML:Model> que contém um atributo com o nome "*isSpecification*".

Como já dito anteriormente, quando da explicação do formato comum, os "*namespaces*" que compõem o arquivo de entrada devem ser ignorados no instante da criação dos mapeamentos e do metamodelo. Depois será necessário um refinamento no formato comum do modelo de interoperabilidade. Este refinamento será feito por um método que adicionará os *namespaces* antes que o arquivo XMI de saída seja gerado.

b) Descrição do metamodelo: para a criação do metamodelo, por meio da implementação das classes em Java, é necessário criar um diagrama de classes que represente a semântica do artefato que se deseje interoperar, e logo em seguida implementar o diagrama de classes utilizando uma linguagem orientada a objetos.

No IMART, foi utilizada a linguagem Java, pelo fato de esta linguagem ter sido adotada para a construção do DiSEN e permitir a utilização do *framework* Castor, que oferece uma estrutura adequada para manipulação de arquivos XML, por meio da criação dos mapeamentos e do metamodelo. Após as descrições dos mapeamentos e do metamodelo, o passo final consiste em construir as classes em Java. No entanto, devem-se seguir os cuidados com a criação dos métodos *get* e *set* discutidos no item a) e apontados nas Figuras 16 e 17.

4.4 Extensibilidade do Modelo de Interoperabilidade

Para Bao e Horowitz (1996), uma habilidade importante para um modelo de interoperabilidade é a capacidade que ele tem de ser extensível. Seguindo-se esta recomendação identificaram-se duas formas de extensibilidade.

(i) Extensibilidade dos componentes do modelo de interoperabilidade: este primeiro tipo diz respeito à capacidade de, a partir da necessidade de representação de uma nova semântica, os componentes do modelo de interoperabilidade sofrerem alterações/adaptações para continuarem válidos e úteis para realizar a tarefa que se propõe.

Um exemplo desta extensibilidade seria uma alteração nos dados que estão sendo compartilhados por duas ferramentas distintas por meio do modelo de interoperabilidade. Isso provocaria uma atualização no metamodelo e nos mapeamentos para a representação daquele artefato e, provavelmente, seria necessário construir *parsers* para permitir as transformações necessárias neste artefato.

Esta extensibilidade é uma característica importante para um modelo de interoperabilidade, tendo-se em vista que, em um processo de desenvolvimento de software, os artefatos utilizados para a construção do produto podem ser de diferentes tipos e sofrer alterações ao longo do tempo.

Neste trabalho, esta extensibilidade é alcançada pela implementação do metamodelo e dos mapeamentos, que são flexíveis o suficiente para manipular e representar a semântica de um artefato de diferentes formas. Esta característica será comprovada na seção 4.5, onde o modelo de interoperabilidade será avaliado.

(ii) Extensibilidade do modelo de interoperabilidade para adaptação em novos domínios: o segundo tipo de extensibilidade se refere à forma de adaptação do modelo de interoperabilidade a alguma arquitetura ou domínio já existente.

A Figura 9 mostrou como os três componentes do modelo de interoperabilidade foram adaptados à arquitetura inicial no DiSEN. Para esta adaptação o ponto de intersecção utilizado foi o serviço de persistência do ambiente, que é responsável pela fragmentação e distribuição dos dados do ADDS.

Escolheu-se este serviço para a adaptação dos componentes do modelo de interoperabilidade, pois o serviço de persistência é que permite o tratamento dos artefatos manipulados do DiSEN de uma forma transparente para os desenvolvedores, uma vez que todos os dados persistidos são recuperados e manipulados por ele.

É possível adaptar os componentes do modelo de interoperabilidade, apresentados na figura 8, às camadas que operem sobre dados e artefatos em outros ambientes de desenvolvimento distribuído de software. A adaptação é possível, pois o modelo de interoperabilidade permite representar os artefatos com que o ADDS deseja interoperar, por meio do metamodelo, e posteriormente a isso, fazê-los interagir com o formato comum e com os filtros de importação e exportação.

O modelo pode ser utilizado como uma interface para acessar e manipular estes dados, o que constitui uma vantagem do modelo de interoperabilidade já apresentada na seção 4.2.

Uma vez apresentadas as funcionalidades, os componentes do modelo de interoperabilidade, a forma como estes foram implementados e as maneiras pelas quais o modelo poderia ser estendido para uma arquitetura e para atender a novos requisitos, apresentam-se na seção 4.5 as avaliações realizadas com o IMART.

4.5 Avaliações do Modelo de Interoperabilidade

Em Travassos, et al (2002) pode ser encontrado um estudo acerca da importância e das hipóteses de experimentos que podem ser realizados para avaliação, sendo os três principais discutidos a seguir:

Survey: os objetivos do *survey* são, por exemplo, determinar a distribuição de atributos ou características (descritivo), explicar o porquê de os desenvolvedores terem escolhido uma técnica (explanatória) ou o estudo preliminar para uma investigação mais profunda (explorativa). O *survey* permite levantar um grande número de variáveis a serem avaliadas (variáveis qualitativas e quantitativas) (TRAVASSOS, G.H.; GUROV, D.; AMARAL, 2002).

Estudo de caso: é utilizado para monitorar projetos, atividades e atribuições. Estudos de caso visam observar um atributo específico e estabelecer relação com atributos diferentes (TRAVASSOS, G.H.; GUROV, D.; AMARAL, 2002).

Experimento: normalmente, é realizado em laboratório e oferece um nível maior de controle sobre os resultados. O seu objetivo é manipular uma ou algumas variáveis e manter as outras fixas a fim de medir o efeito do resultado. Os experimentos são utilizados para confirmar teorias, confirmar o conhecimento convencional, explorar os relacionamentos, avaliar a predição dos modelos ou as medidas (TRAVASSOS, G.H.; GUROV, D.; AMARAL, 2002).

Ainda, segundo Travassos, et al. (2002), o tipo de experimento mais apropriado em uma determinada situação vai depender dos objetivos do estudo, das propriedades do processo de software utilizado durante a experimentação, ou dos resultados finais esperados com o experimento.

Nesta dissertação optou-se por fazer um estudo de caso, tendo-se em vista a necessidade de observar se as funcionalidades foram atendidas após a implementação dos

componentes do modelo de interoperabilidade. Anteriormente, o ambiente DiSEN não oferecia suporte para reutilizar artefatos, permitindo que eles fossem compartilhados entre os desenvolvedores. Os artefatos criados com uma ferramenta nativa do ambiente não conseguiam persistir seus artefatos nem reutilizavam os artefatos produzidos por ferramentas CASE externas ao ambiente.

Para a avaliação foram elaborados estudos de caso com cenários para se testar os mapeamentos e metamodelos construídos, avaliando-se assim os componentes do IMART apresentados na seção 4.2. A seguir serão descritos os cenários de avaliação e a ferramenta de simulação utilizada.

4.5.1 Ferramenta de simulação

Para a realização dos testes definidos nos cenários, foi criada uma ferramenta que simulasse o processo apresentado na figura 18, que mostra a interação dos componentes do modelo de interoperabilidade durante estes testes.

