Universidade Federal do Rio de Janeiro
Instituto de Matemática
Núcleo de Computação Eletrônica

Maria Cristina Ferreira Gomes

Certificação da Utilização de Padrões de Projeto no Desenvolvimento Orientado a Modelos

Rio de Janeiro
2005

Maria Cristina Ferreira Gomes

Certificação da Utilização de Padrões de Projeto no Desenvolvimento Orientado a Modelos

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática do Instituto de Matemática e Núcleo de Computação Eletrônica (IM/NCE) - Universidade Federal do Rio de Janeiro (UFRJ), como parte dos requisitos necessários à obtenção do título de Mestre

Orientador: Maria Luiza Machado Campos

Co-orientador: Paulo de Figueiredo Pires

Rio de Janeiro
2005

FICHA CATALOGRÁFICA

G633 Gomes, Maria Cristina Ferreira Gomes.
Certificação da utilização de padrões projeto no desenvolvimento orientado a modelos / Maria Cristina Ferreira Gomes. – Rio de Janeiro,
2005.
148 f.: il.

Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro, Instituto de Matemática, Núcleo de Computação Eletrônica, 2005.

Orientadora: Maria Luiza Machado Campos
Co-orientador: Paulo de Figueiredo Pires

1. MDA (Model Driven Architecture) – Teses. 2. Padrões de Projeto – Teses. 3. Certificação de Componentes – Teses. 4. J2EE (Java 2 Enterprise Edition) – Teses. 5. MDR (Metadata Repository) 6. JMI (Java Metadata Interface) I. Maria Luiza Machado Campos (Orient.). II. Paulo de Figueiredo (Co-orient.) III. Universidade Federal do Rio de Janeiro. Instituto de Matemática. Núcleo de Computação Eletrônica. III. Título.

Maria Cristina Ferreira Gomes

Certificação da Utilização de Padrões de Projeto no Desenvolvimento Orientado a Modelos

Dissertação de mestrado submetida ao corpo docente do Instituto de Matemática e Núcleo de Computação Eletrônica (IM/NCE) – Universidade Federal do Rio de Janeiro – UFRJ, como parte dos requisitos necessários à obtenção do grau de Mestre.

Rio de Janeiro, 27 de setembro de 2005

_____ Orientador
Profa. Maria Luiza Machado Campos – PhD

_____Co-orientador
Prof. Paulo de Figueiredo Pires – D.Sc.

_____
Profa. Maria Claudia Reis Cavalcanti – D.Sc.

_____
Profa Fernanda Araújo Baião Amorim – D.Sc.

DEDICATÓRIA

Dedico este trabalho aos meus pais, que sempre lutaram com dificuldades para dar o melhor possível para todos os seus filhos, que desde o início, ainda na alfabetização, me incentivaram a procurar sempre meu aprimoramento intelectual e se sentiam muito orgulhosos a cada vitória que eu conquistava.

AGRADECIMENTOS

Agradeço aos meus amigos da PETROBRAS: ao Márcio pelas boas indicações de fontes de referência; ao Evandro pela ajuda na implementação da interface do sistema e a Patrícia e o Marcelo pelo apoio e compreensão nesses últimos momentos da minha jornada.

A minha família, meu marido e meus filhos, por terem suportado o transtorno que eu causei em suas vidas por conta de alcançar o meu objetivo.

Aos meus orientadores, Maria Luiza e Paulo, o meu muito especial e sincero agradecimento, por terem acreditado e me dado todas as chances de chegar ao final apesar de todas as dificuldades que eu tive que superar, pelo seu excelente apoio técnico e pessoal e pela boa vontade em estar sempre disponível para todas as dúvidas.

RESUMO

GOMES, M. Cristina F. **Certificação da Utilização de Padrões de Projeto no Desenvolvimento Orientado a Modelos**. Rio de Janeiro, 2005. Dissertação (Mestrado em Informática)--Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2005.

O modelo de desenvolvimento baseado em componentes (DBC) surgiu como uma proposta que pretendia superar alguns desafios da engenharia de software. Esses desafios estão relacionados com a questão da reutilização organizada e eficiente e a produtividade necessária para o atendimento satisfatório das demandas das organizações. A *Model Driven Architecture* (MDA) é uma proposta da OMG, que define uma abordagem para o processo de desenvolvimento de software baseado na construção de modelos em alto nível de abstração e independentes de plataforma. Podemos simplificar o processo de desenvolvimento do modelo DBC unificando a sua abordagem com a proposta da MDA, resultando no aumento da automatização de várias tarefas dentro do ciclo de vida do desenvolvimento de componentes. A pesquisa desse trabalho se propõe a apresentar um mecanismo que automatiza a verificação da utilização correta de padrões de projeto em todos os níveis de abstração do desenvolvimento de componentes de negócio no ambiente orientado a modelos da MDA. A solução está baseada na representação de padrões através de perfis específicos da *Unified Modelling Language* (UML), na utilização de um repositório de padrões compatível com *Meta Object Facility* (MOF), no acesso e manutenção dos padrões no repositório e na manipulação e comparação desses padrões através do mapeamento do MOF para a linguagem Java, utilizando *Java Metadata Interface* (JMI).

ABSTRACT

GOMES, M. Cristina F. **Certificação da Utilização de Padrões de Projeto no Desenvolvimento Orientado a Modelos**. Rio de Janeiro, 2005. Dissertação (Mestrado em Informática)—Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2005.

The *Component Based Development* (CBD) model came up with a proposal that intended to overcome some software engineering's challenges. These challenges are related with an organized and efficient reuse and the necessary productivity for satisfactory attendance of organizations demands. The *Model Driven Architecture* (MDA) is an OMG proposal that defines an approach for the software development process based on high level abstraction and platform independent models construction. We can simplify the CBD process unifying its approach with the MDA proposal, resulting in the automation increase of some tasks inside the components development life cycle. This research presents a mechanism for verification of the correct use of design patterns at all abstraction levels of business components development in MDA model driven environment. The solution is based on patterns representation through specific *Unified Modelling Language* (UML) profiles, on the use of *Meta Object Facility* (MOF) compliant patterns repository and patterns manipulation and comparison through the MOF Java mapping *Java Metadata Interface* (JMI).

LISTA DE SIGLAS

API – *Application Program Interface*
ASP – *Application Server Pages*
AST – *abstract syntax tree*
BNF – *Backus Naur Form*
CASE - *Computer Aided Software Engineering*
COM – *Component Object Model*
CORBA – *Common Object Request Broker Architecture*
COTS - *Comercial Off-the-shelf*
CWM – *Common Warehouse Metamodel*
DBC – Desenvolvimento Baseado em Componentes
DCOM – *Distributed Component Object Model*
DOM – *Document Object Model*
DTD – *Document Type Definition*
EAI – *Enterprise Application Integration*
EJB – *Enterprise Java Beans*
ESBC – Engenharia de Software Baseada em Componentes
GoF – *Gang of Four*
GoV - *Gang of Five*
HTML – *HyperText Markup Language*
HTTP – *Hypertext Transfer Protocol*
IDE – *Interactive Development Environment*
IDL – *Interface Definition Language*
J2EE – *Java 2 Enterprise Edition*
J2SE – *Java 2 Standard Edition*
JCP – *Java Community Process*
JDBC – *Java database connectivity*
JDM API – *Java DataMining API*
JMI – *Java Metadata Interface*
JMS – *Java message services*
JNDI – *Java naming and directory interface*
JOLAP – *Java OLAP Interface*
JSF – *Java Server Faces*
JSP – *Java Server Pages*
JVM – *Java Virtual Machine*
JVM – *Java Virtual Machine*
MDA – *Model Driven Architecture*
MOF – *Meta Object Facility*
MSIL – *Microsoft Intermediate Language*
OCL - *Object Constraint Language*
OLAP – *Online Analytical Process*
OMG – *Object Management Group*
PIM – *Platform Independent Model*
POSA – Pattern-Oriented Software Architecture
PSM – *Platform Specific Model*
QVT - *Query, Views, and Transformations*
RMI – *Remote Method Invocation*
RM-ODP – *Reference Model for Open Distributed Processing*
RUP – *Rational Unified Process*

SCJ - *Sun Java Center*
SCL – *Software Certification Laboratories*
SOAP – *Simple Object Access Protocol*
SQA - *Software Quality Assurance*
UDDI – *Universal Description, Discovery and Integration*
UML – *Unified Modelling Language*
W3C – World Wide Web Consortium
XLINK – *XML Linking Language*
XMI – *XML Metadata Interchange*
XML – *Extensible Markup Language*
XPATH – *XML Path Language*
XQUERY – *XML Query Language*
XSLT – *Extensible Stylesheet Language Transformation*

LISTA DE ILUSTRAÇÕES

SUMÁRIO

CAPÍTULO 1

Introdução

1.1. Motivação

O crescimento do interesse no desenvolvimento de aplicações através da composição planejada de componentes de software pré-existentes, hoje freqüentemente chamado de Desenvolvimento Baseado em Componentes (DBC) ou Engenharia de Software Baseada em Componentes (ESBC), é o resultado da evolução de um investimento constante e que vem de longo tempo, de se propor métodos de trabalho para o desenvolvimento de software buscando o aumento cada vez maior de produtividade e integração (BROWN, 2000). A evolução dos níveis de amadurecimento em direção ao modelo DBC pode ser ilustrado na figura 1.1.

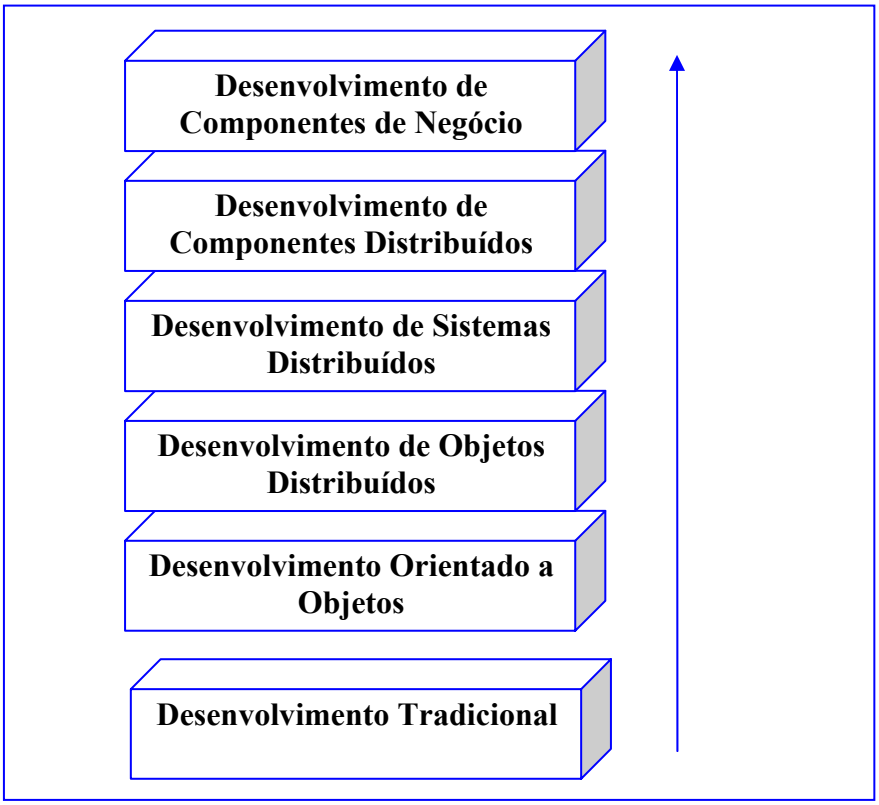


Figura 1.1: A Evolução para o modelo DBC nas Organizações. Adaptado de (HERZUM, SIMS, 1999).

Essa evolução não se apresenta de maneira uniforme, existindo organizações que ainda se encontram no estágio de desenvolvimento tradicional, e muito poucas no estágio do desenvolvimento de componentes de negócio.

Em correspondência com a evolução dos níveis de amadurecimento em direção ao modelo DBC, na indústria de software ocorria uma evolução relacionada com o aumento do nível de abstração no desenvolvimento de software. A história da engenharia de software está relacionada com o aumento do nível de abstração. Linguagens de programação como FORTRAN surgiram e tornaram a tradução de fórmulas uma realidade. Padrões para COBOL e C permitiram a portabilidade entre plataformas de hardware. Nos anos setenta surgiram as linguagens de programação estruturada e vários níveis de modulação começaram a serem aplicados. A teoria e a prática da orientação a objetos evoluiu por um período aproximado de dez anos, tendo seu início na década de oitenta, quando surgiram as primeiras linguagens com essa abordagem, como C++, Smalltalk e por último Eiffel. Na década de noventa a indústria de software introduziu tecnologias de objetos distribuídos como CORBA e surgiu a linguagem Java. Recentemente, a partir do final da década de noventa, surgiram tecnologias específicas para o modelo DBC, como CORBA 3.0 e EJB (HERZUM, SIMS, 1999).

Um novo nível de abstração surgiu recentemente, com a abordagem de desenvolvimento orientado a modelos, onde construímos modelos independentes de plataforma de software, que possibilita também o aumento do nível de reutilização (MELLOR, 2004). A figura 1.2 ilustra a evolução desses níveis.

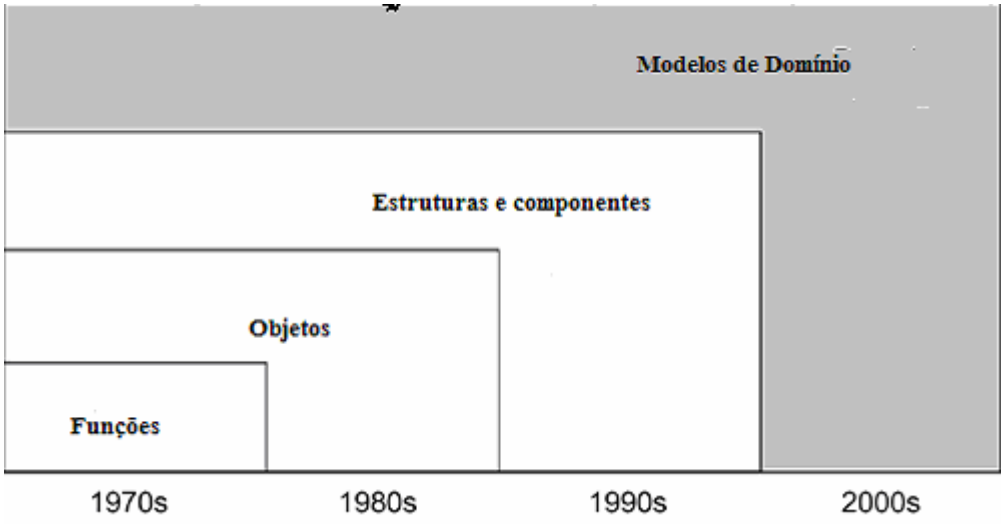


Figura 1.2: A Evolução do nível de reutilização. Adaptado de (MELLOR, 2004).

A *Model Driven Architecture* (MDA) é uma abordagem do OMG, que enfatiza a utilização dos modelos no desenvolvimento de sistemas de software. É orientada a modelos porque fornece um meio de utilizá-los para direcionar todo o ciclo de desenvolvimento (entendimento dos requisitos, projeto, construção, entrega, operação e manutenção).

As características da abordagem da MDA, detalhadas no capítulo 2, nos apresentam diversas diretrizes e mecanismos que podem fornecer um apoio bastante relevante na solução de diversos obstáculos encontrados pelo modelo DBC. GRISS (2001) apresenta uma classificação desses obstáculos quanto ao caráter de negócio, de processo, de organização, de tecnologia e de infra-estrutura. Esse apoio pode ser verificado principalmente nos processos de desenvolvimento, nas atividades de reutilização, no desenvolvimento de componentes corporativos, na utilização de técnicas e ferramentas adequadas, na utilização de padrões e arquiteturas e na convivência com uma grande diversidade de ambientes tecnológicos.

Os componentes de software são especificados com responsabilidades de execução de serviços de acordo com a arquitetura tecnológica utilizada. A figura 1.3 ilustra essas responsabilidades dentro da arquitetura em camadas. Esses diversos componentes se inter-relacionam, e para que isso aconteça de forma eficiente e coesa, é necessário seguir as

melhores práticas já testadas e consolidadas, o que significa a utilização de padrões adequados com a arquitetura e plataforma tecnológica utilizada.

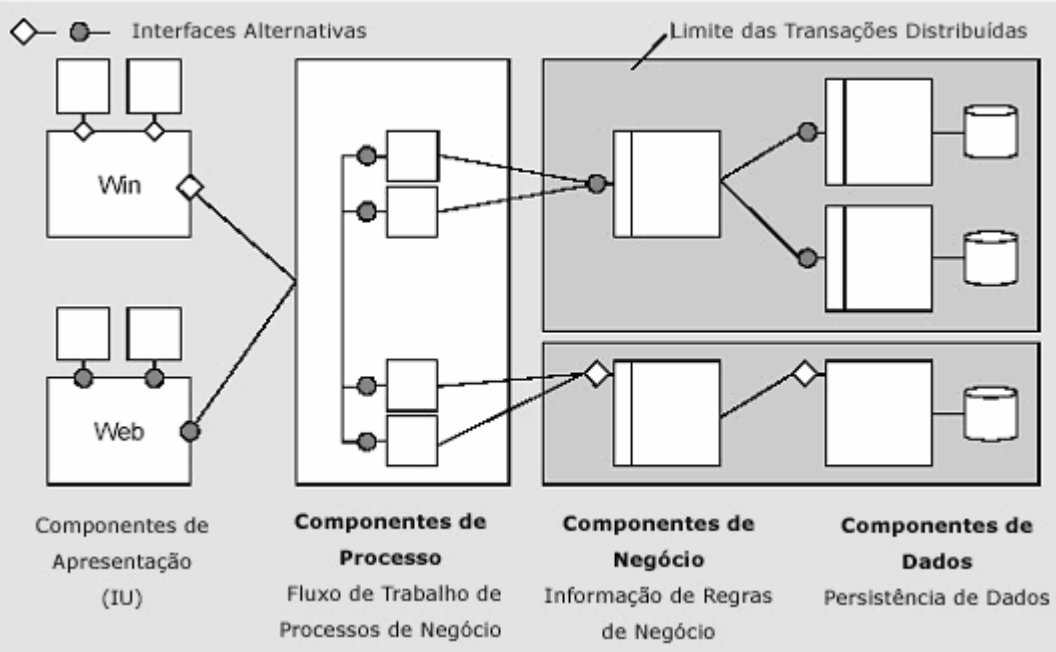


Figura 1.3: Configuração básica da arquitetura em camadas. Adaptado de (CBDi, 1999).

Conforme declaração do *Object Mangement Group* (OMG)[1] os padrões têm um papel fundamental nos procesos de desenvolvimento baseados na MDA. As transformações dos modelos, nos diversos níveis de abstração até a implementação do código, necessitam que esses modelos contenham detalhe suficiente para direcionar um aplicativo de software através de todo o processo. Esses detalhes podem ser incorporados através do uso de padrões, ao invés da elaboração manual das transformações desde o seu início, trazendo enorme ganho de produtividade e qualidade. A cada passagem de um nível de abstração para o outro, é necessario realizar a certificação da utilização de padrões no novo modelo resultante da transformação, antes de se prosseguir com as próximas transformações.

Podemos concluir que a certificação da utilização de padrões é um aspecto dentro do desenvolvimento de software que merece bastante atenção e destaque, tanto no modelo DBC

[1] **http://www.omg.org/mda/faq_mda.htm**

quanto no ambiente orientado e modelos e conseqüentemente num ambiente de desenvolvimento baseado em componentes orientado a modelos.

1.2. Objetivo

O objetivo deste trabalho é a apresentação de um processo de certificação da qualidade de um componente, a fim de verificar a utilização correta de padrões de projeto utilizados em todas as fases do ciclo de desenvolvimento. A abordagem proposta cobre tanto os padrões de projeto independentes de plataforma utilizados nas fases de requisitos e especificação, quanto os padrões dependentes de plataformas compatíveis com a tecnologia de componentes utilizada, adequados à sua respectiva camada na arquitetura da aplicação. Na fase de implementação nosso processo é específico para a plataforma *Java 2 Enterprise Edition* (J2EE).

Nessa proposta unem-se os conceitos e recursos do padrão MDA com o potencial da plataforma J2EE no modelo de desenvolvimento baseado em componentes, utilizando *Java Metadata Interface* (JMI)[1], que fornece um mapeamento do *Meta Object Facility* (MOF) para a linguagem Java (DIRCKZE, BAISLEY, IYENGAR, 2002).

Dessa forma, a idéia é utilizar um repositório de padrões compatível com MOF, acessar e manter esses padrões no repositório, manipular e comparar os modelos de representação dos padrões do repositório, através do mapeamento do MOF para a linguagem Java (JMI), com os modelos dos componentes, certificando a conformidade do componente com os respectivos padrões, apresentando relatórios com diagnósticos e sugestões de correções apropriadas.

[1] Proposta do *Java Community Process* (JCP)

1.3. Estrutura

Esta dissertação está organizada da seguinte forma: no capítulo 2, é apresentada uma visão geral da abordagem da MDA, do desenvolvimento baseado em componentes, a influência da MDA no modelo DBC e o papel dos padrões de projeto nesse contexto. No capítulo 3, é discutida a evolução das estratégias de garantia de qualidade no desenvolvimento de software, evidenciando em especial o momento atual, em que se enfatiza a atividade de certificação de componentes dentro do modelo DBC. São apresentados trabalhos relacionados com certificação de componentes e com verificação da utilização de padrões de projeto, além de uma análise comparativa desses trabalhos segundo os aspectos referentes ao foco da pesquisa desta dissertação. No capítulo 4, são apresentados os requisitos do ambiente de certificação e as características do sistema computacional desenvolvido, incluindo a definição conceitual e a especificação do sistema. No capítulo 5, é apresentado um estudo de caso para ilustrar a utilização do sistema de certificação. Por fim, no capítulo 6 são apresentadas conclusões da pesquisa, suas contribuições e limitações, além de possíveis trabalhos futuros.

CAPÍTULO 2

MDA, Componentes e Utilização de Padrões de Projeto

Apesar da grande evolução no desenvolvimento de software, principalmente os grandes progressos conceituais e teóricos, na prática nós continuamos lidando ainda com diversos problemas, e o processo de desenvolvimento continua exigindo a realização de trabalho bastante intensivo. A cada nova tecnologia, é preciso refazer muita coisa. Sistemas não são construídos com uma única tecnologia e sempre precisam se comunicar com outros sistemas. Outro aspecto importante é a troca contínua dos requisitos. Essas questões em conjunto representam desafios que se tornaram constantes, nos ambientes de desenvolvimento como: produtividade; portabilidade; interoperabilidade, documentação atualizada e agilidade de manutenção.

O modelo de desenvolvimento baseado em componentes fornece conceitos e mecanismos, para gerenciar complexidade em desenvolvimento de software em larga escala, mas na prática, a maioria das equipes de desenvolvimento encontra dificuldades na sua utilização efetiva.

A abordagem do desenvolvimento orientado a modelos (*Model Driven Architecture* – MDA) surge como uma nova alternativa para solucionar, ou pelo menos melhorar muito os problemas apresentados, inclusive a simplificação do processo do desenvolvimento baseado em componentes.

Os padrões de projeto executam papel importante tanto no desenvolvimento baseado em componentes, quanto na MDA. O desenvolvimento de componentes baseado na utilização de padrões de projeto garante componentes com maior facilidade de manutenção, adaptação e integração com outros componentes. Na MDA, eles são fundamentais também na definição das regras de transformação dos modelos nos diversos níveis de abstração.

Neste capítulo, serão apresentadas as características da proposta da MDA, do desenvolvimento baseado em componentes, como as duas abordagens podem conviver conjuntamente e a utilização dos padrões de projeto dentro desse contexto.

2.1 A Abordagem da MDA

A MDA é uma estrutura (*framework*) de desenvolvimento de software definido pela OMG. A chave da MDA é a importância dos modelos no processo de desenvolvimento de software. Na MDA o processo é dirigido pela atividade de modelagem do sistema de software que está sendo desenvolvido.

A MDA define uma abordagem para tecnologia da informação que separa a especificação das funcionalidades do sistema, da especificação da implementação dessas funcionalidades em uma plataforma tecnológica específica. Para isso a MDA define uma arquitetura para modelagem que fornece um conjunto de diretrizes para especificações estruturadas baseadas em modelos (MDA, 2001) (MDA, 2003).

Modelos consistem de um conjunto de elementos que descrevem algo físico, abstrato ou uma realidade hipotética. Bons modelos servem como meio de comunicação de conhecimento, projetos e idéias. O conceito central da MDA é a criação de diferentes modelos em diferentes níveis de abstração e a ligação de todos esses modelos para formar uma implementação. Alguns desses modelos existirão independentes de plataforma enquanto outros serão para uma plataforma tecnológica específica (MELLOR, 2004).

2.1.1 O Processo de Desenvolvimento da MDA

O ciclo de vida de desenvolvimento da MDA, representado na figura 2.1, apesar de parecer semelhante ao ciclo de vida tradicional, produz outros artefatos. Os artefatos desenvolvidos são modelos formais, escritos em linguagem bem definida, que podem ser compreendidos por computadores (KLEPPE, WARMER, BAST, 2003).

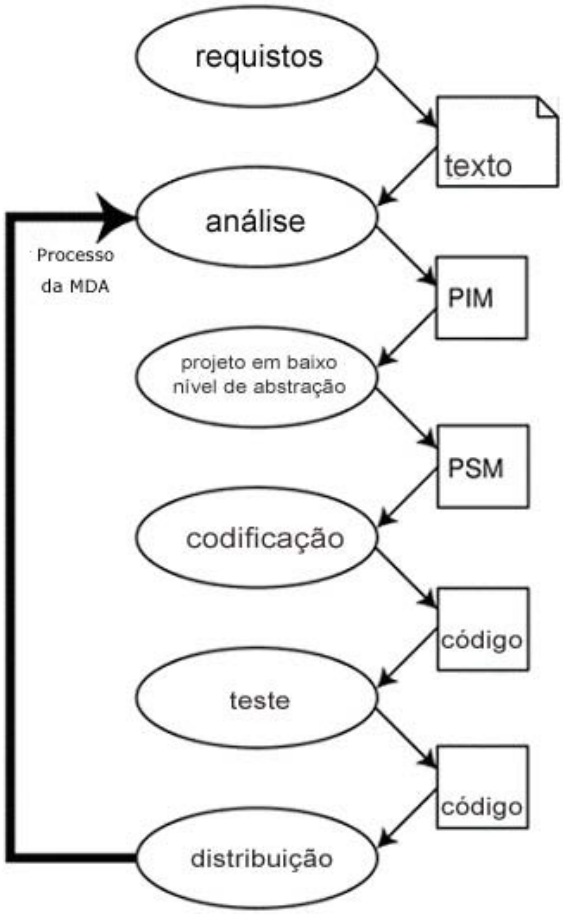


Figura 2.1: Ciclo de vida do desenvolvimento de software na MDA. Adaptado de (KLEPPE, WARMER, BAST, 2003).

Nos processos de desenvolvimento tradicionais, existe uma separação bem definida nos papéis dos modelos e das linguagens de programação, onde os modelos são apenas utilizados como artefatos de projeto. Na MDA os modelos fazem parte de forma direta do processo de produção, sendo transformados em artefatos de desenvolvimento (FRANKEL, 2003).

A MDA especifica um modelo chamado de modelo independente de computação (*computationally independent model* – CIM), que é um modelo de conceitos do domínio de negócios completamente independente de qualquer idéia de computabilidade (ARLOW e NEUSTADT, 2003), porém no processo de desenvolvimento de sistemas de software da MDA, três modelos constituem o seu núcleo. O primeiro define um modelo de alto nível de abstração e independente de qualquer implementação tecnológica. É chamado de modelo

independente de plataforma (*platform independent model* - PIM). Um PIM descreve um sistema que apóia alguma atividade de negócio. O segundo modelo é chamado de modelo de plataforma específica (*platform dependent model* – PSM). Um PSM é desenhado para especificar o sistema em termos de construtores de implementação que estão disponíveis numa plataforma específica de implementação. O terceiro é o modelo de código. Na MDA os códigos implementados também são considerados modelos.

A figura 2.2 ilustra as três principais etapas do processo. Na primeira etapa construimos um PIM. Na segunda etapa um PIM é transformado em um ou mais PSMs dependendo das plataformas tecnológicas desejadas. A etapa final do desenvolvimento é a transformação de cada PSM em código.

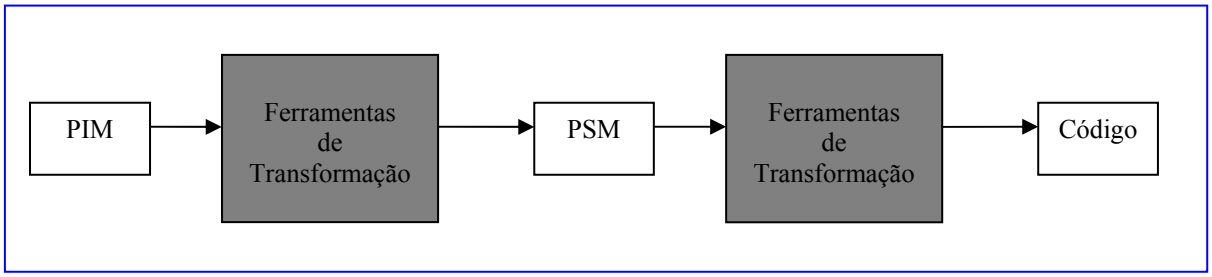


Figura 2.2: As três principais etapas no processo de desenvolvimento da MDA. Adaptado de (KLEPPE, WARMER, BAST, 2003).

Os benefícios da aplicação da MDA no processo podem ser identificados diretamente na solução dos principais problemas identificados no desenvolvimento de software (KLEPPE, WARMER, BAST, 2003):

- Produtividade: na MDA o foco do desenvolvedor esta na construção do PIM, que precisa conter todas as informações necessárias para que o PSM e o código possam ser gerados;
- Portabilidade: o mesmo PIM pode ser automaticamente transformado em vários PSMs para plataformas distintas;

- Interoperabilidade: quando necessitamos de que os diversos PSMs gerados por um PIM se relacionem, a MDA possui o recurso de ponte ilustrado na figura 2.3. Essas pontes podem ser geradas também automaticamente pelas informações constantes no PIM e pelas regras de transformação do PIM para os PSMs;
- Manutenção atualizada e documentação: como o foco do desenvolvimento fica no PIM, onde reside toda a informação necessária para a especificação do sistema, a documentação pode ser mantida atualizada todo o tempo e a manutenção também é bastante facilitada. Na MDA, a documentação é toda construída em alto nível de abstração;

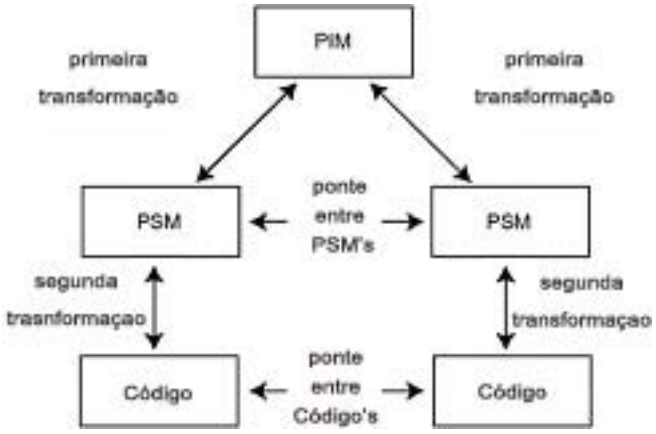


Figura 2.3: Interoperabilidade na MDA utilizando pontes. Adaptado de (KLEPPE, WARMER, BAST, 2003).

Os principais elementos da estrutura da MDA são modelos, PIMs, PSMs, linguagens, transformações, definições de transformações e mecanismos que executam transformações. Todos esses elementos atuam em conjunto na estrutura básica da MDA.

Do ponto de vista do desenvolvedor, o PIM e o PSM são os elementos mais importantes. O desenvolvimento tem seu foco na construção do PIM, descrevendo o sistema em alto nível de abstração. Na próxima etapa escolhemos um ou mais mecanismos que tenham condições de

executar a transformação do PIM, que foi desenvolvido de acordo com uma determinada definição de transformação. O resultado é um PSM que então pode ser transformado em código.

2.1.2 O Uso da Metamodelagem na MDA

No passado as linguagens eram freqüentemente definidas usando uma gramática em *Backus Naur Form* (BNF) (KLEPPE, WARMER, BAST, 2003), que descreve que séries de símbolos é uma expressão correta de uma linguagem. Esse método é apropriado e muito utilizado para linguagens baseadas em texto como linguagens de programação. No contexto da MDA, onde o produto é na maioria das vezes um artefato gráfico, nós necessitamos de um mecanismo diferente para definição de linguagens de modelagem. Esse mecanismo é chamado de metamodelagem (KLEPPE, WARMER, BAST, 2003).

Um modelo define que tipos de elemento (construtos) podem ser usados para construir um sistema. Uma linguagem também define que tipos de elemento podem existir e serem usados num modelo. De forma similar, nós também podemos definir uma linguagem por um modelo. O modelo da linguagem descreve que tipos de elementos podem ser usados para construir programas ou artefatos na linguagem.

Todo tipo de elemento que um arquiteto pode usar na construção de um modelo é definido pelo metamodelo da linguagem que está sendo usada. Um metamodelo é simplesmente um modelo cujas instâncias são tipos em outro modelo. Isso permite que outros modelos sejam capturados e manipulados. Um metamodelo bem definido e conhecido é a especificação da *Unified Modelling Language* (UML), que provê os elementos dos modelos das aplicações. Os metamodelos também podem ser capturados em metametamodelos (MELLOR, 2004). Na UML podemos usar, classes, atributos, associações, e outros tipos de elementos, porque no metamodelo da UML existem esses elementos definidos (UML, 2003).

Como um metamodelo também é um modelo, ele mesmo pode ser escrito numa linguagem bem definida. A linguagem é chamada de metalinguagem, então BNF é uma metalinguagem para as linguagens baseadas em texto (KLEPPE, WARMER, BAST, 2003).

Existem dois motivos principais para a metamodelagem ser tão importante na estrutura da MDA. Primeiro precisamos de um mecanismo para definir linguagens de modelagem, de forma que sejam definidas sem ambigüidade. Assim uma ferramenta de transformação poderá ler, escrever e entender os modelos. Na MDA linguagens são definidas através de metamodelos. Em segundo, as regras de transformação que constituem a definição de transformação, descrevem como um modelo em uma linguagem original pode ser transformado em um modelo numa linguagem destino. Essas regras usam os metamodelos das linguagens original e destino para definir as transformações (MELLOR, 2004).

A figura 2.4 apresenta uma descrição do metamodelo da MDA. Os modelos PIM**,** PSM e as técnicas de transformação são baseados em metamodelos, expressos com as tecnologias da OMG, que compõem o núcleo da MDA (MDA, 2001).

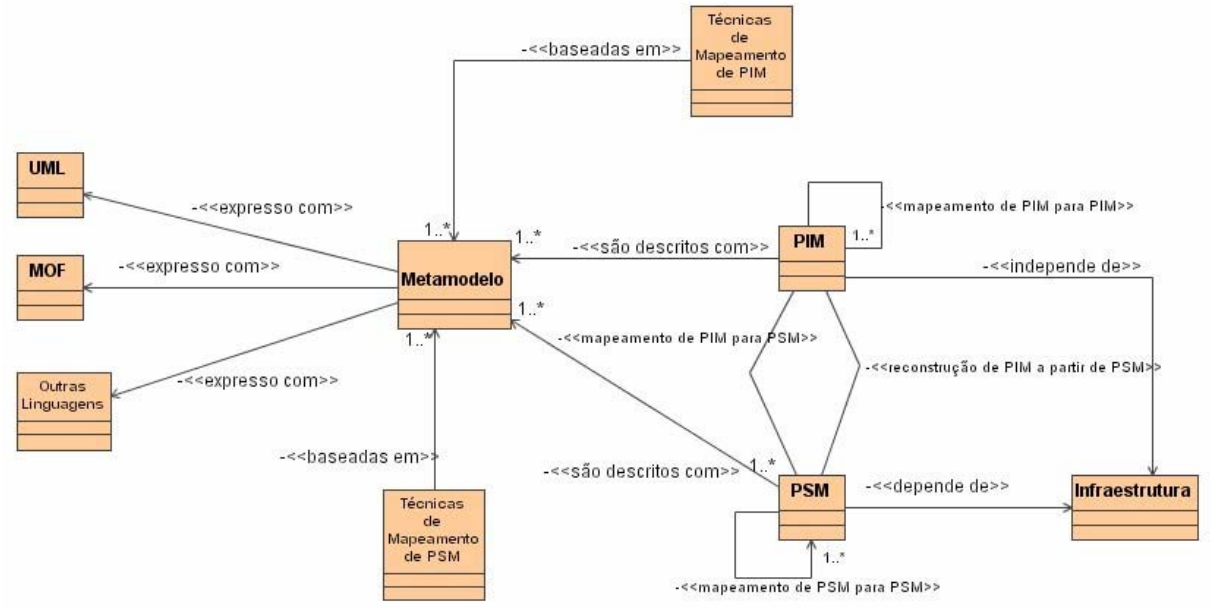


Figura 2.4: Descrição do Metamodelo da MDA. Extraído de (MDA, 2001)

2.1.3 Padrões da OMG e Tecnologias Adicionais

Para entendermos o relacionamento entre os vários padrões da OMG que possuem papéis dentro da estrutura da MDA, nós precisamos conhecer as camadas de metamodelagem onde estão definidos. A OMG usa a arquitetura em quatro camadas para seus padrões (MELLOR, 2004) (MDA, 2001). Na terminologia da OMG essas camadas são chamadas de M0, M1, M2 e M3 (figura 2.5).

Na camada M0 situam-se as instâncias por sobre as quais os sistemas executam. Essas instâncias podem existir de diversas formas, como um dado em um banco de dados ou como um objeto ativo executando num computador.

Na camada M1 situam-se os modelos, por exemplo, um modelo UML de um sistema. Nessa camada temos a definição das instâncias da camada M0.

Os modelos que existem na camada M2 são chamados de metamodelos. Cada modelo UML da camada M1 é uma instância do metamodelo da UML como definido em sua especificação. Quando construímos o modelo do modelo de um sistema executável, estamos construindo um metamodelo.

Da mesma maneira os modelos da camada M2 podem ser vistos como instâncias de um elemento da camada imediatamente superior, a camada M3 ou metametacamada.

Os conceitos que dão suporte a MDA podem ser aplicados sem o uso de padrões. Entretanto, para trazer produtividade no uso da MDA, é fundamental a utilização de um conjunto de padrões relacionados com a modelagem. Isso permite que a indústria desenvolva instrumentos que possam operar em conjunto com soluções.

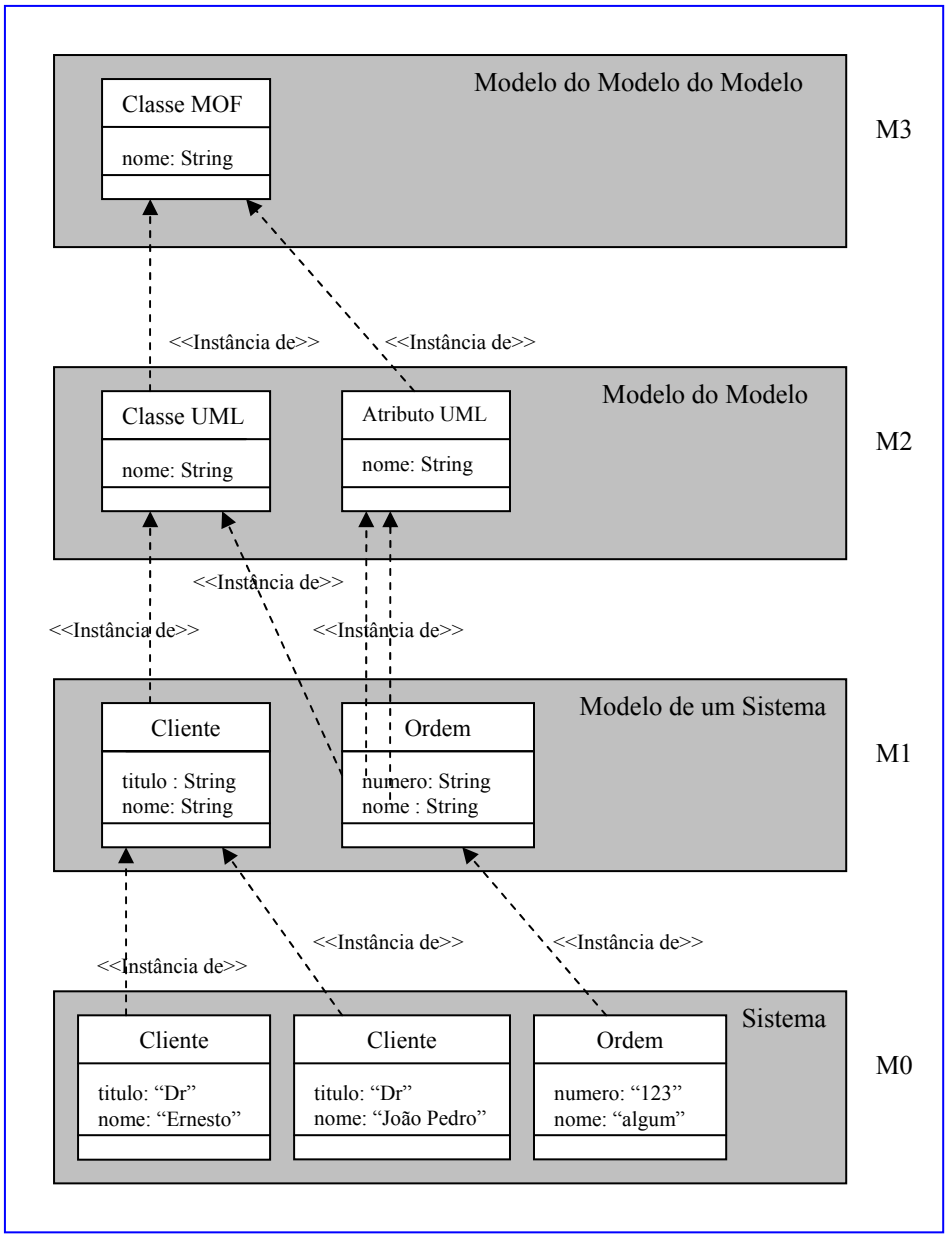


Figura 2.5: Visão geral das camadas M0 a M3. Extraído de (KLEPPE, WARMER, BAST, 2003).