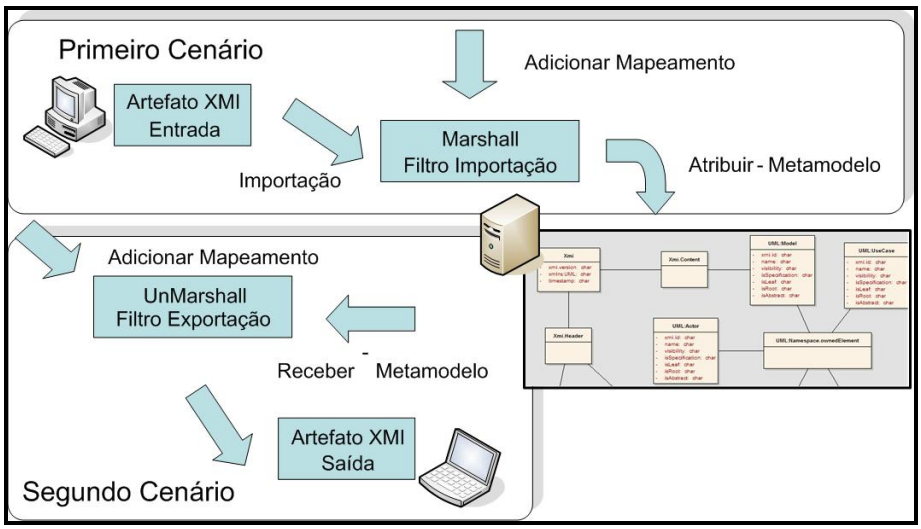

Figura 18 – Processo de interação entre os componentes do modelo de interoperabilidade.

O processo é composto de dois cenários, o primeiro dos quais procura avaliar a obtenção de um arquivo XMI de entrada gerado por uma ferramenta CASE. Para isso é

necessário utilizar o mapeamento correspondente, bem como um filtro de importação para atribuir os valores do arquivo XMI no metamodelo.

O segundo cenário testa a situação inversa ao primeiro cenário, onde um mapeamento e um filtro de exportação são utilizados para receber os dados do metamodelo e gerar o artefato XMI de saída correspondente a outra ferramenta CASE.

A figura 19 mostra a interface da ferramenta de simulação que implementa os componentes da arquitetura do modelo de interoperabilidade descritos nas seções anteriores e permite a avaliação dos dois cenários descritos sucintamente, o quais serão discutidos nas seções 4.5.2 e 4.5.3.

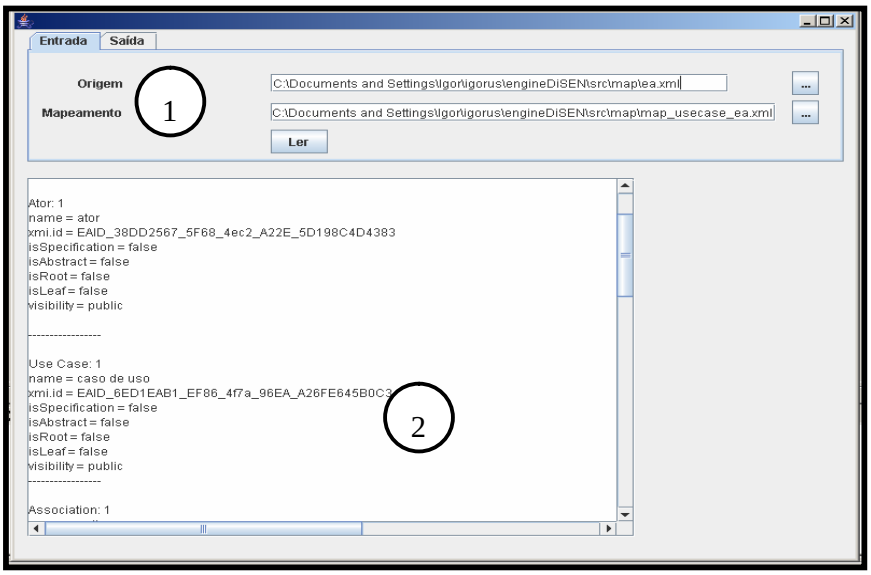

Figura 19 – Interface da ferramenta de simulação, tela de importação.

A primeira parte da Figura 19, no topo, indicada pela marcação (1), mostra as abas de escolha para testar o processo de entrada (importação de um artefato para o ambiente – atribuição no metamodelo) ou saída (exportação do ambiente/metamodelo para uma ferramenta externa – obtenção dos dados do artefato no metamodelo).

Para testar a entrada é necessário indicar um arquivo de origem criado por uma ferramenta, e escolher o mapeamento construído para a sua interpretação. Clicando no botão

ler, ocorre a transformação, em que são atribuídos os valores do arquivo de entrada, do metamodelo do DiSEN. Esses valores são apresentados no espaço abaixo, indicado pela marcação (2) na Figura 19.

Para a transformação do artefato em arquivo XMI de saída, os dados do artefato de entrada foram atribuídos ao metamodelo, indicando posteriormente o mapeamento da ferramenta de destino, o local e o nome do arquivo de saída.

Abaixo estão descritos quatro cenários de avaliação considerados para avaliar o IMART.

4.5.2 Primeiro Cenário: Atribuir dados do Metamodelo

O primeiro cenário a ser testado verificou se o IMART era capaz de atribuir e obter os valores de um arquivo XMI gerado por uma ferramenta CASE *open-source*, Poseidon, (http://gentleware.com, 2005) no metamodelo do IMART. Um segundo teste foi realizado com uma ferramenta CASE proprietária Enterprise Architect (http://www.sparxsystems.com.au, 2005). Para todos os cenários o artefato utilizado para a realização dos testes foi um diagrama de caso de uso, porém no primeiro cenário somente um ator, uma relação e um caso de uso foram descritos e mapeados para o metamodelo.

Após os dois primeiros testes, detectou-se a necessidade de descrever no metamodelo as diferenças encontradas em cada artefato de saída, criando um mapeamento para cada uma das ferramentas. Isso foi essencial para a utilização do metamodelo, tendo-se em vista que cada uma das ferramentas gera o seu próprio arquivo XMI.

A Figura 19, na marcação (2), mostra a obtenção e impressão dos valores obtidos durante os testes do primeiro cenário, validando o metamodelo, o filtro de importação e o mapeamento feito para os diagramas de caso de uso.

4.5.3 Segundo Cenário: Obter dados do Metamodelo

O segundo cenário testou o recebimento dos dados de um artefato gerado por uma das ferramentas, utilizando novamente o seu mapeamento correspondente. Foi possível obter os dados do metamodelo utilizando os métodos (*get*) para o filtro de exportação, o que permitiu a geração de um arquivo de saída.

Como resultado dos testes realizados nos dois primeiros cenários, obteve-se um arquivo XMI de saída, conforme a Figura 20 apresenta.

Figura 20 – Arquivo XMI de saída gerado a partir do metamodelo.