O MOF é por definição, o metamodelo comum para especificação de metamodelos da OMG. É um exemplo de metametamodelo ou modelo de metamodelos (POOLE, 2001). É um padrão que define uma linguagem abstrata para definir linguagens de modelagem. O MOF reside na camada M3 da arquitetura de metadados. É a linguagem utilizada para escrever os metamodelos da UML, *Common Warehouse Metamodel* (CWM) e do próprio MOF.

A versão corrente é a MOF 1.4 (MOF, 2002). A base da UML 2.0 é o MOF 2.0, que está já esta sendo finalizado; já estando disponíveis versões preliminares do núcleo do MOF 2.0,

MOF 2.0 *Interface Definition Language* (IDL) *Mapping* e MOF 2.0 XML *Metadata Interchange* (XMI) *Specification*.

O MOF não é usado apenas para definir linguagens de modelagem, mas também para permitir a construção de ferramentas para definição de linguagens de modelagem. Logo, ele fornece algumas funcionalidades adicionais como:

- Interface de repositório MOF: a definição do MOF incluiu uma especificação da interface para um repositório MOF. Essa interface nos permite obter informações sobre modelos da camada M1 de um repositório baseado em MOF. Essa interface é definida usando *Common Object Request Broker Architecture* (CORBA)-IDL e pode ser usada em muitos ambientes. No ambiente Java existe uma interface nativa que fornece a mesma funcionalidade, que é chamada de JMI (JSR40, 2002). Para aplicações Java essa interface é mais fácil de ser utilizada do que o mapeamento do CORBA para Java;
- Intercâmbio de modelos: o MOF também é utilizado para definir um formato de intercâmbio para modelos da camada M1 baseado em arquivo. O MOF define uma maneira padrão de gerar um formato de intercâmbio para modelos construídos em linguagens definidas por metamodelos descritos em MOF. Esse formato é baseado em XML e é chamado de *XML Metadata Interchange* (XMI).

Aplicações desenvolvidas com base no padrão MOF, não necessitam ter qualquer conhecimento sobre as interfaces do domínio específico de alguma instância de modelo. As aplicações podem ler e atualizar esse modelo, usando operações genéricas com as interfaces reflexivas do MOF (POOLE, 2001).

A UML é o padrão de linguagem de modelagem na camada de nível M2. Para estar habilitado a usar a MDA no desenvolvimento de aplicações, é necessário conhecer UML, pois é a linguagem de modelagem mais utilizada definida em MOF.

A versão corrente é a UML 1.5 (UML, 2003). A especificação da UML 2.0 está em fase final de desenvolvimento, e está voltada para atender os requisitos da MDA. Uma versão preliminar está disponível em quatro partes: UML 2.0 *Infraestruture*; UML 2.0 *Superstruture*; UML 2.0 OCL e UML 2.0 *Diagram Interchange* (especificação que define um padrão de intercambio entre diferentes instrumentos de modelagem, permitindo exportar e importar além dos elementos dos modelos, a definição dos diagramas construídos);

O CWM (CWM, 2003), é uma linguagem de modelagem específica para representar aplicações de data warehousing e domínios de análise de negócios (POOLE, 2001). O metamodelo do CWM tem muita coisa em comum com o metamodelo da UML, mas tem um conjunto específico de metaclasses, por exemplo, para modelagem de banco de dados relacional. A parte comportamental do metamodelo da UML, como diagrama de estados ou colaboração, não existe no CWM.

Como já foi mencionado antes, XMI (XMI, 2003) é o padrão da OMG que mapeia o MOF para o padrão XML do *World Wide Web Consortium* (W3C). O XMI define como as etiquetas do XML são usadas para representar modelos em conformidade com o MOF numa forma serial. Metamodelos são convertidos em XML *Document Type Definitions* (DTD) e mais recentemente em XML *schemas* (esquemas) e modelos são convertidos em documentos XML consistentes com seu DTD ou esquema correspondente. O XMI permite que os metadados (etiquetas) e suas instâncias (conteúdo dos elementos), sejam empacotados juntos num mesmo documento, possibilitando que aplicações interpretem as instâncias através dos metadados associados (POOLE, 2001).

Um padrão novo ainda em desenvolvimento chamado de *Query, Views, and Transformations Standard* (QVT) (QVT, 2002) está direcionado para a maneira como são obtidas as transformações entre modelos cujas linguagens são definidas em MOF. QVT se tornará parte

do MOF e conterá uma linguagem para criação de visões em modelos, uma linguagem para consulta em modelos e uma linguagem para escrever definições de transformações.

Grande parte dos modelos não está suficientemente refinada para fornecer todos os aspectos relevantes de uma especificação. Existe a necessidade de descrever restrições adicionais sobre seus elementos. Essas restrições são freqüentemente descritas em linguagem natural, o que muitas vezes resulta em ambigüidades. Para eliminar essas ambigüidades e permitir que essas descrições sejam compreendidas e processadas por computadores, foram desenvolvidas linguagens chamadas de linguagens formais. As linguagens formais tradicionais exigem forte conhecimento matemático para serem utilizadas.

Object Constraint Language (OCL) (OCL, 2003) é uma linguagem de especificação com a qual podemos expressar regras sobre determinados elementos dos modelos (expressar restrições sobre atributos, expressar o corpo de operações de consulta, invariantes, pré e pós-condições, etc...). OCL pode ser usada em modelos MOF e UML e foi desenvolvida como uma alternativa de linguagem formal de fácil leitura e escrita, mais acessível para quem constrói modelos de sistemas ou negócios.

Uma expressão OCL não pode realizar nenhuma mudança nos modelos, é simplesmente avaliada e retorna um valor. OCL não é uma linguagem de programação e dessa forma, não é possível escrever uma lógica programada ou fluxos de controle em OCL.

Usando OCL, estendemos a capacidade de expressão dos modelos, e podemos criar modelos mais precisos, mais expressivos e mais amplos. OCL pode ser traduzida para linguagens de programação como Java. Além de acrescentar mais precisão aos modelos e definição de linguagens, OCL pode ser muito efetiva na definição de transformações (KLEPPE, WARMER, BAST, 2003).

A linguagem de programação Java fornece uma infra-estrutura, para aplicações distribuídas baseadas na Web e em componentes. Programas em Java são altamente portáveis, porque a

linguagem Java é interpretada e os programas são compilados em um fluxo de bytes que é processado pela *Java Virtual Machine* (JVM). Se a JVM estiver disponível para um determinado sistema operacional ou estiver embutida em um navegador, então os programas poderão executar em qualquer ambiente. A portabilidade do Java também é facilitada pela larga coleção de padrões, serviços opcionais e *Application Program Interfaces* (APIs), desenvolvidas pelo *Java Community Process* (JCP), uma organização aberta, no qual seus participantes contribuem para o desenvolvimento das especificações do Java. J2EE consiste em um conjunto de especificações que definem uma arquitetura multicamada completa baseada em componentes. Dentro do JCP estão sendo desenvolvidos vários projetos para definição de especificações, com o objetivo de mapear os padrões da OMG para a tecnologia Java (POOLE, 2001). Essas especificações são:

- JSR-40, ou JMI: fornece um mapeamento formal do MOF para a linguagem Java, está na sua 1ª versão final (JSR40, 2002). A implementação do JMI permite uma geração programática de pura interface Java e acesso baseado em XMI em repositórios baseados em meta-modelos MOF e suas instâncias;
- JSR-69**,** ou *Java OLAP Interface* (JOLAP): é uma API desenvolvida com puro Java para servidores OLAP e aplicações distribuídas no ambiente J2EE**.** JOLAP usa o meta-modelo OLAP do CWM para descrever metadados OLAP (JSR069, 2003);
- JSR-73, ou *Java DataMining* API (JDM API): é uma API, desenvolvida com puro Java para aplicações de BI (*business intelligence*) que empregam técnicas de mineração de dados para descoberta e análise de conhecimento. JDM é similar ao JOLAP, no sentido de que representa realizações Java do meta-modelo *Data Mining* do CWM, enquanto JOLAP representa a realização Java do meta-modelo OLAP do CWM (JSR073, 2005).

O diagrama da figura 2.6 ilustra o relacionamento entre os padrões da OMG na MDA e os padrões correspondentes da plataforma J2EE.

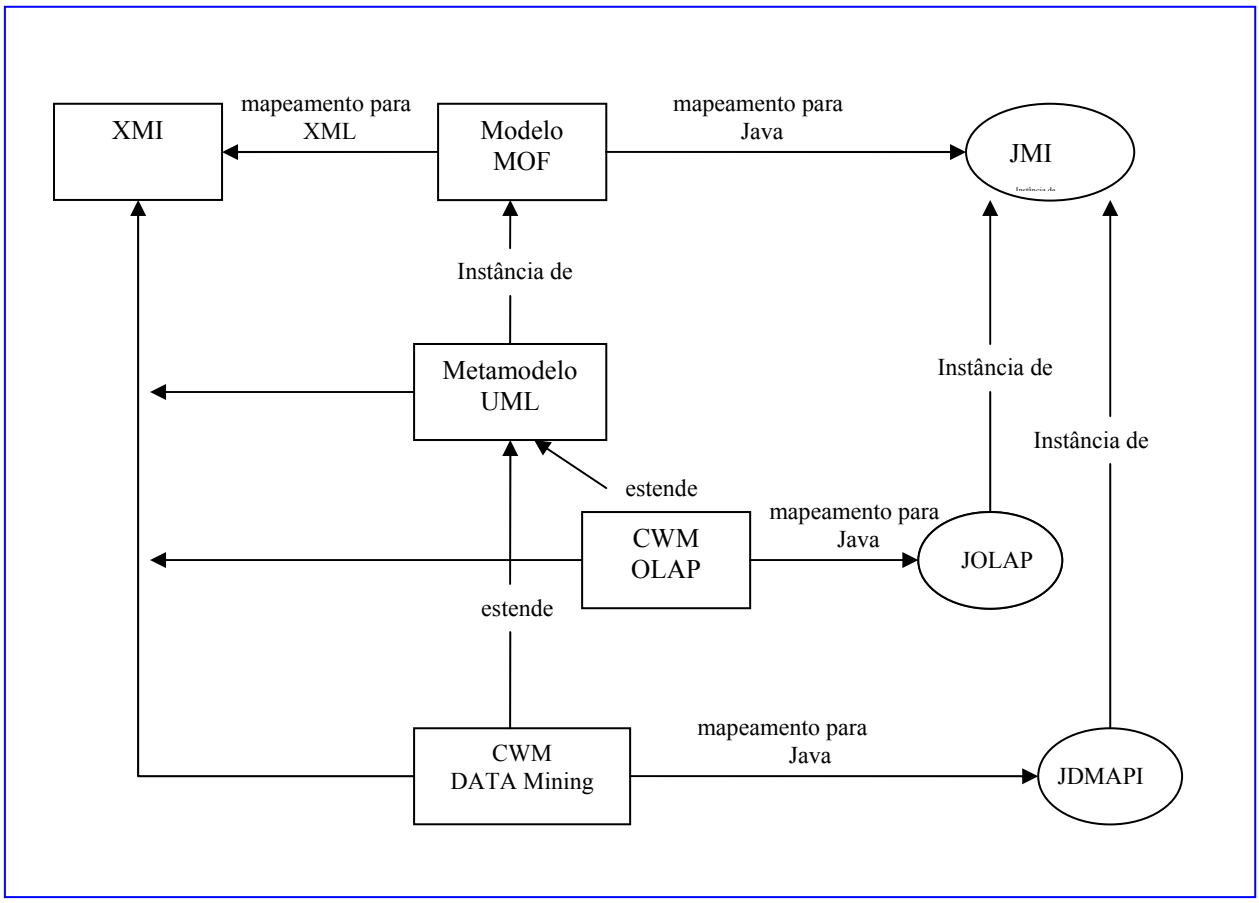


Figura 2.6: Relacionamento entre padrões. Adaptado de (POOLE , 2001).

2.1.4 A Definição de Linguagens de Modelagem

No desenvolvimento de sistemas de software, em algumas situações pode surgir a necessidade de se definir uma linguagem de modelagem, que atenda a determinados requisitos e características específicas. Para se definir uma linguagem de modelagem específica, podemos estender uma linguagem de modelagem existente (um metamodelo existente, como por exemplo, o metamodelo da UML) ou criar uma nova (um novo metamodelo).

Existem diferenças entre os conceitos de estender linguagens de modelagem existentes utilizando MOF ou perfis UML específicos.

O conceito de perfil é um mecanismo de especialização definido como parte da UML. Um perfil define uma forma específica de usar a UML. Por exemplo, o perfil CORBA da UML define uma forma específica de usar a UML para modelar interfaces CORBA, e o perfil Java da UML define uma forma de modelar código fonte Java em UML (KLEPPE, WARMER, BAST, 2003). Dessa forma a UML deixa de ser uma única linguagem, mas uma base para uma família de linguagens baseadas em UML. A MDA se utiliza muito dos mecanismos de extensão da UML, porque necessita dar suporte para diferentes aspectos e níveis de abstração (FRANKEL, 2003), e um perfil define uma nova linguagem simplesmente reutilizando o metamodelo da UML (KLEPPE, WARMER, BAST, 2003).

Um perfil é definido por um conjunto de estereótipos, um conjunto de restrições em OCL relacionadas e um conjunto de valores etiquetados (*tagged values*).

O efeito de um perfil é a definição de uma variação especializada, estendendo a UML para um propósito específico. O metamodelo da UML é definido através do MOF, e pode então ser estendido também através do MOF. As extensões UML que utilizam o poder do MOF são algumas vezes chamadas de extensões peso pesado (*heavyweight extensions*).

Quando um arquiteto se depara com a necessidade de estender a UML, precisa decidir se o fará através de um novo perfil, ou se pela extensão através do MOF. A principal vantagem da abordagem do perfil UML é que os modelos criados utilizando extensões definidas por um perfil podem ser construídos com qualquer editor de modelos UML, que seja configurado para utilizar as extensões específicas do perfil que será utilizado. A principal desvantagem é a restrição imposta ao arquiteto da extensão de não utilizar a força semântica que os mecanismos de modelagem do MOF oferecem. No caso de extensões complexas, as restrições da utilização de perfil podem se tornar bastante sérias (FRANKEL, 2003).

A MDA não exige que se use a UML como linguagem de modelagem, outras linguagens podem ser usadas, desde que um metamodelo MOF seja fornecido para dar suporte a cada

uma dessas linguagens. Alguns tipos de modelos são fundamentalmente diferentes de um modelo UML, porque suas construções de modelagem não se ajustam facilmente nos paradigmas de modelagem da UML. Nesses casos não é aconselhável tentar estender o metamodelo da UML e sim criar um novo metamodelo MOF para definir a sintaxe abstrata dessas construções de modelagem. Em algumas situações temos as duas estratégias: (FRANKEL, 2003). Por exemplo:

- Perfil UML para EJB: a OMG adotou um metamodelo MOF para Java e EJB para complementar o perfil UML utilizado pelo JCP;
- UML para *Enterprise Application Integration* (EAI): essa especificação atualmente define um metamodelo MOF e também um perfil UML para modelagem de integração de aplicações;
- Perfil UML para CORBA: A OMG definiu um metamodelo MOF e um perfil UML para CORBA.

Como já foi mencionada a principal motivação de se utilizar perfil UML, é criar linguagens especializadas que qualquer editor de modelos UML pode manipular, entretanto sua utilização não é recomendada em extensões complexas. Os novos mecanismos criados especificamente para a MDA que deverão surgir, prometem fornecer mais liberdade e eliminar as diferenças entre metamodelos e perfis (FRANKEL, 2003).

2.1.5 As Funcionalidades Necessárias no ambiente da MDA

Embora os mecanismos de transformação sejam o núcleo do ambiente de desenvolvimento da MDA, eles não são as únicas funcionalidades necessárias. Além da funcionalidade fornecida por esses mecanismos de transformação, outras funcionalidades são relevantes. As funcionalidades necessárias num ambiente completo são representadas na figura 2.7 e descritas a seguir:

- Editor de código – *Interactive Development Environment* (IDE): funções comuns que são fornecidas por um ambiente de desenvolvimento interativo, por exemplo, execução passo, a passo, para procura de erros, compilação, edição de código, e etc...
- Arquivos de código: embora possamos considerar código como um modelo, usualmente ele está na forma de arquivos baseados em texto. Arquivo baseado em texto não é um formato que outras ferramentas sejam capazes de entender. Dessa forma precisamos dos dois itens a seguir:
  - Analisador de código: um analisador sintático lê um arquivo de código baseado em texto e o armazena no repositório de modelos no formato baseado em modelo que outras ferramentas possam usar;
  - Gerador de código: lê o modelo no repositório e produz arquivo de código baseado em texto;
- Repositório de modelos: o "banco de dados" de modelos onde os modelos são armazenados e podem ser consultados usando XMI, JMI ou IDL;
- Editor de modelos - ferramenta de modelagem: editor de modelos onde os modelos podem ser construídos e modificados;
- Validador de Modelos: modelos usados para geração de outros modelos precisam ser extremamente bem definidos. Um validador pode verificar modelos contra um conjunto de regras pré-definidas para garantir que o modelo está apropriado para uso numa transformação;
- Editor de definição de transformação: onde as definições de transformação podem ser construídas e modificadas;
- Repositório de definição de transformação: armazenamento para as definições de transformação;

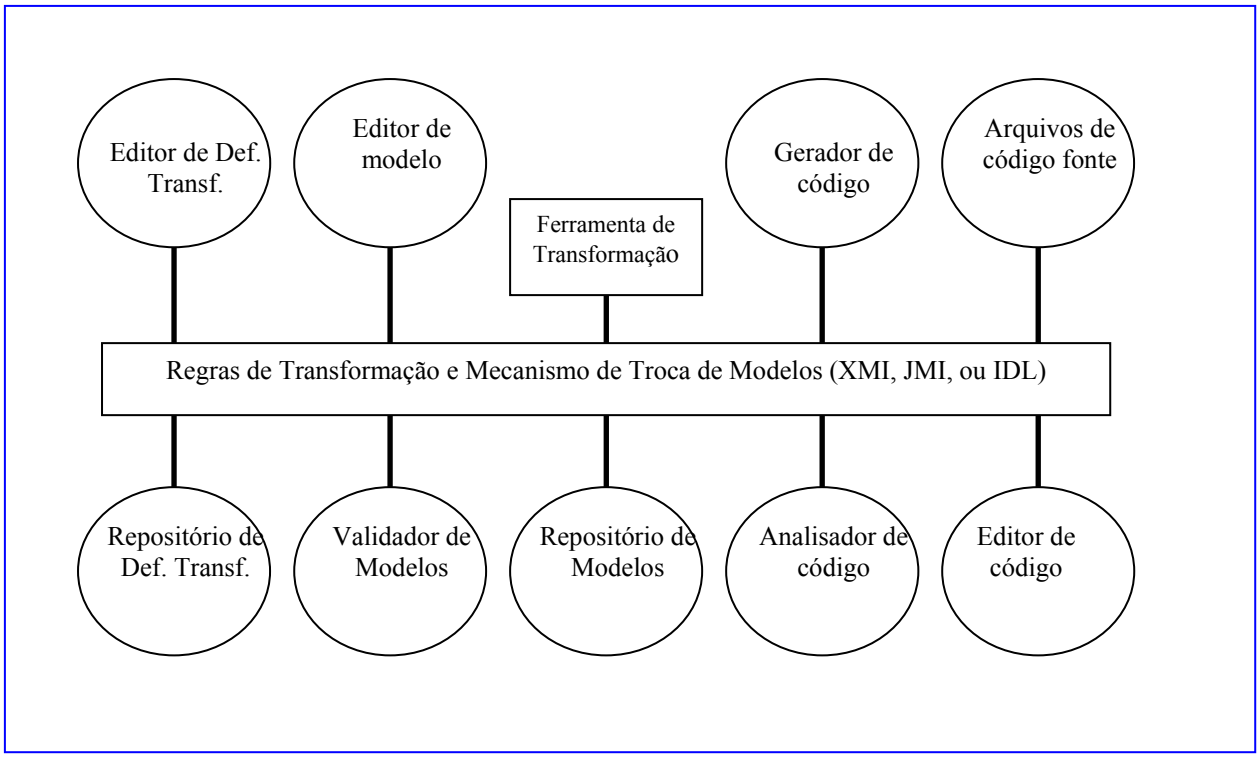


Figura 2.7: Funcionalidades no ambiente de desenvolvimento da MDA. Adaptado de (MELLOR, 2004).

2.2 O Modelo de Desenvolvimento Baseado em Componentes (DBC)

É essencial compreender os conceitos e idéias em que são baseadas as implementações de soluções no modelo DBC.

O objetivo de construir sistemas através de partes bem definidas não é novo. O interesse no modelo DBC vem das metodologias de sistemas modulares, projeto estruturado e, mais recentemente dos sistemas orientados a objetos. O modelo DBC estende essas metodologias, enfatizando o projeto de soluções em termos de pedaços de funcionalidades fornecidas como componentes. Para desenvolvedores e usuários de sistemas de software intensivos, o modelo DBC é visto como uma forma de reduzir custos de desenvolvimento, aumentar a produtividade e fornecer o controle de atualização de sistemas em face da rápida evolução da tecnologia (BROWN, 2000).

A chave para compreender o modelo DBC, é aprofundar o entendimento do que é um componente e como os componentes se tornam os blocos básicos de construção de uma solução. A definição de componente é a base do que se pode obter usando o modelo DBC, e

definem a diferença entre o modelo DBC e outras abordagens orientadas a reutilização. Para o modelo DBC, componente é muito mais que uma sub-rotina da programação estruturada, um objeto ou classe dos sistemas orientados a objetos, ou mesmo um pacote em um módulo de sistema. Um componente é usado como a base para o projeto, implementação e a manutenção de sistemas baseados em componentes (BROWN, 2000). Existem diversas definições do que é um componente, mas HERZUM e SIMS (1999) descreveram as principais características do modelo de componente de software como:

- Um componente é uma construção de software auto contida que tem um uso definido, tem uma interface gerada em tempo de execução, pode ser distribuído autonomamente e é construído com o conhecimento prévio de um soquete específico;
- Um soquete de componente é um software que fornece uma interface bem definida e bem conhecida, em tempo de execução, para uma infra-estrutura de apoio onde o componente será ajustado;
- Um componente é construído para composição e colaboração com outros componentes.

Também segundo HERZUM e SIMS (1999), o desenvolvimento baseado em componentes é uma abordagem onde todos os aspectos e fases do ciclo de desenvolvimento, incluindo análise de requisitos, arquitetura, projeto, construção, teste, distribuição, infra-estrutura de suporte técnico e a gerência de projetos são baseados em componentes.

Para CRNKOVIC e LARSSON (2002) os principais objetivos do modelo DBC são:

- Fornecer suporte no desenvolvimento de sistemas como uma composição de componentes;
- Dar suporte no desenvolvimento de sistemas como entidades reutilizáveis;

- Facilitar a manutenção e atualização de sistemas pela adaptação e substituição de seus componentes.

E apesar do modelo DBC ter surgido como uma alternativa para solucionar diversos problemas relacionados com a Engenharia de Software, CRNKOVIC e LARSSON (2002) consideram que ele ainda está enfrentando muitos desafios. A seguir são apresentados alguns dos principais:

- Ciclo de vida baseado em componentes: o ciclo de vida de um sistema baseado em componentes é mais complexo, pois existem fases separadas em atividades sem sincronismo. O desenvolvimento de um componente pode ser totalmente independente do desenvolvimento de um sistema que use esse componente;
- Componentes confiáveis e certificação de componentes: com a construção de sistemas através da composição de componentes pré-existentes, torna-se imprescindível a criação da atividade de certificação de componentes para garantir a utilização de componentes confiáveis;
- Gerência de configuração de componentes: sistemas complexos podem incluir muitos componentes, que por sua vez incluem outros componentes. Com muita freqüência temos que manipular estruturas complexas de componentes, tornando difícil o controle de instâncias e versões dos componentes utilizados nas composições das aplicações;
- Mecanismos e instrumentos de apoio: as soluções fornecidas pela Engenharia de Software necessitam de instrumentos de apoio apropriados. O modelo DBC requer diversos tipos de instrumentos: ferramentas de seleção e avaliação de componentes; repositório de componentes; instrumentos de certificação de componentes, ferramentas de projeto baseado em componentes; instrumentos de

análise em tempo de execução, mecanismos de configuração de componentes, etc...

Existe uma variedade de processos de desenvolvimento de sistemas corporativos baseados em componentes. Os três mais divulgados são:

- *Rational Unified Process* (RUP): é um processo que cobre todo o ciclo de vida do desenvolvimento. Na arquitetura da solução, direciona na utilização da abordagem baseada em componentes e é fortemente influenciado pela notação da UML (RATIONAL, 2001);
- *Select Perspective*: uma abordagem de projeto baseada em componentes, apoiada pelo produto Select Component Manager. Baseia-se no fluxo de trabalho Consumo-Gerência-Fornecimento de componentes. Utiliza a UML como linguagem de especificação de componentes (APPERLY, *et al,* 2003);
- *Catalysis*: é uma abordagem orientada a componentes originada da comunidade da análise orientada a objetos. É descrita como uma nova geração de metodologia para modelagem e construção de sistemas abertos a partir de componentes e estruturas *(frameworks),* formada por padrões emergentes como a UML, OMG e *Reference Model for Open Distributed Processing* (RM-ODP) (D´SOUSA, WILLS, 1998);

Outros processos incluem: UML Components (CHEESMAN, DANIELS, 2000); CADA (BOERTIEN, STEEN, JONKERS, 2001) (JONKERS, *et al,* 2000); COMO (LEE, *et al,* 1999); COMET (SOLBERG, BERRE, 1997); Odyssey DE (BRAGA, WERNER, MATTOSO, 1999); C&C Scheme (VITHARANA, ZAHEDI, JAIN, 2003);

As abordagens desses processos de desenvolvimento diferem em diversos detalhes, mas as características essenciais são:

- Compreensão: identificação dos componentes existentes no domínio e estabelecimento das suas especificações;
- Fornecimento: entrega da implementação física do componente, que pode ser obtido de diversas origens, como: compra de componentes; assinatura de componentes; modificação de um componente existente, sistemas legados encapsulados como componentes; desenvolvimento de um novo componente;
- Composição: composição dos componentes em aplicações e entrega da solução;
- Implementação: execução e distribuição dos componentes em aplicações;
- Gerência: gerência do processo e do inventário de componentes.

Uma forma específica de se aplicar o modelo DBC é a abordagem de componentes de negócio, onde o desenvolvimento de sistemas começa e termina com o conceito de componentes de negócio. O núcleo da estrutura dessa abordagem é o conceito de granularidade de componente. HERZUM e SIMS (1999) definem as três granularidades que compõem essa estrutura.

- Componente distribuído: é considerado o componente de menor granularidade, e é a forma usual do conceito de componente na indústria. Pode ser implementado como um componente EJB, um componente CORBA ou um componente com a tecnologia *component object model* (COM) +;
- Componente de negócio: é um componente que implementa um único e autônomo conceito de negócio. Usualmente consiste de um ou mais componentes distribuídos, que em conjunto endereçam vários aspectos de distribuição requeridos pelo componente de negócio. Isto é, um componente de negócio é um único artefato que pode fisicamente ser distribuído, através de duas ou mais máquinas. Componente de negócio é o principal conceito em torno do qual toda a abordagem é centrada;

- Sistema de componentes de negócio: Um grupo de componentes de negócio que cooperam para entregar um conjunto coeso de funcionalidades requeridas por uma necessidade específica de negócio. Em outras palavras é um sistema construído através de componentes de negócio;

2.3 A MDA no Desenvolvimento Baseado em Componentes

A MDA enfatiza a utilização dos princípios da produção industrial no desenvolvimento de software, pela procura em automatizar ao máximo o processo de desenvolvimento de componentes e de construção de aplicações através desses componentes. A abordagem de HERZUM E SIMS (1999), anunciou a MDA incluindo artefatos de projeto na definição do conteúdo de um componente de negócio. Eles compreenderam que a reutilização de um componente não dependia apenas da utilização de técnicas para o desenvolvimento de componentes facilmente substituíveis. O projeto de um componente precisa ser compreendido por quem pretende utilizá-lo. Da mesma forma que o empacotamento de artefatos de execução do componente, o empacotamento de artefatos de projeto torna-se uma atividade de importância crítica, de forma que fiquem acessíveis para todos que desejem reutilizá-lo. Com a MDA, alguns modelos de um componente de negócio, que antes eram vistos como artefatos de projeto, são altamente formalizados e adquirem características de artefatos de desenvolvimento. Eles direcionam a automatização da geração de APIs para as camadas de área de trabalho e de negócios de um componente e automatizam pelo menos parte da implementação dessas APIs (FRANKEL, 2003).

No processo de desenvolvimento baseado em componentes dentro da MDA, a partir dos modelos de domínio, os componentes são identificados e separados para um processo de desenvolvimento individual, onde são construídos seus respectivos PIM, PSM e código. E

após o teste e distribuição do componente ele é incorporado no processo de desenvolvimento da aplicação ou sistema.

2.4 A Utilização de Padrões de Projeto na MDA

A MDA unificou o paradigma do desenvolvimento de software baseado em padrões com a construção de modelos (BLANKERS, 2003), e enfatizou a utilização de padrões desde o nível de abstração dos modelos de negócio e domínio. A transformação de padrões de projeto através dos diversos níveis de abstração tornou-se um dos mecanismos de transformação de modelos utilizados no ambiente da MDA. Nas seções a seguir apresentamos uma breve descrição de como surgiram os padrões de projeto na Engenharia de Software, as características dos padrões independentes de plataforma de desenvolvimento utilizada e dos padrões específicos de uma determinada plataforma tecnológica. Como referência, abordamos os padrões da plataforma J2EE. e como esses padrões podem ser utilizados na construção de PIMs e PSMs de um componente ou aplicação.

2.4.1 Padrões de Projeto

Em qualquer disciplina consolidada da Engenharia encontramos manuais que nos orientam na solução de problemas conhecidos. Essas diretrizes possuem uma série de registros de sucesso, e tornam-se padrões de projeto, que são reutilizados no desenvolvimento dos produtos inerentes a cada respectiva Engenharia (SCHMIDT, JOHNSON, FAYAD, 1996).

Um padrão é uma solução recorrente para um problema comum. Quando grupos de padrões relacionados são integrados, eles formam uma linguagem que fornece um mecanismo para desenvolver soluções ordenadas para problemas de uma determinada engenharia ou domínio e revela as estruturas e os relacionamentos inerentes aos elementos que pretendem cumprir um objetivo comum. Linguagens padrão não são linguagens formais, mas sim uma coleção de

padrões inter-relacionados que formam um vocabulário para compreensão e comunicação de conhecimento (SCHMIDT, JOHNSON, FAYAD, 1996) (COPLIEN, 1996).

O uso corrente dos termos padrão e linguagem padrão é derivado de escritos do arquiteto Christopher Alexander que publicou vários livros sobre o assunto, relacionado com planejamento urbano e construções arquitetônicas (COPLIEN, 1997). É dele o formato de descrição de padrão original, chamado de *Alexanderian Form* (COPLIEN, 1996).

A história do padrão de software está muito relacionada com a história da orientação a objetos, já que objetos se transformaram no foco principal dos projetistas de software desde os meados de 1980 (COPLIEN, 1996). Em 1987 Ward Cunningham e Kent Beck estavam trabalhando com Smalltalk e decidiram usar idéias de Alexander para desenvolver cinco pequenas linguagens padrão para guiar programadores iniciantes. De 1990 a 1992 vários membros da *Gang of Four* (GoF), como são conhecidos Erich Gamma, Richard Helm, Ralph Johnson, e John Vlissides, se encontraram e construíram um catálogo de padrões. Em agosto de 1993, Kent Beck e Grady Booch patrocinaram no Colorado o primeiro encontro do que é conhecido agora como o grupo do Hillside[1] grupo dedicado a pesquisa de padrões e práticas para melhorar a qualidade do desenvolvimento de software. Em 1995 o livro *Design Patterns: Elements of Reusable Object-Oriented Software* da GoF (GAMMA, E. *et al,* 1995), foi publicado. A publicação desse livro, voltado para padrões de projeto orientado a objetos, popularizou a utilização de padrões de software. Outro livro de padrões de projeto bastante utilizado pela comunidade de software é *Pattern-Oriented Software Architecture: A System of Patterns* (BUSCHMANN, F. *et al*, 1996) também conhecido como POSA**,** de Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, e Michael Stal, algumas vezes chamado de *Gang of Five* (GoV) (COPLIEN, 1996).

[1] http://hillside.net

O foco inicial em padrões foi direcionado para padrões de projeto, que significavam padrões a serem aplicados durante a fase de projeto da especificação de uma aplicação. Porém existem outros tipos de padrões de software, como padrões de análise (FOWLER, 1996), padrões de arquitetura e mais recentemente nos referenciamos a padrões de negócio ou domínio, que são padrões aplicados durante a modelagem de um determinado aspecto de negócio ou domínio (ARLOW, NEUSTADT, 2003) (COMPUWARE,2004).

Padrões auxiliam na redução da complexidade em várias fases do ciclo de vida de um sistema de software. Embora os padrões não sejam processos ou métodos de desenvolvimento, eles complementam processos e métodos existentes. Padrões auxiliam na ligação entre as abstrações das fases de análise de domínio e projeto de arquitetura, e nas realizações concretas dessas abstrações nas fases de implementação e manutenção. Nas fases de análise e projeto, os padrões ajudam a direcionar os desenvolvedores na seleção das arquiteturas adequadas ao contexto do sistema que está sendo desenvolvido. Nas fases de implementação e manutenção, eles ajudam a documentar as propriedades estratégicas utilizadas, em nível mais alto que o código implementado e modelos de rotinas individuais do sistema (SCHMIDT, JOHNSON, FAYAD, 1996).

No contexto do nosso trabalho, iremos considerar os padrões utilizados em todas as fases de desenvolvimento de componentes e aplicações de software e que possam ser representados através de modelos, utilizando uma linguagem de modelagem formal, que possa ser compreendido e manipulada por instrumentos específicos de software. Convencionamos tratar todos como padrões de projeto, independentemente do nível de abstração em que é utilizado. Podemos entender que um padrão de negócio ou domínio é um padrão de projeto na camada de modelagem de negócios ou domínio.

2.4.2 Padrões de Projeto no Modelo DBC

Como estamos abordando o modelo de desenvolvimento baseado em componentes, é importante verificarmos de que forma a utilização de padrões pode ser aplicada e de que forma influencia esse modelo.

Segundo LARSSON e SANDBERG (2000), padrões de projeto são muito adequados para descrever o poder de diferentes estratégias no modelo DBC. Programadores desenvolvem componentes que podem utilizar padrões de projeto já existentes e a comunidade de padrões pode extrair novos ou aprimorar padrões de projeto de aplicações de sucesso baseadas em componentes, que podem trazer benefícios no desenvolvimento de outros componentes.

Existem padrões que dão diretrizes em como estabelecer conexões flexíveis entre diferentes subsistemas. Quando se trata de trabalhar internamente num módulo ou componente, existem padrões que auxiliam a identificar a implementação mais apropriada.

Ainda segundo LARSSON e SANDBERG (2000), o domínio dos problemas inerentes ao modelo DBC são interfaces versus implementação, interdependências, interações e conjunto. Padrões de projeto com foco nesses temas podem ser de grande auxílio.

- Interface vs. Implementação**:** é difícil documentar e descrever como um componente se comunica com o sistema e com outros componentes. Um componente precisa invocar um método em outro componente que retorna para um método no primeiro componente. Usualmente existe pouca ou nenhuma documentação descrevendo como os componentes interagem. Com a utilização de um conjunto de padrões seria possível descrever soluções adequadas e de documentação mais fácil, para interação entre componentes;
- Interdependências: não é um problema novo, e foi herdado pelo modelo DBC; minimizá-lo torna-se crucial. O desenvolvimento pode parar se as interdependências crescerem demais. Por exemplo, se cada componente tiver uma

dependência explicita em uma interface com cada um dos demais componentes do sistema, o número de interdependências será N(N-1), onde N é o número total de componentes em um determinado sistema;

- Interações: reunir componentes que não são compatíveis envolve escrever código de ligação. Isso representa consumo de tempo e freqüentemente resulta em código que não pode ser reutilizado numa versão posterior. Essa é uma área onde padrões de projeto podem beneficiar os desenvolvedores de componentes;
- Montagem: sempre existirão interdependências entre componentes que não poderão ser eliminadas. Alguns componentes podem ser de terceiros, e sem conformidade com a desejada estrutura de interdependência. Nesse caso, a capacidade de substituição fica comprometida e código de ligação precisa ser escrito. Podemos aplicar padrões de projeto para solucionar esse tipo de problema, permitindo uma montagem baseada numa camada intermediária onde responsabilidades são transferidas para uma camada entre dois componentes.

No catálogo de padrões GoF (GAMMA, E. *et al,* 1995), existe uma seleção de padrões de projeto muito úteis e ajustados para o modelo DBC. Por exemplo, *Observer*, *Proxy*, *Mediator* e *Facade* resolveriam algumas das dificuldades de implementação e interdependências e interações de componentes:

- Padrão *Adapter*: também chamado *Wrapper,* permite a um cliente usar um componente alvo com uma interface incompatível. Ele traduz pedidos feitos de acordo com a interface esperada em pedidos correspondentes para o componente alvo com interface incompatível;
- Padrão *Proxy*: introduz um intermediário que cuida de toda comunicação com o componente alvo, alimentando a eficiência e proteção do alvo. A interface *Proxy* pode também fornecer acesso mais fácil ao alvo;

- Padrão *Observer*: também chamado *Publisher-Subscriber*, regula como uma modificação num objeto pode ser refletida em um número de objetos dependentes não específicos. Ajuda a evitar um acoplamento forte entre objetos envolvidos, com grande aumento de flexibilidade e reuso de possibilidades;
- Padrão *Mediator*: oculta como um conjunto de objetos se comunica entre si. Introduz um acoplamento fraco por não permitir referência direta entre um conjunto de objetos. Em vez disso, objetos se comunicam através de um mediador;
- Padrão *Facade*: fornece uma única e simples interface para um subsistema, reduzindo o acoplamento do subsistema com seus clientes. *Facade* não fornece qualquer nova funcionalidade e as classes no subsistema não conhecem nada sobre ele. Portanto, seu protocolo é unidirecional.

Os padrões POSA (BUSCHMANN, F. *et al,* 1996), também contêm um conjunto de padrões de projeto que são bastante ajustados para o modelo DBC como, *Blackboard, Broker, Whole-Part, Master-Slave.*

YAU (2000) apresenta uma abordagem para o uso de padrões de projeto no desenvolvimento baseado em componentes. Essa abordagem utiliza uma representação formal para padrões de projeto e uma técnica de instanciação dos padrões de projeto para empacotar componentes de forma automática a partir dos padrões de projeto. Os padrões de projeto são organizados num repositório de padrões de projeto, usando a representação formal de padrões e os componentes e especificações são recuperados de um repositório de componentes.

2.4.3 Padrões de Projeto Independentes de Plataforma

Os padrões de projeto independentes de plataforma são padrões que se aplicam em níveis de abstração em que ainda não estamos preocupados com a plataforma tecnológica que será

utilizada no desenvolvimento do sistema, aplicação ou componente. Esses padrões posteriormente podem ser transformados em padrões aplicados especificamente numa determinada tecnologia de implementação.

Nessa categoria de padrões nós podemos incluir, entre outros, os padrões conhecidos como GoF e GoV ou POSA.

Podemos considerar os padrões de projeto GoF, como são conhecidos os padrões apresentados por GAMMA, E. *et al*, (1995), os mais conhecidos e utilizados no desenvolvimento de software desde sua apresentação, sendo considerados como referência para qualquer arquiteto de software. STELTING e MAASSEN (2001) e METSKER (2002) desenvolveram exemplos da utilização dos padrões de projeto GoF pela perspectiva da linguagem Java.

Os padrões de projeto POSA (BUSCHMANN, F. *et al*, 1996) representam uma evolução da abordagem de padrões em aplicações de larga escala e cobrem vários níveis de abstração, desde padrões arquiteturais e padrões de projeto de nível médio, até idiomas de baixo nível. Ele auxilia os especialistas em desenvolvimento de software, a configurar os padrões para as suas necessidades, e refinar padrões individuais produzindo eficientes sistemas de padrões.

ARLOW e NEUSTADT (2003) apresentam um conjunto de conceitos que formam a estrutura de uma proposta de utilização de padrões corporativos, que eles chamam de padrões de arquétipos. Os conceitos dessa estrutura são:

- Arquétipo de negócio: representa as coisas principais que ocorrem consistentemente e universalmente em domínios de negócio e sistemas de software de negócios;
- Padrão de arquétipo de negócios: é uma colaboração entre arquétipos de negócio que ocorre consistentemente e universalmente em ambientes e sistemas de software de negócios;

Segundo ARLOW e NEUSTADT (2003), se uma empresa possui seus modelos de alto nível que correspondem aos modelos CIM no ambiente da MDA, eles podem ser muito úteis como entrada para o processo de desenvolvimento na geração do PIM de mais alto nível. Entretanto defendem a idéia de que é mais fácil e rápido gerar PIMs através da utilização de padrões de arquétipo.

Esses padrões de arquétipo podem possuir elementos opcionais, e conseqüentemente possuem variabilidade e possibilidade de configuração. Essa proposta de padrões de arquétipo apresenta também um perfil UML específico para sua representação e utilização.

Temos como exemplo dos padrões de arquétipos apresentados, um padrão de *Customer Relationship Management* (CRM), um padrão de Produto, um Padrão de Inventário, um padrão de ordem (venda ou compra), um padrão de quantidade, um padrão de Dinheiro, um padrão de regra, etc...

Todos esses padrões usam notações formais para denotar relacionamentos e interações entre classes e objetos, e podem ser representados utilizando dois tipos de diagramas da UML para sua representação:

- Diagrama de Classes: utiliza classes, suas estruturas e os relacionamentos estáticos entre elas;
- Diagrama de Interação: mostra o fluxo de solicitações entre objetos.

2.4.4 Padrões de Projeto Dependentes de Plataforma

No processo de desenvolvimento da MDA, um PIM pode ser transformado em vários PSMs, um para cada plataforma tecnológica específica. Em cada plataforma tecnológica específica, existem catálogos de padrões de projeto específicos, que são utilizados para construções de componentes e aplicações robustas.

Considerando que a arquitetura utilizada nas principais plataformas tecnológicas atualmente é a arquitetura em camadas, o sistema é visualizado em termos de camadas, onde uma camada equivale a uma partição lógica dos diversos aspectos tratados em um sistema. A cada camada é atribuída sua responsabilidade distinta, ou única, no sistema. O enfoque de camadas foi utilizado para organizar a maioria dos catálogos de padrões, como os catálogos de padrões da plataforma J2EE. Na figura 2.8 é apresentado esse modelo de camadas.

No contexto do nosso trabalho, a plataforma tecnológica utilizada no ambiente da MDA será a plataforma J2EE. Por isso, nosso foco se volta para os padrões específicos J2EE. Esses padrões fornecem soluções para problemas normalmente encontrados em aplicativos para a plataforma J2EE. Descrevem soluções com ênfase em tecnologias J2EE importantes como *Java Server Pages* (JSP), Servlets, componentes EJB, *Java Message Server* (JMS), *Java Database Connectivity* (JDBC) e *Java Naming and Directory Interface* (JNDI) e foram reunidos para criar o catálogo de padrões *Core* J2EE (ALUR, CRUPI, MALKS, 2001).

Outro catálogo de padrões para a plataforma J2EE é apresentado por (MARINESCU, 2002), que aborda padrões avançados específicos para a tecnologia EJB.

Os padrões da camada de apresentação contêm os padrões relacionados à tecnologia de servlets e JSP. Os padrões da camada de negócios contêm os padrões relacionados à tecnologia EJB**.** Os padrões da camada de integração contêm os padrões relacionados ao JMS e JDBC**.** O catálogo de padrões EJB Design Patterns atua na camada de negócios, pois aborda padrões relacionados com a tecnologia EJB.

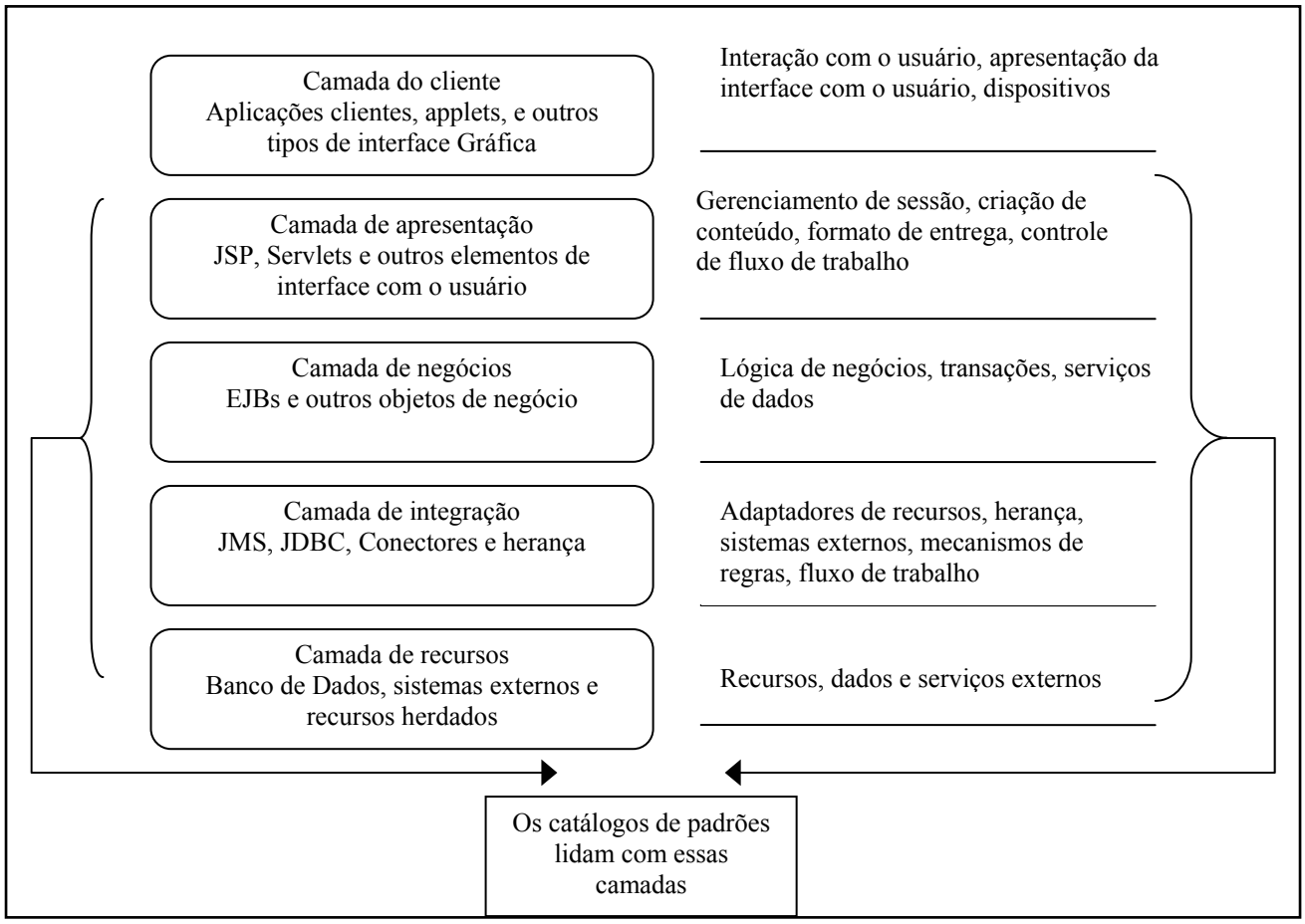


Figura 2.8: Modelo de cinco camadas. Adaptado de (ALUR, CRUPI, MALKS, 2001).

A UML é utilizada largamente nos catálogos de padrões J2EE, particularmente como a seguir (ALUR, CRUPI, MALKS, 2001):

- Diagrama de Classes**:** são utilizados para mostrar a estrutura da solução do padrão e as estruturas das estratégias de implementação. Isso fornece a visualização estática da solução;
- Diagramas de Seqüência (ou interação): são utilizados para mostrar as interações entre participantes diferentes em uma solução ou uma estratégia. Isso fornece a visualização dinâmica ou comportamental da solução;
- Estereótipos: são utilizados para indicar tipos diferentes de objetos e de funções nos diagramas de classe e de interação.

2.4.5 A Construção de PIMs e PSMs Utilizando Padrões de Projeto

No modelo de desenvolvimento baseado em componentes, na modelagem da arquitetura de negócios, o foco da atenção se volta para a identificação e especificação de uma solução orientada a componentes e serviços. Essa fase é totalmente independente da plataforma tecnológica que será utilizada para a implementação. Esse modelo de arquitetura, na MDA, é classificado como um PIM e seria utilizado para identificar os componentes de negócio existentes e identificar os serviços que esses componentes devem fornecer. No momento em que é verificada a necessidade de se construir um novo componente, ao invés de reutilizar um componente já existente ou adquirir de terceiros, a especificação do componente é separada para um novo modelo onde será refinado para construção do seu projeto detalhado. Nessa fase são utilizados padrões de projeto independentes de plataforma. Esse modelo na MDA também é classificado como um PIM do componente específico.

Após o projeto do componente ser finalizado, decide-se a tecnologia em que ele será implementado. Nessa fase é gerado o PSM do componente aplicando os padrões da plataforma tecnológica escolhida, de acordo com o processo de desenvolvimento da MDA.

POELS (2003) apresenta uma metodologia de quantificação de tamanho de sistemas de software desenvolvidos usando MDA, do ponto de vista dos requisitos funcionais dos usuários. Esse trabalho descreve a MDA como uma abordagem de desenvolvimento de software que prevê a construção de sistemas de software através da transformação, via padrões de projeto independentes de plataforma e padrões de implementação dependentes de plataforma, e da representação conceitual das organizações, independente de computação e seus sistemas de informação requeridos.

Esses dois trabalhos se referem a processos de desenvolvimento na MDA com a utilização de padrões de projeto ao longo das fases do processo, desde o mais alto nível de abstração.

Ferramentas desenvolvidas para a MDA, como o OptimalJ, são construídas com base na utilização de padrões em todos os níveis de abstração e na transformação desses padrões na passagem de um nível para o outro (CRUPI, BAERVELDT, 2005) (COMPUWARE, 2004).

Como a própria abordagem da MDA define, o desenvolvimento de software é realizado através da construção de modelos num nível alto de abstração e independente de plataforma (PIMs), utilizando padrões de projeto, independentes de plataforma, na transformação desses PIMs em modelos de plataforma específica (PSMs) e em modelos de código. Nessas transformações, os padrões independentes de plataforma são mapeados nos padrões de projeto correspondentes da plataforma específica.

Essas transformações podem se dar em diversos níveis podendo existir mais de um modelo independente de plataforma. Porém sempre ocorre a redução do nível de abstração quando a transformação se dá em direção à implementação dos sistemas de software, e na correspondência direta, a redução do nível de abstração dos padrões de projeto utilizados em cada transformação. A figura 2.9 apresenta um exemplo dos níveis que podem existir no processo de desenvolvimento da MDA.

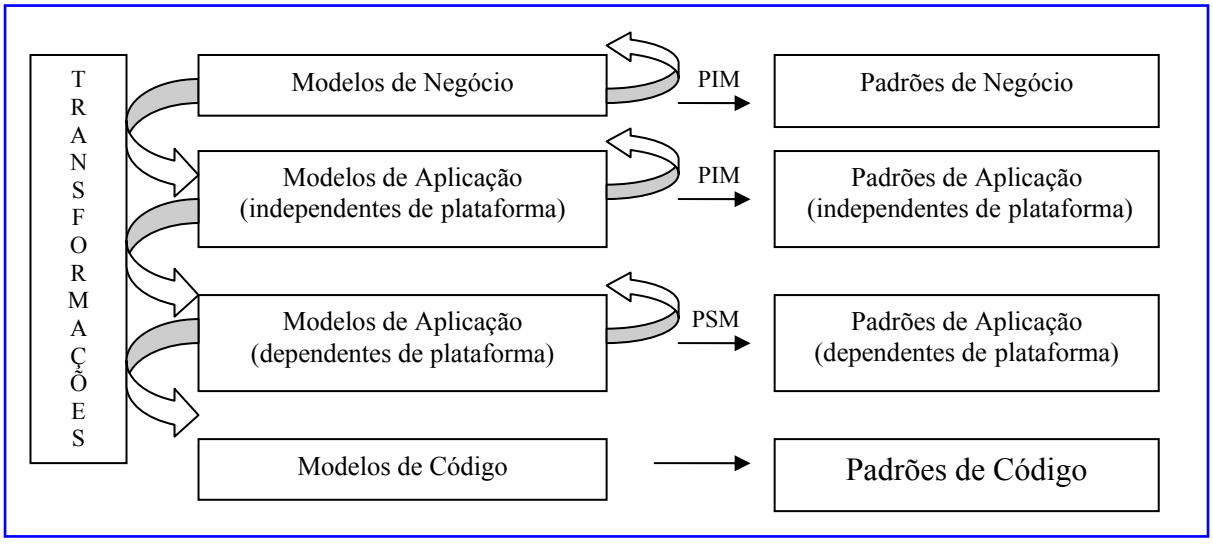


Figura 2.9: Exemplo de níveis de transformação de padrões na MDA

CAPÍTULO 3

A Atividade de Certificação no Desenvolvimento de Software

A garantia de qualidade é uma área de pesquisa que vem merecendo crescente atenção na Engenharia de Software. Existem várias abordagens de *software quality assurance* (SQA) com a utilização de diversas propostas de modelos, métricas, e padrões.

Os padrões de projeto foram introduzidos na Engenharia de Software, almejando maior qualidade nos sistemas desenvolvidos, já que representam a aplicação de soluções conhecidas e aprovadas para problemas comuns. A necessidade de se criar formalismos de representação, mecanismos de identificação, verificação e validação da utilização de padrões no desenvolvimento de software, ficou logo evidente. Diversos trabalhos de pesquisa foram realizados nesse sentido. No modelo DBC, a atividade de certificação de componentes assumiu um papel de importância significativa, pela necessidade de se garantir a confiabilidade do componente antes de torná-lo disponível para reutilização. Verificar e validar a correta utilização de padrões de projeto ao longo do ciclo de desenvolvimento de um componente é parte de sua certificação, que envolve a validação de outros atributos de qualidade. Nesse capitulo será apresentada uma visão geral de garantia de qualidade de software e sua relação com utilização de padrões de projeto e certificação de componentes, além de uma discussão sobre os principais trabalhos, ligados à certificação.

3.1 Garantia de Qualidade de Software

Na Engenharia de Software, garantia de qualidade de software é uma prática para assegurar as características de qualidade requeridas de um determinado produto de software. SQA é definida como uma abordagem sistemática e planejada para avaliação de qualidade e aderência a padrões, processos e procedimentos. SQA inclui o processo de garantia de que

padrões e procedimentos são estabelecidos e seguidos através do ciclo de vida de desenvolvimento de software.

A conformidade com padrões e procedimentos acordados é avaliada através de processo de monitoração, avaliação de produto e auditorias. O desenvolvimento de software e o processo de controle devem incluir pontos de aprovação de garantia de qualidade, onde uma avaliação SQA do produto pode ser feita em relação aos padrões aplicáveis.

O estabelecimento de padrões e procedimentos para o desenvolvimento de software é fator fundamental na criação de uma estrutura (*framework)* a partir da qual o software evolui. Padrões são critérios estabelecidos através dos quais os produtos de software são comparados. Procedimentos são critérios estabelecidos através dos quais os processos de controle e desenvolvimento são comparados. Padrões (de projeto, de codificação) e procedimentos estabelecem os métodos prescritos para o desenvolvimento de software. O papel do SQA é assegurar a existência e adequação desses padrões e procedimentos (FUTRELL, SHAFER, SAFER, 2002).

Verification and Validation (V&V) é um processo da Engenharia de Software que utiliza metodologias rigorosas para avaliação da qualidade e exatidão de produtos de software ao longo do seu ciclo de vida (IEEE, 1998). A diferença entre verificação e validação é que a verificação garante que o produto foi projetado para entregar toda a funcionalidade desejada pelo cliente enquanto que a validação garante que a funcionalidade definida nos requisitos está contemplada no comportamento do produto. A verificação procura detectar e corrigir erros em cada etapa do ciclo de vida do software, para determinar se os produtos de uma atividade satisfazem os seus requisitos. Entretanto a verificação não é suficiente para garantir que o software final cumpre seus propósitos e atende as necessidades dos usuários. A validação determina se o software atende as necessidades para o seu uso pretendido.

Independent Verification and Validation (IV&V)[1] é um processo V&V independente do ponto de vista técnico e financeiro, onde a responsabilidade do processo está numa organização independente da organização responsável pelo desenvolvimento do produto (IEEE, 1998).

SQA é um conjunto de atividades que formam um programa de garantia de qualidade, com diferentes técnicas para quantificar a qualidade de um produto de software. V&V é um processo explicitamente para verificação e validação, e conseqüentemente seus resultados são modelos de muita utilidade para os desenvolvedores e de difícil compreensão para os usuários. V&V é um tipo de modelo de garantia de qualidade que pode ser utilizado dentro de um programa de SQA (ARTHUR e SARGENT, 1999).

Posteriormente surgiu uma variação do método V&V, chamado de VV&A, mais voltado para aplicação em modelos e simulações, onde foi acrescentado o termo acreditado ou credenciado (*accreditation)*. Esse termo na engenharia de software corresponde à certificação, uma certificação oficial para um determinado modelo, simulação, federação de modelos e simulações, para utilização em um propósito específico (BALCI *et al*, 2002).

Existem diversos tipos de modelos de qualidade, de acordo com as características do software que será avaliado. TRENDOWICZ e PUNTER (2003) desenvolveram um modelo de qualidade para linha de produtos de software, determinando as características de qualidade a serem avaliadas nessa situação específica.

3.2 Suporte e Verificação da Utilização de Padrões de Projeto

A utilização de padrões de projeto é fundamental para elevar a qualidade de software e o nível de abstração. Entretanto, escolher, aplicar e recuperar padrões de projeto no desenvolvimento de software não é uma tarefa simples.

[1] www.ivv.nasa.gov.br

Padrões de projeto são micro-arquiteturas que podem executar um papel muito efetivo na construção de modelos e metamodelos em um ambiente de desenvolvimento orientado a modelos (WILLIAMS, HUGHES e OROOJI, 2002).

Por outro lado, dentro de um programa de garantia de qualidade, a utilização de padrões de desenvolvimento em todo ciclo de vida é um item obrigatório.

Mecanismos e instrumentos de apoio para a automatização dessas tarefas, e ferramentas de verificação, validação e certificação da utilização de padrões precisam ser fornecidos.

Para PETRI e CSERTAN (2003), verificar se a implementação está em conformidade com a descrição dos padrões é uma das principais funcionalidades fornecidas por mecanismos e ferramentas relacionadas com padrões de projeto.

CARR III e BALCI (2000) descrevem a utilização de verificação e validação em artefatos orientados a objetos ao longo do ciclo de vida de desenvolvimento dos modelos de simulação. Apontam as principais questões da aplicação da V&V em casos de uso, em diagramas de casos de uso, diagramas de seqüência e diagramas de classe. Nos diagramas de classe os autores fazem referência à utilização de padrões de projeto, e a identificação de padrões de projetos nos diagramas é contemplada pela aplicação da V&V.

3.3 Certificação de Componentes

Gerenciar recursos de software tem sido sempre um desafio. Antes da emergência dos componentes, softwares eram usualmente controlados ao nível de sistemas ou sub-sistemas ou mesmo código fonte, com uma visão total e comum do ambiente operacional ou infra-estrutura local. Com o movimento em direção ao desenvolvimento baseado em componentes, recursos de software passaram a requisitar um gerenciamento como múltiplas colaborações entre vários pacotes componentes com interdependências específicas. Esses pacotes

componentes podem ser originados de uma variedade de fontes, incluindo: (SELECT, 2003) (SELECT B.S., 2003).

- Soluções de software completas fornecidas por vendedores de pacotes de software;
- Coleções de serviços e componentes fornecidos por vendedores de software;
- Componentes configurados desenvolvidos internamente ou por parceiros;
- Sistemas não baseados em componentes com interfaces como componentes;
- Serviços Web via *Universal Description, Discovery and Integration* (UDDI) (TRIREME, 1999);

A forma natural de ter sucesso na implantação de um processo de desenvolvimento baseado em componentes é pela utilização de um processo que inclua gerência e repositório de componentes (APPERLY, 2001). O uso de um repositório garante que uma cópia de um componente usada comumente pode ser encontrada e reutilizada em múltiplas aplicações (SELECT, 2003) (SELECT B.S., 2003).

SAMETINGER (1997) definiu repositório de componentes como um banco de dados para armazenamento e recuperação de componentes reutilizáveis. Esses repositórios contêm componentes de software com todas as informações relevantes sobre eles, incluindo sua arquitetura, história, interações com outros componentes, classificação (para recuperação) e documentação. Um repositório é a ligação entre o desenvolvimento para o reuso e o desenvolvimento com reuso.

Gerência de componentes não aborda apenas a organização de milhares de componentes. É um conceito muito útil utilizado nas disciplinas de engenharia, que pode ser aplicado no desenvolvimento de software em diferentes escalas e diferentes níveis de complexidade (APPERLY, 2001).

O processo de desenvolvimento de software *Select Perspective* (APPERLY, 2001), é baseado na integração entre três fluxos de trabalho fornecedor-gerência-consumidor (*supply-manage-*

consume). Aborda explicitamente a gerência de componentes, no seu fluxo de trabalho Gerência de Componentes (*Component management)* com uma atividade exclusiva para certificação de componentes. Nesse processo o repositório também faz a integração entre os fornecedores de componentes (desenvolvimento para o reuso) e consumidores de componentes (desenvolvimento com reuso) através da gerência de componentes.

A definição de certificação encontrada no dicionário (FERREIRA, 1986), nos dá uma idéia do significado convencional de certificação e expõe de que forma o termo está relacionado com componentes de software. Certificação significa afirmar a certeza de, atestar, passar a certidão de; convencer da verdade ou da certeza de algo, garantir, endossar.

Segundo ADDY (1998), embora certificação de componentes e V&V estejam ambos relacionados com avaliação de recursos de software e qualificação, significam coisas distintas. A certificação de componentes seria uma parte de todo V&V da aplicação e usualmente se refere a componentes reutilizáveis no nível de código de implementação. V&V tem o escopo mais amplo e também considera modelos de domínio e arquiteturas genéricas, bem como a conexão entre artefatos de domínio e artefatos da aplicação. Porém algumas pesquisas abordam a certificação segundo a visão do método VV&A, onde uma atividade de certificação oficial foi adicionada para modelos e simulações, mas que também pode ser aplicada em componentes, considerando modelos ou simulações como componentes (BALCI e SAADI, 2002).

A idéia de certificação mostrada na figura 3.1**,** é que o usuário de um componente confie no resultado da verificação e testes executados por um certificador, que pode ser outra pessoa, mecanismos e instrumentos ou contratação de serviço terceirizado, economizando tempo de cada usuário do componente em verificar seu funcionamento, apenas consultando seu certificado.

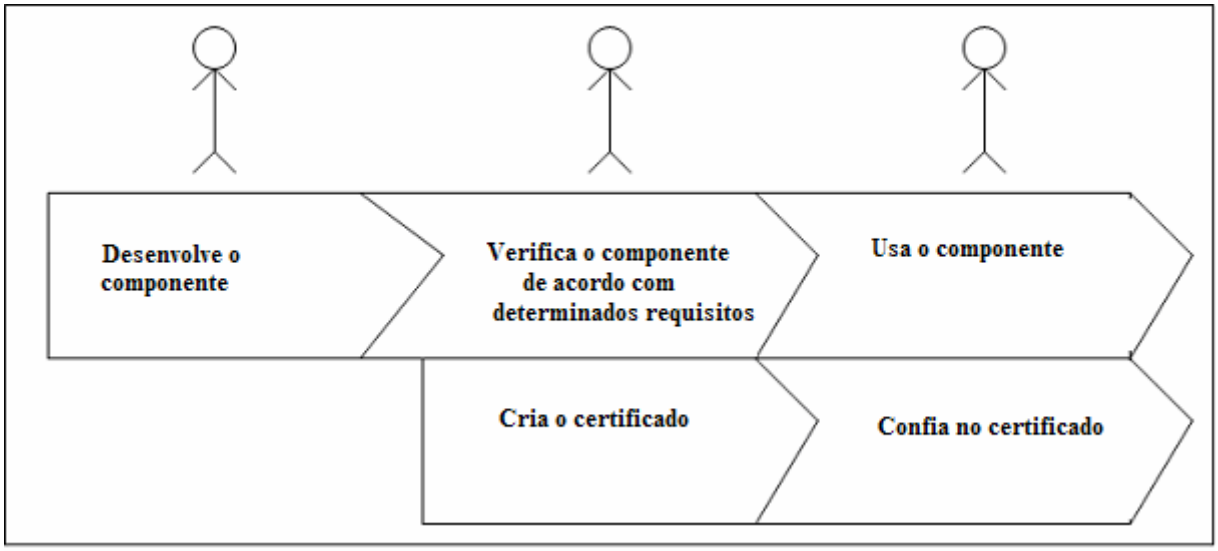


Figura 3.1: Processo de certificação de componentes. Adaptado de (PISTER, 2004).

Existe um variado número de dimensões ou atributos de qualidade de software que estão relacionados e servem para dimensionar características operacionais como confiabilidade (probabilidade de operação livre de falhas), características de programação (custo e duração de desenvolvimento), características de manutenção (facilidade de adaptação para um novo contexto), etc. (THOMAS e CERINO, 1995). WOODMAN *et al* (2001) classificaram esses atributos de qualidade de acordo com o desenvolvimento baseado em componentes. Esses atributos são:

- Capacidade de Reutilização (*Reusability*): O aumento da reutilização tem sido um dos principais objetivos de negócio e a proposta do modelo DBC proporciona a colocação da reutilização em níveis bem mais elevados;
- Capacidade de Manutenção (*Maintainability*): Redução de custos de manutenção é visto como o principal benefício do modelo DBC. Em condições ideais, componentes certificados e sem defeitos estariam disponíveis e seriam distribuídos por fornecedores que aperfeiçoariam as funcionalidades de acordo com necessidades demandadas;

- Exatidão (*Accuracy*): A capacidade do componente fornecer as funcionalidades especificadas e com a precisão necessária precisa ser projetada no componente;
- Clareza (*Clarity*): Especificação clara e rigorosa das interfaces dos componentes é um requisito obrigatório para o sucesso do modelo DBC;
- Capacidade de Substituição (*Replaceability*): Um componente precisa ser facilmente substituído por outro com implementação diferente. Exatidão e clareza de interface são imprescindíveis nesse caso, acrescido da forma como o componente utiliza a respectiva tecnologia de componentes;
- Capacidade de Atividade Conjunta (*Interoperability*): Refere-se a capacidade do componente interagir com outros componentes e com outras tecnologias. Do ponto de vista de uso do componente, é um atributo de qualidade de alta importância. É atingido pela sua especificação. A principal preocupação do usuário do componente é sua aderência à especificação;
- Capacidade de Alteração de Escala (*Scalability*): A propriedade de aumentar de escala é bem conhecida em muitos domínios, como sistemas de banco de dados. Em sistemas baseados em componentes distribuídos, é um atributo de qualidade de grande importância;
- Desempenho (*Performance*): O componente precisa fornecer desempenho apropriado abaixo do pico de carga relativo a quantidade de recursos alocados;
- Flexibilidade (*Flexibility*): O componente precisa estar livre do desenvolvimento e distribuição tradicional;
- Adaptabilidade (*Adaptability*): Os componentes precisam ser reutilizados e estendidos em uma forma "*plug and play*" em diferentes contextos de negócio;
- Confiabilidade (*Reliability*): O componente necessita fornecer um determinado nível de imunidade à falha. Em sistemas de segurança crítica os níveis de

confiabilidade necessários são declarados em termos de freqüência de falha. O desenvolvimento baseado em componentes em sistemas de alta confiabilidade, em primeiro lugar exige componentes com a confiabilidade declarada e em segundo lugar que o sistema resultante atinja os requisitos de confiabilidade.

Com a descrição desses atributos, vemos que o conceito de qualidade, em diferentes perspectivas pode significar coisas distintas. Por exemplo, na aquisição de componentes de terceiros, pode primeiramente se concentrar no custo do ciclo de desenvolvimento, enquanto uma empresa de manutenção de software, com a facilidade de adaptação. O que é comum entre as perspectivas é que a qualidade é definida de acordo com o contexto das necessidades (THOMAS, CERINO, 1995). Dessa forma os objetivos de negócio das organizações determinam e direcionam seu significado de qualidade.

Executando as atividades de certificação de componentes, estamos garantindo que os atributos de qualidade imprescindíveis para o contexto em que ele será utilizado estão presentes, e fornecem meios para estabelecer e verificar informações de linha de base de um componente, permitindo uma melhor compreensão do benefício de sua reutilização. A certificação fornece instrumentos para determinar informações críticas sobre os componentes reutilizáveis (THOMAS, CERINO, 1995). Para cada atributo de qualidade identificado como relevante para o desenvolvimento baseado em componentes, atividades específicas e adequadas de certificação precisam ser adotadas.

SAMETINGER (1997) descreve o conceito de nível de certificação. Os esforços aplicados na certificação de um componente dependem da natureza, freqüência de uso e importância do componente. São definidos quatro níveis conforme a seguir:

- Nível 1: O componente é descrito através de palavras chaves e uma abstração é armazenada para recuperação automática. Nenhum teste é executado. O grau de completeza é desconhecido;

- Nível 2: O código fonte do componente precisa ser compilado para ser validado. Métricas são determinadas para linguagens específicas;
- Nível 3: Testes de dados e resultados de testes são adicionados;
- Nível 4: Um Guia de reutilização é adicionado;

SAMETINGER (1997) também descreve várias técnicas de garantia de qualidade, e como cada uma delas se adequou, para verificação de determinados atributos de qualidade de um componente. As definições dessas técnicas apresentadas por ele são:

- Análise Estática: pode verificar várias propriedades, especialmente do código fonte, sem necessidade de execução. Essas propriedades incluem código sem utilização (que nunca será executado), anomalias de variáveis, coesão do componente, etc...
- Inspeção Formal: algumas propriedades de componentes podem escapar da análise automática, necessitando de inspeção humana. Essas propriedades incluem consistência de especificação, qualidade de documentação, exatidão de projeto, etc...
- Teste: testes podem ser aplicados em qualquer tipo de componente executável. Em primeiro lugar verificam o código fonte, mas também suas especificações e projetos. Testes não garantem a ausência de erros, mas são úteis para encontrá-los. Testes podem auxiliar no estabelecimento de um certo grau de garantia na confiabilidade de um componente;
- Modelos de Uso: modelos de uso são adotados para modelar a visão externa de uso do componente. Fornecem a base para o controle estatístico de qualidade. Um componente pode ser certificado pela modelagem de seu uso, derivando perfis de uso, gerando e executando casos de teste, coletando falha de dados e predizendo a confiabilidade futura;

- Verificação Formal: utilização de meio formal de prova de garantia do atributo de qualidade. Necessita de ferramentas específicas. Ideal para verificar a implementação completa do componente em relação a sua especificação, porém essa tarefa é apresentada como quase impossível de ser atingida;
- Comparação (*benchmarks*): medidas de desempenho como velocidade de execução, tempo de resposta e utilização de memória do componente podem ser determinadas por comparação.

Existem diversas técnicas de garantia de qualidade que podem ser utilizadas de acordo com os atributos que se deseja verificar. Conseqüentemente, existem diversas abordagens de certificação de componentes:

- Existe a certificação quanto ao atendimento dos requisitos funcionais da sua especificação. Nesse caso, a implementação não é levada em consideração e a certificação é baseada na execução dos casos de teste apropriados, de forma a garantir os resultados esperados com o uso do componente. Esse tipo de certificação é útil para os consumidores do componente na construção de aplicações;
- Existe a certificação quanto ao atendimento de especificações técnicas e quanto à utilização das melhores práticas do modelo DBC e da tecnologia utilizada. Esse tipo de certificação é mais voltado para quem desenvolve e fornece componentes de software. Essa certificação garante o desenvolvimento de componentes genéricos com alto índice de reutilização e flexibilidade, atributos de qualidade importantes nessa situação.

3.4 Trabalhos Relacionados com Certificação

Como pudemos verificar nas seções anteriores, SQA, V&V, IV&V, VV&A, padrões e certificação, são temas bastante interligados. Num determinado processo de SQA, podemos incorporar métodos V&V. Aplicando-se o método V&V nos modelos que utilizam padrões, naturalmente estaremos aplicando o método também nos padrões de projeto, com condições de diagnosticar problemas na instanciação e implementação desses padrões. Concluindo, a atividade de certificação está sendo incorporada nos métodos V&V, para ser utilizada como uma certificação formal para dar credibilidade a aplicações de modelos e simulações, gerando um novo método conhecido como VV&A.

Dessa forma todos os trabalhos analisados estão sendo considerados como trabalhos de certificação, mesmo que não abordem explicitamente o assunto. Nesse caso se encontram os diversos trabalhos de pesquisa com foco na verificação e validação da utilização de padrões de projeto, verificação de inconsistências de software em geral e com a certificação de componentes propriamente dita. A análise dos principais trabalhos avaliados será apresentada nessa seção, organizados com relação a sua abordagem principal, isto é, trabalhos de verificação da utilização de padrões, verificação de inconsistências de software em geral e certificação de componentes. Dentro do grupo de certificação de componentes tentaremos classificar os trabalhos segundo as técnicas de garantia de qualidade de SAMETINGER (1997).

Os trabalhos de pesquisa sobre utilização de padrões abordam o tema de várias maneiras:

- Em alguns, o foco é dar apoio a sua utilização, criando mecanismos que automatizam em algum nível a geração do código desses padrões. Dentro dessa categoria se encontram as pesquisas de:
    - WILLIAMS, HUGHES e OROOJI (2002), que fornece os construtores essenciais de um formalismo matemático, inserido no contexto da MDA para especificação

de padrões de projeto, constituídos de uma extensão dos conceitos básicos da álgebra de múltiplos domínios (AABY, 2002);

- ALBIN-AMIOT E GUEHENEUC (2001) desenvolveram um metamodelo específico para representação de padrões de projeto que permite geração automática de código e detecção de padrões de projeto, contemplando os aspectos da estrutura dos padrões. O objetivo da pesquisa é propor um conjunto de ferramentas para utilizar padrões de projeto em um modelo de ida e volta (*round-trip*);
- A aplicação MVCASE[1] utiliza o padrão XMI para armazenar especificações numa arquitetura de repositório distribuído de padrões de projeto. O objetivo é auxiliar a criação e utilização desses padrões e automatizar o processo de implementação utilizando geração de código (LUCRÉDIO, D., *et al,* 2002) (LUCRÉDIO, D., *et al,* 2003).
- A aplicação Fujaba Tool Suíte RE[2] é a proposta de engenharia reversa do projeto Fujaba Tool Suite[3]. Contém um cartucho (*plug in*) para especificação de padrões de projeto, através de gráficos de transformação de regras. Essas especificações auxiliam no processo de reconhecimento dos padrões nos códigos existentes. A ferramenta já contém um catálogo com os padrões GoF (NICKEL, *et al*, 2000) (NIERE, WADSACK, ZÜNDORF, 2001).
- A ferramenta comercial Optimalj[4] é uma implementação da MDA, e sua arquitetura é toda baseada em padrões, permitindo que novos padrões sejam incorporados no seu ambiente realizando consistências durante todo desenvolvimento. Utiliza três níveis de abstração, o mais alto corresponde ao nível

[1] http://www.recope.dc.ufscar.br/mvcase/
[2] http://wwwcs.upb.de/cs/fujaba/index.html
[3] http://wwwcs.upb.de/cs/fujaba/projects/reengineering/index.html
[4] http://www.compuware.com/products/optimalj/default.htm

de modelos de domínio, PIM, o segundo ao nível de modelos de aplicação, PSM, e o último ao nível de modelos de código. Possui uma ferramenta UML integrada para análise, mas utiliza uma notação estrutural um pouco diferente na sua parte MDA.

- E finalmente os trabalhos cujo foco é a verificação da utilização dos padrões em conformidade com as suas descrições e que estão relacionados de alguma forma com o tema dessa dissertação. A seguir temos a relação das pesquisas com essas características:
  - Uma proposta de formalização e verificação de padrões de projeto através de linguagens de texto utiliza uma técnica de atributo de extensão, baseada na gramática de atributos. Extensão de atributo é um mecanismo para forçar e descrever convenções de programação através de declarações de regras semânticas (HEDIN, 1997);
  - A *RAISE Specification Language* (RSL) é uma linguagem de especificação de modelos formais orientados a objetos, e foi utilizada para definição de modelos de representação formal de padrões para serem utilizados em mecanismos de verificação de uso de padrões GoF em aplicações orientadas a objetos (FLORES, MOORE, 2000) (FLORES, REYNOSO, MOORE, 2001) (ARANDA, MOORE, 2000) (ARANDA, MOORE, 2002) (REYNOSO, MOORE, 2000). Esses mecanismos foram testados com a ferramenta de código aberto, FUJABA (NICKEL, *et al*, 2000) (NIERE, WADSACK, ZÜNDORF, 2001);
  - A aplicação OOPDTool foi desenvolvida para ser utilizada como apoio na busca de construções problemáticas existentes em sistemas desenvolvidos com técnicas de orientação a objetos (OO). OOPDTool é baseada em uma base de conhecimento de boas construções de projeto correspondentes à aplicação de heurísticas e

padrões de projeto, assim como de construções problemáticas, os anti-padrões (CORREA, WERNER, ZAVERUCHA, 2000);

- O *framework Knowledge Centric Software* (KCS) é baseado em padrões para o desenvolvimento de ferramentas de apoio na detecção de anomalias de software**.** O uso da abordagem baseada em padrões fornece a flexibilidade necessária para endereçar necessidades específicas de domínio, com respeito aos tipos de problemas que as ferramentas devem detectar e as estratégias usadas para inspecionar e adaptar o código. A estrutura (*framework)* inclui uma linguagem intermediária extensível e comum chamada *Extensible Common Intermediate Language* (XCIL)**,** uma linguagem de especificação de padrões extensível chamada *Extensible Pattern Specification Language* (XPSL)**,** um catálogo de padrões que capturam o conhecimento de um domínio específico, uma ferramenta de apoio para a análise de inspeção, um banco de dados de repositório para armazenar e recuperar os resultados das análises. Nessa estrutura (*framework*) a noção de padrão foi planejada para captura de conhecimento e não somente relativo a projetos, mas também contemplando outros aspectos de análise e transformação de software. O XCIL fornece independência de linguagem. É baseado em JVM (Java), MSIL (.NET) e UML, e incluí extensões para cobrir as semânticas do C e C++**.** Utiliza também as tecnologias *Extensible Stylesheet Language Transformation* (XSLT) e *XML Query Language* (XQUERY), na conversão do código fonte na sua representação em XML e na criação dos resultados da inspeção de código em representação XML**.** As diferentes linguagens são convertidas para e do XCIL (KOTHARI, *et al*, 2004).

Os trabalhos de pesquisa sobre inconsistências de software em geral estão relacionados com verificação de implementação e verificação de inconsistências em artefatos de software no

ciclo de vida de desenvolvimento, incluindo inconsistências de especificação. Dentro desse grupo foram analisados os seguintes trabalhos:

- A ferramenta de engenharia reversa, chamada REV-SA, é uma proposta de verificação de implementação Java, utilizando XMI para validar a coerência do código com o modelo original da especificação do componente (COOPER *et al,* 2004*)*. O modelo de implementação é recuperado do código e convertido em XMI através do processo de engenharia reversa utilizando a biblioteca NSUML que inclui o metamodelo da UML em Java. Uma proposta semelhante e anterior ao JMI (JSR40, 2002). O modelo de especificação em UML também é convertido em XMI, e as duas especificações são comparadas na representação XMI. As diferenças são filtradas a priori, eliminando itens que sabidamente existem apenas nos modelos UML de especificação e não são recuperados do código Java;
- O XLINKIT (NENTWICH, *et al,* 2003) é uma proposta de utilização de uma estrutura (*framework)* para verificação de consistência de documentos (artefatos) de software heterogêneos e distribuídos. Sua solução é baseada nas tecnologias XML, como *Document Object Model* (DOM), e nas voltadas para Internet como *XML Linking Language* (XLINK) e *XML Path Language* (XPATH)**,** que são recomendadas pela W3C;

Os trabalhos de pesquisa sobre certificação de componentes, cobrem diversos aspectos de certificação e controle de qualidade de componentes. Os principais trabalhos analisados são apresentados a seguir, organizados de forma simplificada segundo o tipo da técnica de garantia de qualidade de SAMETINGER (1997) utilizada ou pela sua abordagem teórica e metodológica.

Com a técnica de teste:

- Trabalho onde o processo de certificação é do tipo "caixa preta", para componentes como *Comercial Off-the-shelf* (COTS), também conhecidos como softwares de prateleira (VOAS, 1998).

Com a técnica de modelos de uso:

- MORRIS, *et al* (2001) apresentam um método de autocertificação de componentes, ao invés da utilização dos chamados Laboratórios de Certificação de Software (*Software Certification Laboratories– SCL*), que são agentes independentes de certificação. As vantagens desse método são a simplicidade e redução de custos, apropriados para os pequenos construtores de componentes, para certificar componentes de segurança crítica ou para a comunidade de software livre. O documento de especificação de testes é baseado em XML. Esse modelo de certificação foi utilizado na pesquisa para o desenvolvimento de um pacote de ferramentas conhecido como – SCL *Component Test Bed* (CTB)[1], que cria um banco de teste baseado em XML, consistindo de padrões de testes que podem ser vendidos junto com o componente;
- RAMACHANDRAN (2003) apresenta uma técnica de testar componentes, baseada no princípio de executar testes a partir de modelos de objetos, pela decomposição do componente em modelos de objetos. A partir dai, com a análise das interfaces dos componentes, é possível gerar os casos de teste essenciais e encontrar a quantidade de testes necessária;
- As pesquisas de WOHLIN, RUNESON e REGNELL (1994) (1998) utilizam modelos e perfis de uso, onde a certificação é realizada com a geração dos casos de teste, apresentando um procedimento geral de certificação de confiabilidade fornecendo diretrizes de como certificar componentes de software.

[1] http://xml.coverpages.org/scl.html

Trabalhos relacionados com a técnica de Inspeção Formal:

- Uma estrutura (*framework)* de certificação, denominada (*Certification Framework*-CF), tem uma abordagem de certificação que pode ser adaptada de acordo com o domínio, estratégia de negócio e tipo de componente (ROHDE, *et al*, 1996). Apresenta um algoritmo de certificação que define os processos e tarefas que devem ser isolados e a análise de defeitos por tipo e severidade, fornecendo um modelo de custo de certificação, baseado no tipo de defeito, capacidade de detectar sua presença e impacto no custo de re-trabalho. As ferramentas utilizadas no processo são: AdaWise para análises estatísticas; Logiscope para fornecer análise estática e dinâmica dos fluxos de controle dos diagramas e teste estrutural; e AdaQuest para fornecer análise estática das diretrizes de estilo, tamanho e complexidade. Considera quatro etapas de certificação: Simulações de funcionamento, análises estatísticas, inspeção de código (com a técnica de engenharia de certificação) e testes;
- O projeto *Jade Bird* (HONG, *et al*, 2000), é um ambiente integrado de desenvolvimento de software para apoiar a reutilização, o gerenciamento e recuperação de componentes, tendo em seu núcleo um sistema de biblioteca de componentes, *Jade Bird Component Library* (JBCL). JBCL qualifica o componente utilizando um modelo de métricas de componentes baseado no modelo Fator-Critério-Métrica. Esse modelo é utilizado considerando duas perspectivas, da qualidade e reutilização do componente, que possuem cada uma seu modelo de métricas específico;
- KARLSSON, ELES e PENG (2002) propõem uma abordagem formal de verificação integrada com a metodologia utilizada, em nível de projeto, em sistemas baseados em componentes. A verificação é baseada nas propriedades das interfaces dos

componentes conectados e nos modelos abstratos de suas funcionalidades, sem assumir qualquer conhecimento de suas implementações;

Existem ainda trabalhos com abordagens relacionadas com estruturas (*frameworks)* e modelos teóricos de certificação:

- O trabalho de CAI, LYU e WONG (2000) é uma proposta de um modelo de garantia de qualidade para ser aplicado na construção de sistemas baseados em componentes do tipo COTS;
- O trabalho de (GHOSH, e MATHUR) (2001) está relacionado com a certificação de componentes e aplicações em conformidade com o padrão CORBA 3. Apesar de focar a pesquisa no padrão CORBA 3, a abordagem é muito genérica, podendo ser aplicada para outros padrões.
- O *framework Component Deployment Testing* (CDT) (BERTOLINO e POLINI, 2003) foi proposto com a finalidade de fornecer tanto uma técnica para especificar previamente um conjunto de testes de distribuição, quanto um ambiente para execução e reuso de testes especificados para qualquer implementação de componente. O CDT também pode ser utilizado como veículo de entrega do conjunto de testes do fornecedor de componentes, que posteriormente pode ser re-executado;
- KALLIO e NIEMELÄ (2001) desenvolveram um modelo geral para documentar componentes de software (COTS e *Original software Component Manufacturer* - OCM) em uma forma padrão, na qual as visões do consumidor e do fornecedor de componentes foram levadas em consideração;
- COUNCILL (2001) aborda os aspectos relacionados especificamente com a certificação de componentes de terceiros. Essa certificação é baseada em *UL 1998, 2nd ed., UL Standard for Safety for Software in Programmable Components.*

- THOMAS e CERINO (1995) apresentam utra estrutura (*framework)* para certificação de reutilização que fornece uma informação de custo efetivo de certificação de componentes reutilizáveis. A estrutura orienta os usuários em selecionar e executar as principais atividades de certificação de acordo com as características das organizações, domínio das aplicações, ferramentas utilizadas e diferentes métodos de certificação;
- MEYER (2003) desenvolveu uma pesquisa onde examina os trabalhos realizados com a pretensão de enriquecer o conceito de componente confiável, trazendo todo o seu potencial para a industria de software;

3.5 Considerações Gerais

Após a análise dos diversos trabalhos e pesquisas sobre padrões de projeto, verificação de inconsistências de software e certificação de componentes, podemos concluir que a grande maioria das pesquisas realizadas sobre padrões de projeto está inserida no contexto do desenvolvimento de aplicações orientadas a objetos e são focados nos padrões de projeto independentes de plataforma GoF, atuando no nível dos modelos de aplicação sem a especificação da tecnologia utilizada. Os trabalhos analisados que contemplam padrões de outros níveis de abstração foram: o KCS, que possui um catálogo de padrões que capturam o conhecimento de um domínio específico, incluindo padrões de projeto, padrões de aspectos de análise e transformação de software; e a ferramenta comercial Optimalj, uma ferramenta voltada especificamente para um ambiente de desenvolvimento orientado a modelos utilizando a plataforma tecnológica J2EE.