É possível perceber que o arquivo de saída não apresenta *namespaces*, por não ter sido submetido ao processo de refinamento no segundo componente da arquitetura, o formato comum de dados. No formato comum é que são adicionados os *namespaces* de cada ferramenta. No entanto já se pode perceber, conforme mostra a Figura 20, que o metamodelo atendeu perfeitamente ao seu propósito, que era o de atribuir ao metamodelo os valores de um artefato de entrada e gerar o artefato de saída baseado no mapeamento da ferramenta desejada. Estes testes foram aplicados para a transformação tanto do Poseidon – Metamodelo do Ambiente - Enteprise Architect, quanto do inverso Enteprise Architect - Metamodelo do Ambiente - Poseidon.

Os testes realizados neste cenário não implicaram em alteração alguma nos mapeamentos ou metamodelos criados, durante a execução do primeiro cenário.

Outros testes foram realizados inserindo-se mais atores, relações e casos de uso ao diagrama de casos de uso. Estas inserções refletiram em incremento na representação semântica dos artefatos. Isto foi feito para verificar se as atribuições e obtenções de dados dos arquivos XMI de entrada e a geração de arquivos XMI de saída, como foram ilustradas na figura 20, continuaram a ser realizadas com sucesso conforme descritas a seguir.

4.5.4 Terceiro Cenário: Adição de complexidade na semântica dos artefatos

Este cenário de testes implicou na adição de complexidade dos artefatos considerados para verificar se os componentes do modelo de interoperabilidade continuavam válidos e poderiam atender a novos requisitos, como, por exemplo, a adição de nova semântica ao artefato que estava sendo manipulado.

Com esse cenário, foi possível testar a adição de novas classes e atributos nos mapeamentos, e também testar se o metamodelo poderia ser estendido. Para estes testes, os diagramas de caso de uso foram gerados com atores, casos de uso, relações e estereótipos do tipo <<include>> e <<extend>>. Os testes utilizaram a ferramenta de simulação descrita na seção 4.5.1 e mostrada na Figura 19. Estes testes, no entanto, requereram um novo metamodelo, que fosse capaz de representar semanticamente os estereótipos. Novos mapeamentos também precisaram ser construídos de acordo com as alterações realizadas no metamodelo, e estes deveriam ser escolhidos pela ferramenta de simulação no instante de testar a obtenção e atribuição dos dados do metamodelo.

Os testes mostraram que as diferenças na representação semântica dos artefatos de saída de cada ferramenta demandaram a criação de métodos de transformação, principalmente no que se refere à forma como cada ferramenta tratava os estereótipos <<include>> e <<extend>>.

Um exemplo dessa diferença pode ser visto na Figura 21, que mostra a ferramenta Enterprise Architect (EA) tratando o estereótipo <<extend>>. No EA, um estereótipo <<extend>> se torna uma associação entre dois casos de uso. A ferramenta Poseidon trata os estereótipos como uma outra estrutura, como mostra a Figura 22.

```
<UML.Stereotype xmi.id="EAID_A9E775EB_83FF_4971_B364_198D0E0AE205" name="extend" isSpecification="false" isRoot="fa
        <UML.Stereotype.baseClass>Association</UML.Stereotype.baseClass>
</UML.Stereotype>
<UML.Association xmi.id="EAID_27C2816D_8C4B_4cb3_A1B6_5CC952965FCE" visibility="public" isRoot="false" isLeaf="fals
        <UML.ModelElement.stereotype>
                <UML.Stereotype xmi.idref="EAID_A9E775EB_83FF_4971_B364_198D0E0AE205"/>
        </UML.ModelElement.stereotype>
        <UML.Association.connection>
                <UML.AssociationEnd visibility="public" aggregation="none" isOrdered="false" isNavigable="false" ty
                        <UML.AssociationEnd.multiplicity>
                                <UML.Multiplicity>
                                        <UML.Multiplicity.range>
                                                <UML.MultiplicityRange lower="0" upper="-1"/>
                                        </UML.Multiplicity.range>
                                </UML.Multiplicity>
                        </UML.AssociationEnd.multiplicity>
                        <UML.AssociationEnd.participant>
                                <UML.UseCase xmi.idref="EAID_D30E2FBB_DF84_476a_85B2_E23C2C097066"/>
                        </UML.AssociationEnd.participant>
                </UML.AssociationEnd>
                <UML.AssociationEnd visibility="public" aggregation="none" isOrdered="false" isNavigable="true" typ
                        <UML.AssociationEnd.multiplicity>
                                <UML.Multiplicity>
                                        <UML.Multiplicity.range>
                                                <UML.MultiplicityRange lower="0" upper="-1"/>
                                        </UML.Multiplicity.range>
                                </UML.Multiplicity>
                        </UML.AssociationEnd.multiplicity>
                        <UML.AssociationEnd.participant>
                                <UML.UseCase xmi.idref="EAID_0B541C34_A031_48d9_B9B4_7A19141B4036"/>
                        </UML.AssociationEnd.participant>
                </UML.AssociationEnd>
        </UML.Association.connection>
</UML.Association>
```

Figura 21 - Fragmento do XMI da ferramenta EA, mostrando um esteriótipo <<extend>>.

```
<UML:Extend xmi.id = 'I24ae0e96m1093f6cf4f0mm78de' isSpecification = 'false'>
    <UML:Extend.base>
      <UML:UseCase xmi.idref = 'I24ae0e96m1093f6cf4f0mm78e5'/>
    </UML:Extend.base>
    <UML:Extend.extension>
      <UML:UseCase xmi.idref = 'I24ae0e96m1093f6cf4f0mm790f'/>
    </UML:Extend.extension>
 </UML:Extend>
```

Figura 22 – Fragmento do XMI da ferramenta Poseidon, mostrando um esteriótipo <<extend>>.