Os trabalhos relacionados explicitamente com certificação de componentes que incluem inspeção de código, não abordam a certificação da utilização de padrões de projeto.

No ambiente da MDA, a ferramenta Optimalj, fornece um ambiente todo baseado em padrões desde o nível dos modelos de domínio até a codificação para a plataforma J2EE, mas fixa os níveis de abstração em três níveis (domínio, aplicação e código).

Para consolidar a avaliação dos principais trabalhos apresentados anteriormente, foi construído um quadro resumo, relacionando as principais características analisadas.

Característica	**MVCASE**	**FUJABA**	**OOPDTool**	**KCS**	**REV - SA**	**XLINKIT**	**OPTIMALJ**	**Trabalhos de certificação de componentes**
Inspeção de Código		**X**		**X**	**X**	**X**		**X**
Validação de atendimento de especificação				**X**	**X**	**X**		**X**
Formalismo de representação de padrões	**X**		**X**	**X**			**X**	
Verificação da utilização de padrões		**X**	**X**	**X**			**X**	
Repositório de padrões	**X**	**X**	**X**	**X**			**X**	
Quantidade flexível de níveis de abstração								
Verificação da utilização de padrões em todos os níveis de abstração				**X**			**X**	
Orientação a modelos (alinhamento com a MDA)							**X**	

CAPÍTULO 4

Ambiente de Certificação de Padrões de Projeto

Para que a abordagem da MDA no desenvolvimento de software se torne uma realidade, um dos requisitos é a existência de ferramentas de apoio às equipes de desenvolvimento, na criação, refinamento e validação de seus modelos de projetos nos diferentes níveis de abstração.

Neste capítulo serão apresentadas as características de um ambiente de certificação cujo objetivo é a verificação dos modelos criados ao longo do ciclo de vida do desenvolvimento de componentes e aplicações com relação à correta utilização de padrões de projetos.

Inicialmente será apresentada a descrição das características das validações realizadas em cada nível de abstração; a descrição detalhada de todas as atividades envolvidas com a determinação de papéis e responsabilidades associados a essas atividades e os diagramas de casos de uso relacionados com o funcionamento do ambiente proposto.

Por fim serão apresentadas a arquitetura conceitual do ambiente proposto e a especificação do sistema computacional de certificação, incluindo os aspectos relacionados com formalismo de representação de padrões, com reconstrução de arquitetura de software e com a plataforma tecnológica utilizada.

4.1 Certificação em um Ambiente Orientado a Modelos

No desenvolvimento orientado a modelos e baseado em padrões da MDA, existem padrões específicos para serem aplicados em cada nível de abstração. Dessa forma, para que um ambiente de certificação da utilização de padrões seja bem sucedido, é necessário que a certificação também seja dividida em níveis, com etapas de certificação específicas para cada nível de abstração existente em um processo de desenvolvimento.

Esse ambiente de certificação precisa ser totalmente aberto e flexível, podendo ser utilizado em qualquer quantidade de níveis de abstração. A formação desses níveis pode se dar de diversas formas, não existindo uma quantidade fixa de modelos de negócio e modelos de aplicação (independentes e dependentes de plataforma). A distribuição em níveis é determinada pelas características específicas de cada ambiente de desenvolvimento.

Nos níveis principais dessa estrutura, caracterizados pelos modelos de negócio, modelos de aplicação independentes de plataforma e modelos de aplicação dependentes de plataforma, podem existir vários modelos em diferentes níveis de abstração, sendo que o modelo de um determinado nível é derivado a partir do modelo do nível de abstração imediatamente superior. Essa estrutura pode ser interpretada como uma composição de níveis e subníveis de abstração.

Independentemente da quantidade de níveis, todos os tipos de certificação que podem ser realizados nesse ambiente estão ilustrados na figura 4.1 e descritos a seguir:

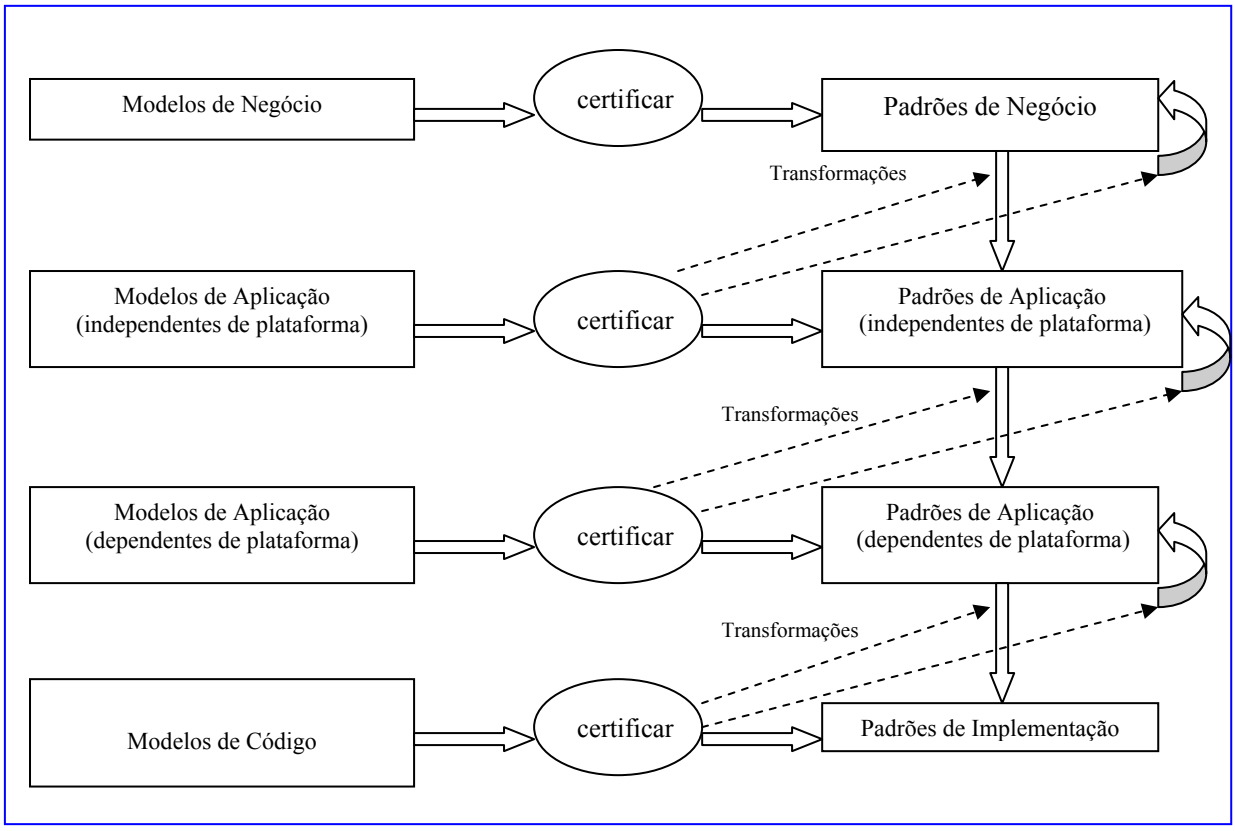


Figura 4.1: As certificações realizadas pelo sistema nos diversos níveis de abstração

- Certificação dos padrões de negócio utilizados na construção dos modelos de negócio;
- Certificação dos padrões de aplicação independentes de plataforma utilizados na construção dos modelos de aplicação independentes de plataforma;
- Certificação dos padrões de aplicação dependentes de plataforma utilizados na construção dos modelos de aplicação dependentes de plataforma;
- Certificação dos padrões de implementação utilizados na construção dos modelos de código;
- Certificação das transformações de padrões de negócio ocorridas entre diferentes e adjacentes níveis de modelos de negócio;
- Certificação das transformações de padrões de negócio em padrões de aplicação independentes de plataforma ocorridas entre um nível de modelo de negócio e um nível adjacente de modelo de aplicação independente de plataforma;
- Certificação das transformações de padrões de aplicação independentes de plataforma ocorridas entre diferentes e adjacentes níveis de modelos de aplicação independentes de plataforma;
- Certificação das transformações de padrões de aplicação independentes de plataforma em padrões de aplicação dependentes de plataforma ocorridas entre um nível de modelo de aplicação independente de plataforma e um nível adjacente de modelo de aplicação dependente de plataforma;
- Certificação das transformações de padrões de aplicação dependentes de plataforma ocorridas entre diferentes e adjacentes níveis de modelos de aplicação dependentes de plataforma;

- Certificação das transformações de padrões de aplicação dependentes de plataforma em padrões de implementação ocorridas entre um modelo de aplicação dependente de plataforma e o modelo de código correspondente;

4.2 Requisitos do Ambiente de Certificação Proposto

Através das pesquisas realizadas sobre certificação e sobre utilização de padrões de projeto, e com o levantamento das necessidades do ambiente de certificação proposto, foram identificados os principais requisitos desse ambiente. Para descrever esses requisitos, definimos a UML como linguagem de modelagem e baseado no conhecimento adquirido e apresentado nos capítulos 2 e 3 foram elaborados diagramas de atividades e diagramas de casos de uso que serão apresentados a seguir.

4.2.1 Diagrama de Atividades do Ambiente de Certificação

Neste diagrama (figura 4.2) as atividades estão agrupadas nas macro-atividades do ambiente de certificação na qual elas se inserem, possibilitando uma visualização do contexto geral de cada uma dessas macro-atividades.

Essas macro-atividades são:

- Administração de Repositório: manter um repositório dos padrões utilizados no ambiente de desenvolvimento. Os modelos de representação dos padrões de um ambiente precisam ser construídos segundo um determinado formalismo para posteriormente serem armazenados no repositório;
- Reconstrução de Arquitetura: recuperar do código fonte dos componentes e aplicações seu modelo de implementação, na linguagem de modelagem utilizada;

- Análise de Modelos: comparar os modelos dos componentes e aplicações, de todos os níveis de abstração existentes no ciclo de vida do desenvolvimento, com os modelos dos padrões utilizados e armazenados no repositório de padrões.

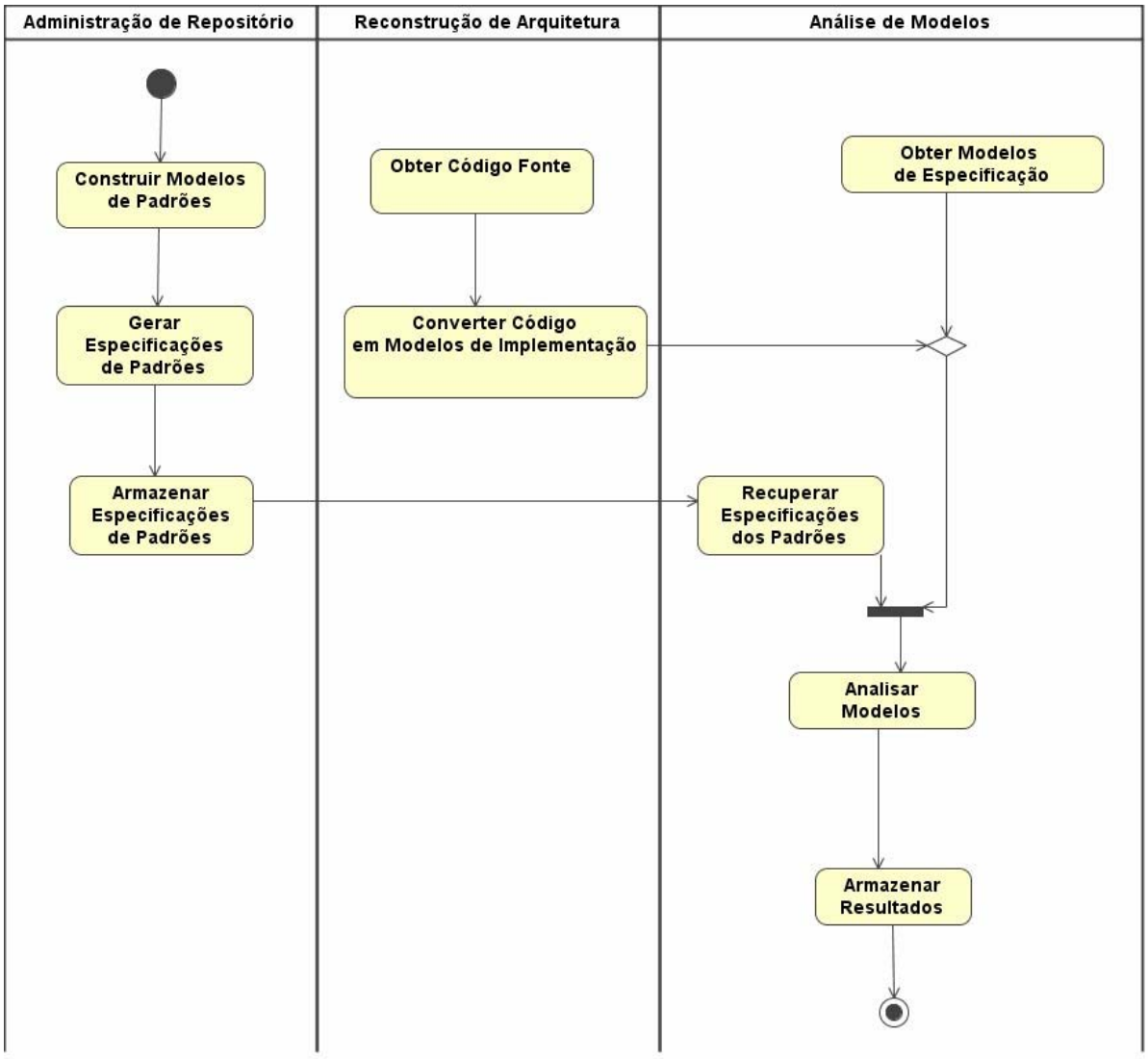


Figura 4.2: Diagrama de Atividades organizado por macro-atividades

As principais atividades identificadas possuem as seguintes atribuições:

1. Construir Modelos de Padrões: Processo de construção dos modelos de representação dos padrões que serão utilizados nas validações realizadas utilizando o ambiente de certificação. Inclui a representação dos padrões dos modelos de negócio e dos padrões utilizados nos modelos de aplicação (independentes e dependentes de plataforma).

2. Gerar Especificações de Padrões: Processo de converter os modelos de representação dos padrões em uma forma padrão de intercâmbio de modelos.
3. Armazenar Especificações de Padrões: Processo de armazenamento das especificações dos padrões no repositório de padrões.
4. Obter Código Fonte: Processo de obter o código fonte do componente que será certificado.
5. Converter código em modelos de implementação: Processo de conversão do código implementado em modelos.
6. Obter Modelos de Especificação: Processo de obter as especificações referentes ao componente que será certificado, como modelos de negócio e de aplicação, permitindo a comparação da especificação de um nível com a especificação do nível imediatamente superior, e a validação das transformações dos padrões de projeto utilizados.
7. Recuperar Especificações dos Padrões: Processo de recuperar do repositório, as especificações dos padrões, necessárias para cada validação de modelo que será realizada.
8. Analisar Modelos: Processo de comparar as especificações (modelos de negócio, modelos de aplicação ou modelos resultantes da conversão da implementação do componente) com as especificações dos padrões obtidas do repositório de padrões.
9. Armazenar Resultados: Processo de armazenamento dos resultados obtidos no processo de análise.

4.2.2 Papéis e Responsabilidades Dentro do Ambiente

Considerando o diagrama de atividades apresentado na seção 4.2.1. Os papéis envolvidos no ambiente de certificação relacionados com cada macro-atividade são:

- Arquiteto de Negócios: responsável pela construção das especificações dos padrões utilizados nos modelos de negócio;
- Arquiteto de Aplicação: responsável pela construção das especificações dos padrões utilizados nos modelos de aplicação dos componentes.
- Administrador do Repositório: Armazena os modelos no repositório de padrões. É de sua responsabilidade manter atualizado o catálogo dos padrões utilizados no respectivo ambiente de desenvolvimento;
- Gerente de Reutilização: responsável pela interação com as macro-atividades de reconstrução de arquitetura e análise. Executa as seguintes atividades:
  - Converte o código dos componentes em modelo de implementação;
  - Analisa as especificações (modelos de negócio, modelos de arquitetura ou modelos recuperados dos códigos dos componentes) pela comparação com as especificações dos padrões obtidas do repositório de padrões. É de sua responsabilidade analisar todos os modelos dos componentes desenvolvidos, antes da liberação desses componentes para consumo;

Na figura 4.3 temos um novo diagrama de atividades, agora organizado pelos papéis envolvidos. Essa nova organização do diagrama de atividades permite uma visualização do escopo total das responsabilidades de cada papel dentro do ambiente de certificação.

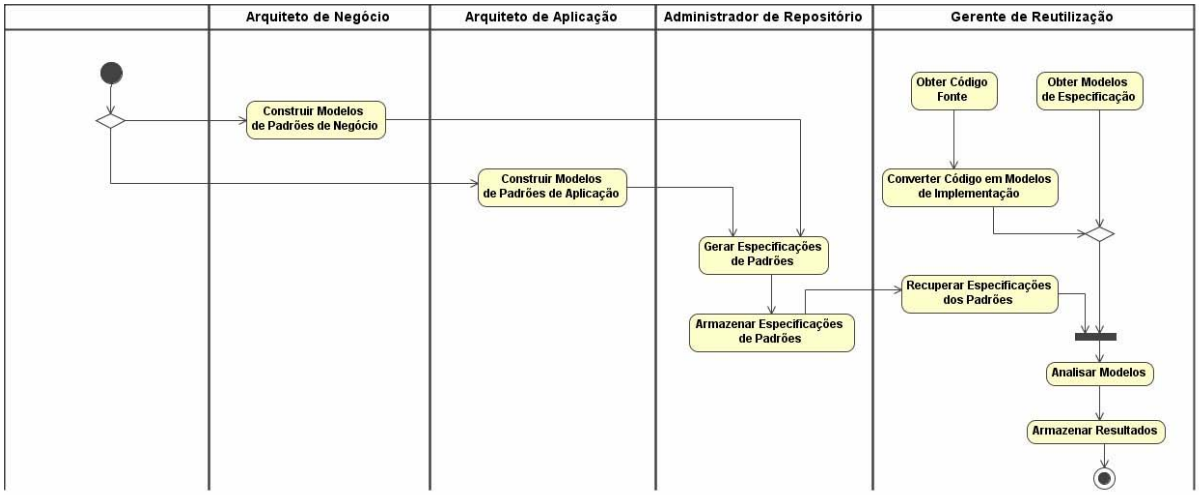


Figura 4.3: Diagrama de Atividades organizado por papel

Esse modelo de atribuição de responsabilidades pode ser alterado de acordo com as características e necessidades de cada ambiente particular.

4.2.3 Diagrama de Casos de Uso

O diagrama de casos de uso (figura 4.4) reflete o envolvimento dos arquitetos de negócios e componentes, do administrador do repositório e gerente de reutilização nas principais funcionalidades existentes no ambiente de certificação.

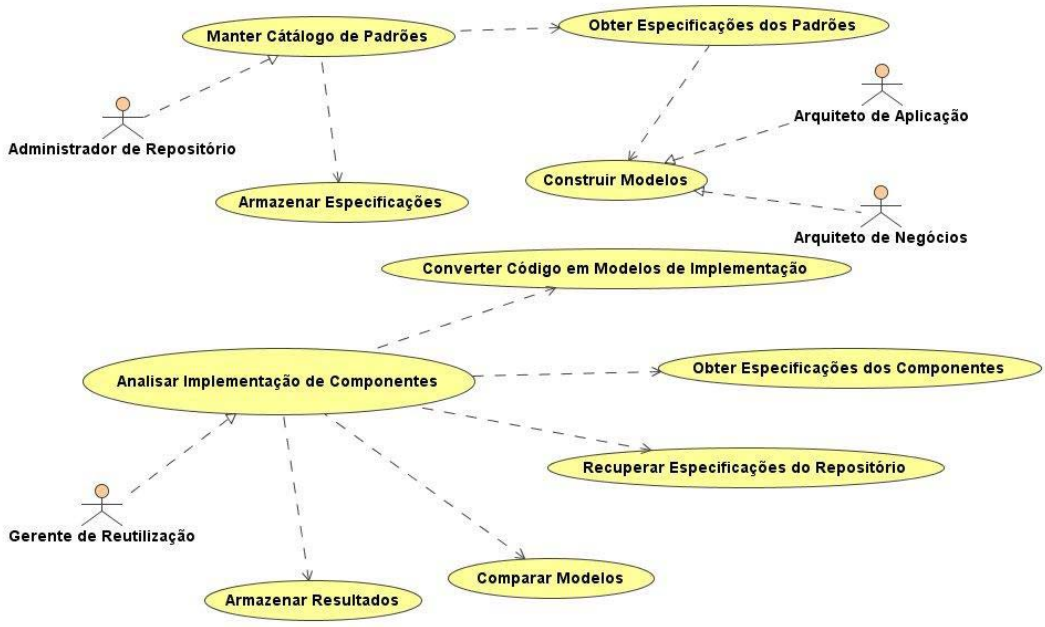


Figura 4.4: Diagrama de Casos de Uso do Sistema

- **Manter Catálogo de Padrões:** Esse caso de uso descreve a interação do administrador do repositório com o sistema na atualização das especificações dos padrões utilizados no respectivo ambiente de desenvolvimento. É subdividido nos casos de uso:
  - **Obter Especificações dos Padrões:** Consiste na exportação dos modelos dos padrões de projeto em um padrão de intercâmbio de modelos pela ferramenta de modelagem utilizada. Depende do caso de uso:
  - **Construir Modelos**: Consiste na construção dos modelos das especificações dos padrões, pelos arquitetos de negócio e aplicação;
  - **Armazenar Especificações**: Esse caso de uso é executado pelo sistema de certificação. Lê os modelos em um padrão de intercâmbio de modelos e armazena no repositório de padrões;
- **Analisar Implementação de Componentes**: Esse caso de uso descreve a interação do gerente de reutilização com o sistema, na certificação da utilização dos padrões apropriados nos modelos de desenvolvimento de um determinado componente. Depende dos casos de Uso:
  - **Converter Código em Modelos de Especificações**: Esse caso de uso é executado pelo sistema de certificação. Lê o código do componente, identifica os elementos da linguagem de código e faz a conversão para os elementos da linguagem de modelagem utilizada;
  - **Obter Especificações dos Componentes:** Esse caso de uso é executado pelo sistema de certificação, e consiste em ler os modelos de especificação dos componentes em um formato padrão de intercâmbio de modelos (PIMs e

PSMs), que correspondem aos modelos de negócio e de aplicação, para serem comparados e certificados.

- **Recuperar Especificações do Repositório:** Esse caso de uso é executado pelo sistema de certificação. Obtém do repositório de padrões**,** as especificações armazenadas dos padrões identificados no modelo de especificação do componente que está sendo certificado.
- **Comparar Especificações**: Esse caso de uso é executado pelo sistema de certificação. Compara os modelos das especificações dos componentes com os modelos dos padrões utilizados.
- **Armazenar Resultados**: Esse caso de uso é executado pelo sistema de certificação. Armazena os resultados da comparação dos modelos como diagnóstico do processo de certificação do respectivo componente. Esses resultados armazenados poderão ser utilizados para elaboração de relatórios de análise da situação do componente avaliado.

4.3. Arquitetura Conceitual do Ambiente de Certificação

Para atender os requisitos definidos nas seções anteriores, foi construída a arquitetura conceitual da solução (figura 4.5), com a definição da estrutura do sistema computacional utilizado.

Dentro dessa arquitetura identificamos que as funcionalidades necessárias no ambiente de certificação proposto coincidem com funcionalidades da MDA, descritas na seção 2.1.5 (figura 4.6).

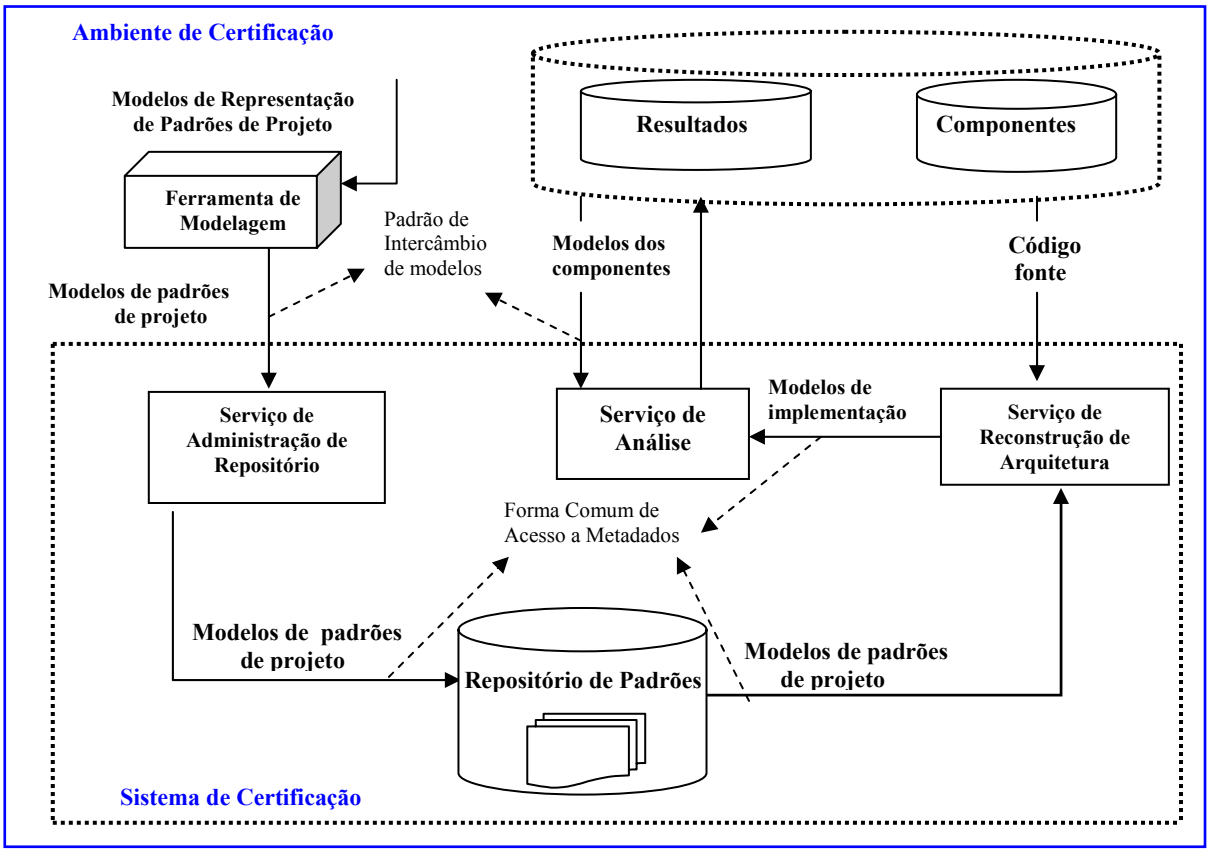


Figura 4.5 : Arquitetura Conceitual

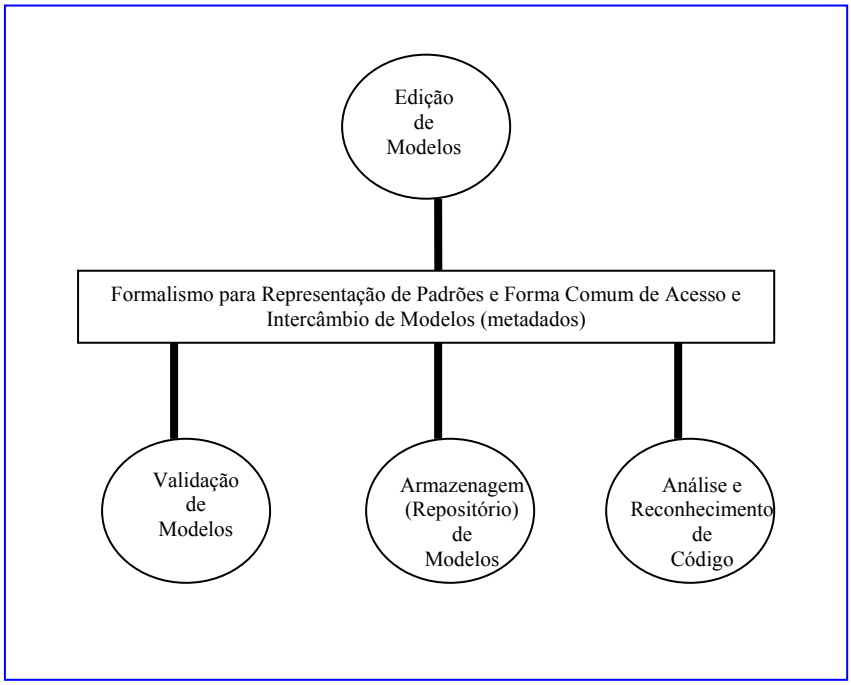


Figura 4.6: Funcionalidades do ambiente de certificação proposto

Essas funcionalidades são:

1. Edição de Modelos: utilizada na construção dos modelos de representação dos padrões de projeto e dos modelos de desenvolvimento dos componentes e aplicações. Precisa exportar os modelos para uma forma comum de intercâmbio de modelos (metadados);
2. Formalismo para Representação de Padrões: uma linguagem de especificação capaz de representar padrões, que seja formal o suficiente para ser compreendida e processada por computadores e flexível o suficiente para ser capaz de representar padrões independentes e dependentes de plataforma, tanto os padrões já divulgados em publicações específicas (ALUR, 2001) (MARINESCU, 2002) (GAMMA, *et al*, 1995) (BUSCHMANN, 1996), (ARLOW, NEUSTADT, 2003), quanto qualquer padrão novo que venha a se criado para atender as necessidades de um domínio ou aplicação específica;
3. Forma Comum de Acesso e Troca de Modelos (metadados): utilizada para acesso, manipulação e intercâmbio de modelos entre os diferentes ambientes do sistema. Interoperabilidade e integração requerem API´s de acesso comum a metadados e formato comum de intercâmbio. As API´s de acesso comum a metadados de qualquer modelo MOF são derivadas através de mapeamento de linguagem. Dado um modelo MOF, a API de mapeamento gera API´s de linguagens específicas para criação, acesso e atualização de instâncias (DIRCKZE, BAISLEY, IYENGAR, 2002);
4. Validação de Modelos: verifica e valida a correta construção dos modelos (negócio, aplicação e código) dos componentes e aplicações, com relação à utilização dos padrões de projeto (Análise de Modelos);
5. Repositório de Padrões: É utilizado para armazenar os modelos dos padrões de projeto. Importa e exporta os modelos na forma comum de intercâmbio de modelos

(metadados). Os modelos são acessados no repositório utilizando a forma comum de acesso a modelos (metadados) (Administração de Repositório);

6. Análise e Reconhecimento de Código: É utilizada para interpretar o código implementado dos componentes, identificando os elementos da linguagem de implementação utilizados e converter a definição desses elementos na linguagem de modelagem utilizada (Reconstrução de Arquitetura).

   Reconstrução de arquitetura é um processo pelo qual as arquiteturas de sistemas são obtidas da sua implementação. Entre as diversas abordagens existentes, a reconstrução de arquitetura para avaliar conformidade com a arquitetura construída e a arquitetura documentada (STOERMER, O'BRIEN e VERHOEF, 2002) é a abordagem que será utilizada no sistema de certificação desenvolvido. Essa abordagem é chamada de Padrão de Arquitetura Executada: padrão que cobre o problema de consistência entre a arquitetura construída e a arquitetura projetada de um sistema.

4.4. Características do Sistema Computacional do Ambiente de Certificação

Para automatizar as atividades do ambiente de certificação, é necessária a utilização de um sistema computacional de apoio. Considerando os requisitos e as macro-atividades executadas no ambiente de certificação, foram identificadas as características funcionais a serem incorporadas nesse sistema de certificação.

O sistema computacional desenvolvido pode ser utilizado para certificar modelos de componentes e aplicações em qualquer plataforma, com exceção da certificação dos modelos de implementação recuperados do código dos componentes. Nesse caso ele é específico para a plataforma J2EE.

4.4.1. Especificação do Sistema de Certificação

A tecnologia Java (GOSLING et al, 2005) (JSR244, 2005) foi escolhida para a implementação do ambiente computacional por ser considerada a mais adequada no contexto da pesquisa. Existem diversos padrões e propostas de padrões da OMG e do JCP, relacionadas com o ambiente orientado a modelos da MDA, como o JMI (JSR40, 2002), JOLAP (JSR69, 2003), JDMAPI (JSR73, 2005), metamodelos e perfis UML para Java e EJB (OMG, 2004) (JSR26, 2001).

A solução está inserida num ambiente de desenvolvimento orientado a modelos, dessa forma foram escolhidas tecnologias relacionadas com a MDA para a representação dos diversos elementos que compõem a arquitetura conceitual apresentada na figura 4.5.

Na composição tecnológica dessa arquitetura temos:

1. Padrão de Intercâmbio e Forma Comum de Acesso a Dados: Os padrões escolhidos foram o XML *Metadata Interchange* (XMI) e o JMI (JSR40, 2002), ambos são mapeamentos do MOF para a linguagem Java e para XML respectivamente. A API Java para metadados conhecida como JMI fornece um mapeamento nativo da linguagem Java das interfaces do MOF. Permite que aplicações Java especifiquem, armazenem, acessem e troquem metadados usando serviços padrões de metadados. Resulta numa forma mais simples e intuitiva de programação do que com o mapeamento para CORBA/IDL, mapeamento definido na própria especificação do MOF. Junto com XMI, JMI fornece uma completa estrutura de acesso e troca dinâmica de metadados independente de plataforma e de fabricante (ECKERSON, MANES, 1999). A implementação do JMI permite uma geração programática de pura interface Java para acesso em repositórios baseados em metamodelos MOF e suas instâncias. Isso significa que implementações Java de qualquer serviço de metadados baseado em MOF podem expor tanto as interfaces dos metamodelos genéricos quanto

dos metamodelos específicos derivados das regras de mapeamento das interfaces do MOF. Os clientes Java possuem portabilidade completa de acesso aos serviços de metadados via JMI (POOLE, J., *et al,* 2003). A semântica de qualquer sistema, que tenha sido construído através de modelos baseados em MOF, pode ser recuperada e manipulada (MOSHER, 2005). JMI conduz a computação dirigida por modelos para a tecnologia J2EE, sendo atualmente, uma tecnologia complementar para a arquitetura J2EE. JMI é tanto uma estrutura para facilitar o desenvolvimento de aplicações centradas em processos e modelos de negócio, quanto uma base para ferramentas de integração (DIRCKZE, BAISLEY, IYENGAR, 2002).

2. Repositório de Padrões: Foi escolhido o repositório de metamodelos MDR, compatível com MOF/JMI**.** Está habilitado a carregar qualquer metamodelo MOF (descrição de metadados) e armazenar instâncias desses metamodelos (os metadados em conformidade com o metamodelo). Metamodelos e metadados podem ser importados e exportados do MDR usando XML no padrão XMI. Metadados no repositório podem ser manipulados programaticamente usando API JMI reflexiva ou específica do metamodelo (MATULA, 2003).
3. Serviço de Administração de Repositório: responsável pela importação em XMI dos modelos construídos pelas ferramentas de modelagem e por armazená-los no repositório no padrão JMI;
4. Serviço de Reconstrução de Arquitetura: responsável pela reconstrução da arquitetura dos componentes, recuperando a partir do código o seu respectivo modelo UML descrito em JMI;
5. Serviço de Análise: responsável pela análise e certificação dos modelos dos componentes (de negócio, de arquitetura e de implementação) contra os modelos dos

padrões utilizados, e recuperados do repositório. A análise é executada pelo acesso e manipulação dos modelos no padrão JMI;

4.4.2. Serviços do Sistema de Certificação

Nesta seção será apresentada a descrição das especificações tecnológicas dos serviços do sistema de certificação desenvolvido. No Apêndice A, são apresentadas as descrições de todas as bibliotecas de software utilizadas na implementação desse sistema.

4.4.2.1 Serviço de Administração do Repositório

Este serviço é utilizado na execução da macro-atividade "Administração de Repositório" apresentada na seção 4.2 e tem a responsabilidade de importar os modelos dos padrões para o repositório MDR. Isso é feito através da leitura, da validação da formação correta e do armazenamento no repositório, dos modelos de representação dos padrões exportados pelo editor de modelo utilizado no padrão XMI, padrão de intercâmbio de modelos utilizado.

Os modelos são lidos no formato XMI**,** através da implementação MDR do javax.jmi.xmi.XmiReader (JSR40, 2002), interface da especificação JMI para leitura de modelos no padrão XMI.

Após a leitura dos modelos, é realizada a validação da formação correta dos modelos quanto ao formalismo de representação de padrões utilizado apresentado na seção 4.4.2.1.1. Os modelos em conformidade são aceitos e armazenados no repositório através de uma variável do tipo org.omg.uml.UmlPackage[1] que representa uma instância do metamodelo MOF da UML 1.4 dentro do repositório MDR. Os modelos são organizados no repositório de acordo com o seu tipo (PNIP, PAIP e PAPE). Cada tipo de modelo é armazenado dentro de extensões (*extents*)[2] específicas, chamadas de MODELOS_UML_PNIP, MODELOS_UML_PAIP, MODELOS_UML_PAPE, para os tipos PNIP, PAIP e PAPE respectivamente. O repositório

[1] uml-1.4.jar descrita no Apêndice A
[2] API do MDR (http://www.netbeans.org/download/dev/javadoc/MdrAPIs

MDR é estruturado por extensões (*extents*). Cada extensão é considerada como um subdiretório[1] dentro do repositório. Esse subdiretório é determinado por um metamodelo MOF e suas instâncias. Na implementação deste serviço, são utilizados três subdiretórios para o metamodelo da UML, um para cada tipo de padrão que está sendo representado.

4.4.2.1.1. O Formalismo de Representação de Padrões Utilizado

Foram analisados diversos trabalhos relacionados com representação de padrões, os quais utilizam diferentes instrumentos de representação e possuem diferentes escopos de aplicação. Das abordagens encontradas, optamos por incorporar no desenvolvimento da solução as abordagens de modelagem de papéis e a de utilização de mecanismos de extensão da UML através de perfis específicos para uma determinada finalidade.

BACHMAN e DAYA (1977) apresentaram o conceito de papéis na modelagem de dados. Esse conceito permite que a representação de diferentes papéis do mundo real seja realizada por uma entidade. A partir daí, vários trabalhos e pesquisas foram desenvolvidos utilizando esse conceito. Dos trabalhos analisados sobre representação de padrões de projeto utilizando o conceito de modelagem baseada em papéis, os principais são: metamodelos de colaboração de papéis participantes de padrões e restrições em OCL, (LE GUENNEC, SUNYE, JEZEQUEL, 2000); Diagramas *Role-Elements of a Pattern* (REP - elementos papéis de um padrão) (MONTES, VELA, 2003); *Role-Based Metamodeling Language* (RBML) *Research Group*[2] (FRANCE, *et al,* 2002) (FRANCE, *et al,* 2003) (FRANCE, *et al,* 2004);

Relacionados com a abordagem de utilização dos mecanismos de extensão da UML, foram analisados os trabalhos de FONTOURA, PREE e RUMPE (2000) (2001) sobre um perfil UML chamado UML-F para representação de estruturas (*frameworks*) de arquitetura e de

[1] "key features" – http://mdr.netbeans.org
[2] http://www.cs.colostate.edu/~dkkim/ReuseResearch /main.html)

DONG e YANG (2002) (2003) sobre um perfil UML para identificação e visualização de padrões de projeto em diagramas UML.

As principais notações formais utilizadas para representar padrões de projeto, apresentadas nas seções 2.4.3 e 2.4.4 foram diagramas de classe e seqüência da UML. Os modelos de papéis podem ser considerados metamodelos, pois podemos enxergá-los como se estivessem numa camada acima dos modelos das instâncias que utilizam esses papéis, e a utilização dos mecanismos de extensão da UML**,** estereótipos e etiquetas (*taggedValues*), através de perfis UML específicos, nos permite lidar num mesmo modelo com duas camadas de informação. A utilização de um perfil UML na construção de modelos, possibilita a representação das informações da camada de metamodelos.

Dessa forma os diagramas de classe dos papéis existentes num determinado padrão e os diagramas de seqüência com as interações entre esses papéis, podem ser considerados como os metamodelos dos padrões que se deseja representar e a utilização de perfis UML de representação e identificação de padrões nos permite lidar no mesmo modelo, com as informações dos metamodelos dos padrões e com as informações dos modelos de instanciação desses padrões.