A Figura 23 mostra um arquivo de saída gerado pela ferramenta de simulação. É possível perceber que o arquivo de saída apresenta os *namespaces*, por ter sido submetido ao método de refinamento definido no formato comum de dados. Também é possível verificar que o metamodelo resultou em um arquivo de saída correto para a transformação desejada, validando, assim, as adições no metamodelo, o *parser* do formato comum de dados que realiza as transformações. Estes testes foram aplicados tanto para a transformação Poseidon - Metamodelo do Ambiente - Enteprise Architect, quanto o inverso.

```
<?xml version="1.0" encoding="UTF-8"?>
    <XMI xmi.version="1.2" xmlns:UML="org.omg.xmi.namespace.UML" timestamp="2006-02-06 16:55:20">
        <XMI.content>
            <UML:Model name="Use Case Model" xmi.id="MX_EAID_E77A5D80_0EE2_4dfc_99C4_C8E98BA4992F" isSpecification="false" is
                <UML:Namespace.ownedElement>
                    <UML:Actor name="Actor2" xmi.id="EAID_35E3E7B1_491B_41d2_8A7D_FD15F0D5A9BD" isSpecification="false" is
                    <UML:Actor name="Actor1" xmi.id="EAID_821FB01E_4C84_4f3c_8892_87D1A4956477" isSpecification="false" is
                    <UML:UseCase name="Use Case5" xmi.id="EAID_0B541C34_A031_48d9_B9B4_7A19141B4036" isSpecification="false
                    <UML:UseCase name="Use Case1" xmi.id="EAID_101E7E17_5FDC_4c76_826D_B4F37199A547" isSpecification="false
                    <UML:UseCase name="Use Case6" xmi.id="EAID_13D7C169_C029_4194_A1C5_C1570476A4C5" isSpecification="false
                    <UML:UseCase name="Use Case2" xmi.id="EAID_47CA3369_A815_437b_947B_69533285DF6C" isSpecification="false
                    <UML:UseCase name="Use Case3" xmi.id="EAID_A999F86B_CEFF_4ecc_98F2_FA5F324FBBB3" isSpecification="false
                    <UML:UseCase name="Use Case4" xmi.id="EAID_D30E2FBB_DF84_476a_85B2_E23C2C097066" isSpecification="false
                    <UML:Association xmi.id="EAID_D518282D_9D91_4064_80FC_BFF2AEB6A7D3" visibility="public" isRoot="false"
                        <UML:Association.connection>
                            <UML:AssociationEnd visibility="public" isNavigable="true" isOrdered="unordered" aggregation=
                                <UML:AssociationEnd.multiplicity>
                                    <UML:Multiplicity>
                                        <UML:Multiplicity.range>
                                            <UML:MultiplicityRange lower="0" upper="-1"/>
                                        </UML:Multiplicity.range>
                                    </UML:Multiplicity>
                                </UML:AssociationEnd.multiplicity>
                                <UML:AssociationEnd.participant>
                                    <UML:Actor xmi.idref="EAID_821FB01E_4C84_4f3c_8892_87D1A4956477"/>
                                </UML:AssociationEnd.participant>
                            </UML:AssociationEnd>
                            <UML:AssociationEnd visibility="public" isNavigable="true" isOrdered="unordered" aggregation=
                                <UML:AssociationEnd.multiplicity>
                                    <UML:Multiplicity>
```

Figura 23 – Arquivo de Saída gerado pela ferramenta de simulação.

Pode-se perceber na Figura 23 que o arquivo gerado possui a mesma identação dos arquivos originais exportados pelas ferramentas Poseidon e EA. Isso foi feito para que a técnica de criação do metamodelo e dos mapeamentos possa ser aplicada sem prejudicar a visualização do arquivo XMI, e seguir o padrão que, normalmente, é utilizado pelas ferramentas CASE.

A Figura 24 representa um dos testes realizados com os elementos de um diagrama de caso de uso. Percebe-se no diagrama a presença de atores, casos de uso, relações e os estereótipos <<include>> e <<extend>>.

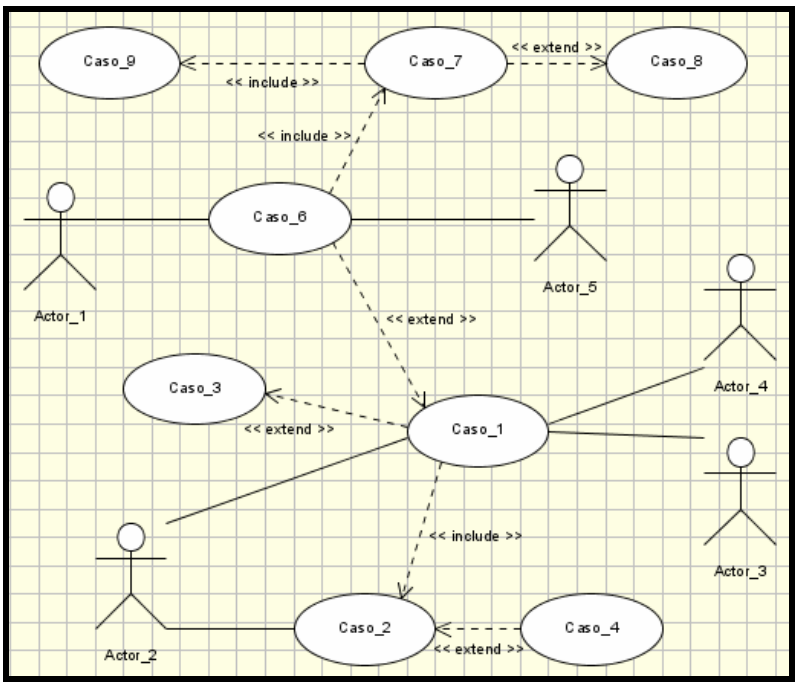

Figura 24 – Diagrama de caso de uso realizado para teste do Metamodelo.

Durante o último teste, deste terceiro cenário, percebe-se que a ferramenta Enterprise Architect – EA possuía um erro na exportação e importação do arquivo XMI quando era selecionada a versão 1.2 do XMI. O erro pode ser observado comparando-se as Figuras 25 e 26. A Figura 25 representa um caso de uso criado na ferramenta e exportado na versão 1.2 do XMI; logo depois foi importado esse caso de uso e o resultado pode ser visto na Figura 26.

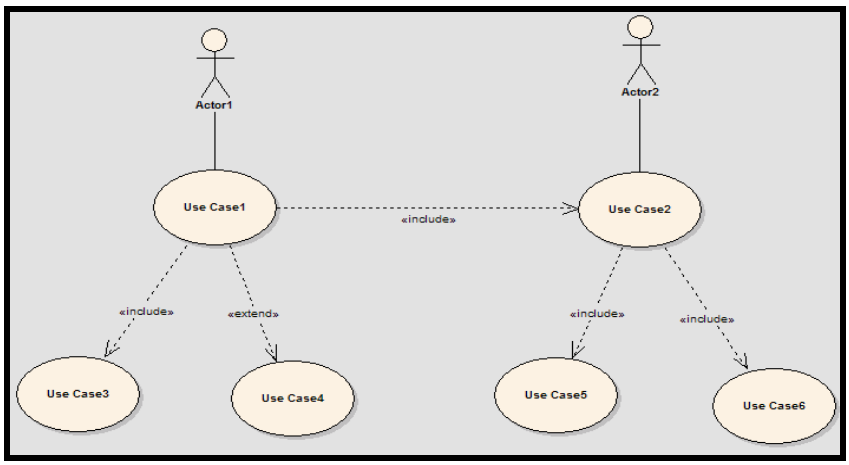

Figura 25 – Diagrama de caso de uso correto da ferramenta EA.