A escolha da modelagem de papéis utilizando o metamodelo da UML estendido por perfis UML específicos, em detrimento da definição de um novo metamodelo específico de representação de padrões, tem como vantagens, a vasta utilização da UML pelos arquitetos e desenvolvedores de sistemas de software, tendo se tornado um padrão de fato como linguagem de modelagem, e a existência de um grande número de ferramentas (editores de modelos) para UML. Além de também estar em conformidade com a abordagem da MDA na construção da solução, já que perfil UML é um dos padrões da OMG incorporados na MDA.

O sistema de certificação utilizará dois perfis UML. No Apêndice B são apresentados todos os detalhes da especificação do perfil UML para representação de padrões e do perfil UML para utilização de padrões na construção dos modelos dos componentes e aplicações.

4.4.2.2 Serviço de Reconstrução de Arquitetura

Esse serviço tem a responsabilidade de recuperar o modelo UML de implementação do componente a partir de seu código Java. Para certificação da utilização correta de padrões de projeto, e da transformação correta do modelo PSM de último nível de abstração no código implementado, precisamos reconstruir a arquitetura do código fonte e compará-la com a sua especificação (modelo PSM).

Existem muitas abordagens e ferramentas de reconstrução de arquiteturas para dar suporte aos padrões de reconstrução. No desenvolvimento deste sistema de certificação será utilizada a abordagem de reconstrução através de linguagens de consulta. Nessa abordagem estão incluídos os processos de reconhecimento de linguagem, onde são utilizados *Abstracts Syntax Trees* (ASTs). Analisadores de código fonte de um sistema geram ASTs, específicas para cada linguagem de programação.

Conforme vimos na seção 2.1.5, softwares analisadores sintáticos de código que lêem os arquivos com código fonte baseado em texto e convertem para a forma baseada em modelo é uma das necessidades do ambiente da MDA. Da mesma forma, na construção do ambiente computacional de certificação, teremos que incluir essa categoria de software, pois precisamos transformar os códigos fonte dos componentes na forma de modelo para análise e validação.

Em um ambiente de desenvolvimento orientado a modelos, precisamos especificar como a transformação de um modelo para outro é realizada. Através de regras bem definidas, chamadas regras de transformação, especificamos as transformações entre modelos.

Ferramentas específicas de transformação utilizam essas regras para saber como executar essas transformações.

Já existem diversas regras definidas com relação à transformação de modelos UML em código fonte de uma determinada linguagem de programação (HARRISON W., BARTON C., RAGHAVACHARI, 2000), mas não são suficientes para esse trabalho, pois não contemplam mapeamento para estereótipos e etiquetas, para elementos relacionados com os aspectos de comportamento de modelos UML, para elementos dos diagramas de seqüência, bem como um mapeamento direto para associações de cardinalidade n. Dessa forma, foram acrescentadas algumas regras para contemplar esses mapeamentos, possibilitando a recuperação do código dos componentes e aplicações, de todas as associações, estereótipos e etiquetas (*tags),* elementos de diagramas de seqüência, utilizados na construção dos modelos de especificação desses componentes e aplicações na plataforma tecnológica utilizada.

No desenvolvimento do serviço de reconstrução de arquitetura utilizamos a ferramenta de reconhecimento de linguagem ANTLR[1] e uma adaptação da classe *Modeller* da ferramenta JavaRE[2].

Essa adaptação consistiu na substituição da biblioteca NSuml[3] pela utilização do repositório MDR e das API's JMI do metamodelo da UML 1.4. Essa classe *Modeller* e a ferramenta ANTLR também são utilizadas na engenharia reversa do ArgoUML[4] e Poseidon[5]**.**

A ferramenta ANTLR utiliza uma AST específica para a gramática da linguagem Java (PARR. 1999). No sistema de certificação desenvolvido, a descrição da gramática foi atualizada da versão Java 1.2 para a versão Java 5, e adaptações necessárias foram realizadas nessa nova versão para gerar modelos UML, adicionando a recuperação de anotações, associações e elementos de modelo utilizados nos diagramas de seqüência, não contemplados

[1] http://www.antlr.org/
[2] http://javare.sourceforge.net/index.php
[3] http://nsuml.sourceforge.net/
[4] http://argouml.tigris.org/
[5] http://www.gentleware.com/

na versão da ferramenta JavaRE. Através das classes JavaLexer, JavaRecognizer, JavaTreeParser, JavaTokenTypes, JavaTreeParserTokenTypes geradas pelo ANTLR, fazemos o reconhecimento da linguagem, identificamos os elementos a serem mapeados para UML e através da classe *Modeller*, adicionamos esses elementos num atributo org.omg.uml.modelmanagement.Model[1], representação de um modelo UML na API JMI da UML 1.4. Após todo o código Java ser convertido, temos o modelo UML completo.

Conforme já foi citado na seção 4.3.3, foram definidas regras de mapeamento adicionais da UML para o código fonte na linguagem Java. Essas regras permitem a recuperação de todas as associações, estereótipos e etiquetas (*tags),* elementos dos diagramas de seqüência, utilizados na construção dos modelos de plataforma específica (MAPE) da plataforma J2EE, do código dos componentes:

- Na implementação dos componentes, os estereótipos e etiquetas (*tags)*, utilizados para identificar os elementos de modelo participantes de padrões de projeto, são transformados em anotações (*Annotation Types*). Recurso adicionado no J2SE 5.0;
- As associações com cardinalidade n, são transformadas em atributos de algum tipo que implemente a interface *Collection* da linguagem Java*,* utilizando o recurso *Generics.* Recurso também adicionado no J2SE 5.0, que permite determinar o tipo dos objetos existentes numa coleção.
- As chamadas de métodos da linguagem Java são mapeadas para os elementos *Stimulus* e *CallAction* utilizados nos diagramas de seqüência;
- As instâncias de uma determinada classe são mapeadas para os elementos *Stimulus* e *CreateAction* utilizados nos diagramas de seqüência;

[1] uml-1.4.jar descrita no Apêndice B

O recurso de *Annotation* foi adicionado no J2SE 5.0, no pacote *java.lang.annotation*. Esse novo recurso é resultado de uma nova proposta do JCP identificada como JSR-000175 *A Metadata Facility for the Java Programming Language* (JSR 175, 2004)**.**

As anotações trazem uma capacidade de lidar com metadados muito necessária à linguagem Java. A utilização de metadados tem sido uma tendência muito forte ultimamente na programação Java. Os metadados podem ser usados para criar documentação, para capturar dependências no código, e mesmo executar controle rudimentar de compilação. Uma solução parcial para metadados, como o *XDoclet*[1], adiciona essas características à linguagem Java e tem sido parte do cenário da programação em Java desde então (MCLAUGHLIN, 2004).

Na proposta do JCP para uma nova versão de EJB contida no JSR 220: *Enterprise JavaBeans™ 3.0* estão contidas várias utilizações de anotações. Na proposta de especificação do EJB 3.0, (EJB, 2005) estão relacionadas anotações de metadados para várias finalidades, como: especificar o tipo de EJB; especificar a acessibilidade local ou remota do EJB; gerar as interfaces dos EJB; especificar atributos de transação; especificar segurança e métodos de permissão; especificar referências e recursos de ambiente e outras. Parte do trabalho da especificação ainda em andamento tem o objetivo de utilizar as anotações para especificar metadados de mapeamento objeto/relacional, o que eliminaria a utilização de arquivos XML, com esses descritores.

No sistema de certificação, as anotações são utilizadas para tornar possível identificar os padrões de projeto existentes no código gerado e quais são os elementos participantes desses padrões. Foram definidos três tipos de anotações que substituirão o papel dos estereótipos e etiquetas (*taggedValues*) dos modelos. Essas anotações estão demonstradas abaixo:

[1] (http://xdoclet.sourceforge.net/xdoclet/index.html)

- **Tipo de anotação para as classes envolvidas em padrões.**

```
// ClassePadrao.java
@Target(ElementType.TYPE)
public @interface ClassePadrao {
   String[] padraoExecutado();
}
```

- **Tipo de anotação para as operações envolvidas em padrões.**

```
// OperacaoPadrao.java
@Target(ElementType.METHOD)
public @interface OperacaoPadrao {
   String[] padraoExecutado();
}
```

- **Tipo de anotação para os atributos envolvidos em padrões.**

```
// AtributoPadrao.java
@Target(ElementType.FIELD)
public @interface AtributoPadrao {
   String[] padraoExecutado();
}
```

4.4.2.3 Serviço de Análise de Modelos

Esse serviço tem a responsabilidade de analisar e certificar os modelos (de negócio, de arquitetura e recuperados do código Java). Utiliza a API JMI para acessar os metadados, definições dos elementos de modelo contidos em org.omg.uml.modelmanagement.Model, identificando os padrões utilizados através dos estereótipos e etiquetas (*tags*) existentes nos modelos ou convertidos das anotações encontradas nos códigos dos componentes. Os padrões identificados são recuperados do repositório MDR, pela busca de uma instância do

metamodelo UML 1.4 (org.omg.uml.UmlPackage)[1] com o nome e tipo do padrão de projeto. Os modelos são comparados através da comparação das interfaces JMI do metamodelo UML 1.4 (forma comum de acesso de modelos definida para o sistema de certificação) que representam os elementos existentes nos modelos comparados. Existem dois tipos de análise fornecidos pelo serviço:

- Análise dos Padrões Utilizados: Análise do modelo importado quanto à utilização correta dos padrões existentes no modelo e identificados através dos estereótipos e etiquetas (*tags*) específicas para o respectivo nível de abstração. Compara a construção dos padrões identificados no modelo, com a representação desses padrões armazenados no repositório, certificando se estão em conformidade;
- Análise das Transformações de Padrões: Análise do modelo importado quanto às definições de transformação de padrões existentes no modelo fonte do nível de abstração imediatamente superior. Nesse caso os padrões utilizados e identificados, através dos estereótipos e etiquetas (*tags*), no modelo que está sendo analisado, são comparados com as definições dos mapeamentos dos padrões do modelo original no nível de abstração anterior, certificando se as transformações de padrões entre os dois níveis estão em conformidade;

Podemos certificar modelos MNIP, MAIP e MAPE (modelos recuperados do código J2EE, segundo regras de transformação específicas para a plataforma J2EE, ou modelos de arquitetura gerados a partir de transformações sucessivas originadas em modelos de negócio independentes de plataforma, MNIP, segundo um mapeamento de padrões definido pelos arquitetos do ambiente de desenvolvimento).

[1] uml-1.4.jar descrita no Apêndice B

4.5. Avaliação do Ambiente de Certificação Proposto

A nossa proposta de certificação inserida num ambiente orientado a modelos pode ser considerada como um método semelhante a VV&A. O foco do sistema de certificação proposto é certificar os modelos construídos no desenvolvimento de componentes e aplicações em todo ciclo de desenvolvimento, fornecendo credibilidade suficiente para os modelos certificados de forma que o processo de desenvolvimento prossiga para os próximos níveis de abstração.

Os trabalhos cujas abordagens possuem mais pontos de identificação com a nossa proposta de certificação são realmente o KCS e o OptmailJ. Serão apresentadas a seguir as análises comparativas específicas para cada um:

- KCS:

  Com relação às semelhanças temos: a manutenção de um repositório de padrões sobre o conhecimento de um determinado domínio; a utilização de uma representação formal de padrões; e a utilização de uma linguagem intermediária comum de representação para onde as linguagens são convertidas para análise;

  Com relação às diferenças temos: a não utilização pelo KCS dos padrões de metamodelagem da MDA na especificação da sua linguagem de representação de padrões e da sua linguagem intermediária comum; e não estar inserido no ambiente de desenvolvimento orientado a modelos, não existindo a verificação da interligação e das transformações entre os padrões nos diversos níveis de abstração.

- Optmalj:

  Com relação às semelhanças temos: é totalmente voltado para um ambiente orientado a modelos e baseado em padrões;

  Com relação às diferenças temos: os níveis de abstração são fixos (domínio, aplicação e código); é uma ferramenta proprietária, exigindo que a implantação do ambiente seja feita com a sua utilização; não é específica sobre certificação e sim mais um ambiente de desenvolvimento dentro da MDA.

Encontramos ainda algumas semelhanças nas abordagens dos trabalhos:

- O mecanismo de extensão de atributo de HEDIN (1997): é semelhante à utilização do recurso de anotações no código implementado utilizado no nosso trabalho;
- A ferramenta de engenharia reversa, chamada REV-AS: também recupera os modelos UML correspondentes do código implementado para comparação com a sua especificação, porém o foco não é verificar a utilização de padrões e sim verificar a conformidade da implementação com a especificação.

CAPÍTULO 5

Exemplo de Aplicação: Sistema de Vendas de Rede de Lojas de Departamento

Neste capítulo é apresentado um exemplo de certificação de padrões no desenvolvimento orientado a modelos da MDA, utilizando o sistema de certificação desenvolvido. Para tanto, um ciclo completo de transformação de padrões é apresentado com as suas respectivas certificações, realizadas em cada etapa de transformação. Esse ciclo inicia no primeiro modelo de negócios de nível mais alto de abstração até o modelo de implementação. São demonstrados os mapeamentos realizados que permitem o rastreamento dos padrões utilizados em todos os níveis. A verificação da formação correta dos modelos é feita pela comparação com os modelos dos padrões de projeto utilizados, armazenados no repositório de padrões baseado em metamodelos MOF e construídos utilizando o perfil UML de representação de padrões.

A criação do exemplo foi inspirada nos padrões de arquétipos de ARLOW e NEUSTADT (2003), apresentados na seção 2.4.3, na transformação de modelos PIM em seus respectivos PSMs de KLEPPE, WARMER e BAST (2003), e na arquitetura utilizada pela ferramenta OptimalJ (CRUPI, BAERVELDT, 2005).

5.1. Cenário: Rede de Lojas de Departamento

O cenário do exemplo de aplicação não corresponde a um exemplo real e representa o processo de certificação do desenvolvimento de um sistema de registro das vendas de uma grande rede de lojas de departamento localizadas em diversas cidades do país. A figura 5.1 apresenta o nosso modelo de domínio, correspondente ao PIM de nível mais alto de abstração no ciclo de desenvolvimento.

Conforme o modelo demonstra, a associação de uma venda com cliente não é obrigatória. Essa associação existe quando a venda realizada é referente a um produto de preço mais

elevado, e cuja forma de pagamento foi parcelada ou que a entrega do produto foi realizada pela loja, exigindo o cadastramento do cliente no sistema.

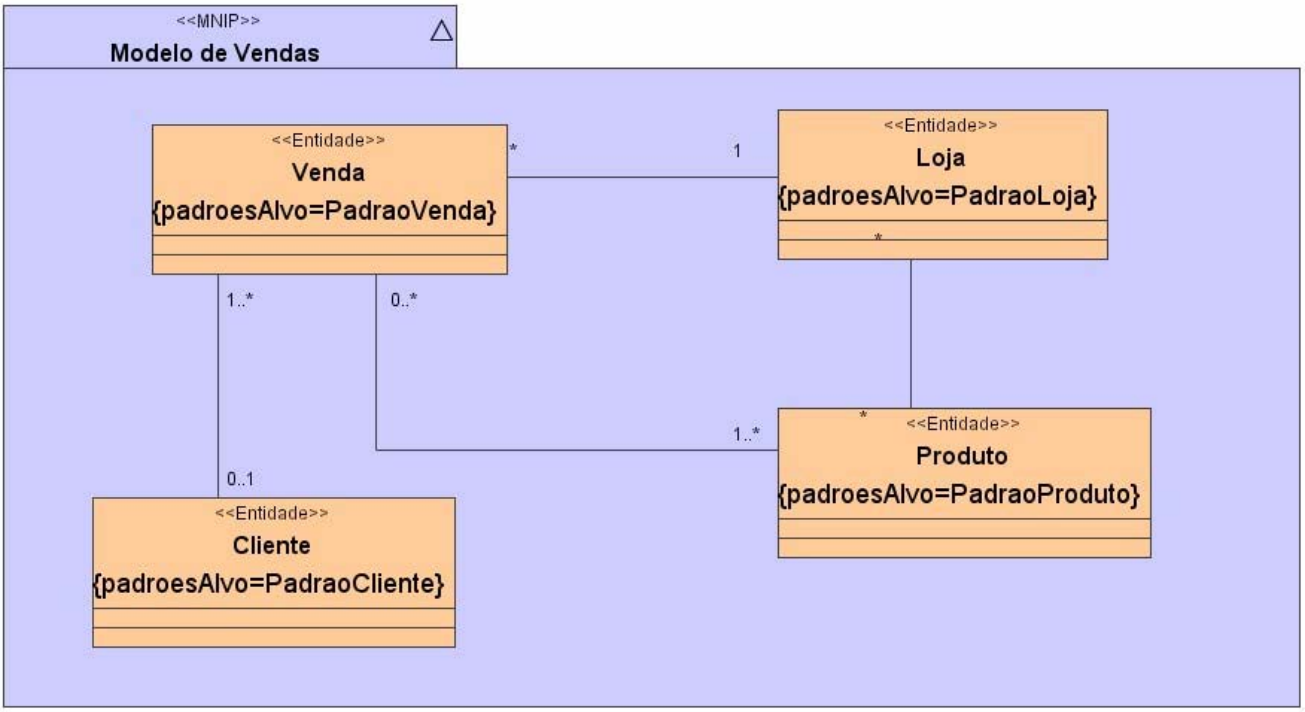


Figura 5.1: PIM: MNIP - Sistema de Vendas

O modelo de negócio do sistema de vendas obedece ao perfil UML para utilização de Padrões apresentado na seção 4.4.2.1.1. Possui o estereótipo <<MNIP>> (modelo de negócio independente de plataforma) com a etiqueta nivelModelo igual a um, indicando o nível mais alto de abstração. Os seus elementos de classe possuem o estereótipo <<Entidade>> indicando que representam o conceito de entidade do domínio do sistema, com a etiqueta padroesAlvo, indicando o padrão que será utilizado na transformação para o modelo do próximo nível de abstração.

5.2. As Transformações de Padrões na Construção do Sistema

Transformamos o modelo de negócio acima, aplicando nossos exemplos de padrões corporativos de negócios, ou padrões de arquétipos de negócios, conforme denominação de

ARLOW e NEUSTADT (2003). Eles representam nossos padrões hipotéticos e bastante simplificados dos conceitos de venda, cliente, loja e produto, apenas para serem utilizados, de uma maneira fácil de ser compreendida, na demonstração da aplicabilidade do ambiente de certificação. O modelo inicial do exemplo do sistema de vendas é refinado através da substituição dos elementos Venda, Cliente, Loja e Produto, pelos padrões de negócios Venda, Cliente, Loja e Produto respectivamente, conforme indicado na etiqueta padroesAlvo associada ao estereótipo Entidade dos seus elementos com papel de classe. As figuras 5.2, 5.3 e 5.4 apresentam os modelos desses padrões.

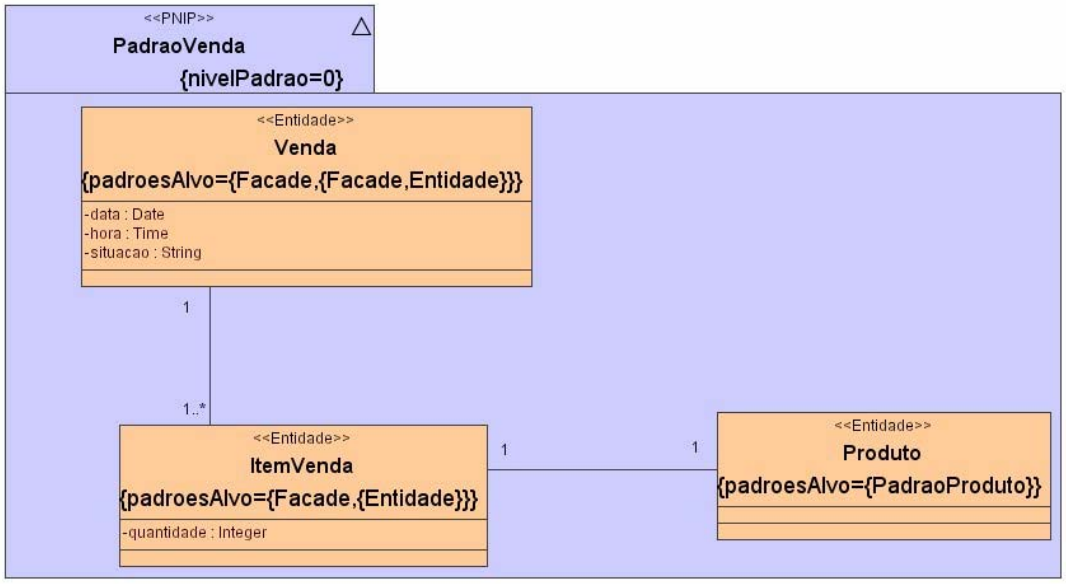


Figura 5.2: Padrão de Negócio Independente de Plataforma: PNIP – Venda

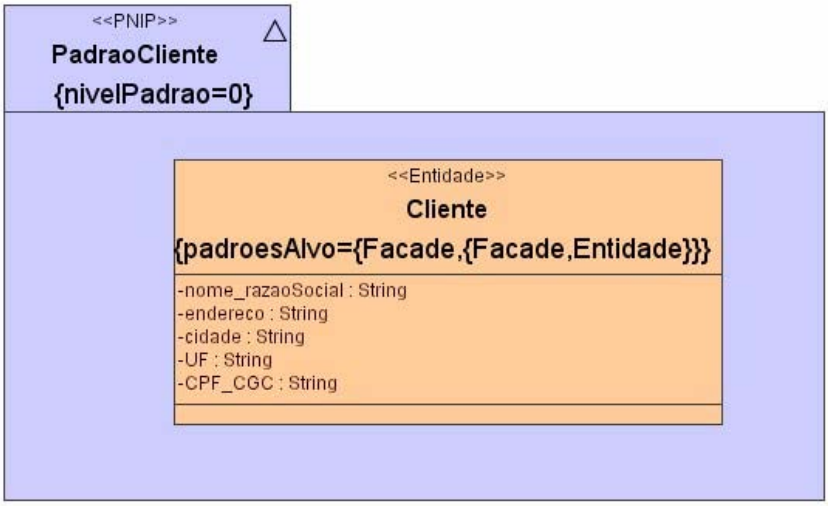


Figura 5.3 : Padrão de Negócio Independente de Plataforma : PNIP – Cliente

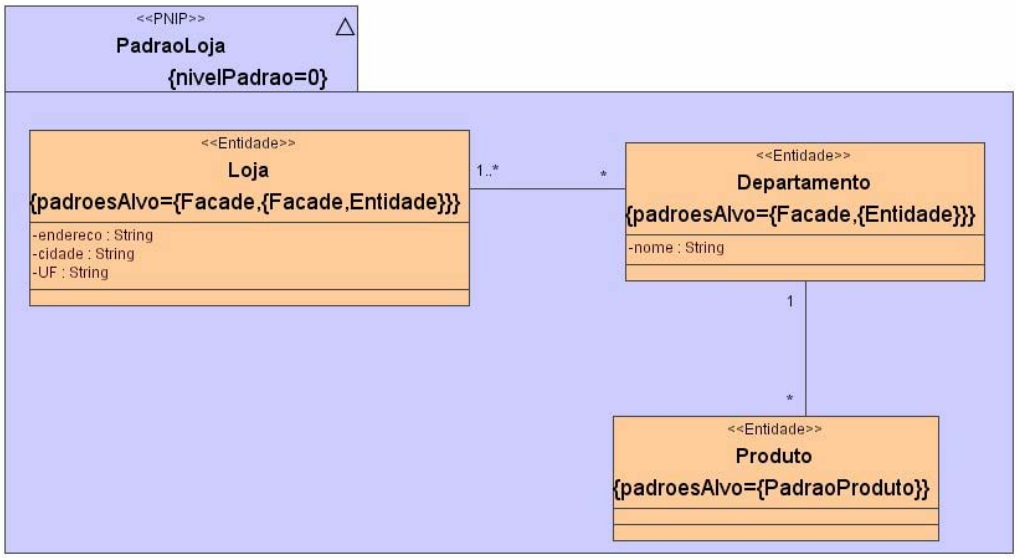


Figura 5.4: Padrão de Negócio Independente de Plataforma: PNIP – Loja

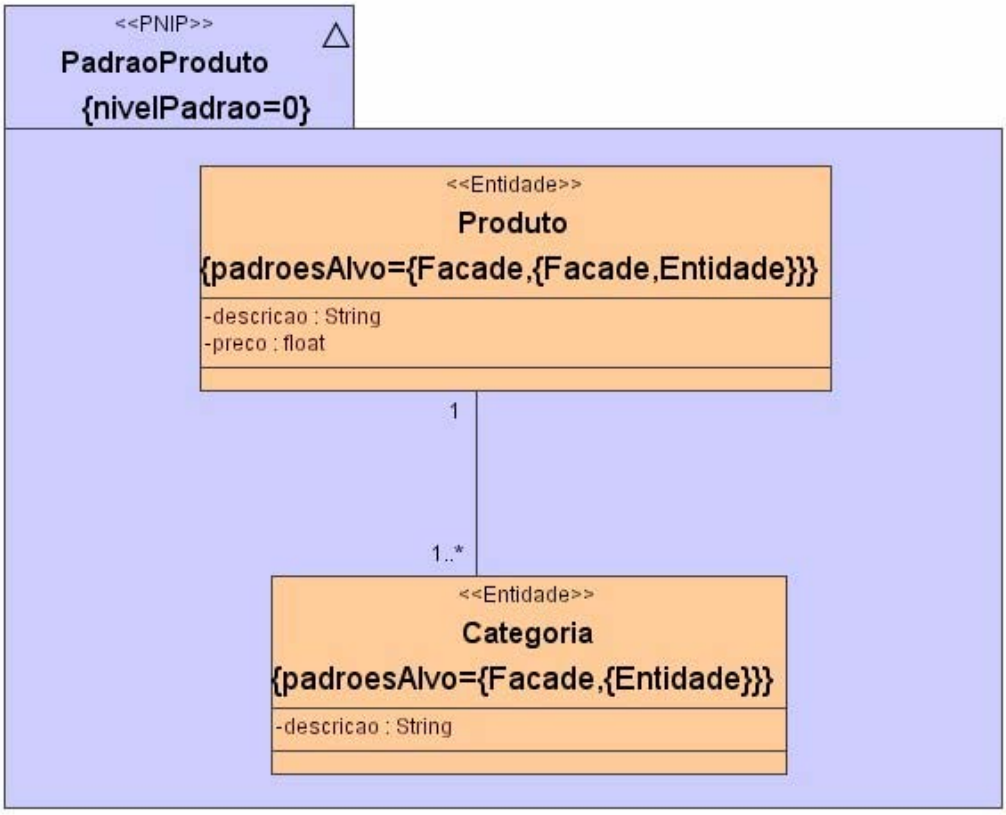


Figura 5.5: Padrão de Negócio Independente de Plataforma: PNIP – Produto

Os quatro modelos de padrões de negócio obedecem ao perfil UML para representação de Padrões descrito na seção 4.4.2.1.1. Todos possuem o estereótipo <<PNIP>> (padrão de negócio independente de plataforma) com a etiqueta nivelPadrao igual a zero, indicando o último nível de refinamento antes da transformação para um padrão <<PAIP>> (padrão de

aplicação independente de plataforma), e todos os seus elementos com papel de classe, possuem o estereótipo Entidade, indicando a representação do conceito de entidade do domínio do padrão, com a etiqueta padroesAlvo indicando os papéis de padrões que podem ser mapeados na transformação desses elementos para o próximo nível de abstração.

No primeiro refinamento, após a aplicação dos padrões de negócios utilizados, geramos um <<MNIP>>, que corresponde a um modelo de negócio mais detalhado, com a etiqueta nivelModelo igual a zero, indicando que o próximo nível será um modelo de aplicação. Nesse modelo ainda não encontramos os detalhes da tecnologia a ser utilizada. A figura 5.6 apresenta esse modelo do sistema, correspondente ao PIM de segundo nível de abstração no ciclo de desenvolvimento.

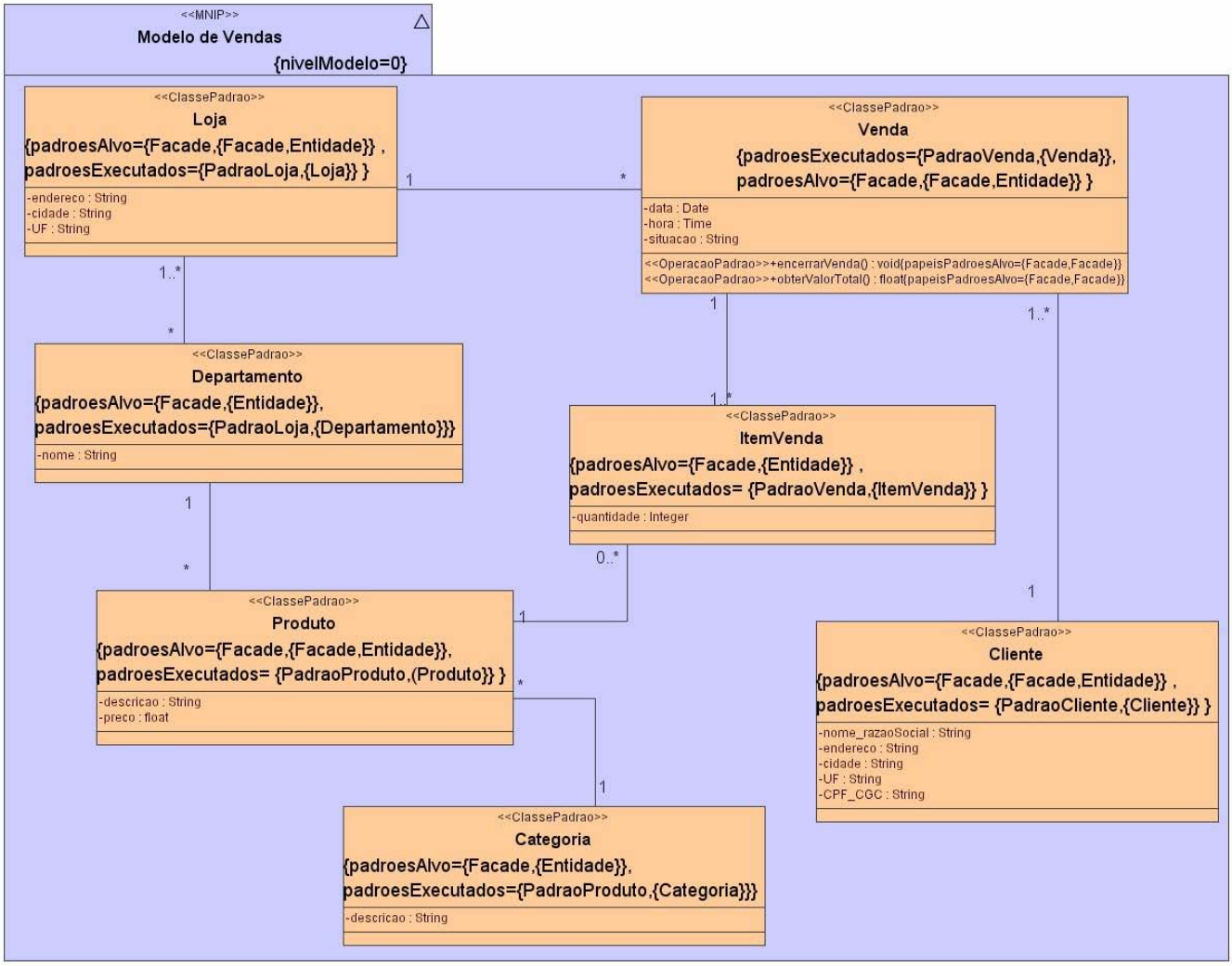


Figura 5.6 : PIM transformado : MNIP - Sistema de Vendas

O padrão de negócio PadraoProduto, aparece mapeado no modelo do sistema de vendas e nos modelos dos padrões PadraoVenda e PadraoLoja. No momento da transformação para o próximo nível de abstração, esse mapeamento é instanciado apenas uma única vez para atender todos os mapeamentos.

A certificação realizada nesse modelo correspondente a certificação da primeira etapa de transformação ou refinamento, resulta nos resultados, gerados pelo sistema de certificação, apresentados nas figuras 5.7, 5.8 e 5.9.

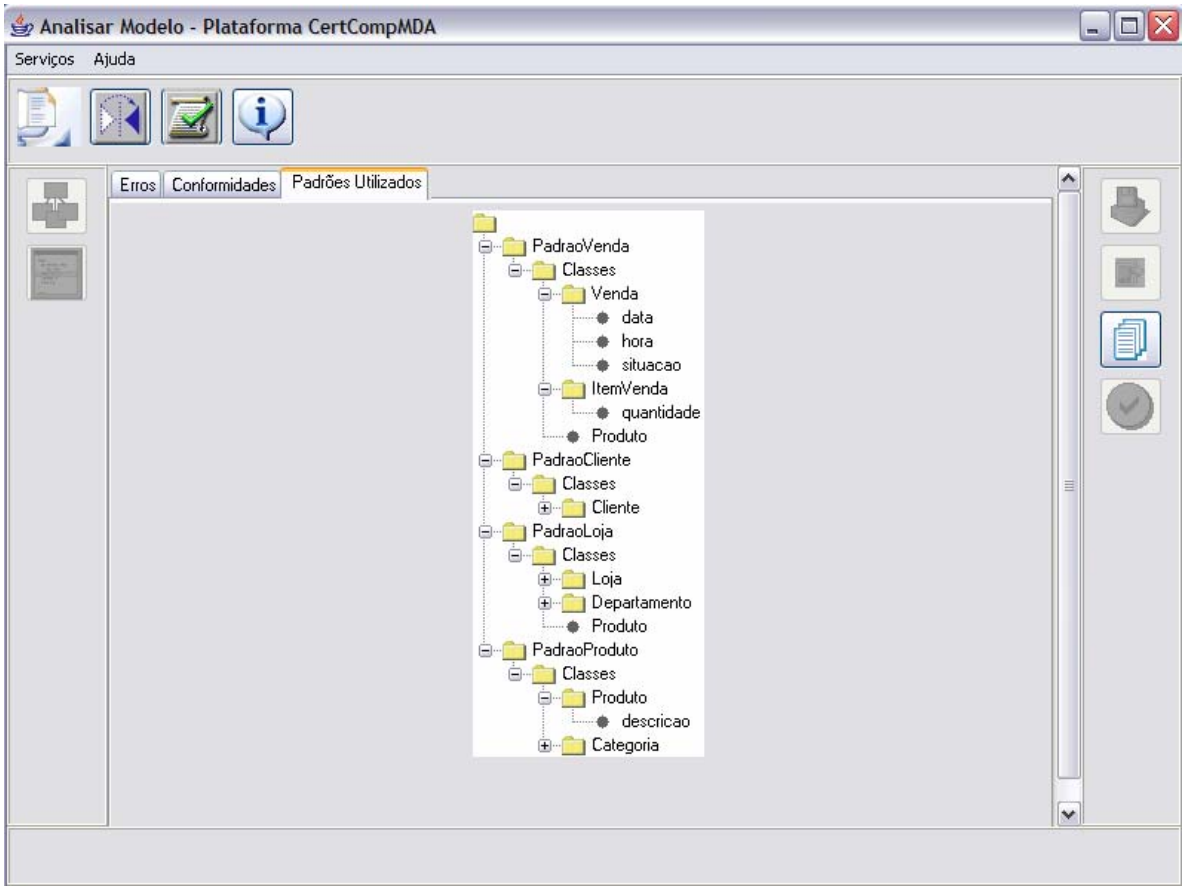


Figura 5.7 : Resultado da Análise do Modelo : Padrões Utilizados

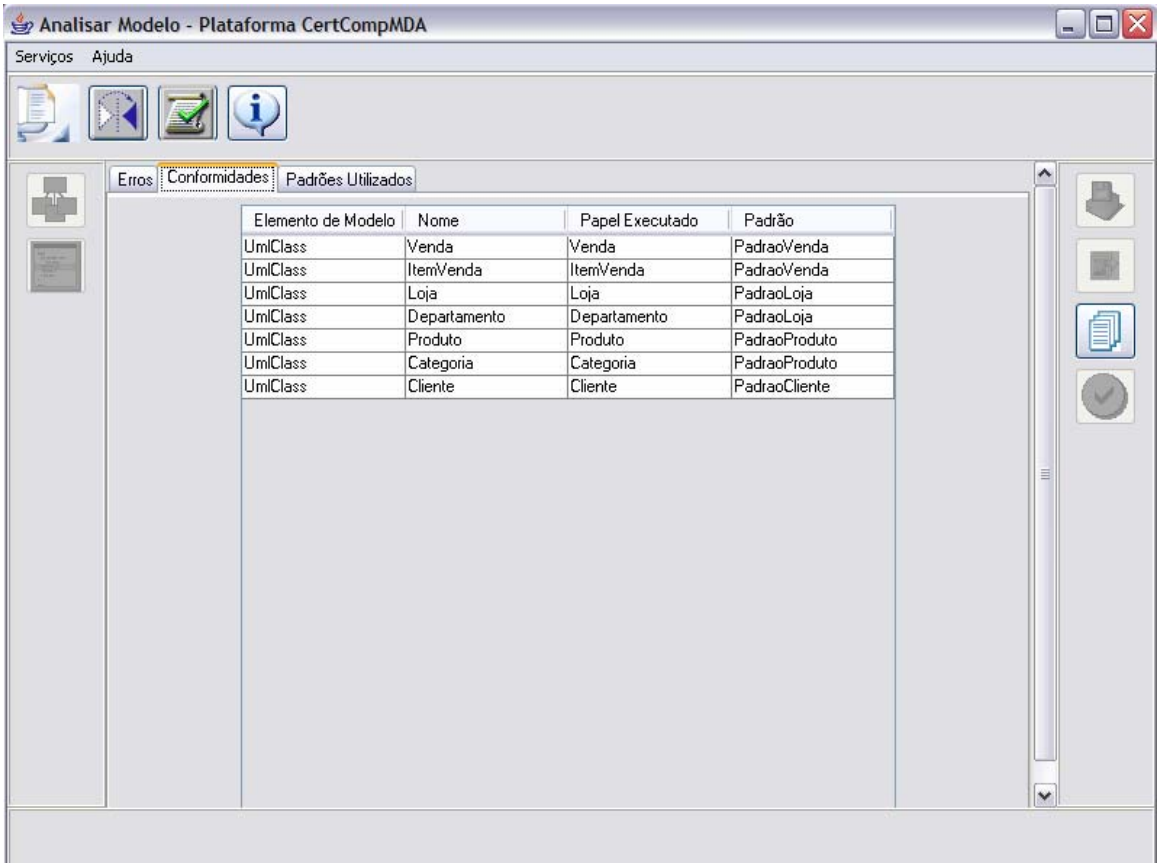


Figura 5.8 : Resultado da Análise do Modelo : Padrões Utilizados – Conformidades

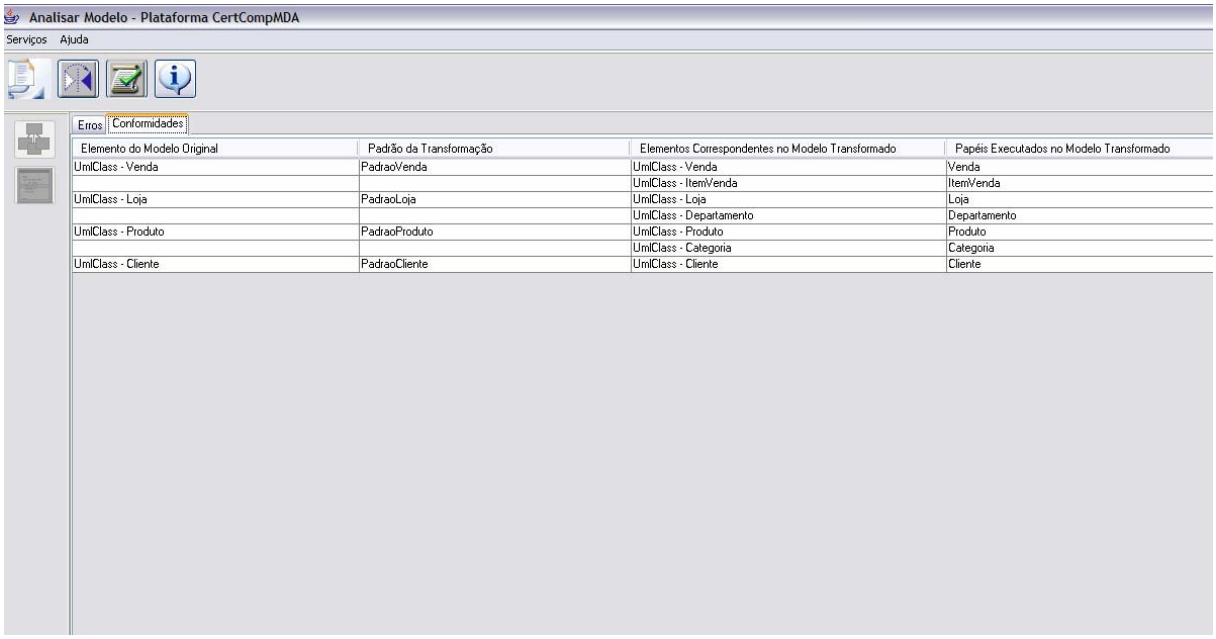


Figura 5.9 : Resultado da Análise do Modelo : Transformação de Padrões – Conformidades

A segunda etapa de transformação ou refinamento, utilizou o mapeamento definido pela etiqueta padroesAlvo em todos os elementos do modelo anterior, aplicando a transformação para os papéis do padrão Facade, A figura 5.10 ilustra o modelo desse padrão.