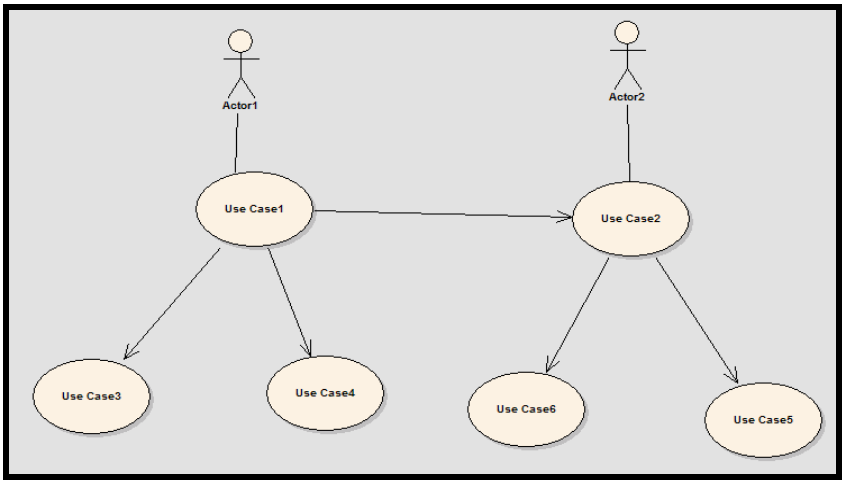

Figura 26 – Diagrama de caso de uso importado de maneira errada pela ferramenta EA.

As relações entre os casos de uso que continham estereótipos de <<include>> ou <<extend>> foram mantidas na direção correta, no entanto o estereótipo foi perdido e a sua representação gráfica foi alterada, tendo as linhas tracejadas sido substituídas por linhas contínuas.

Como um mapeamento já havia sido construído e adicionado às classes no metamodelo para representar o artefato gerado pela ferramenta EA, foi possível construir um método que corrigisse o artefato exportado. Após a aplicação da transformação foi possível importar o artefato para a própria ferramenta de acordo com a representação da figura 23, que era a representação correta para o artefato.

Uma vez que os testes foram concluídos, foi então construída uma nova interface para a transformação dos artefatos. A nova interface já não exige que os mapeamentos de entrada e saída sejam informados, mas basta que o usuário selecione um artefato de entrada para ser transformado em um artefato de saída. É necessário somente informar o tipo de transformação por meio dos *radio buttons* e solicitar a transformação. A figura 27 mostra a nova interface.

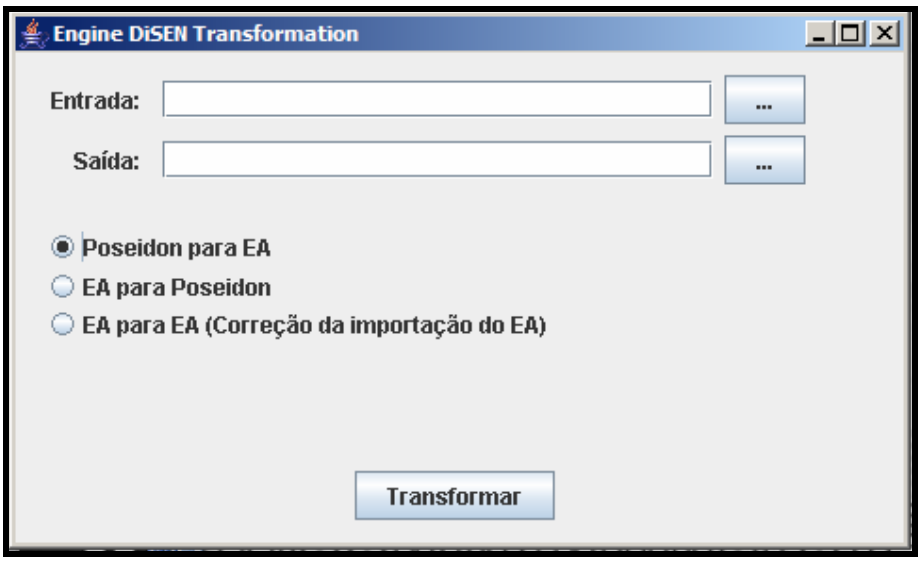

Figura 27 – Nova interface para a transformação dos artefatos.

As seções 4.5.2, 4.5.3 e 4.5.4 apresentaram cenários que permitissem testar e avaliar se os componentes do modelo de interoperabilidade atendiam às funcionalidades requeridas por um ambiente de desenvolvimento distribuído de software, o DiSEN. Foi possível mostrar como o modelo obtinha e atribuía dados ao metamodelo, permitindo que filtros e parsers pudessem atuar sobre o metamodelo, a fim de gerar os arquivos XMI necessários para a reutilização entre ferramentas distintas instanciadas no DiSEN.

Também foi possível avaliar como o modelo de interoperabilidade se comportava quando o ADDS solicitasse uma inclusão na semântica de representação do artefato manipulado pelas ferramentas, mostrando que o metamodelo e os demais componentes do modelo de interoperabilidade podiam ser extensíveis quanto à necessidade de atender a um novo requisito do ADDS.

O quarto cenário avaliará a vantagem de utilização do metamodelo como representação dos artefatos, e novamente avaliará quanto ele pode ser extensível para resolução de outros problemas relacionados com a tarefa de interoperabilidade de ferramentas e reutilização dos artefatos do ADDS.

4.5.5 Quarto Cenário: Adição de funcionalidades na Ferramenta REQUISITE.

O quarto cenário permitiu realizar um novo teste, para validar se o metamodelo poderia ser utilizado por outras ferramentas construídas para o ambiente - por exemplo, uma ferramenta CASE, uma ferramenta para métricas, enfim, qualquer ferramenta que pudesse ser criada para atender a uma necessidade do DiSEN. Conforme visto na seção 2.1.1, que apresentou a ferramenta REQUISITE, algumas das funcionalidades apresentadas na Figura 2 foram reimplementas utilizando-se o metamodelo para representar a semântica dos artefatos (diagramas de caso de uso) manipulados pela ferramenta.

Nesta nova implementação os objetos que antes eram instanciados sem preocupar-se com a semântica dos artefatos, pois uma instância deles representava uma referência gráfica do objeto na tela, passaram a representar um objeto pertencente ao metamodelo, composto de atributos e relações com outros objetos, bem como a representação gráfica.

Como a REQUISITE passou a utilizar o metamodelo para a criação dos elementos no diagrama, era possível mapear os objetos (elementos do diagrama de caso de uso descritos no metamodelo) e realizar neles as transformações e aplicações dos métodos do formato comum e dos filtros de importação e exportação, para importação e criação de arquivos XMI.

A Figura 28 ilustra um artefato construído nas ferramentas CASE Poseidon e EA, o qual pôde ser reutilizado após a implantação do metamodelo na REQUISTE. É possível visualizar o diagrama e, no canto esquerdo, a árvore com os elementos do diagrama importados.