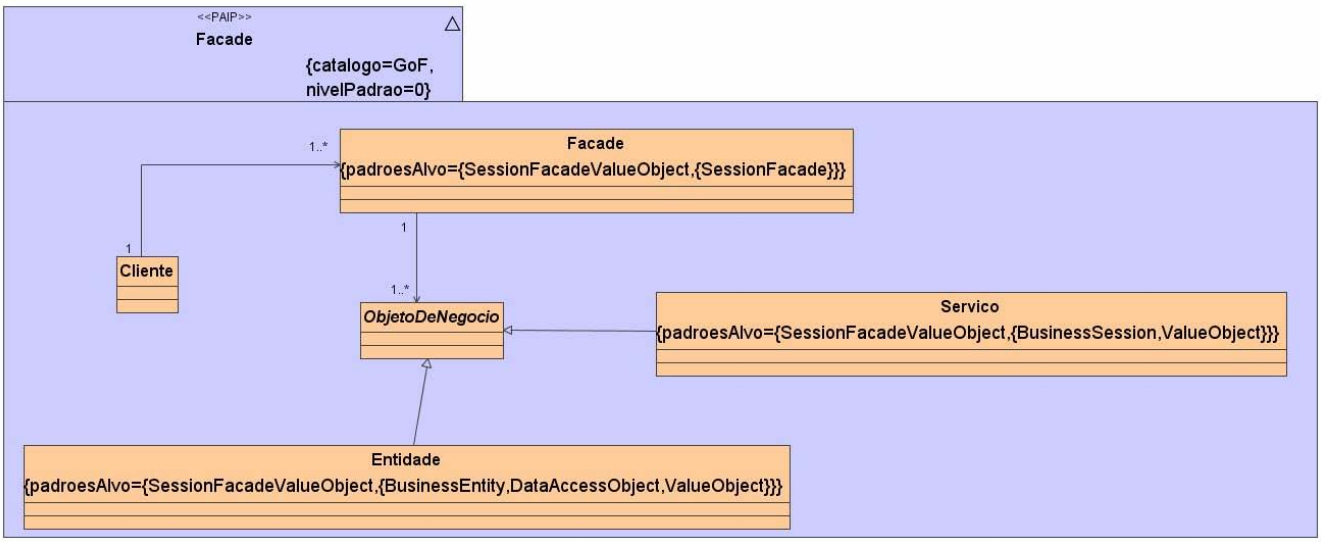
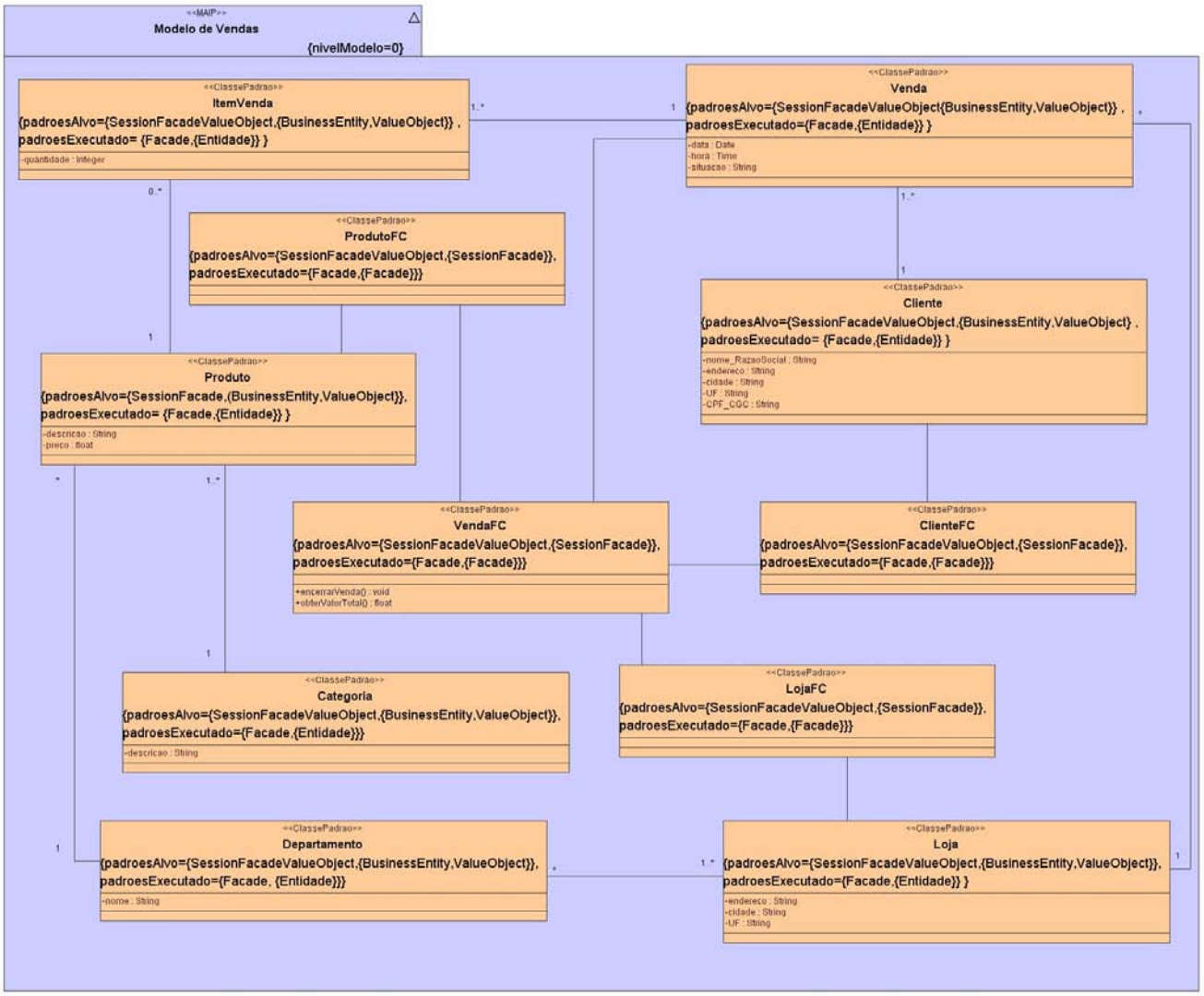


Figura 5.10 : Padrão de Aplicação Independente de Plataforma : Facade

Nessa etapa, após a aplicação do padrão de aplicação independente de plataforma Facade, pertencente ao catálogo GoF, geramos o primeiro e único modelo de aplicação independente de plataforma (MAIP) de nível 0, onde ainda continuamos não encontrando os detalhes da tecnologia a ser utilizada. A figura 5.11 apresenta o modelo do sistema de vendas após esse segundo refinamento.

As operações encerrarVenda() e obterValorTotal() foram transferidas para a classe VendaFC que representa o papel Facade, conforme determinado no mapeamento do modelo <<MNIP>> de nível de abstração anterior na etiqueta papeisPadroesAlvo dos elementos operação.



5.11: PIM transformado: MAIP - Sistema de Vendas

A certificação realizada nesse modelo correspondente a certificação da segunda etapa de transformação ou refinamento, resulta nos resultados, gerados pelo sistema de certificação, apresentados nas figuras 5.12, 5.13 e 5.14.

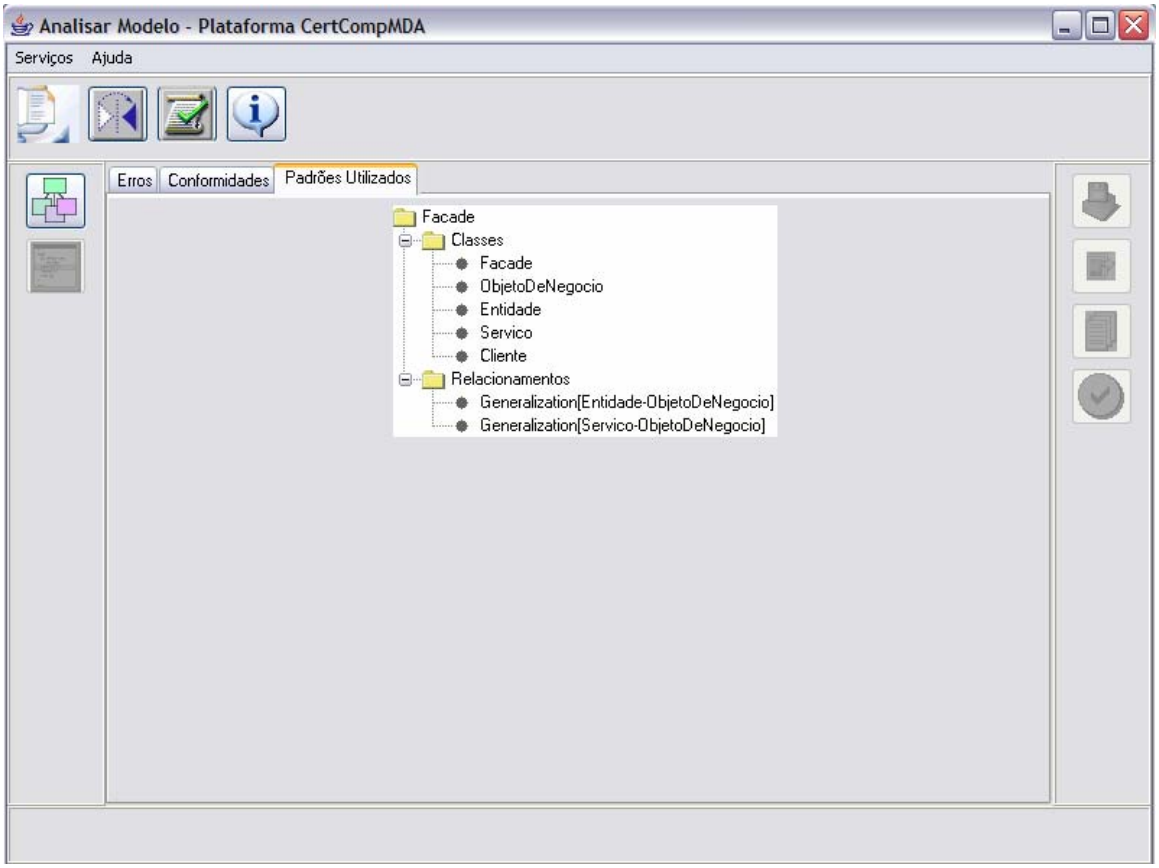


Figura 5.12 : Resultado da Análise do Modelo : Padrões Utilizados

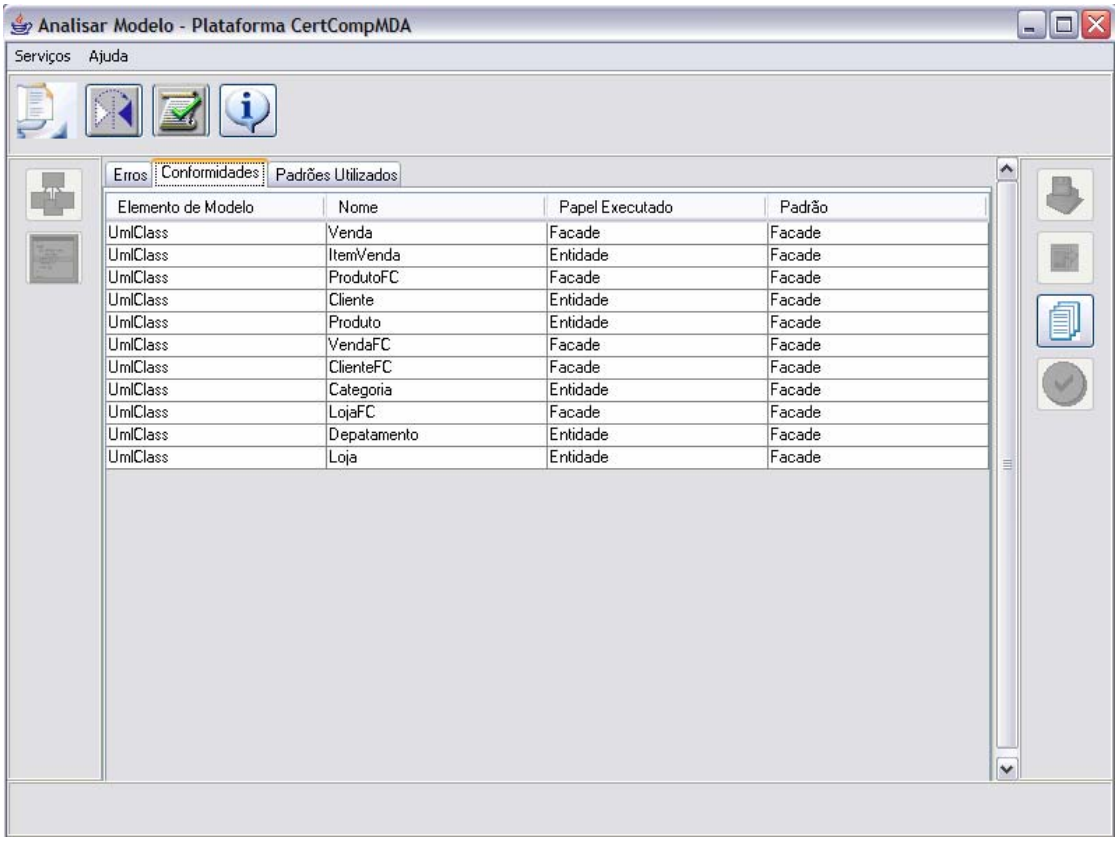
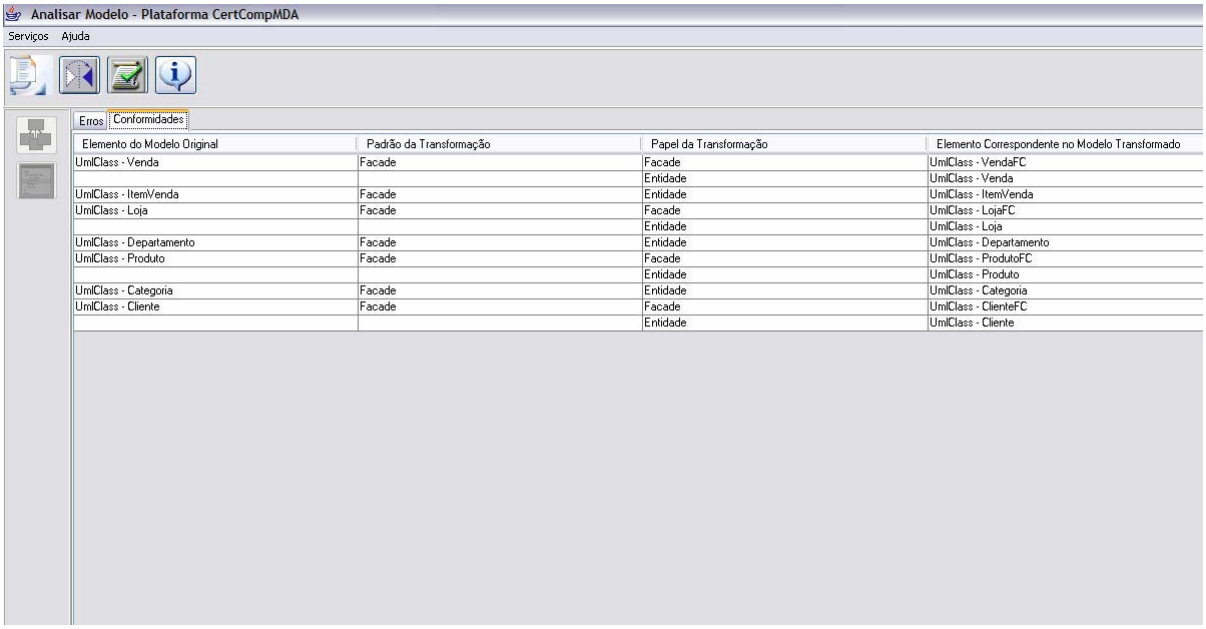


Figura 5.13 : Resultado da Análise do Modelo : Padrões Utilizados – Conformidades



Elemento do Modelo Original	Padrão da Transformação	Papel da Transformação	Elemento Correspondente no Modelo Transformado
UmlClass - Venda	Facade	Facade	UmlClass - VendaFC
		Entidade	UmlClass - Venda
UmlClass - ItemVenda	Facade	Entidade	UmlClass - ItemVenda
UmlClass - Loja	Facade	Facade	UmlClass - LojaFC
		Entidade	UmlClass - Loja
UmlClass - Departamento	Facade	Entidade	UmlClass - Departamento
UmlClass - Produto	Facade	Facade	UmlClass - ProdutoFC
		Entidade	UmlClass - Produto
UmlClass - Categoria	Facade	Entidade	UmlClass - Categoria
UmlClass - Cliente	Facade	Facade	UmlClass - ClienteFC
		Entidade	UmlClass - Cliente

Figura 5.14 : Resultado da Análise do Modelo : Transformação de Padrões – Conformidades

Esse ultimo modelo gerado, correspondente ao PIM de terceiro nível de abstração no ciclo de desenvolvimento, pode gerar diversos PSMs, para diversas plataformas tecnológicas e camadas da arquitetura. Considerando que o J2EE foi escolhido como plataforma tecnológica de desenvolvimento de componentes do nosso ambiente orientado a modelos da MDA, que a interface da aplicação será baseada na Web e que o sistema de gerenciamento de banco de dados é objeto-relacional, temos a transformação desse modelo do sistema para três modelos PSM. A figura 5.15 demonstra como se dão as transformações dos modelos do sistema de vendas.

Nesse exemplo de utilização do ambiente de certificação, nos limitaremos a certificar as transformações existentes para gerar os componentes EJB, sem contemplar a certificação das transformações para SQL (fora do escopo do ambiente) e para os componentes Web.

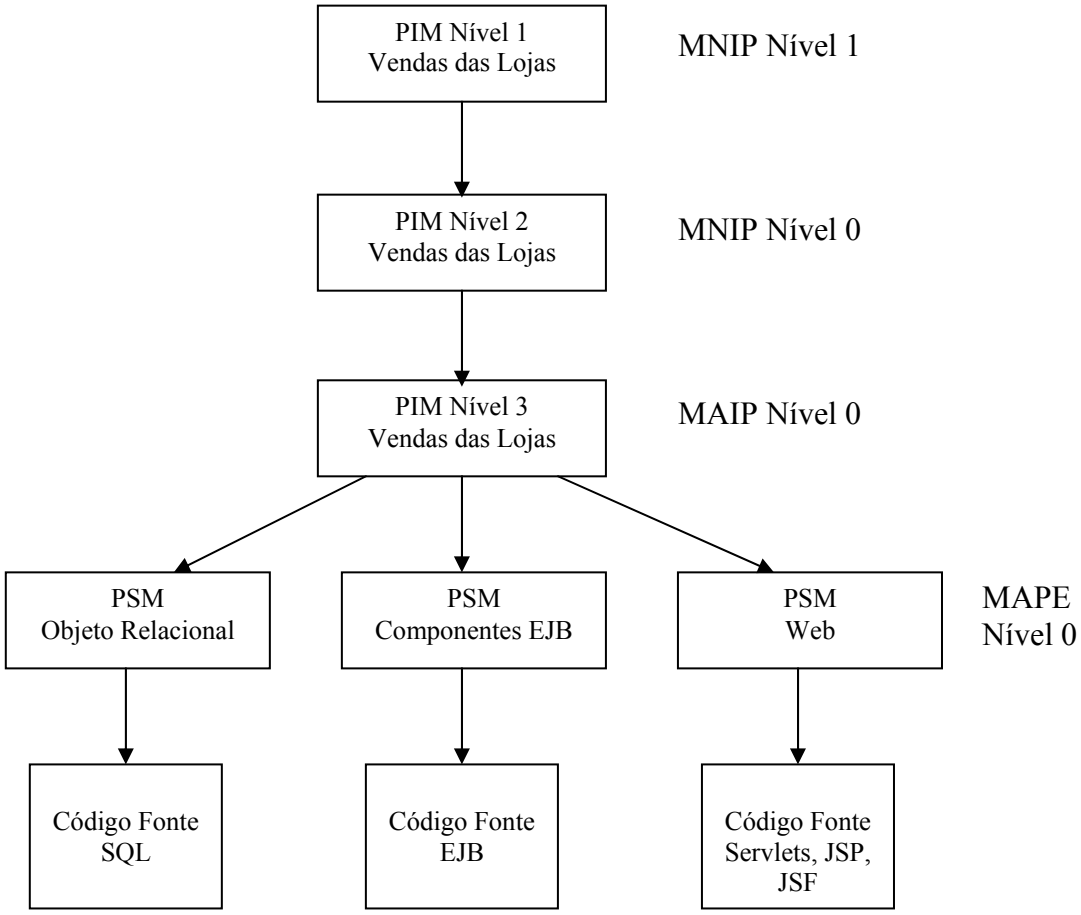


Figura 5.15: Os quatro níveis de modelo do sistema

Na terceira etapa de transformação ou refinamento, após a aplicação do padrão da plataforma J2EE, SessionFacadeValueObject, resultante de uma configuração nova de padrão de projeto, com a junção dos padrões SessionFacade e ValueObject do CORE J2EE (figura 5.16), e indicado nas etiquetas padroesAlvo do modelo anterior para a camada de negócios, o modelo PIM da aplicação resulta no primeiro e último modelo PSM (5.17), antes da transformação para o modelo de implementação (código) J2EE.

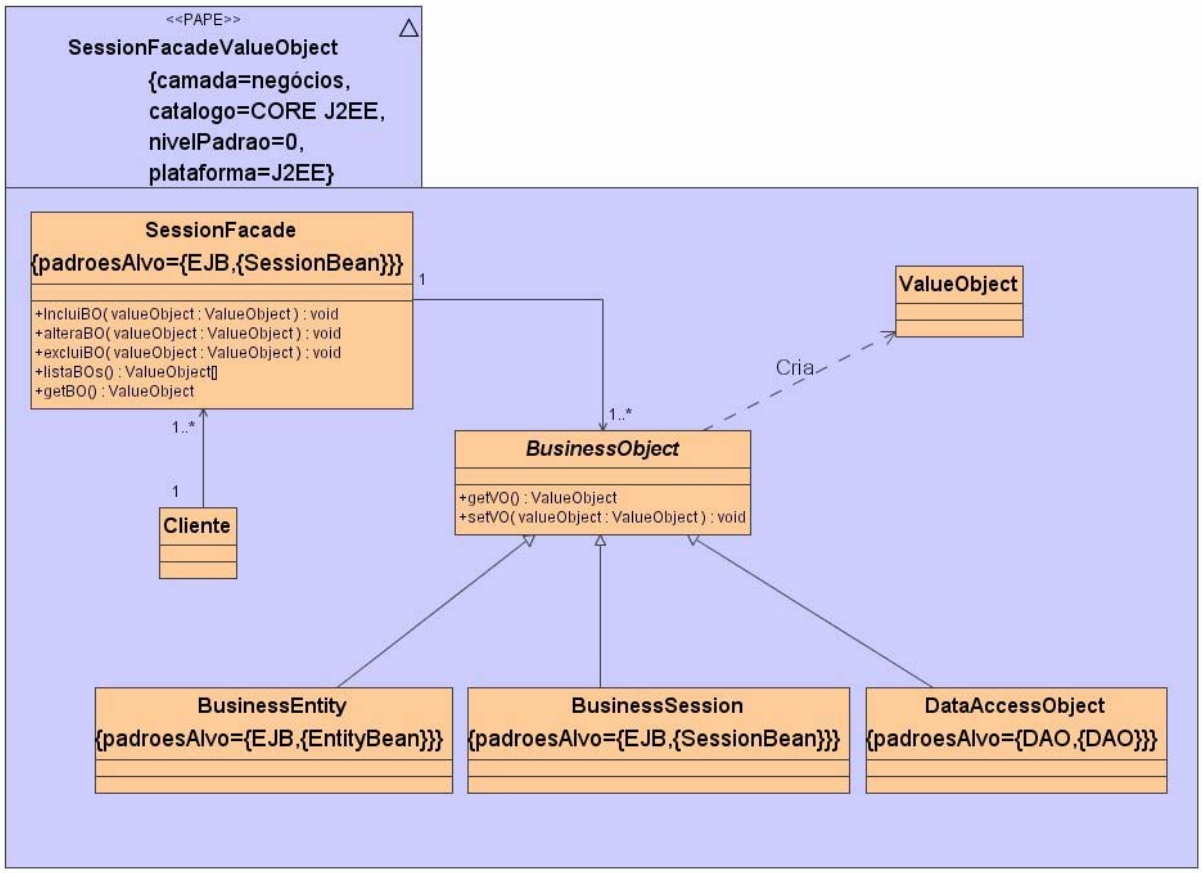
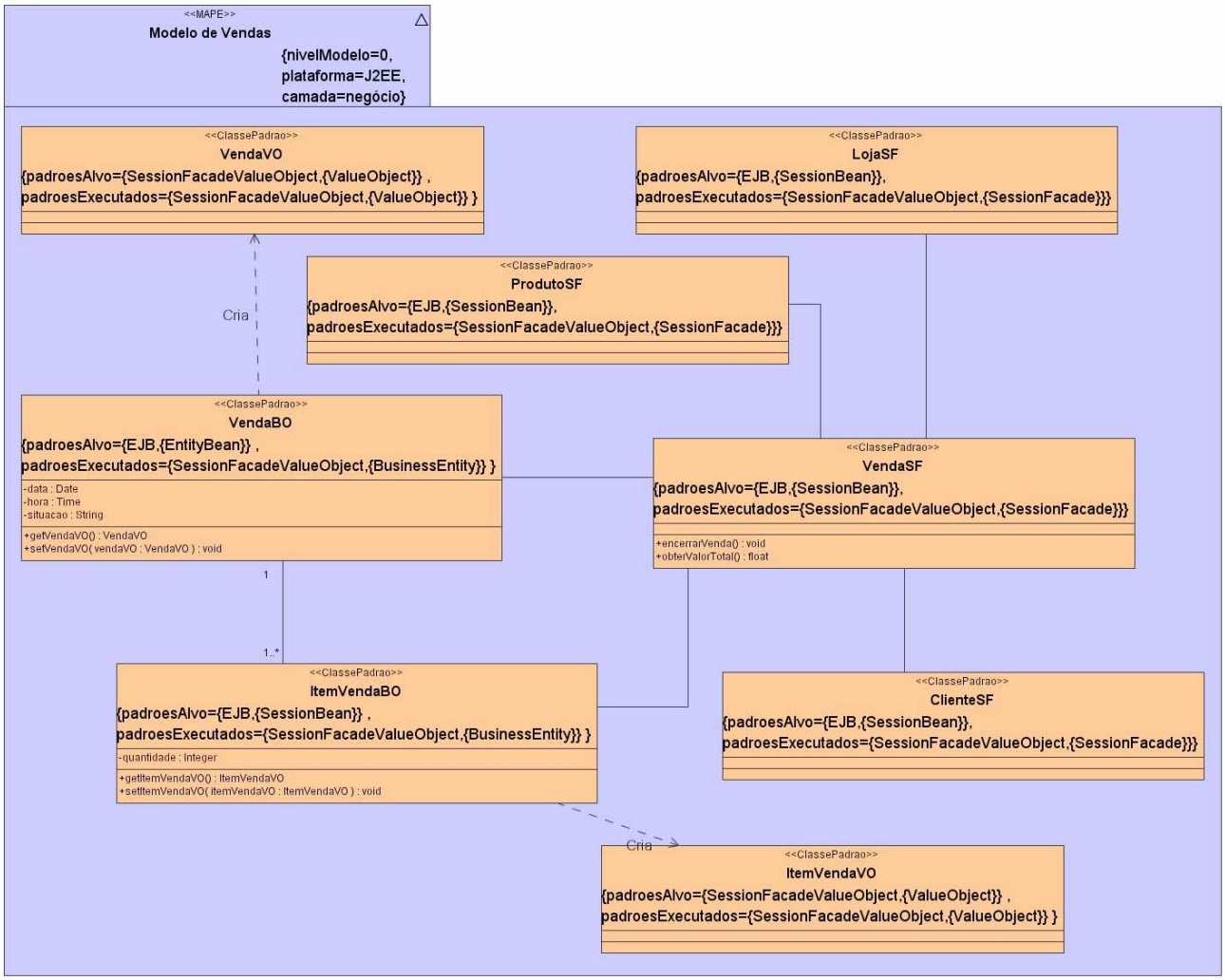


Figura 5.16 : Padrão da Plataforma J2EE : SessionFacadeValueObject

A cada transformação para um nível de abstração menor aumentam as informações necessárias nos diagrama de classes dificultando sua visualização. Dessa forma, para que fosse possível ilustrar todos os detalhes relevantes dessa nova transformação, foram transportadas apenas as classes referentes à representação do PadraoVenda relacionado com a fachada dos demais padrões de negócio. Além do mapeamento dos papéis de padrões indicado pelas etiquetas padroesAlvo e padroesExecutados, as regras de transformação demonstradas nesse modelo são:

- As classes com papel de ValueObject possuem todos os atributos das classes com papel de BusinessObject à qual possuem relação de dependência;
- As classes com papel de SessionFacade possuem as operações de papéis: incluiBO; alteraBO; excluiBO; listaBO e getBO. Essas operações contêm a

implementação do código responsável pela execução das inclusões, alterações, exclusões, listas e acesso as informações das vendas realizadas, através das entidades Venda e ItemVenda.



5.17 : PSM transformado : MAPE – Camada de Negócios, Plataforma J2EE

A certificação realizada nesse modelo correspondente a certificação da terceira etapa de transformação ou refinamento, resulta nos resultados gerados pelo sistema de certificação, apresentados nas figuras 5.18, 5.19 e 5.20.

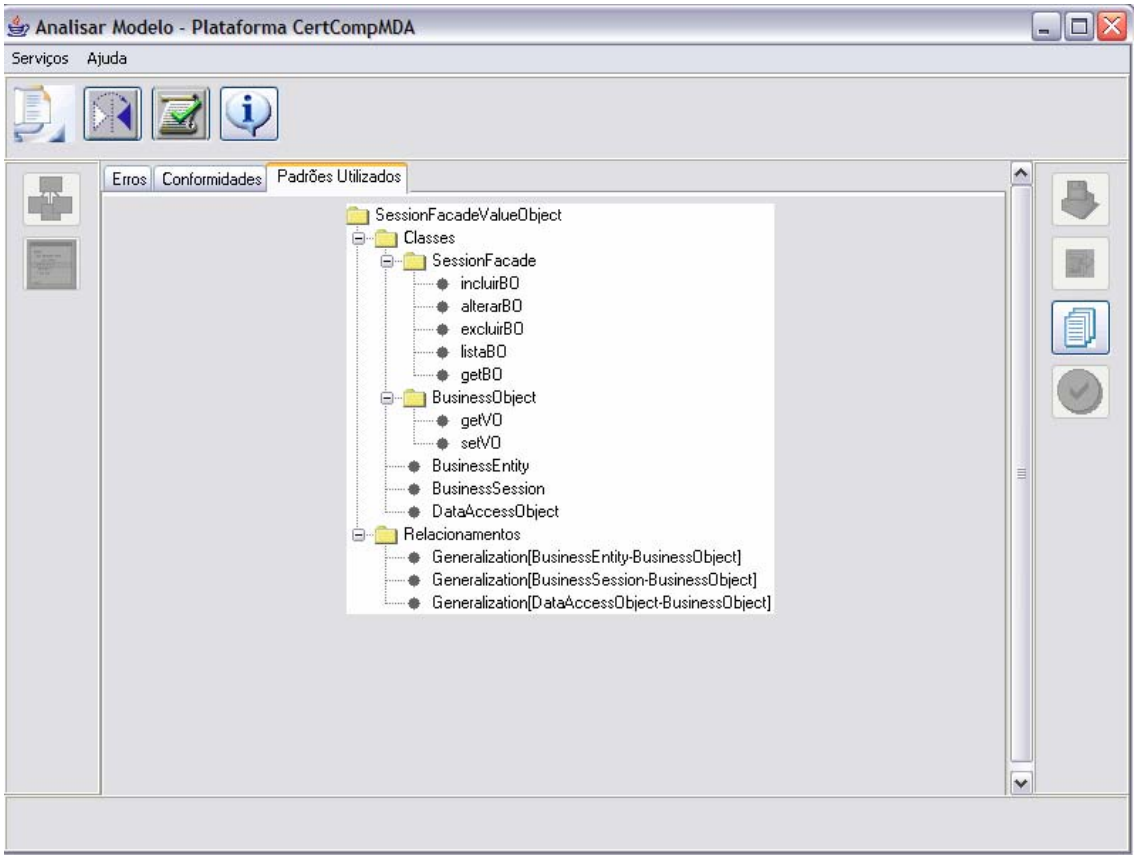


Figura 5.18 : Resultado da Análise do Modelo : Padrões Utilizados

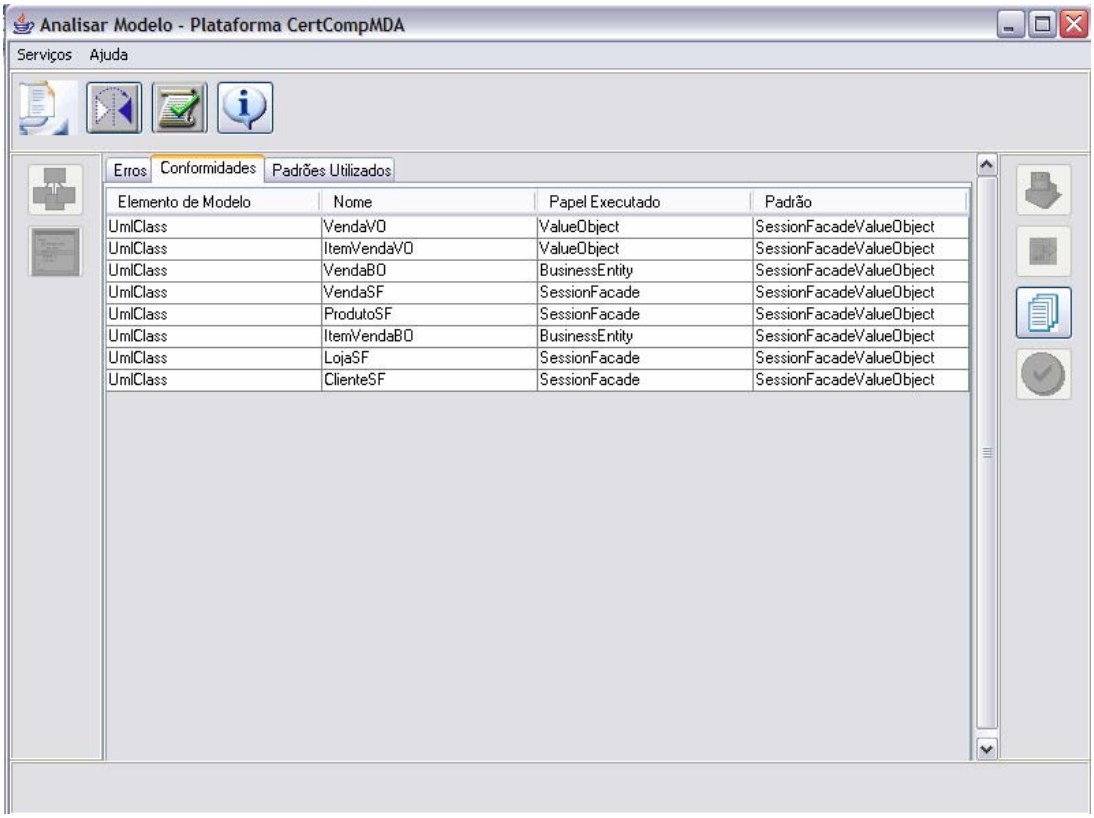
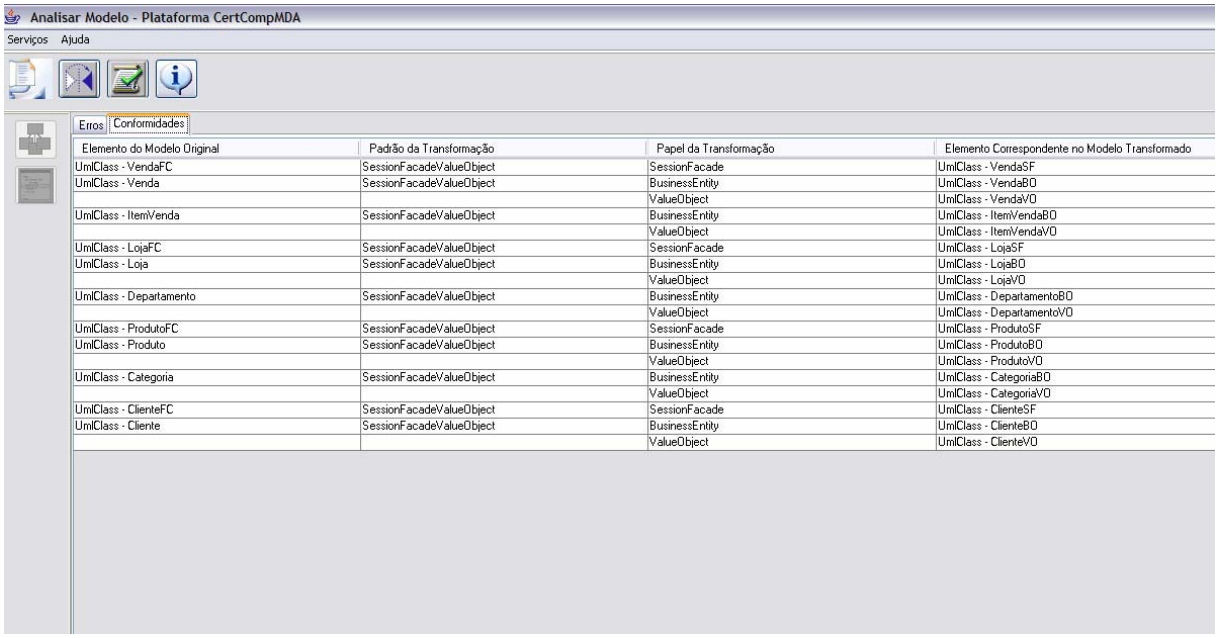


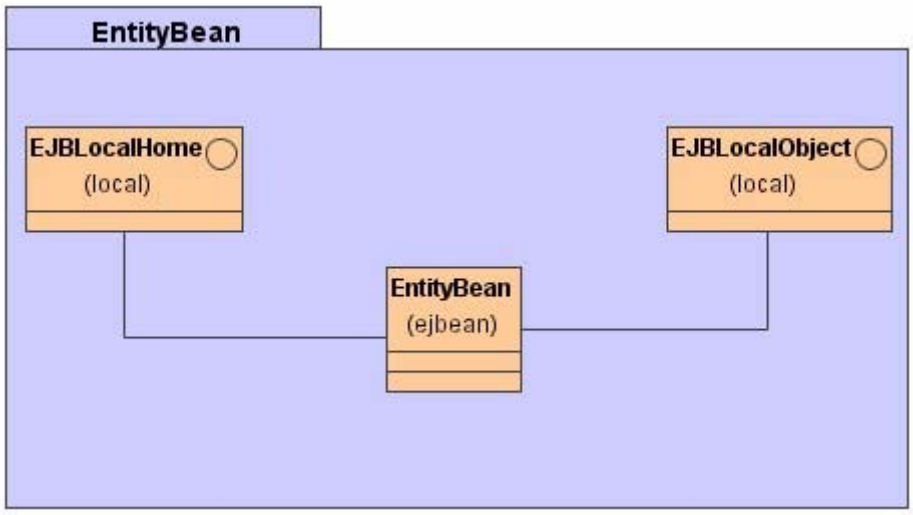
Figura 5.19 : Resultado da Análise do Modelo : Padrões Utilizados - Conformidades



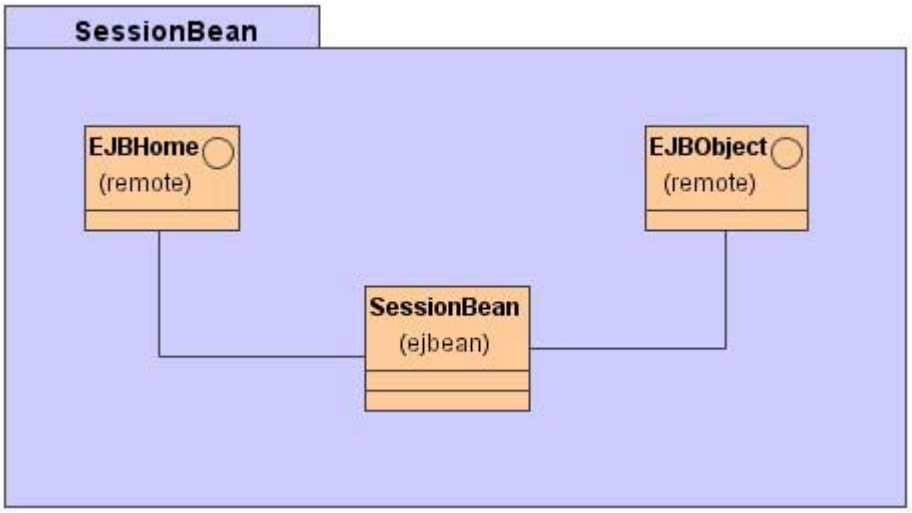
Elemento do Modelo Original	Padrão da Transformação	Papel da Transformação	Elemento Correspondente no Modelo Transformado
UmlClass - VendaFC	SessionFacadeValueObject	SessionFacade	UmlClass - VendaSF
UmlClass - Venda	SessionFacadeValueObject	BusinessEntity	UmlClass - VendaBO
		ValueObject	UmlClass - VendaVO
UmlClass - ItemVenda	SessionFacadeValueObject	BusinessEntity	UmlClass - ItemVendaBO
		ValueObject	UmlClass - ItemVendaVO
UmlClass - LojaFC	SessionFacadeValueObject	SessionFacade	UmlClass - LojaSF
UmlClass - Loja	SessionFacadeValueObject	BusinessEntity	UmlClass - LojaBO
		ValueObject	UmlClass - LojaVO
UmlClass - Departamento	SessionFacadeValueObject	BusinessEntity	UmlClass - DepartamentoBO
		ValueObject	UmlClass - DepartamentoVO
UmlClass - ProdutoFC	SessionFacadeValueObject	SessionFacade	UmlClass - ProdutoSF
UmlClass - Produto	SessionFacadeValueObject	BusinessEntity	UmlClass - ProdutoBO
		ValueObject	UmlClass - ProdutoVO
UmlClass - Categoria	SessionFacadeValueObject	BusinessEntity	UmlClass - CategoriaBO
		ValueObject	UmlClass - CategoriaVO
UmlClass - ClienteFC	SessionFacadeValueObject	SessionFacade	UmlClass - ClienteSF
UmlClass - Cliente	SessionFacadeValueObject	BusinessEntity	UmlClass - ClienteBO
		ValueObject	UmlClass - ClienteVO

Figura 5.20 : Resultado da Análise do Modelo : Transformação de Padrões – Conformidades

Na última etapa de transformação ou refinamento temos a aplicação de padrões da plataforma J2EE, indicando a utilização da tecnologia EJB. Componentes EJB possuem também um padrão específico de implementação. De acordo com o metamodelo e perfil UML da tecnologia EJB (OMG, 2004) (JSR026, 2001), uma configuração específica de implementação de componentes EJB foi desenvolvida para ser aplicada nesse exemplo de certificação (figura 5.21 e 5.22). Essas configurações novas, a partir de padrões existentes, servem para demonstrar a flexibilidade da solução, que permite a utilização de qualquer padrão de projeto criado pelos arquitetos de um determinado ambiente de desenvolvimento. Com a representação desses padrões de implementação podemos certificar o código construído, através de um mecanismo flexível e aberto.



5.21 : Padrão EJB – EntityBean



5.22 : Padrão EJB – SessionBean

Nas configurações do padrão EJB acima, foi determinado em que pacotes as classes e as interfaces dos *SessionBeans* e *EntityBeans* seriam colocadas no código implementado.

Os modelos PSM dos componentes EJB em UML são transformados no modelo de código em Java. A seguir é apresentada parte do código implementado do exemplo, ilustrando a utilização das anotações representando os estereótipos e etiquetas e a utilização do recurso *Generics* para especificar o tipo das coleções que representam relacionamentos:

Classes com a declaração dos tipos de anotação utilizados (AtributoPadrao, ClassePadrao e OperacaoPadrao):

```
package anotacao;

import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.FIELD;

@Target(FIELD)
public @interface AtributoPadrao {
    String[] padroesExecutados();
}
```

```
package anotacao;

import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.TYPE;

@Target(TYPE)
public @interface ClassePadrao {
    String[] padroesExecutados ();
}
```

```
package anotacao;

import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.METHOD;;

@Target(METHOD)
public @interface OperacaoPadrao {
    String[] padroesExecutados();
}
```

Classe do *EntityBean* com a anotação *ClassePadrao* indicando que essa classe executa um papel de padrão.

```
package ejbean;

import java.rmi.RemoteException;
import java.util.Collection;
import java.sql.*;
import anotacao.*;
import javax.ejb.*;
import javax.naming.*;
import local.ClienteBOLocal;
import local.ItemVendaBOLocal;
import local.LojaBOLocal;
import remote.VendaVO;


@ClassePadrao(padroesExecutados={"SessionFacadeValueObject-1-
BusinessEntity"})
public abstract class VendaBO implements EntityBean {
```

```
}
```

Operações da classe do *EntityBean* com a anotação *OperacaoPadrao*, indicando que essas operações executam um papel de padrão.

```
      @OperacaoPadrao(padroesExecutados={"SessionFacadeValueObject-1-
getVO"})
      public VendaVO getVendaVO () {
            VendaVO vendaVO = new VendaVO();
            vendaVO.setId (getId());
            vendaVO.setData(getData());
            vendaVO.setHora(getHora());
            vendaVO.setSituacao (getSituacao());
            return vendaVO;
      }


      @OperacaoPadrao(padroesExecutados={"SessionFacadeValueObject-1-
setVO"})
      public void setVendaVO (VendaVO vendaVO) throws CreateException {
            setData(vendaVO.getData());
            setHora(vendaVO.getHora());
            setSituacao (vendaVO.getSituacao());
      }

}
```

Classe do *SessionBean* com a anotação *ClassePadrao* indicando que essa classe executa um papel de padrão.

```
package ejbean;

import java.rmi.RemoteException;
import java.util.*;
import javax.ejb.*;
import javax.naming.InitialContext;
import local.*;
import remote.*;
import anotacao.*;


@ClassePadrao(padroesExecutados={"SessionFacadeValueObject-1-
SessionFacade"})
public class VendaSFBean implements SessionBean {


}
```

Operações da classe do *SessionBean* com a anotação *OperacaoPadrao*, indicando que essas operações executam um papel de padrão.

```
      // fachada para VendaBO

      @OperacaoPadrao(padroesExecutados={"SessionFacadeValueObject-1-
incluiBO"})
```

```
	public void incluiVendaBO( VendaVO vendaVO ) throws CreateException,
FinderException{
		VendaBOLocal vendaBO = homeVendaBO.create( vendaVO );
		Collection <ItemVendaVO> itensVenda =
vendaBO.getVendaVO().getItensVenda();
		for (ItemVendaVO itemVendaVO : itensVenda){
			ItemVendaBOLocal itemVendaBO = homeItemVendaBO.create(
itemVendaVO );
			itemVendaBO.setVendaBO(vendaBO);
		}
	}

	@OperacaoPadrao(padroesExecutados={"SessionFacadeValueObject-1-
alteraBO"})
	public void alteraVendaBO( VendaVO vendaVO ) throws CreateException,
FinderException{
		VendaBOLocal vendaBO = homeVendaBO.findByPrimaryKey(
vendaVO.getId() );
		vendaBO.setVendaVO( vendaVO );
	}

	@OperacaoPadrao(padroesExecutados={"SessionFacadeValueObject-1-
excluiBO"})
	public void excluiVendaBO( Integer id ) throws FinderException,
RemoveException{
		VendaBOLocal vendaBO = homeVendaBO.findByPrimaryKey(id);
		try {
			homeVendaBO.remove( id );
		}
		catch (TransactionRolledbackLocalException te) {
			if (te.getCausedByException() instanceof
NoSuchObjectLocalException)
				throw new ObjectNotFoundException ("primKey="+id);
		}
		catch (NoSuchObjectLocalException ne) {
			throw new ObjectNotFoundException ("primKey="+id);
		}
	}

	@OperacaoPadrao(padroesExecutados={"SessionFacadeValueObject-1-
listaBO"})
	public Collection listaVendaBOs() throws FinderException{
		return homeVendaBO.findAll();
	}

	@OperacaoPadrao(padroesExecutados={"SessionFacadeValueObject-1-
getBO"})
	public VendaVO getDadosVendaBO( Integer id ) throws FinderException{
		VendaBOLocal vendaBO = homeVendaBO.findByPrimaryKey( id );
		VendaVO vendaVO = vendaBO.getVendaVO();
		return vendaVO;
	}
```

Classe do *ValueObject* com a anotação *ClassePadrao* indicando que essa classe executa um papel de padrão.

```
package remote;
import java.io.Serializable;
import java.sql.*;
import java.util.Collection;
```

```
import local.ItemVendaBOLocal;
import anotacao.*;


@ClassePadrao(padroesExecutados={"SessionFacadeValueObject-1-ValueObject"})
public class VendaVO implements Serializable {

}
```

Utilização do recurso Generics para determinar o tipo permitido numa coleção, utilizado para mapear uma cardinalidade n de uma associação.

```
	public Collection<ItemVendaVO> getItensVenda() {
		return itensVenda;
	}

	public void setItensVenda(Collection<ItemVendaVO> itensVenda) {
		this.itensVenda = itensVenda;
	}

}
```

A certificação realizada no modelo de implementação corresponde à certificação da quarta e última etapa de transformação ou refinamento, que resulta nos resultados gerados pelo sistema de certificação, apresentados nas figuras 5.23, 5.24 e 5.25.

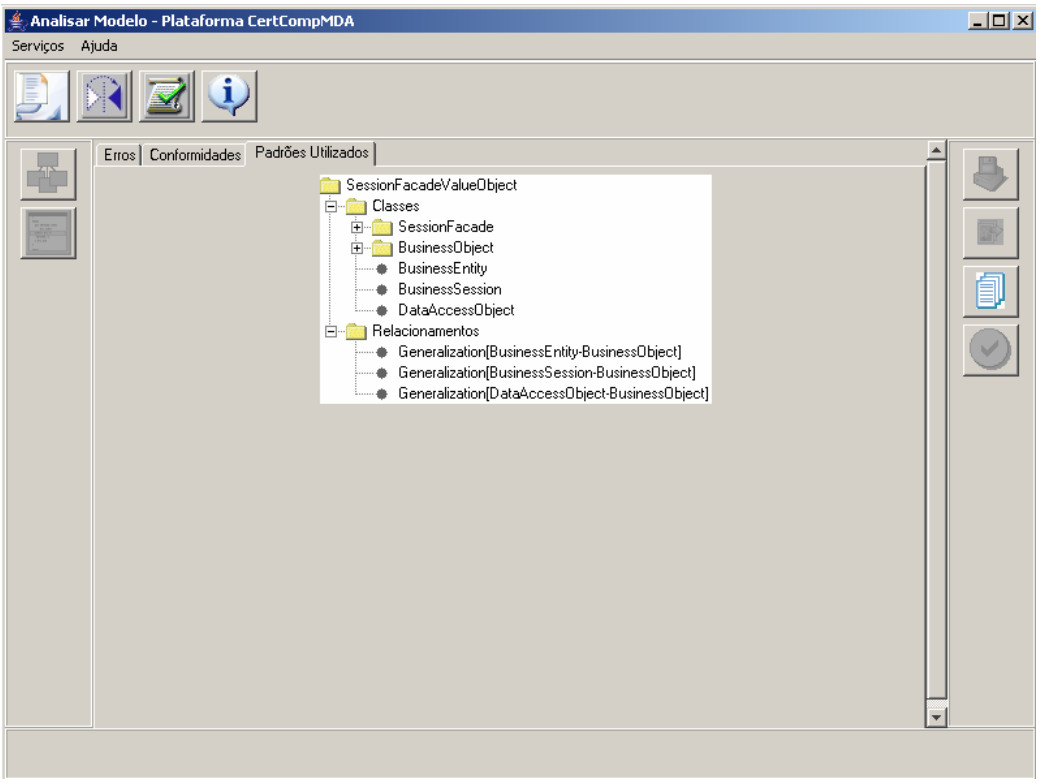


Figura 5.23 : Resultado da Análise do Modelo : Padrões Utilizados

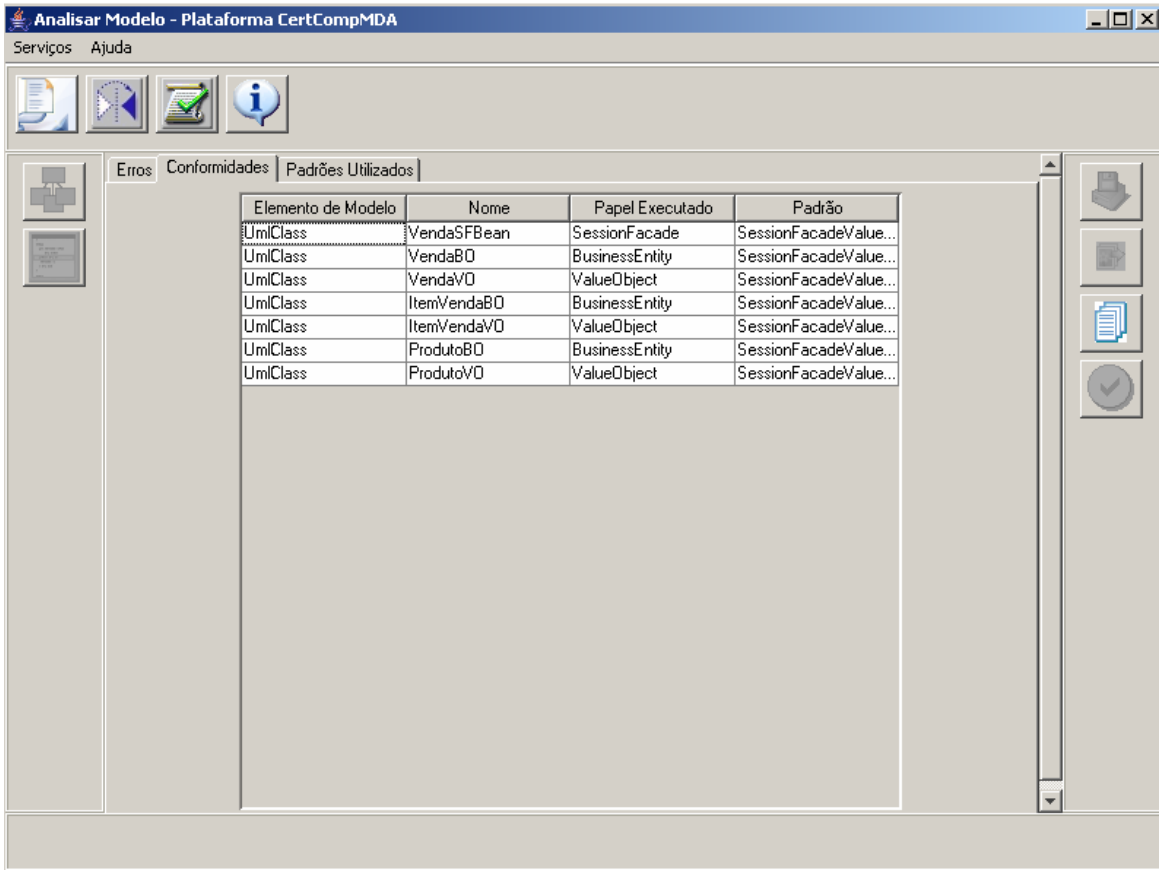


Figura 5.24 : Resultado da Análise do Modelo : Padrões Utilizados - Conformidades

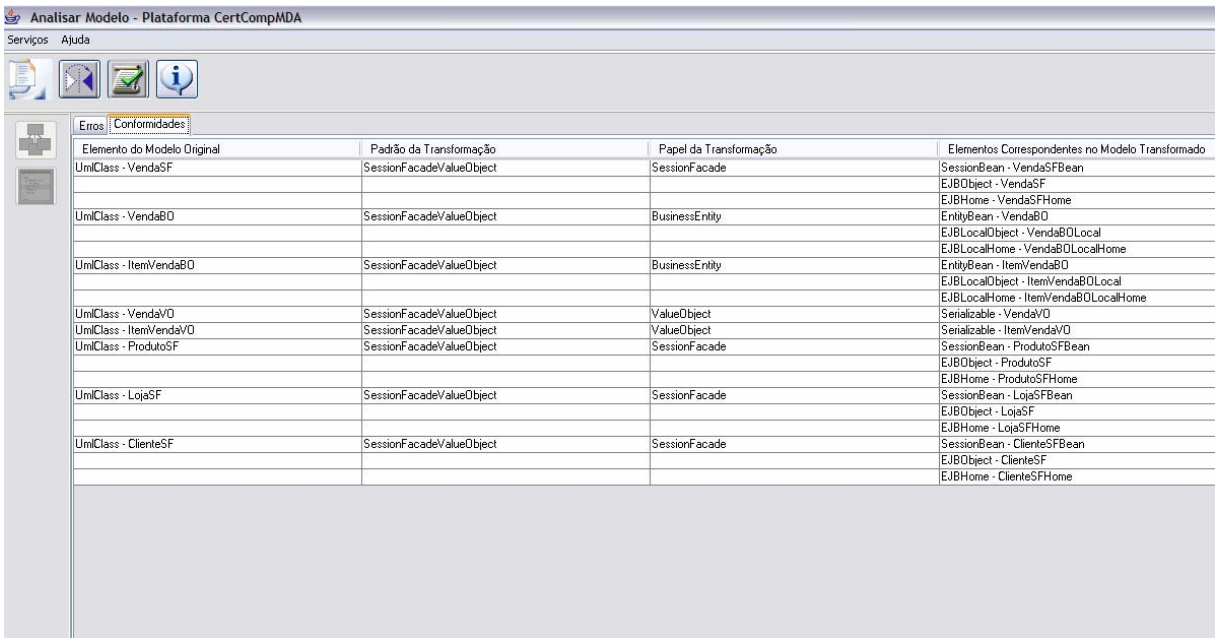


Figura 5.25 : Resultado da Análise do Modelo : Transformação de Padrões - Conformidades

Nos modelos <<MNIP>>, <<MAIP>> e <<MAPE>> da aplicação, construídos para os respectivos níveis de abstração, cada elemento classe, com o estereótipo <<ClassePadrao>>,

pode estar executando um ou mais papéis correspondentes aos padrões instanciados pela transformação. A etiqueta padroesExecutados contém a informação dos papéis de padrões executados por cada elemento do modelo, enquanto que a etiqueta padroesAlvo indica os papéis de padrão para os quais cada elemento será mapeado na transformação para o próximo nível de abstração.

Todos os elementos atributo e operação que vem do nível anterior de abstração, devem possuir os estereótipos indicando que fazem parte de um padrão e o conteúdo das etiquetas informando os papéis do padrão que estão executando. Dessa forma eles ficam diferenciados de qualquer atributo ou operação que seja adicionada fora do processo de transformação.

5.3. As Certificações nos Modelos do Sistema

Os padrões utilizados na construção dos modelos do componente e nas transformações foram representados de acordo com o seu perfil específico numa ferramenta de modelagem em UML, exportados em XMI e importados e armazenados no repositório de padrões MDR, conforme definido no ambiente de certificação proposto (serviço de Administração de Repositório).

O modelo de código em Java do Sistema de Vendas foi convertido no seu respectivo modelo UML, correspondente ao PSM da camada de negócios, conforme definido no ambiente de certificação proposto (serviço de Reconstrução de Arquitetura).

As validações dos modelos nos diversos níveis de abstração foram realizadas conforme definido no ambiente de certificação proposto (serviço de Análise de Modelos).

No Apêndice C temos a apresentação do guia de utilização do sistema de certificação, com a orientação nos passos a serem executados nos seus serviços (Administração de Repositório, Reconstrução de Arquitetura e Análise de Modelos).

CAPÍTULO 6

Conclusões e Trabalhos Futuros

A implantação de um efetivo ambiente de reutilização requer a utilização de processos de desenvolvimento e ferramentas específicas e eficientes de apoio, que forneçam um grau maior de automatização em todos os aspectos dentro do ciclo de desenvolvimento de componentes e aplicações.

O objetivo desta dissertação foi desenvolver um ambiente automatizado de certificação da utilização de padrões de projeto, no desenvolvimento de sistemas baseados em componentes em um ambiente orientado a modelos.

O modelo de desenvolvimento baseado em componentes enfatiza a importância da garantia de qualidade de software. Esse modelo apregoa a construção de sistemas através da composição de componentes reutilizáveis e pré-existentes, o que requer a certeza da qualidade e conformidade destes componentes aos requisitos funcionais e não funcionais determinados na sua especificação.

A construção de componentes, em um ambiente de desenvolvimento orientado a modelos, se dá através da modelagem desses componentes desde o mais alto nível de abstração, onde são criados os modelos de domínio de componentes de negócio. Todos os modelos (independentes, dependentes de plataforma e código) são criados através de transformações e refinamentos sucessivos, em direção ao maior nível de detalhamento.

Na construção desses modelos, são aplicados padrões específicos e adequados a cada nível de abstração. Nas transformações executadas ao longo do ciclo de desenvolvimento, os padrões utilizados são transformados em outros padrões correspondentes associados aquele nível de abstração.

A MDA, padrão do OMG para um ambiente de desenvolvimento orientado a modelos, surgiu como uma proposta capaz de elevar a automatização e o nível de abstração do processo de

desenvolvimento de software, atingindo, por conseqüência, aumento da produtividade, com mais agilidade no acompanhamento das mudanças tecnológicas e de negócio e mantendo a atualidade da documentação referente às aplicações desenvolvidas.

Este aumento de automatização só é alcançado através da utilização de mecanismos e instrumentos específicos para as diversas funcionalidades existentes no ambiente de desenvolvimento da MDA. Como por exemplo: instrumentos de definição de transformações e armazenamento dessas definições em repositórios; mecanismos de análise e reconhecimento de código que recuperam modelos dos códigos fonte implementados; ferramentas de transformação, onde as definições das transformações são compreendidas, processadas e as transformações de modelos executadas automaticamente até a geração de código; ferramentas de validação de modelos, que garantem a formação correta desses modelos, segundo determinados critérios e restrições, etc...

A principal contribuição deste trabalho é a apresentação de um ambiente, aberto e flexível, de certificação que inclui processo e sistema computacional, automatizando a validação dos modelos criados em todos os níveis de abstração com relação à utilização de padrões de projeto, garantindo a sua formação correta, para que a cadeia de transformação possa prosseguir até a geração final do código de execução dos componentes.

O processo inclui a definição de um formalismo de representação de padrões, com flexibilidade suficiente para representar qualquer padrão de projeto utilizado no ciclo de desenvolvimento. Define um mecanismo de mapeamento entre os padrões através dos diversos níveis de abstração, permitindo a validação das transformações de padrões na passagem de um nível para outro imediatamente adjacente. Esse sistema de certificação pode ser aplicado em qualquer ambiente orientado a modelos, independente da quantidade de níveis de abstração e refinamentos utilizados. Essa flexibilidade permite que os níveis de

abstração sejam definidos de acordo com as necessidades do ambiente de desenvolvimento e de acordo com as características de seus componentes e aplicações.

O sistema computacional inclui a definição de um repositório para armazenamento e catalogação dos padrões de projeto, da definição do mecanismo de acesso desses padrões no repositório e de manipulação de modelos para análise e comparação.

A utilização de perfis UML específicos para representação e utilização de padrões de projeto nos modelos das aplicações e componentes, torna a solução adotada de grande alcance dentro da comunidade de desenvolvimento de software, já que nesse ambiente a UML é intensamente conhecida e adotada, sendo o padrão de fato de linguagem de modelagem de sistemas de software.

Outra contribuição importante deste trabalho foi utilizar como ambiente computacional, a tecnologia Java e a plataforma J2EE, referências hoje no desenvolvimento de aplicações robustas e distribuídas. A utilização de um repositório compatível com MOF e JMI trouxe bastante flexibilidade no desenvolvimento do sistema de certificação. A especificação do JMI deverá se tornar um padrão no desenvolvimento de serviços, ferramentas e aplicações para o ambiente orientado a modelos. A facilidade de manipulação das definições de modelos através das interfaces Java geradas pelo mapeamento do MOF para a linguagem Java, torna o desenvolvimento nesse ambiente muito mais produtivo.

Como contribuições específicas podemos citar o fornecimento de uma implementação de domínio publico de transformação de código Java em modelos UML (em XMI ou JMI) e a recuperação do código Java dos aspectos comportamentais dos modelos UML através de diagramas de seqüência.

Este trabalho apresenta algumas limitações e deixa alguns problemas ainda em aberto, que poderão ser resolvidos em trabalhos futuros. A seguir são sugeridas algumas extensões a serem realizadas:

- Um aprimoramento nas consistências relativas aos aspectos comportamentais dos modelos de representação de padrões. Na versão atual, os elementos de modelo correspondentes aos diagramas de seqüência são utilizados para comparação e validação, mas ainda num grau de precisão insatisfatório. Esses aprimoramentos necessários são:
  - Manipulação e comparação nos modelos, dos elementos com as informações da seqüência em que as operações são invocadas e executadas;
  - Recuperação do código fonte dos componentes e aplicações, dos elementos com as informações da seqüência em que as operações são invocadas e executadas;
- Restrições em OCL podem ser adicionadas nos modelos de representação dos padrões de projeto, aumentando a precisão semântica e conseqüentemente a precisão dos diagnósticos das avaliações realizadas;
- Desenvolvimento de extensão da ferramenta de automatização da certificação para outras plataformas de desenvolvimento, incluindo o serviço de reconstrução de arquitetura (recuperação dos modelos de implementação) do código fonte dessas plataformas;
- Construção de uma aplicação de automatização do desenvolvimento de componentes e aplicações no ambiente orientado a modelos, através das transformações de modelos utilizando os mesmos perfis UML e mecanismo de associação entre padrões definidos neste trabalho. Na construção de um modelo

em um determinado nível de abstração, essa ferramenta disponibilizaria interfaces que permitiriam a escolha dos padrões a serem mapeados na transformação para o modelo do próximo nível de abstração, atualizando automaticamente as etiquetas que registram esse mapeamento. Na passagem de um modelo de aplicação independente de plataforma para o primeiro modelo de plataforma específica, após a definição da camada da aplicação e plataforma tecnológica desse modelo, a ferramenta de transformação só permitiria padrões da mesma camada e plataforma, para definição do mapeamento. A classificação dos modelos por níveis configuráveis dá total flexibilidade para que os arquitetos de um determinado ambiente de desenvolvimento definam a quantidade de refinamentos (níveis de abstração) que deverão existir, de acordo com as características das aplicações e componentes;

- Incorporação dos perfis UML para Java e EJB do OMG e JCP na transformação dos modelos de plataforma específica em código.

- Adaptação desse sistema de certificação para ser utilizado como aditivos (*plug-ins*) para ferramentas IDEs utilizadas em ambiente Java, como por exemplo o Eclipse[1] e NetBeans[2];

[1] http://www.eclipse.org/
[2] http://www.netbeans.org/

Referências Bibliográficas.

AABY**,** A. **Many-sorted Algebra**. 2002. Disponível em : http://cs.wwc.edu/~aabyan/Math/Algebra.html Acesso em : 27/09/2005

ADDY, E. A., **Performing Verification and Validation in Reuse-Based Software Engineering.** NASA/WUV Software Research Laboratory. Software Engineering Workshop, novembro, 1998. Disponível em : http://sel.gsfc.nasa.gov/website/sew/1998/topics/addy-p.pdf Acesso em : 27/09/2005

ALBIN-AMIOT, H., GUEHENEUC**,** Y. G. **Meta-Modelling Design Patterns: Application to Pattern Detection and Code Synthesis**. In Proceedings of ECOOP Workshop on Automating Object-Oriented Software Development Methods, 2001. Disponível em http://www.yann-gael.gueheneuc.net/Work/Publications/Documents/ECOOP01AOOSDM.doc.pdf Acesso em : 27/09/2005

ALBIN-AMIOT, H., GUEHENEUC**,** Y. G. **Design Patterns: A Round-trip**. Proceedings of ECOOP Workshop for PhD Students in Object-Oriented Systems, 2001. Disponível em http://www.yann-gael.gueheneuc.net/Work/Publications/Documents/ECOOP01PHDOOS.doc.pdf Acesso em : 27/09/2005

ALUR, D., CRUPI, J., MALKS, D. **Core J2EE Patterns: Best Practices and Design Strategies.** Prentice Hall PTR; 1a edição, junho, 2001.

APPERLY, H. **Configuration Management and Component Libraries**. Component-based software engineering: putting the pieces together, pág. 513 – 526, Addison-Wesley, 2001. Disponível em : http://www.selectbs.com/downloads/techpapers/tp_Component_Management_a4.pdf Acesso em : 27/09/2005

APPERLY, H., *et al*. **Service - and Component-based Development: Using Select Perspective™ and UML**. Addison Wesley, janeiro, 2003.

ARANDA, G., MOORE, R. **GoF Creational Patterns: A Formal Specification**. Technical Report 224, UNU/IIST, P.O. Box 3058, Macau, 2000. Disponível em http://citeseer.ist.psu.edu/aranda00gof.html Acesso em : 27/09/2005

ARANDA, G., MOORE, R. **A formal model for verifying compound design patterns**,. Proceedings of the 14th international conference on Software engineering and knowledge engineering Proceedings, Ischia, Itália, 2002. Disponível em http://portal.acm.org/citation.cfm?id=568797 Acesso em : 27/09/2005

ARLOW, J., NEUSTADT**,** I. **Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML.** Addison-Wesley, dezembro, 2003, ISBN: 032111230X.

ARTHUR, J. D., SARGENT, R. G. **Verification and Validation: What Impact *Should* Project Size and Complexity Have on Attendant V&V Activities and Supporting**

Infrastructure? Proceedings of the 1999 Winter Simulation Conference, 1999. Disponível em : http://eprints.cs.vt.edu/archive/00000510/ Acesso em : 27/09/2005

BACHMAN, C.W., DAYA, M. **The role concept in data models**. In Proceedings of the Third International Conference on Very Large Databases, pages 464-476, Tokyo, 1977.

BALCI, O., *et al*. **Expanding our Horizons in Verification, Validation, and Accreditation Research and Practice.** Proceedings of the 32nd conference on Winter simulation, 2002. Disponível em : http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/8343/26360/01172944.pdf?arnumber=1172944 Acesso em : 27/09/2005

BALCI, O., SAADI, S. D. **Proposed Standard Processes For Certification of Modeling And Simulation Applications.** Proceedings of the 32nd conference on Winter simulation, 2002. Disponível em : http://www.informs-cs.org/wsc02papers/222.pdf Acesso em : 27/09/2005

BERTOLINO, A., POLINI, A. **A Framework for Component Deployment Testing.** Proceedings of the 25th international conference on Software engineering, Portland, Oregon, 2003. Disponível em : http://portal.acm.org/citation.cfm?id=776843&dl=ACM&coll=GUIDE Acesso em : 27/09/2005

BLANKERS, T. **Combining models and patterns: delivering on the promise of increased IT productivity.** COMPUWARE White Paper Series, 2003. Disponível em : http://www.compuware.com/products/optimalj/1794_ENG_HTML.htm#MDA Acesso em : 27/09/2005

BOERTIEN, N., STEEN, M. W. A., JONKERS, H. **Evaluation of Component-Based Development Methods.** In EMMSAD'2001, Sixth CAiSE/IFIP8.1, 2001. Disponível em: http://ww5.introcom.net/~nkcho/article/TI_evaluation_paper_EMMSAD01.pdf . Acesso em: 27/09/2005

BRAGA, R.; WERNER, C.; MATTOSO, M. **Odyssey: A Reuse Environment based on Domain Models**. IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'99), pp.50-57, Texas, março, 1999. Disponível em http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/asset/1999/0122/00/0122toc.xml&DOI=10.1109/ASSET.1999.756751 . Acesso em : 27/09/2005

BRAGA, R.; WERNER, C.; MATTOSO, M. **The Use of Mediators for Component Retrieval in a Reuse Environment**. Workshop on Component-Based Software Engineering Process, Technology of Object-Oriented Languages and Systems Conference, Santa Bárbara, agosto, 1999. Disponível em http://reuse.cos.ufrj.br/odyssey/site/publicacoes/tools99f.pdf . Acesso em: 27/09/2005

BROWN, A.W. **Large-Scale, Component-Based Development**. Prentice-Hall, 2000.

BUSCHMANN, F. *et al*. **Pattern-Oriented Software Architecture: A System of Patterns**. John Wiley & Sons, 1996.

CAI, X., LYU, M.R., WONG, K, Ko, R. **Component-based software engineering: technologies, development frameworks, and quality assurance schemes,** Seventh Asia-Pacific Software Engineering Conference (APSEC'00), Singapura, 05-08, dezembro, 2000. Disponível em : http://csdl.computer.org/comp/proceedings/apsec/2000/0915/00/09150372abs.htm Acesso em : 27/09/2005

CARR III, J. T., BALCI, O. **Verification and Validation of Object-Oriented Artifacts Throughout the Simulation Model Development Life Cycle.** Winter Simulation Conference. Proceedings of the 32nd conference on Winter simulation, Flórida, 2000.

CBDi. **Component Based Development, Using Componentised Software. The Forum for Component Based Development and Integration.** A CBDi Forum Report., maio, 1999. Disponível em : http://www.cbdiforum.com/bronze/cbdguide/cbd_report.pdf Acesso em : 27/09/2005

CHEESMAN, J., DANIELS, J. **UML Components.** Addison-Wesley, outubro, 2000

COMPUWARE. **OptimalJ: How transformation patterns transform UML models into high-quality J2EE applications.** COMPUWARE White Paper Series, 2004. Disponível em : http://www.compuware.com/products/optimalj/1794_ENG_HTML.htm#MDA Acesso em : 27/09/2005

COMPUWARE White Paper Series, 2005. Disponível em : http://www.compuware.com/products/optimalj/1794_ENG_HTML.htm#MDA Acesso em : 27/09/2005.

COOPER, D., *et al*. **Java Implementation Verification Using Reverse Engineering.** ACM International Conference Proceeding Series. Proceedings of the 27th conference on Australasian computer science - Volume 26, pag. 203 – 211, Dunedin, Nova Zelandia, 2004. Disponível em http://crpit.com/confpapers/CRPITV26Cooper.pdf Acesso em : 27/09/2005

COPLIEN, J. O. **Software Patterns.** AT&T, 1996. Disponível em : http://www.cs.chalmers.se/~kentp/Xjobb/asman_engene.pdf Acesso em : 27/09/2005

COPLIEN, J. O. **On the Nature of *The Nature of Order*.** The Chicago Patterns Group, por Brad Appleton, agosto 1997. Disponível em : http://www.cmcrossroads.com/bradapp/docs/NoNoO.html Acesso em : 27/09/2005

CORREA, A. L., WERNER, C. M. L., ZAVERUCHA, G. **Uma Arquitetura de Suporte à Avaliação de Modelos Orientados a Objetos.** Terceiro Workshop Ibero-americano de Engenharia de Requisitos e Ambientes de Software (IDEAS'00). v.1. p.385 – 396, 2000. Disponível em : http://reuse.cos.ufrj.br/odyssey/site/publicacoes/Ideas00Final.pdf Acesso em : 27/09/2005

CORREA, A. L., WERNER, C. M. L., ZAVERUCHA, G. **Object Oriented Design Expertise Reuse: an Approach Based on Heuristics, Design Patterns and Anti-Patterns.** 6th International Conerence, ICSR-6, Viena, Áustria, 27-29, junho, 2000 Disponível em : http://citeseer.ist.psu.edu/cache/papers/cs/21991/http:zSzzSzwww.cos.ufrj.brzSz~odysseyzSzpublicacoeszSzIcsrFinal.pdf/object-oriented-design-expertise.pdf Acesso em : 27/09/2005

COUNCILL, B. **Third-Party Certification and Its Required Elements.** Proceedings of 4th ICSE Workshop on Component Based Software Engineering - Component Certification and System Prediction, Toronto, Canada, maio, 2001. Disponível em : http://www.sei.cmu.edu/pacc/CBSE4_papers/Councill-CBSE4-20.pdf Acesso em : 27/09/2005

CRNKOVIC, I., LARSSON, M. **Building Reliable Component-Based Software Systems.** Artech House Publishers, Julho, 2002. ISBN 1-58053-327-2.

CRUPI, J., BAERVELDT, F. **Implementing Sun Microsystems Core J2EE Patterns.**

CWM. **OMG Common Warehouse Metamodel Specification. Versão 1.1.** OMG, março, 2003. Disponível em http://www.omg.org/docs/formal/03-03-02.pdf Acesso em : 27/09/2005

D´SOUSA, D. F., WILLS, A. C. **Objects, Componentes and Frameworks with UML – *The Catalysis Approuch***. Addison Wesley, 1998

DIRCKZE, R., BAISLEY, D., IYENGAR, S. **JMI Brings MDA to the J2EE Environment – An Overview;** Unisys Corporation white paper, 2002. Disponível em http://ecommunity.unisys.com/unisys/solution_center/sc_documents/WhitePaper/jmiWP.pdf Acesso em : 27/09/2005

DONG, J. **UML Extensions for Design Pattern Compositions.** In Journal of Object Technology, novembro-dezembro, 2002. Disponível em http://www.jot.fm/issues/issue_2002_11/article3.pdf Acesso em : 27/09/2005

DONG, J., YANG, S. **Extending UML To Visualize Design Patterns In Class Diagrams.** The Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE), pp124-131, San Francisco Bay, California, USA, julho, 2003. Disponível em http://www.utdallas.edu/~jdong/papers/seke03.pdf Acesso em : 27/09/2005

DONG, J., YANG, S. **Visualizing Design Patterns With A UML**. The Proceedings of the IEEE Symposium on Visual/Multimedia Languages (VL), pag.123-125, Auckland, Nova Zelandia, outubro, 2003. Disponível em http://www.utdallas.edu/~jdong/papers/UTDCS-11-03.pdf Acesso em : 27/09/2005

ECKERSON, W. W., MANES, A. T. **Paving the Way for Transparent Application and Data Interchange. A Java API for Metadata.** SUN JMI White Papers, novembro, 1999. Disponível em : http://java.sun.com/products/jmi/pdf/JavaMetadataAPI.pdf Acesso em : 27/09/2005

EJB. **Enterprise JavaBeans ™ 3.0**. Sun Microsystems, Public Review, agosto, 2005. Disponível em: http://jcp.org/aboutJava/communityprocess/pr/jsr220/index.html Acesso em : 27/09/2005

FERREIRA, B. H. A. **Novo Dicionário Aurélio da Lingua Portuguesa.** Editora Nova Fronteira, 1986.

FLORES, A., MOORE, R. **GoF Structural Patterns: A Formal Specification.** Technical Report 207. UNU/IIST, P.O. Box 3058, Macau, agosto, 2000.

FLORES, A., REYNOSO, L., MOORE, R. **A Formal Model of Object Oriented Design and GoF Design Patterns**. In proceedings of the FME 2001, Formal Methods Europe, Berlin, Germany, 2001.

FONTOURA, M., PREE, W., RUMPE, B. **The UML Profile for Framework Architectures,** Tutorial in ECOOP, Cannes, junho, 2000. Disponível em http://www.almaden.ibm.com/cs/people/fontoura/papers/tutorial-ecoop2000.pdf Acesso em : 27/09/2005

FONTOURA, M., PREE, W., RUMPE, B. **UML-F A Modeling Language for Object-Oriented Frameworks**. ECOOP, Cannes, junho, 2000. Disponível em http://www.almaden.ibm.com/cs/people/fontoura/papers/ecoop2000.pdf Acesso em : 27/09/2005

FONTOURA, M., PREE, W., RUMPE, B. **The UML Profile for Framework Architectures**. Addison-Wesley, dezembro, 2001.

FOWLER, M. **Analysis Patterns: Reusable Object Models.** Addison-Wesley, 1996.

FRANCE, R., *et al*. **Using Role-Based Modeling Language (RBML) to Characterize Model Families**. Eighth International Conference on Engineering of Complex Computer Systems, Greenbelt, Maryland, dezembro, 2002. Disponível em http://csdl.computer.org/comp/proceedings/iceccs/2002/1757/00/17570107abs.htm Acesso em 27/09/2005

FRANCE, R., *et al*. **A UML-Based Metamodeling Language to Specify Design Patterns**. In Proceedings of Workshop on Software Model Engineering (WiSME) with UML 2003, San Francisco, outubro, 2003. Disponível em http://www.metamodel.com/wisme-2003/01.pdf Acesso em : 27/09/2005

FRANCE, R., *et al*. **A Role-Based Metamodeling Approach to Specifying Design Patterns**. In Proceeding of 27th IEEE Annual International Computer Software and Applications Conference (COMPSAC) 2003, Dallas, Texas, 3-6, novembro, 2003. Disponível em http://csdl.computer.org/comp/proceedings/compsac/2003/2020/00/20200452abs.htm Acesso em : 27/09/2005

FRANCE, R., *et al*. **A UML-Based Pattern Specification Technique**. IEEE Transactions on Software Engineering, 2004. Disponível em http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=1271174 Acesso em : 27/09/2005

FRANKEL, D. S. **Model Driven Architecture**. **Applying MDA to Enterprise Computing.** John Wiley & Sons OMG Press, 2003.

FUTRELL, R. T., SHAFER, D. F., SAFER, L. I. **Quality Software Project Management.** Prentice Hall, janeiro, 2002. ISBN 0-13-091297.

GAMMA, E., *et al*. **Design Patterns: Elements of reusable object-oriented software**. Addison-Wesley, 1995.

GHOSH, S., MATHUR, A. P. **Certification of Distributed Component Computing Middleware and Applications.** Proceedings of the 4th CBSE workshop during ICSE 2001, Toronto, Canada, 14-15, maio, 2001,. Disponível em : http://www.sei.cmu.edu/pacc/CBSE4_papers/Ghosh-CBSE4-3.pdf Acesso em : 21/12/2004

GOSLING, J. *et al.* **The Java Language Specification**. Third Edition. Addison-Wesley. maio, 2005. ISBN 0-321-24678-0.

GRISS, M. L. **CBSE Success Factors: Integrating Architecture, Process, and Organization. Component-Based Software Engineering: Putting the Pieces Together.** Cap. 9, Addison-Wesley, maio, 2001. Disponível em : http://martin.griss.com/pubs/cbse-success-factors.pdf Acesso em : 27/09/2005

HARRISON W., BARTON C., RAGHAVACHARI M. **Mapping UML Designs to Java.** Conference on Object Oriented Programming Systems Languages and Applications, Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 2000. Disponível em http://www.prolangs.rutgers.edu/refs/docs/Harrison-oopsla00.pdf Acesso em : 27/09/2005

HEDIN, G. **Language Support for Design Patterns using Attribute Extension**. In Proceedings of the Workshop on Language Support for Design Patterns and Frameworks (LSDF'97), held in conjunction with ECOOP'97, Jyväskylä, Finlandia, junho, 1997. Disponível em http://www.cs.lth.se/Research/ProgEnv/LSDF.html Acesso em : 27/09/2005

HERZUM, P., SIMS, O. **Business Component Factory**. John Wiley & Sons, dezembro, 1999.

HONG, M., *et al*. **Component Metrics in Jade Bird Component Library System.** Journal of Software, Vol. 11, No.5, pp.634-641, maio, 2000.

IEEE. **IEEE Standards for Software Verification and Validation**. IEEE Standard 1012. Washington, DC: IEEE Computer Society, 1998.

JONKERS, H., *et al*. **Component-Based Rapid Service Development - State-of-the-Art and Definitions.** GigaTS/D1.3.4 - Telematica Instituut, maio de 2000. Disponível em: https://doc.telin.nl/dscgi/ds.py/Get/File-9157/D1.3.4_CBD_for_RSD.pdf. Acesso em: 27/09/2005

JSR 175**. A Program Annotation Facility for the Java^{tm} Programming Language.** JSR-175 final release specification. JCP, setembro, 2004. Disponível em http://www.jcp.org/en/jsr/detail?id=175 Acesso em : 27/09/2005

JSR026. **UML/EJB Mapping Specification.** JCP Public Draft, agosto, 2001. Disponível em : http://www.jcp.org/aboutJava/communityprocess/review/jsr026/ Acesso em : 27/09/2005

JSR069. **Java OLAP Interface (JOLAP)**. JCP Proposed Final Draft, setembro, 2003. Disponível em : http://jcp.org/aboutJava/communityprocess/first/jsr069/index.html Acesso em : 27/09/2005

JSR073. **Data Mining API.** JCP Maintenance Release, agosto, 2005. Disponível em : http://jcp.org/aboutJava/communityprocess/mrel/jsr073/ Acesso em : 27/09/2005

JSR244. **Java Platform Enterprise Edition 5 Specification.** JCP Close of Public Review, agosto, 2005. Disponível em : http://jcp.org/aboutJava/communityprocess/pr/jsr244/index.html Acesso em : 27/09/2005

JSR40. **Java Metadata Interface(JMI) Specification. Versão 1.0.** OMG**,** junho**,** 2002. Disponível em http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html Acesso em : 27/09/2005

KALLIO, P., NIEMELÄ**,** E. **Documented Quality of COTS and OCM Components**. Proceedings of 4th ICSE Workshop on Component Based Software Engineering - Component Certification and System Prediction, Toronto, Canada, maio, 2001. Disponível em : http://www.sei.cmu.edu/pacc/CBSE4_papers/Kallio-CBSE4-2.pdf Acesso em : 27/09/2005

KARLSSON, D., ELES, P., PENG, Z. **Formal Verification in a Component-based Reuse Methodology.** Proceedings of the 15th international symposium on System Synthesis (ISSS '02), Kyoto, Japão, outubro, 2002,.
Disponível em : http://www.ida.liu.se/labs/eslab/publications/pap/db/ISSS02.pdf Acesso em : 27/09/2005

KLEPPE, A., WARMER, J., BAST, W. **MDA Explained The Model Driven ArchitectureÖ Practice and Promise.** Addison Wesley, abril, 2003.