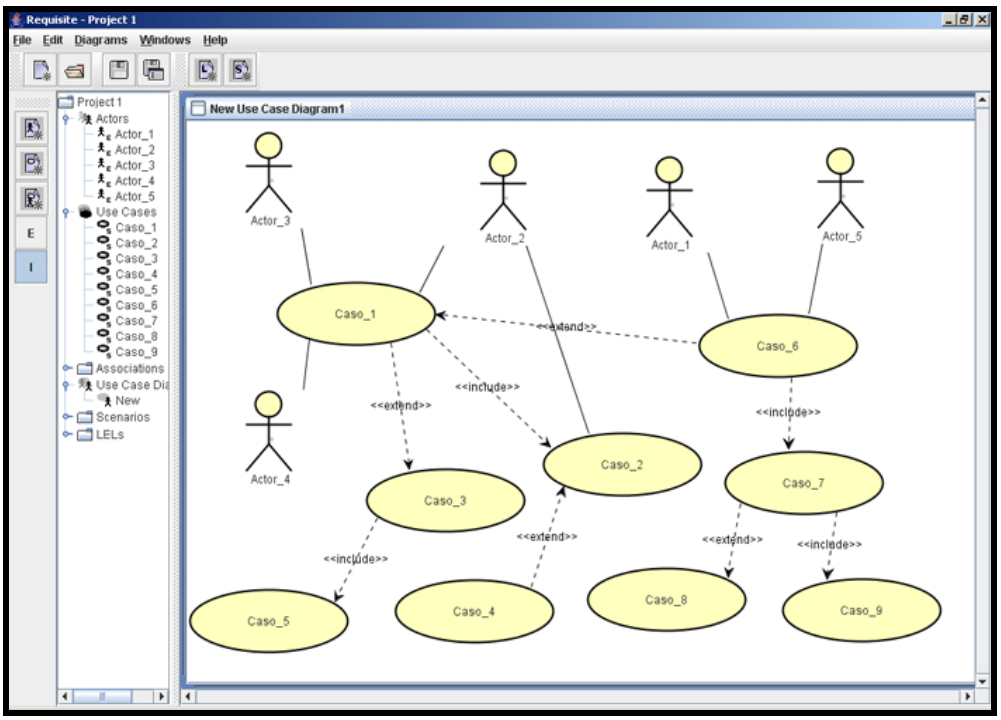

Figura 28 – Representação de um Diagrama de Caso de Uso na Ferramenta Requisite

A seção 4.5.5 apresentou como foram adicionadas funcionalidades à ferramenta nativa do DiSEN, a REQUISITE. As funcionalidades de recuperar e salvar o modelo agora permitem à ferramenta REQUISITE recuperar um artefato que esteja armazenado no ambiente DiSEN que tenha sido criado pela ferramenta EA ou Poseidon. Também foi possível reimplementar, usando o filtro de exportação, a funcionalidade de salvar modelo. Esta funcionalidade foi implementada para salvar o artefato criado na REQUISITE serializando-o em um arquivo XMI, baseado no metamodelo do formato comum do DiSEN.

Os quatro cenários apresentados puderam avaliar e mostrar a aplicação dos modelos de interoperabilidade. Foi possível mostrar a sua extensibilidade tanto para representar um artefato e permitir que ele fosse reutilizado e compartilhado, bem como a adaptação (extensibilidade) do modelo a um ADDS.

5 Conclusão

Os resultados esperados com a especificação e implementação do IMART foram alcançados, pois as funcionalidades que foram propostas e implementadas no modelo, ou seja, permitir que os conhecimentos semântico sobre os artefatos gerados por ferramentas CASE pudessem ser armazenados e manipulados por um ambiente de desenvolvimento distribuído de software. Os componentes previstos na arquitetura do modelo de interoperabilidade também puderam ser implementados e atenderam aos requisitos solicitados, pois foi possível criar um metamodelo para representar, semântica e sintaticamente, os artefatos de diferentes ferramentas CASE. Os filtros permitiram a leitura e escrita dos artefatos por meio da utilização do metamodelo, e foram construídos *parsers* para efetuar as transformações necessárias nos artefatos.

É importante ressaltar que, para a construção do modelo e o processo de validação, levaram-se em conta os questionamentos levantados por Bao e Horowitz (1996). O IMART foi aplicado a um domínio, o DiSEN. O modelo atendeu aos requisitos do ambiente, que necessitava que os seus artefatos tivessem seu conhecimento armazenado e manipulado, bem como exigia que este metamodelo oferecesse formas de transformação nos dados.

O modelo também se mostrou extensível o suficiente para permitir a inclusão de novos requisitos do ADDS que está utilizando o IMART. No terceiro teste foram adicionadas novas semânticas aos artefatos manipulados, com a adição dos estereótipos de <<include>> e <<extend>>, e estes puderam ser representados. Tanto o metamodelo como os mapeamentos puderam ser incrementados sem a necessidade de grandes esforços de implementação. Foi necessário o conhecimento sobre o artefato manipulado.

Também foi possível adicionar novos transformadores quando se percebeu a necessidade de correção de uma importação/exportação da ferramenta CASE Enterprise Architect, que apresentava um erro quando tentava importar um artefato exportado por ela

própria. O método de transformação foi adicionado, e sem a necessidade de qualquer outra alteração, foi possível gerar o formato correto para importação da ferramenta que corrigia o problema da representação dos estereótipos <<include>> e <<extend>>.

Quanto aos esforços de implementação, concluiu-se que é difícil mensurar o conceito de complexidade para construir os mapeamentos, o metamodelo, os filtros de importação e exportação e os *parsers* para transformação. No entanto, procurou-se minimizar essa dificuldade com a criação da técnica do metamodelo e mapeamentos. A tarefa de criação do metamodelo e dos mapeamentos pode ser minimizada à medida que aumenta o conhecimento que se tem sobre o artefato a ser manipulado. Quanto maior o conhecimento sobre a semântica do artefato, menor a dificuldade de criação do metamodelo e dos mapeamentos. Conhecimentos da abordagem orientada a modelos, de modelagem orientada a objetos e Java também são essenciais para a criação do metamodelo e mapeamentos.

Para a criação dos *parsers* e filtros de importação e exportação, a dificuldade é proporcional ao conhecimento da linguagem Java e ao conhecimento sobre o metamodelo criado. Percebeu-se que, se o desenvolvedor dos *parsers* e filtros de importação e exportação for o mesmo desenvolvedor dos mapeamentos e metamodelo, a tarefa pode ser menos complexa. No entanto, percebeu-se também que não é uma tarefa trivial conhecer a semântica do artefato a ser manipulado, especialmente em relação aos testes realizados. É importante muito conhecimento em diagramas UML, metamodelo UML e conhecimentos sobre as extensões da família XML, no nosso caso, XMI.

5.1 Contribuições.

O presente trabalho de dissertação destaca as seguintes contribuições.

- Os artefatos do DiSEN, que antes não eram compartilhados com ferramentas externas ao ambiente, depois da implantação do IMART

puderam ser reutilizados e compartilhados entre os desenvolvedores, com a utilização de diferentes ferramentas CASE, permitindo o trabalho cooperativo e aumentando a produtividade dos desenvolvedores.

- O metamodelo do ambiente permitiu a manipulação dos artefatos de distintas maneiras. Duas delas foram apresentadas na dissertação, por meio da criação de *parsers* para transformações dos artefatos e dos filtros de importação e exportação que possibilitaram a atribuição e obtenção da semântica dos diagramas de caso de uso, bem como a geração dos arquivos de entrada para persistência no ADDS ou de exportação para utilização das ferramentas CASE. O metamodelo também se mostrou fundamental, quando, por meio da sua adaptação e implementação na ferramenta REQUISITE, foi possível persistir e compartilhar seus artefatos com as ferramentas Poseidon e EA.
- A especificação do modelo foi concebida de forma que ela possa ser reutilizada ou sirva como base para construção de mecanismos de interoperabilidade em outros ADDS. Isso ficou comprovado durante a adaptação dos componentes na atualização da arquitetura do DiSEN, onde foram adicionados os componentes do IMART na arquitetura do DiSEN.
- O modelo também se mostrou extensível quando é necessário adicionar semântica na representação de um artefato já mapeado e representado no metamodelo.

5.2 Limitações do Modelo de Interoperabilidade

Os arquivos que representam os artefatos do DiSEN devem estar no formato XML/XMI. Isso implica que as ferramentas externas que venham a ser integradas ao ambiente, necessariamente, deverão exportar seus artefatos no formato XML/XMI. Atualmente essa limitação não trará tantos problemas com relação à adição de ferramentas externas para o ambiente, tendo-se em vista que o XML se tornou um padrão para intercâmbio de dados e tem sido utilizado por quase todas as ferramentas CASE do mercado, sejam elas proprietárias ou gratuítas.

O primeiro protótipo permite a interoperabilidade de artefatos (elementos de diagrama de caso de uso) produzidos por duas ferramentas CASE (Enterprise-Architect e Poseidon), no entanto é possível estendê-lo para outras ferramentas CASE.

5.3 Trabalhos Futuros

Como trabalho futuro pretende-se estender o IMART para outros tipos de artefato e verificar o seu funcionamento, além deste, contribuições também em trabalhos futuros:

- estender o metamodelo para representar outros artefatos diferentes destes utilizados no estudo de caso, como por exemplo, representar diagramas de classes, seqüência, estados, documentos de requisitos, dados de gerenciamento de projeto, etc.;
- criar uma interface no mecanismo de fragmentação e distribuição dos dados do DiSEN, para permitir a replicação dos artefatos em bases de dados distribuídas;
- desenvolver uma ferramenta para facilitar a criação dos mapeamentos e do metamodelo, de forma que este processo fique o mais automatizado

possível, refinando assim a técnica de criação de mapeamento e do metamodelo;

- implementar um *plugin* que facilite a implementação das funcionalidades do modelo para outras ferramentas do ambiente DiSEN e criar novas ferramentas com base no metamodelo, permitindo que estes artefatos possam ser reutilizados e utilizados de outras formas ao longo do desenvolvimento do software;
- criar mecanismos de transformação dos artefatos, baseados na abordagem orientada a modelos, permitindo as transformações bidirecionais de modelos PIM-PSM (MDA) e posteriormente a geração de código a partir dos artefatos criados e mapeados no ADDS;
- estudar outras formas de mapeamento e construção do metamodelo, por exemplo, considerando a utilização de ontologias para esta tarefa.

Referências

ALBUQUERQUE, Nikolai Dimitrii.; KERN, Vinícius Medina. **Uma Arquitetura para o Compartilhamento do Conhecimento em Bibliotecas Digitais**. XIII SEMINCO: Seminário de Computação da Universidade Regional de Blumenau. Páginas: 149-160, 2004.

AVERSANO, Lerina.; CIMITILE, Aniello.; LUCIA, Andrea de; STEFANUCCI, Silvio; VILANN, Maria Luisa. **Workflow Management in the GENESIS Environment**. Research Center on Software Technology, Department of Engineering, University of Sannio, 2003.

BAO, Yimin. HOROWITZ, Ellis. **New Approach to Software Tool Interoperability**, In: Proceedings of ACM 11th Annual Symposium on Applied Computing, Philadelphia, Pennsylvania, United States. Pages: 500-509, 1996.

BATISTA, Sueleni Mendes. **Uma ferramenta de apoio á fase de requisitos da MDSODI no contexto do ambiente DiSEN.** 2003, 83 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Informática, Universidade Federal do Paraná, Curitiba, 2003.

BOLDYREFF, Cornelia; NUTTER, David; and RANK, Stephen. **Architectural Requirements for an Open Source Component and Artefact Repository System within GENESIS**. In: WORKSHOP OPEN SOURCE SOFTWARE DEVELOMPMENT, 2002, Newcastle, UK. Proceedings Newcastle, UK, 2002a. p.176-196.

BOLDYREFF, Cornelia; NUTTER, David; and RANK, Stephen. **Open-Source Artefact Management**. In: WORKSHOP OPEN SOURCE SOFTWARE ENGINEERING, 2, 2002, Proceedings Orlando, Florida, USA, 2002b.

BRODSKY, Stephen A. **XMI Opens Application Interchange**. 1999. [online] Disponível na Internet via WWW. URL: http://www-306.ibm.com/software/awdtools/standards/ xmiwhite0399.pdf. Acessado em 18 de fevereiro de 2006.

BROEKSTRA, Jeen.; KAMPMAN, Arjohn.; and HARMELEN, Frank van. **Sesame: A Generic Architecture for Storing and Querying RDF**. In: Proceedings of the first International Semantic Web Conference (ISWC2002), number 2342 in Lecture Notes in Computer Science, pages 54-68, Sardinia, Italy, June 9 - 12, 2002. Springer Verlag, Heidelberg Germany. See also http://www.openrdf.org/

COULOURIS, George., DOLLIMORE, Jean., KINDBERG, Tim.. **Distributed System: Concepts and Design**, 2 ed. Harlow: Person Education Edinburgh Gate Ltd, 2001. 632 p.

CZARNECKI Krzysztof., HELSEN, Simon. **Classification of Model Transformation Approaches**. OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture. 2003.

FALBO, Ricardo .A. **Integração de Conhecimento em um Ambiente de Desenvolvimento de Software**. Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, 1998.

FALBO, Ricardo .A. NUNES, Vanessa B., SOARES, Andrea O. **Apoio à Documentação em um Ambiente de Desenvolvimento de Software**. VII Workshop Iberoamericano de Ingeniería de Requisitos y Desarrollo de Ambientes de Software (IDEAS'2004), pp. 50-55, Arequipa, Perú, Maio 2004.

GILLILAND-SWETLAND, Anne J.. **Introduction to Metadata**. [online] Disponível na Internet via WWW. URL: http://www.getty.edu/research/conducting_research/standards/intrometadata/2_articles/index.html. Ultimo acesso 15 de maio de 2005.

GRAVENA, Juliana Pelandre. **Aspectos Importantes de uma Metodologia para Desenvolvimento de Software com Objetos Distribuídos**. Trabalho de graduação, Universidade Estadual de Maringá, Departamento de Informática, 2000.

HARRISON, Willian., OSSHER, Harold., TARR, Peri. **Software Engineering Tools and Environments: A Roadmap**, In: Proceedings of The Future of Software Engineering, ICSE'2000, Limerick, Ireland, 2000.

HUZITA, Elisa Hatsue Moriya., **MOOPP – Uma Metodologia para Auxiliar o Desenvolvimento de Aplicações para Processamento Paralelo.** 1995. 213 p. Tese (Doutorado) - Escola Politécnica, USP, São Paulo.

HUZITA, Elisa Hatsue Moriya. **Suporte à Reutilização em Ambientes Distribuídos de Desenvolvimento de Software. Projeto em andamento (CNPq).** Projeto de pesquisa em andamento, Universidade Estadual de Maringá. Departamento de Informática, 2004.

ISO - International Organization for Standardization. **ISO 8879:1986(E)**. Information processing - Text and Office Systems - Standard Generalized Markup Language (SGML). First edition - 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

JACOBSON, Ivar.; BOOCH, Grady.; RUMBAUGH, James. **The Unified Software Development Process.** [S.l]: Addison-Wesley, 1999.

KLEPPE, Anneke., WARMER, Jos., WIM Bast.: **MDA Explained, The Model-Driven Architecture: Practice and Promise**. Addison Wesley. 2003.

LIMA, Fabiana. **Mecanismos de Apoio ao Gerenciador de Recursos Humanos no Contexto de um Ambiente Distribuído de Software**. 2004, 86 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, Universidade Estadual de Maringá, Maringá, 2004.

MAIA, Natanael Elias Nascimento. ; BLOIS, Ana Paula Terra Bacelo. ; WERNER, Cláudia Maria Lima. . Odyssey-MDA: **Uma abordagem para transformação de modelos de componentes**. In: Quinto Workshop de Desenvolvimento Baseado em Componentes, 2005, Juiz de Fora. Quinto Workshop de Desenvolvimento Baseado em Componentes, 2005. p. 89-96.

MANDVIWALLA, Munir; GRILLO, Peter. **TeamBox: an exploration of collaborative interoperability**. In: Proceedings of conference on Organizational computing systems - Conference on Supporting Group Work, 1995, Milpitas, California, United States. Pages: 347-353, 1995.

MENS, Tom., CZARNECKI Krzysztof, GORP, Pieter Van. **A Taxonomy of Model Transformations**. Internationales Begegnungs- und Forschungszentrum (IBFI),Schloss Dagstuhl, Germany. 2005. acessado em: http://drops.dagstuhl.de/opus/volltexte/2005/11/.

MOURA, Lygia Maria V.; ROCHA, Ana Regina C.. **Ambientes de Desenvolvimento de Software**, Publicações Técnicas COPPE/UFRJ, ES-271/92, Rio de Janeiro.

NISO - National Information Standards Organization. **Understanding Metadada**. 2004 (booklet - ISBN: 1-880124-62-9). [online] Disponível na Internet via WWW. URL: www.niso.org. Acessado em 10 de maio de 2005.

OMG MDA. ***MDA Guide 1.0.1***. 2003. [online] Disponível na Internet via WWW. URL: http://www.omg.org. Ultimo acesso 26 de Abril de 2005.

OMG MOF. **MOF Specification 1.4**. 2003. Disponível na Internet via WWW. URL: http://www.omg.org. Ultimo acesso 26 de Abril de 2005.

OMG XMI. **XMI Specification 2.1**. 2005. Disponível na Internet via WWW. URL: http://www.omg.org/technology/documents/formal/xmi.htm. Ultimo acesso 15 de Fevereiro de 2006.

PASCUTTI, Márcia. Cristina Dadalto. **Uma proposta de arquitetura de um ambiente de desenvolvimento de software distribuído baseado em agentes**. 2002, 102 f. Dissertação (Mestrado) - Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002.

PEDRAS, Maria Edith Villela. **Uma Ferramenta de Apoio ao Gerenciamento de Desenvolvimento de Software Distribuído.** 2003, 113 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Informática, Universidade Federal do Paraná, Curitiba, 2003.

PELTIER. Mikaël., ZISERMAN François., BÉZIVIN Jean. **On levels of model transformation**. In: XML Europe. 2000, Paris, France, June 2000, Graphic Communications Association, pp. 1–17

POZZA, Rogério dos Santos. **Proposta de um Framework para Aspectos do Gerenciador de Workspace no Ambiente DiSEN**. Proposta de Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, Universidade Estadual de Maringá, Maringá, 2004.

PRESSMAN, Roger .S. **Software Engineering: A Practitioner's Approach**, 6th Edition, New York: McGraw-Hill, 2006.

ODYSSEY. **Página oficial do Projeto Odyssey**. [online] Disponível na Internet via WWW. URL: http://www.cos.ufrj.br/~odyssey/. Acessado em 08 de maio de 2005.

RABELLO, Abraham Lincoln.; REIS, Carla. Alessandra. Lima.; REIS, Rodrigo Quites.; PIMENTA, Marcelo Soares.; NUNES, Daltro José; **Analisando a Interação entre Desenvolvedores em Ambientes de Processo de Software: Classificação e Exemplos**, VII Semana Acadêmica, Universidade Federal do Rio Grande do Sul, 2001.

ROSETTO, Márcia. **Metadados e formatos de metadados em sistemas de informação**: **caracterização e definição**. São Paulo, 2003. 112 p. (Dissertação de mestrado apresentada ao Curso de Pós – Graduação da Escola e Comunicações e Artes da Universidade de São Paulo)

SOFTWARE ENGINEERING INSTITUTE (SEI). **Current Perspectives on Interoperability**. ISR Technical Report # CMU/SEI-2004-TR-009 ESC-TR-2004-009. Carnegie Mellon, Pittsburgh, EUA, 2004. 2p.

SPINOLA, Rodrigo ; KALINOWSKI, Marcos ; TRAVASSOS, Guilherme Horta. **Uma Infra-Estrutura para Integração de Ferramentas CASE**. In: Simposio Brasileiro de Engenharia de Software - SBES, 2004, Brasilia - DF. Anais do XVIII SBES. Porto Alegre : Sociedade Brasileira de Computacao, 2004. p. 147-162.

TANENBAUM, Andrew., STEEN, Maarten Van. **Distributed System – Principles and Paradigms**, 1 ed. New Jersey: Prentice Hall, 2002. 803 p.

TRAVASSOS, Guilherme Horta.; GUROV, Dmytro.; AMARAL, Edgar Augusto Gurgel do, 2002, **Introdução à Engenharia de Software Experimental**. In: Relatório Técnico ES-590/02-Abril, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ.

VAZ, Maria Salete Marcon Gomes, **Introdução a Metadados**. [online] Disponível na Internet via WWW. URL: http://www.sqlmagazine.com.br/Colunistas/MariaSalete/02_Metadados.asp. Ultimo acesso 15 de maio de 2005.

WEGNER, Peter. **Interoperability**. ACM Computing Surveys, vol. 28, março 1996.

WILEDEN, Jack C.; KAPLAN, Alan. **Software Interoperability: Principles and Practice** (Tutorial). ICSE 1997: 631-632.

W3C. **XML Specification**: Página oficial da especificação do XML na W3C. [online] Disponível na Internet via WWW. URL: http://www.w3.org/TR/2004/REC-xml11-20040204/. Acessado em 20 de fevereiro de 2006.

Livros Grátis

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

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

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