KOTHARI, S. C. , *et al***. A Pattern-Based Framework for Software Anomaly Detection.** Software Quality Control Volume 12, Issue 2, pag. 99 – 120, junho, 2004. Disponível em: http://portal.acm.org/citation.cfm?id=984558.984580&dl=GUIDE&dl=GUIDE&CFID=58657070&CFTOKEN=74981093 Acesso em : 27/09/2005

LARSSON, T., SANDBERG, M. **Building Flexible Components Based on Design Patterns.** In Component-based Software Engineering -- State of the Art, Technical report, Department of Computer Engineering, Mälardalen University, pp. 144-152, Västerås, Suécia, 2000. Disponível em : http://www.idt.mdh.se/personal/tla/publ/patbacom_tr2000.pdf Acesso em : 27/09/2005

LE GUENNEC, A., SUNYE, G., JEZEQUEL, J. M. **Precise Modeling of Design Patterns.** Third International Conference, York, UK, outubro, 2000. Disponível em http://www.irisa.fr/triskell/publis/2000/LeGuennec00a.pdf Acesso em : 27/09/2005

LEE, S. D., *et al*; **COMO: a UML-based component development methodology.** Software Engineering Conference, 1999, (APSEC '99) Proceedings. Sixth Asia Pacific, Page(s): 54 – 61, 1999

LUCRÉDIO, D., *et al*. **Integrando Tecnologias para o Desenvolvimento de Software Baseado em Componentes Distribuídos**. Simpósio Brasileiro de Engenharia de Software

(SBES'2002) - VII Workshop de Teses em Engenharia de Software (WTES), Pág 25 - 28. ISBN: 85-88442-33-7, Gramado/RS – Brasil, outubro, 2002. Disponível em : http://www.recope.dc.ufscar.br/engenhariadesoftware/publicacoes/wtese.pdf Acesso em : 27/09/2005

LUCRÉDIO, D., *et al*. **MVCASE Tool – Working with Design Patterns**. The Third Latin American Conference on Pattern Languages of Programming (SugarLoafPlop), Writers' Workshop, Porto de Galinhas/Recife, Brasil, 2003. Disponível em : http://www.cin.ufpe.br/~sugarloafplop/final_articles/16_MVCase.pdf Acesso em : 27/09/2005

MARINESCU, F. **EJB Design Patterns. Advanced Patterns, Processes and Idioms**. John Wiley & Sons, 2002.

MATULA, M. **NetBeans Metadata Repository.** SUN white Paper, março, 2003. Disponível em http://mdr.netbeans.org/MDR-whitepaper.pdf Acesso em : 27/09/2005

MCLAUGHLIN**,** B. **Annotations in Tiger, Part 1: Add metadata to Java code. How to use Java 5's built-in annotations.** IBM developerWorks, Java technology, setembro, 2004. Disponível em http://www-106.ibm.com/developerworks/java/library/j-annotate1/ Acesso em : 27/09/2005

MDA. **Model Driven Architecture.** Document number ormsc/2001-07-01, julho, 2001. Disponível em http://www.omg.org/docs/ormsc/01-07-01.pdf Acesso em : 27/09/2005

MDA. **Guide Version 1.0.1.** Document Number: omg/2003-06-01, junho, 2003. Disponível em: http://www.omg.org/docs/omg/03-06-01.pdf Acesso em : 27/09/2005

MELLOR, S. J., *et al*. **MDA Distilled: Principles of Model-Driven Architecture.** Addison Wesley, março, 2004.

METSKER, S. J. **Design Patterns Java Workbook**. Addison-Wesley, março, 2002.

MEYER, B. **The Grand Challenge of Trusted Components**, International Conference on Software Engineering**,** Proceedings of the 25th international conference on Software engineering, Portland, Oregon, 2003. Disponível em : http://portal.acm.org/citation.cfm?id=776911&dl=ACM&coll=portal Acesso em : 27/09/2005

MOF. **Meta-Object Facility Specification, version 1.4.** OMG**,** abril, 2002. Disponível em http://www.omg.org/technology/documents/formal/mof.htm Acesso em : 27/09/2005

MONTES, J. L. I., VELA, F. L. G. **Structural Modeling of Design Patterns: REP Diagrams**. Proceedings 2003 (the third Latin American Conference on Pattern Languages of Programming), pág. 301-309, Porto de Galinhas, Pernambuco, Brasil, 12-15, agosto, 2003. Disponível em http://www.cin.ufpe.br/~sugarloafplop/final_articles/19_REP.pdf Acesso em : 27/09/2005

MORRIS, J., *et al*. **Software Component Certification,** IEEE Computer Society Press, Volume 34 , Issue 9, setembro, 2001. Disponível em : http://csdl.computer.org/comp/mags/co/2001/09/r9030abs.htm Acesso em : 27/09/2005

MOSHER, C. **A New Specification for Managing Metadata.** SUN white Paper, abril, 2002. Disponível em: http://java.sun.com/developer/technicalArticles/J2EE/JMI/ Acesso em: 27/09/2005.

NENTWICH, C., *et al*. **Flexible Consistency Checking.** ACM Transactions on Software Engineering and Methodology (TOSEM) Volume 12 , Issue 1, pag. 28 – 63, janeiro, 2003. Disponível em : http://portal.acm.org/citation.cfm?id=839268.839271&dl=GUIDE&dl=ACM&idx=839268&part=periodical&WantType=periodical&title=ACM%20Transactions%20on%20Software%20Engineering%20and%20Methodology%20(TOSEM) Acesso em : 27/09/2005

NICKEL, U.A., *et al*. **Roundtrip Engineering with FUJABA.** In Proc. of 2nd Workshop on Software-Reengineering (WSR), Bad Honnef, Alemanha, Technical Report , agosto, 2000. Disponível em : http://www.uni-koblenz.de/~ist/WSR2000/slides/NickelNiereWadsackZuendorf.pdf Acesso em : 27/09/2005

NIERE, J., WADSACK, J.P., ZÜNDORF , A. **Recovering UML Diagrams from Java Code using Patterns.** Proc. of 2nd Workshop on Soft Computing Applied to Software Engineering, Enschede, The Netherlands, Lecture Notes in Computer Science (LNCS). Springer, 2001. Disponível em : http://wwwcs.upb.de/fachbereich/AG/schaefer/Personen/Ehemalige/Zuendorf/SCASE2001.pdf Acesso em : 27/09/2005

OCL. **OMG UML 2.0 OCL Final Adopted specification.** OMG**,** Document -- ptc/03-10-14, outubro**,** 2003. Disponível em: http://www.omg.org/docs/ptc/03-10-14.pdf Acesso em : 27/09/2005

OMG. **Metamodel and UML Profile for Java and EJB Specification.** Version 1.0, fevereiro, 2004. Disponível em : http://www.omg.org/docs/formal/04-02-02.pdf Acesso em : 27/09/2005

PARR, T. **Building Recognizers By Hand.** Practical Computer Language Recognition and Translation, 1999. Disponível em http://www.antlr.org/book/byhand.pdf Acesso em : 27/09/2005

PARR, T. **Language.** Practical Computer Language Recognition and Translation, 1999. Disponível em http://www.antlr.org/book/language.pdf Acesso em : 27/09/2005

PETRI, D., CSERTAN, G. **Design Pattern Matching.** PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 47, NO. 3–4, PP. 205–212, 2003.

PISTER, M. **Examples for Component Certification.** EMPRESS Consortium, Alemanha, janeiro, 2004, versão 1.0. Status: Final. Versão pública. Disponível em : http://www.empress-itea.org/deliverables/D3.7_v1.0_Public_Version.pdf Acesso em : 27/09/2005

POELS, G**. Functional Size Measurement of Multi-Layer Object Oriented Conceptual Models**. Working Paper series, 2003. Disponível em : http://www.feb.ugent.be/fac/research/WP/Papers/wp_03_184.pdf Acesso em : 27/09/2005

POOLE, J. D. **Model-Driven Architecture: Vision, Standards And Emerging Technologies.** Workshop on Metamodeling and Adaptive Object Models, abril, 2001. Disponível em: http://www.cwmforum.org/Model-Driven%20Architecture.pdf Acesso em : 27/09/2005

POOLE, J., *et al*. **Common Warehouse Metamodel Developer's Guide**. John Wiley & Sons, janeiro, 2003, ISBN: 0-471-20243-6.

QVT. **OMG MOF 2.0 Query / Views / Transformations RFP.** OMG**,** Document -- ad/02-04-10, outubro**,** 2002. Disponível em: http://www.omg.org/docs/ad/02-04-10.pdf Acesso em : 27/09/2005

RAMACHANDRAN, M. **Testing Reusable Software Components from Object Specification.** ACM SIGSOFT Software Engineering Notes, Volume 28 edição2, março, 2003. Disponível em : http://portal.acm.org/citation.cfm?id=638750.638790&dl=GUIDE&dl=ACM&idx=638750&part=periodical&WantType=periodical&title=ACM%20SIGSOFT%20Software%20Engineering%20Notes Acesso em : 27/09/2005

RATIONAL. **Rational Unified Process – Best Practice for Software Development Teams**. Rational Software Corporation White Paper, novembro, 2001.

REYNOSO, L., MOORE, R. **GoF Behavioural Patterns: A Formal Specification**. Technical Report 201, UNU/IIST, P.O. Box 3058, Macau, maio, 2000.

ROHDE, S. L., *et al*. **Certification of reusable software components: summary of work in progress,** 2nd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '96), Montreal, CANADA, 21 - 25, outubro, 1996. Disponível em : http://csdl.computer.org/comp/proceedings/iceccs/1996/7614/00/76140120abs.htm Acesso em : 27/09/2005

SAMETINGER, J. **Software Engineering with Reusable Components.** Springer, 1997.

SCHMIDT, D. C., JOHNSON, R. E., FAYAD, M. **Software Patterns.** Editorial for the Communications of the ACM, **Special Issue on Patterns and Pattern Languages.** Vol. 39, No. 10, outubro, 1996. Disponível em : http://www.cs.wustl.edu/~schmidt/CACM-editorial.html Acesso em : 27/09/2005

SELECT B.S. **Managing Components White Paper Version 3.1.** Select Business Solutions, Inc, 2003. Disponível em : http://www.selectbs.com/forms/whitepapers_form.htm Acesso em : 27/09/2005

SELECT. **Select Component Manager and Select Component Portal.** Select Business Solutions Technical Overview**,** Version 2.0**,** 2003. Disponível em : http://www.selectbs.com/forms/whitepapers_form.htm Acesso em : 27/09/2005

SELECT. **Suporting the OMG Model Driven Architecture (MDA) for Service and Component Based Development**. Select Business Solutions White Paper, 2004. Disponível em : http://www.omg.org/mda/mda_files/SBS_and_MDA_(v5).pdf Acesso em : 27/09/2005

SOLBERG, A., BERRE, A. J. **Component based methodology handbook.** Norway: SINTEF, 1997.

STELTING, S., MAASSEN, O. **Applied Java Patterns**. Prentice Hall, dezembro, 2001.

STOERMER, C., O'BRIEN, L., VERHOEF, C. **Practice Patterns for Architecture Reconstruction**. Ninth Working Conference on Reverse Engineering (WCRE'02), Richmond, Virginia, outubro-novembro, 2002. Disponível em: http://www.sei.cmu.edu/organization/staff/lob/Stoermer_Patterns.pdf Acesso em: 27/09/2005

STOERMER, C., O'BRIEN, L., VERHOEF, C**. Software Architecture Reconstruction: Practice Needs and Current Approaches.** Technical Report CMU/SEI-2002-TR-024, 2002. Disponível em http://www.sei.cmu.edu/publications/documents/02.reports/02tr024.html Acesso em : 27/09/2005

THOMAS, W. M., CERINO, D. A. **Predicting Software Quality for Reuse.** Proceedings. TRI-Ada 95. Adas Role in Global Markets: Solutions for a Changing Complex World, pp. 367-77, Anaheim, California, United States, 1995. Disponível em : http://portal.acm.org/citation.cfm?id=376663&dl=ACM&coll=portal Acesso em : 27/09/2005

TRENDOWICZ, A., PUNTER, T. **Quality Modelling for Software Product Lines.** Proceedings of 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, 2003.

TRIREME. **Reuse and Component Management**.TriReme International White Paper, 1999. Disponível em : http://www.trireme.com/whitepapers/design/components/reuse_and_component_management.html. Acesso em : 27/09/2005

UML. **OMG Unified Modeling Language Specification. Versão 1.5.** OMG, março**,** 2003. Disponível em http://www.omg.org/docs/formal/03-03-01.pdf Acesso em : 27/09/2005

VITHARANA, P., ZAHEDI, F. M., JAIN, H. **Design, Retrieval, and Assembly in Component-based Software Development.** Communications of the ACM, Volume 46,

novembro 2003. Disponível em http://portal.acm.org/citation.cfm?id=948387&jmp=cit&dl=GUIDE&dl=ACM . Acesso em : 27/09/2005

VITHARANA, P., ZAHEDI, F. M., JAIN, H. **Knowledge-Based Repository Scheme for Storing and Retrieving Business Components: A Theoretical Design and an Empirical Analysis.** IEEE Transactions on Software Engineering, Volume 29, 2003. Disponível em http://csdl.computer.org/comp/trans/ts/2003/07/e0649abs.htm Acesso em : 27/09/2005

VOAS, J. M. **Certifying Off-the-Shelf Software Components,** IEEE Computer Society Volume 31 , Issue 6, junho, 1998. Disponível em : http://csdl.computer.org/comp/mags/co/1998/06/r6053abs.htm Acesso em : 27/09/2005

WILLIAMS, D. R. E., HUGHES, C. E., OROOJI, A. **A Mathematical Formalism for Specifying Design Patterns.** 17th International Symposium on Computer and Information Sciences (ISCIS XVII), Orlando, FL 28-30, outubro, 2002. Disponível em http://www.cs.ucf.edu/~ceh/Papers/ISCIS2002FormalismForDesignPatternsWilliams.pdf Acesso em : 27/09/2005

WOHLIN, C., REGNELL, B. **Reliability Certification of Software Components.** Proceedings of the 5th International Conference on Software Reuse, p.56, 02-05, junho, 1998. Disponível em : http://www.tts.lth.se/Personal/bjornr/Papers/ICSR98.pdf Acesso em : 27/09/2005

WOHLIN, C., RUNESON, P.; **Certification of Software Componentes**, IEEE Transactions on Software Engineering, Volume 6, nº 6, junho, 1994. Disponível em http://csdl.computer.org/comp/trans/ts/1994/06/e0494abs.htm Acesso em : 27/09/2005

WOODMAN, M., *et al*. **Issues of CBD Product Quality and Process Quality.** Proceedings of 4th ICSE Workshop on Component Based Software Engineering - Component Certification and System Prediction, maio, 14-15, Toronto, Canada, 2001. Disponível em : http://www.sei.cmu.edu/pacc/CBSE4_papers/Woodman+-CBSE4-15.pdf Acesso em : 27/09/2005

XMI. **OMG XML Metadata Interchange. Versão 2.0.** OMG**,** maio**,** 2003. Disponível em http://www.omg.org/docs/formal/03-05-02.pdf Acesso em : 27/09/2005

YAU, S. S. **Achieving Quality Software Development for Distributed Environments.** Proceedings of First Asia-Pacific Conference on Quality Software, 2000. Disponível em : http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=883790 Acesso em : 27/09/2005

APÊNDICE A

Descrição das Bibliotecas Utilizadas

Nesse apêndice é apresentada a relação de todas as bibliotecas utilizadas na implementação do sistema de certificação, com a descrição das suas funcionalidades.

jmi[1]**:** Essa biblioteca contém a API do JMI**,** que contém as interfaces reflexivas especificadas pelo JSR-40 (JMI). Essas interfaces podem ser usadas para acessar os metadados gerenciados pelo repositório MDR sem um conhecimento explícito do metamodelo desses metadados. Não possui nenhuma dependência de outra API;

mof[2]: Essa biblioteca contém a biblioteca com as interfaces JMI geradas para o metamodelo MOF 1.4**.** Pode ser usada especificamente para acessar metadados MOF (metadados de acordo com o metamodelo MOF 1.4**).** Possui dependência da API JMI**.** Qualquer interface JMI gerada estende as interfaces JMI reflexivas;

mdrapi[3]: Biblioteca da API do MDR**.** Define uma API genérica, adicional a API padrão, definida no JMI**,** para repositórios de metadados**.** É uma extensão da API do JMI**.** Possui dependência da API JMI e parte do openide-util (Lookup API);

jmiutils[4]: Biblioteca contendo utilitários genéricos que operam com as interfaces JMI reflexivas. Incluí a implementação do XMI reader, XMI writer, mapeamento JMI e outros. Todos esses utilitários podem ser utilizados com qualquer repositório compatível

[1] http://java.sun.com/products/jmi/download.html
[2] http://java.sun.com/products/jmi/download.html
[3] http://mdr.netbeans.org/download/daily.html
[4] http://mdr.netbeans.org/download/daily.html

com JMI, eles não são uma implementação JMI específica. Possuí dependência da API do MDR, API do JMI, API do MOF e parte do openide-util (Lookup API);

nbmdr[1]: Biblioteca contendo a implementação do núcleo do MDR, e um pré-repositório compatível com JMI, que pode ser utilizado com qualquer aplicação Java. Possuí dependência da API do MDR, API do JMI, API do MOF e parte do openide-util (Lookup API);

uml 1.4[2]: Essa biblioteca contém a biblioteca com as interfaces JMI geradas para o metamodelo UML 1.4. Pode ser usada especificamente para acessar metadados UML (metadados de acordo com o metamodelo UML 1.4**).** Possui dependência da API JMI**.** Qualquer interface JMI gerada estende as interfaces JMI reflexivas;

openide[3]: Biblioteca contendo APIs que fornecem um *framework* de propósito geral para criação de aplicações cliente, como por exemplo ferramentas de ambiente de desenvolvimento integrado. É utilizada na implementação do NetBeans. Possuí vários módulos como bibliotecas Java, que contém pedaços de funcionalidades (edição, suporte HTML, construção de interfaces gráficas, analisador Java, e outras);

antlr[4]: Biblioteca da ferramenta de reconhecimento de linguagem. Fornece um *framework* para construção de reconhecedores, compiladores e tradutores a partir de descrições gramaticais contendo ações do Java, C# ou C++. ANTLR fornece excelente suporte para construções em árvore e transformações;

[1] http://mdr.netbeans.org/download/daily.html
[2] http://mdr.netbeans.org/uml2mof/download/licenses.html
[3] http://openide.netbeans.org/,http://mdr.netbeans.org/download/daily.html ; http://www.netbeans.org/project/www/download/dev/javadoc/OpenAPIs/org/openide/util/package-tree.html
[4] http://www.antlr.org/

APÊNDICE B

Descrição dos Perfis UML Utilizados

Perfil UML para Representação de Padrões

Precisamos encontrar diversas informações sobre as características dos padrões nos seus modelos de representação. A certificação dos modelos de desenvolvimento dos componentes e aplicações que utilizam esses padrões é realizada de acordo com essas características. Para que essa certificação possa ser realizada num ambiente computacional, é necessário que essas informações possam ser compreendidas e processadas por ferramentas específicas. A definição de um perfil UML específico para representação de padrões, permite que essas informações sejam incorporadas nos modelos através de estereótipos e etiquetas.

No funcionamento do sistema de certificação proposto, é necessário processar as informações das seguintes características:

- Tipo de padrão: identifica se o padrão é utilizado na construção de modelos de negócio, de modelos de aplicação independentes de plataforma ou de modelos de aplicação dependentes de plataforma. Para incorporar essa informação nos modelos foram criados os estereótipos <<PNIP>>, <<PAIP>> e <<PAPE>>;
- Papel de classe no domínio: identifica nos modelos de negócio, se o elemento classe representa um conceito de entidade ou de serviço do domínio que o padrão representa. Para incorporar essa informação nos modelos foram criados os estereótipos <Entidade>> e <<Serviço>>;
- Catálogo do padrão: Identifica o catálogo de padrões a qual o respectivo padrão pertence. Essa informação não interfere no processamento da certificação, é uma informação de caráter documental. Resultou na criação da etiqueta (*tag*) catálogo, associada a elementos modelos;

- Camada da aplicação do padrão: Dentro da arquitetura em camadas, indica em que camada da aplicação o padrão é aplicado. Resultou na etiqueta (*tag*) camada associada a elementos modelos;
- Plataforma tecnológica: Indica a plataforma tecnológica onde o padrão é aplicado. Resultou na etiqueta (*tag*) plataforma associada a elementos de modelos;
- Nível de abstração do padrão: Considerando que não temos níveis de abstração pré-fixados num ambiente de desenvolvimento orientado a modelos. Precisamos identificar em que nível dentro da classificação de tipo, o padrão é aplicado (<<PNIP>>, <<PAIP>> e <<PAPE>>). Resultou na etiqueta (*tag*) nivelPadrao associada a elementos modelos;
- Mapeamentos possíveis entre padrões: Na transformação de um modelo para o próximo nível de abstração, precisamos conhecer que padrões podem ser utilizados no novo modelo gerado, com todos os mapeamentos possíveis para os elementos classe com papéis de padrão. Para incorporar essa informação nos modelos foi criada a etiqueta padroesAlvo associada a elementos classe. Essa etiqueta possui duas leis de formação descritas a seguir;
  - {nome do padrão}: utilizada nos modelos PNIP de nível maior que zero (retrata as transformações de um padrão de negócio em outro padrão de negócio de menor nível de abstração);
  - {nome do padrão, {nome do papel executado}}: utilizada nos demais tipos de modelo e níveis de abstração;
- Relacionamento entre papel de operação e papel de classe: Na definição do mapeamento, na passagem de padrões de domínio para padrões de aplicação, podemos mapear uma classe para mais de um papel do mesmo padrão. Nesse caso é preciso identificar para qual papel, as operações dessa classe irão ser transferidas.

Resultou na etiqueta papeisPadroesAlvo, associada aos elementos operação. Essa etiqueta possui a lei de formação descrita abaixo:

- {nome do padrão, nome do papel executado pela classe}.

A seguir são apresentados os quadros com o resumo dos estereótipos e das etiquetas (*tags*) criadas para este perfil.

Estereótipo	**Aplicado em**	**Definição**
<<PNIP>>	modelos	Padrão de negócio independente de plataforma, padrão utilizado na construção de modelos de negócio.
<<PAIP>>	modelos	Padrão de aplicação independente de plataforma, padrão utilizado na construção de modelos de aplicação que estão num nível de abstração em que a plataforma de desenvolvimento ainda não é especificada.
<<PAPE>>	modelos	Padrão de aplicação de plataforma específica, padrão utilizado na construção de modelos de arquitetura de aplicação e componente, onde a plataforma de desenvolvimento já está especificada.
<<Entidade>>	classe de modelos <<PNIP>>	Conceito de entidade no domínio do padrão.
<<Serviço>>	classe de modelos <<PNIP>>	Conceito de serviço no domínio do padrão.

Etiqueta (*tag*)	**Aplicada em**	**Definição**
catalogo	modelos	Catálogo a que o padrão pertence. Por exemplo “GoF”, “Core J2EE”, etc...
camada	modelos <<PAPE>>	Camada da aplicação em que o padrão deve ser utilizado. Seus valores podem ser: apresentação, negócio ou integração.
plataforma	modelos <<PAPE>>	Plataforma específica onde o padrão é aplicado
nivelPadrao	modelos	Nível de abstração em que o padrão é utilizado.
padroesAlvo	classes	Multivalorada, indicando os papéis de padrões que podem ser utilizados na transformação do respectivo elemento classe do modelo para o próximo nível de abstração.
papeisPadroesAlvo	operações	Multivalorada, indicando em que papel de padrão da classe a qual pertence, o respectivo elemento operação ficará na transformação para o próximo nível de abstração.

Na arquitetura em camadas, atualmente mais utilizada no desenvolvimento de software e padrão arquitetural da plataforma J2EE, a definição dos mapeamentos para a transformação de um padrão independente de plataforma em seus respectivos padrões de plataforma específica de primeiro nível de abstração, deverá considerar a plataforma tecnológica e a camada da aplicação para qual será realizada a transformação. A figura B.1 apresenta um exemplo, onde não é obrigatório que haja sempre um mapeamento para todas as camadas.

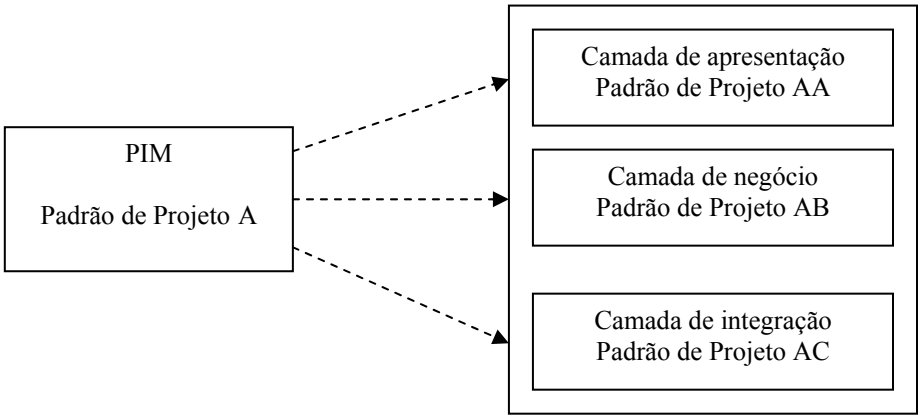


Figura B1: Mapeamento entre padrões (PIM → PSM).

Uma convenção importante é utilizar o nome do padrão no modelo construído para sua representação. A pesquisa no repositório é realizada pela busca de modelos com o nome dos padrões desejados e com o tipo específico, como padrões de negócio independentes de plataforma, padrões de aplicação independente de plataforma e de plataforma específica;

O menor nível de abstração de modelo PNIP, PAIP e PAPE é zero. Por exemplo, dentro do tipo PAIP, se nós quisermos trabalhar com três níveis de refinamento, teremos os níveis dois, um e zero, sendo o nível dois, o de maior nível de abstração, e o nível zero o último nível de padrão PAIP, obrigando que a próxima transformação seja para um padrão PAPE. No caso de um padrão PAPE ser do nível zero, indica que a próxima transformação será para o código;

Perfil UML para Utilização de Padrões

Para rastrear e identificar os padrões presentes nos modelos de componentes e aplicações, precisamos acrescentar algumas informações nos modelos de uma forma que possam ser compreendidas e processadas por ferramentas específicas de automatização do processo.

A definição de um perfil UML específico para representação de padrões, permite que essas informações sejam incorporadas nos modelos através de estereótipos e etiquetas.

O sistema de certificação proposto precisa identificar os padrões existentes nos modelos e todos os elementos com papéis nesses padrões para realizar a certificação da construção correta desses padrões. Essa certificação é realizada pela comparação com as construções dos modelos de representação desses padrões existentes no repositório de padrões.

As informações necessárias para a execução da certificação e que precisam ser encontradas nos modelos são:

- Tipo de modelo: identifica se os modelos se referem aos aspectos de negócio, de aplicação independente de plataforma ou de aplicação dependentes de plataforma. Para incorporar essa informação nos modelos foram criados os estereótipos <<MNIP>>, <<MAIP>> e <<MAPE>>;
- Papel de classe no domínio: identifica nos modelos de negócio, se o elemento classe representa um conceito de entidade ou de serviço do domínio que o modelo representa. Para incorporar essa informação nos modelos foram criados os estereótipos <Entidade>> e <<Serviço>>;
- Camada da aplicação do modelo: Dentro da arquitetura em camadas, indica em que camada da aplicação o modelo é utilizado. Resultou na etiqueta (*tag*) camada associada a elementos modelos;
- Plataforma tecnológica: Indica a plataforma tecnológica onde o modelo é utilizado. Resultou na etiqueta (*tag*) plataforma associada a elementos modelos;
- Nível de abstração do modelo: Precisamos identificar em que nível dentro da classificação de tipo (<<MNIP>>, <<MAIP>> e <<MAPE>>), o modelo é utilizado. Resultou na etiqueta (*tag*) nivelModelo associada a elementos modelos. A etiqueta nivelModelo nesse perfil tem o mesmo significado da etiqueta correspondente nivelPadrao no perfil de representação de padrões. Essa

correspondência permite a validação se os níveis dos modelos e dos respectivos padrões utilizados estão em conformidade;

- Mapeamento entre padrões: Na transformação de um modelo para o próximo nível de abstração, precisamos conhecer os padrões que serão utilizados no novo modelo gerado, com a definição de todos os mapeamentos dos elementos classe com papeis de padrão no modelo original. Para incorporar essas informações nos modelos foi criada a etiqueta padroesAlvo associada aos elementos classe. Essa etiqueta possui duas leis de formação descritas a seguir;
  - {nome do padrão}: utilizada nos modelos MNIP de nível maior que zero (retrata as transformações de um padrão de negócio em outro padrão de negócio de menor nível de abstração);
  - {nome do padrão, {nome do papel executado}}: utilizada em todos os demais tipos de modelo e níveis de abstração;
- Elementos com papéis de padrão: Identifica no modelo os elementos classe, atributo e operação que executam papéis de padrão. Para incorporar essas informações nos modelos foram criados os estereótipos <<ClassePadrao>>, <<AtributoPadrao>> e <<OperacaoPadrao;
- Papéis de padrões executados: Idêntica que papéis são executados por cada elemento classe, atributo ou operação pertencente a uma construção de padrão. Resultou na etiqueta papeisExecutados, associada aos elementos classe, atributo e operação, com a lei de formação descrita abaixo:
  - {nome do padrão}: utilizada nos modelos MNIP de nível maior que zero (retrata as transformações de um modelo de negócio em outro modelo de negócio de menor nível de abstração);

- {nome do padrão[número da instância], nome do papel executado}: utilizada nos demais tipos de modelo e níveis de abstração;

A seguir são apresentados os quadros com o resumo dos estereótipos e das etiquetas (*tags*) criadas para este perfil.

Estereótipo	**Aplicado em**	**Definição**
<<MNIP>>	modelos	Modelo de negócio independente de plataforma.
<<MAIP>>	modelos	Modelos de aplicação que estão num nível de abstração em que a plataforma de desenvolvimento ainda não é especificada.
<<MAPE>>	modelos	Modelo de aplicação de plataforma específica, modelos de arquitetura de aplicação e componente, onde a plataforma de desenvolvimento já está especificada.
<<Entidade>>	classe em modelos <<MNIP>> de maior nível de abstração	Conceito de entidade do domínio.
<<Serviço>>	classe em modelos <<MNIP>> de maior nível de abstração	Conceito de serviço do domínio.
<<ClassePadrao>>	classe	Elemento classe que executa papel de padrão nos modelos
<<AtributoPadrao>>	atributo	Elemento atributo que executa papel de padrão nos modelos.
<<OperacaoPadrao>>	operação	Elemento operação que executa papel de padrão nos modelos.

Etiqueta (*tag*)	**Aplicada em**	**Definição**
nivelModelo	modelos	Nível de abstração em que o modelo está inserido.
plataforma	modelos <<MAPE>>	Plataforma específica para qual o modelo foi construído.
camada	modelos <<MAPE>>	Camada da aplicação para qual o modelo foi construído. Seus valores podem ser: apresentação, negócio ou integração.
padroesExecutados	classes, atributos e operações (MNIP, MAIP, MAPE)	Multivalorado, indicando os papeis de padrões que o respectivo elemento do modelo está representando.
padroesAlvo	classes, atributos e operações (MNIP, MAIP, MAPE)	Multivalorada, indicando os papeis de padrões selecionados para a transformação do respectivo elemento do modelo para o próximo nível de abstração.

As leis de formação das etiquetas padroesAlvo, papeisPadroesAlvo e padroesExecutados nos dois perfis, são composições de determinadas informações sobre padrões definidas a seguir:

- nome do padrão : indica para que padrão será realizada uma transformação ou de que padrão o respectivo elemento está representando um papel**;**
- número da instância : indica de que ocorrência do padrão o respectivo elemento está representando um papel, considerando que o mesmo padrão pode ocorrer mais de uma vez num determinado modelo;
- nome do papel executado : indica para que papel será realizada uma transformação ou que papel o respectivo elemento está representando na ocorrência do padrão**;**

- nome do papel executado pela classe : indica em que papel executado pela classe dona do elemento, o papel desse elemento operação será incorporado após a execução da transformação;

APÊNDICE C

Guia de Utilização do Sistema de Certificação

Na interação com o sistema de certificação, podemos utilizar os menus ou a barra de ferramentas, conforme apresentado nas figuras D.1, D.2, D.3 e D.4.

De acordo com a opção escolhida no menu ou na barra de ferramentas, os ícones apropriados encontrados à esquerda e à direita da tela da aplicação são habilitados, permitindo a execução do respectivo serviço.

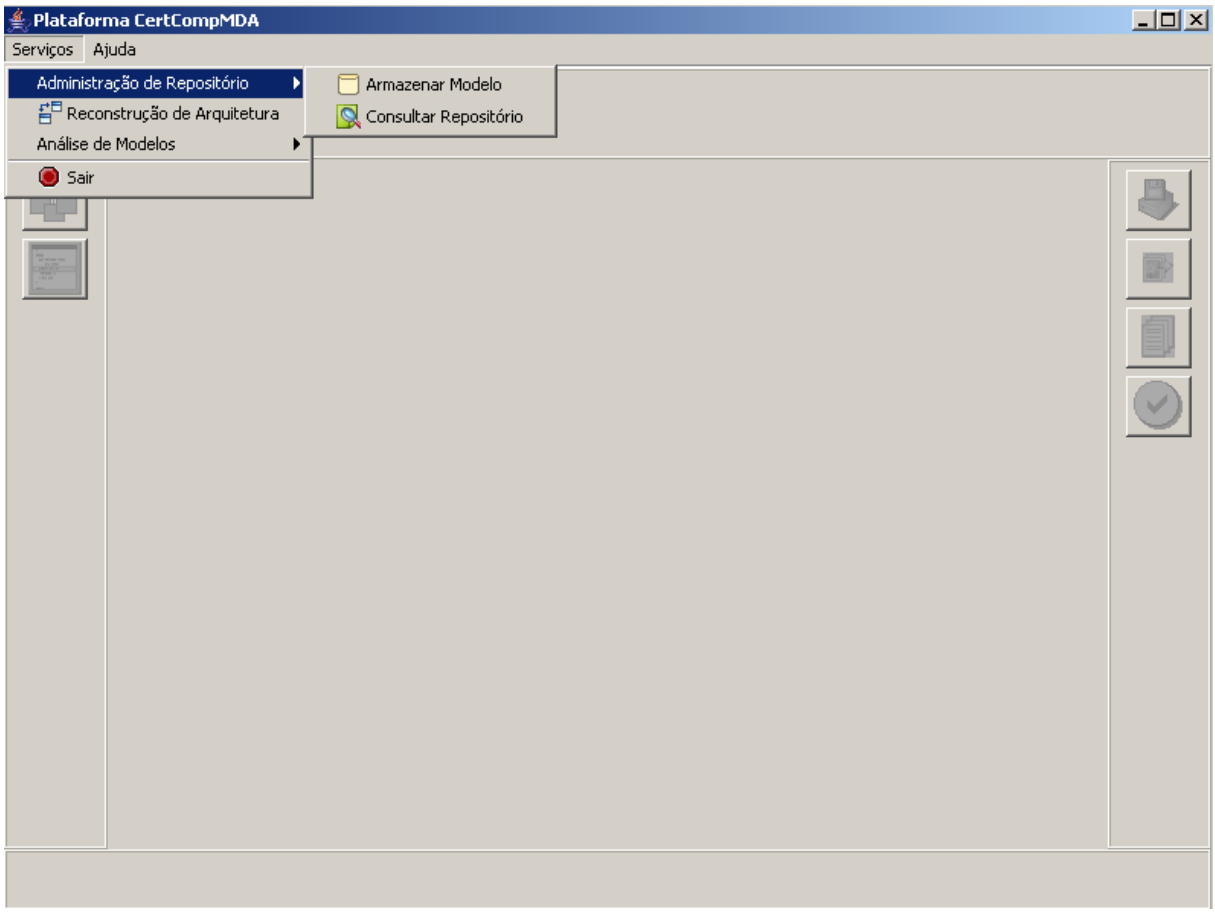


Figura D.1: Menus da aplicação de certificação (Administração do Repositório)

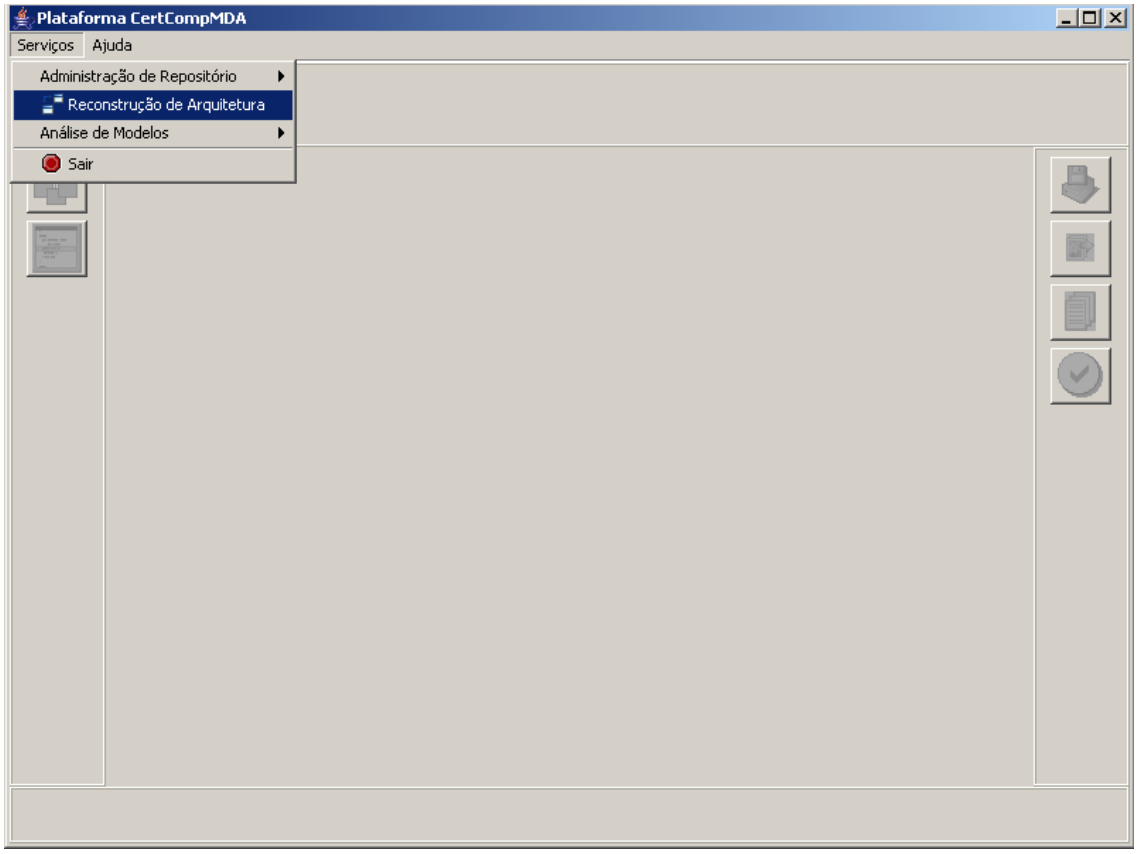


Figura D.2: Menus da aplicação de certificação (Reconstrução de Arquitetura)

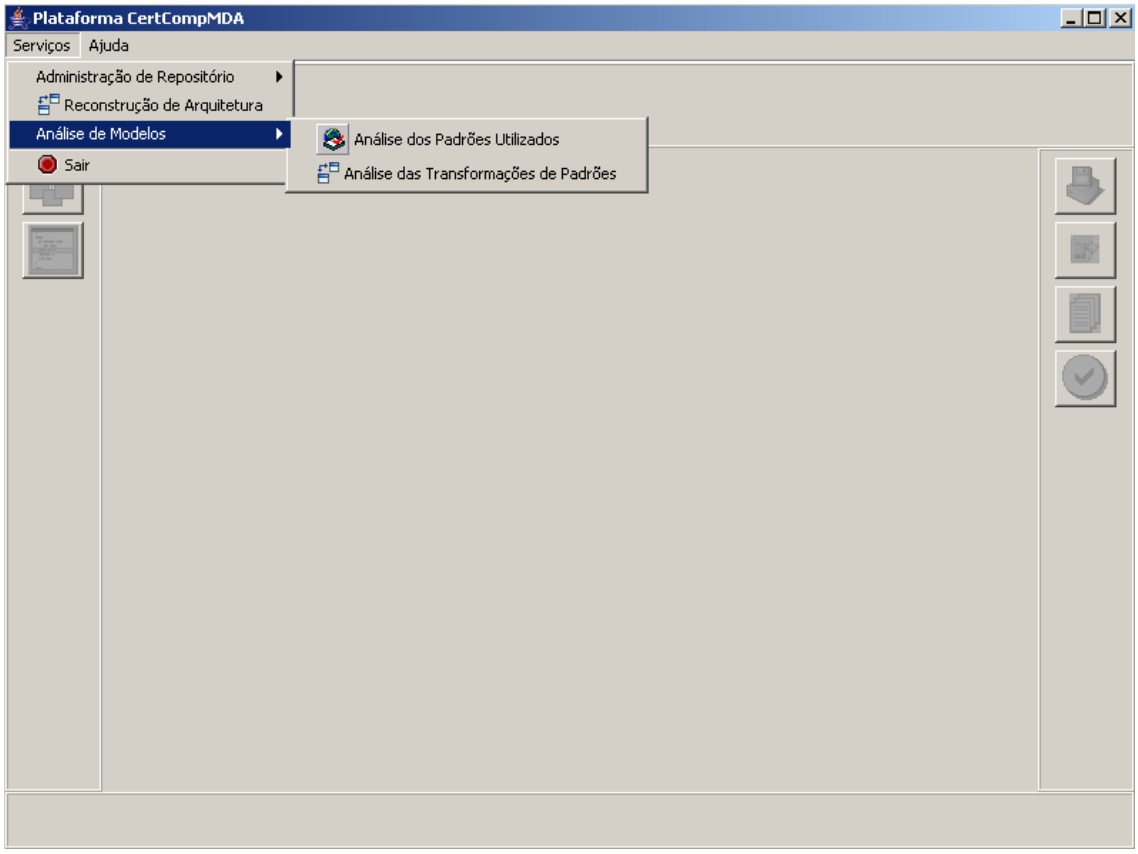


Figura D.3: Menus da aplicação de certificação (Análise)

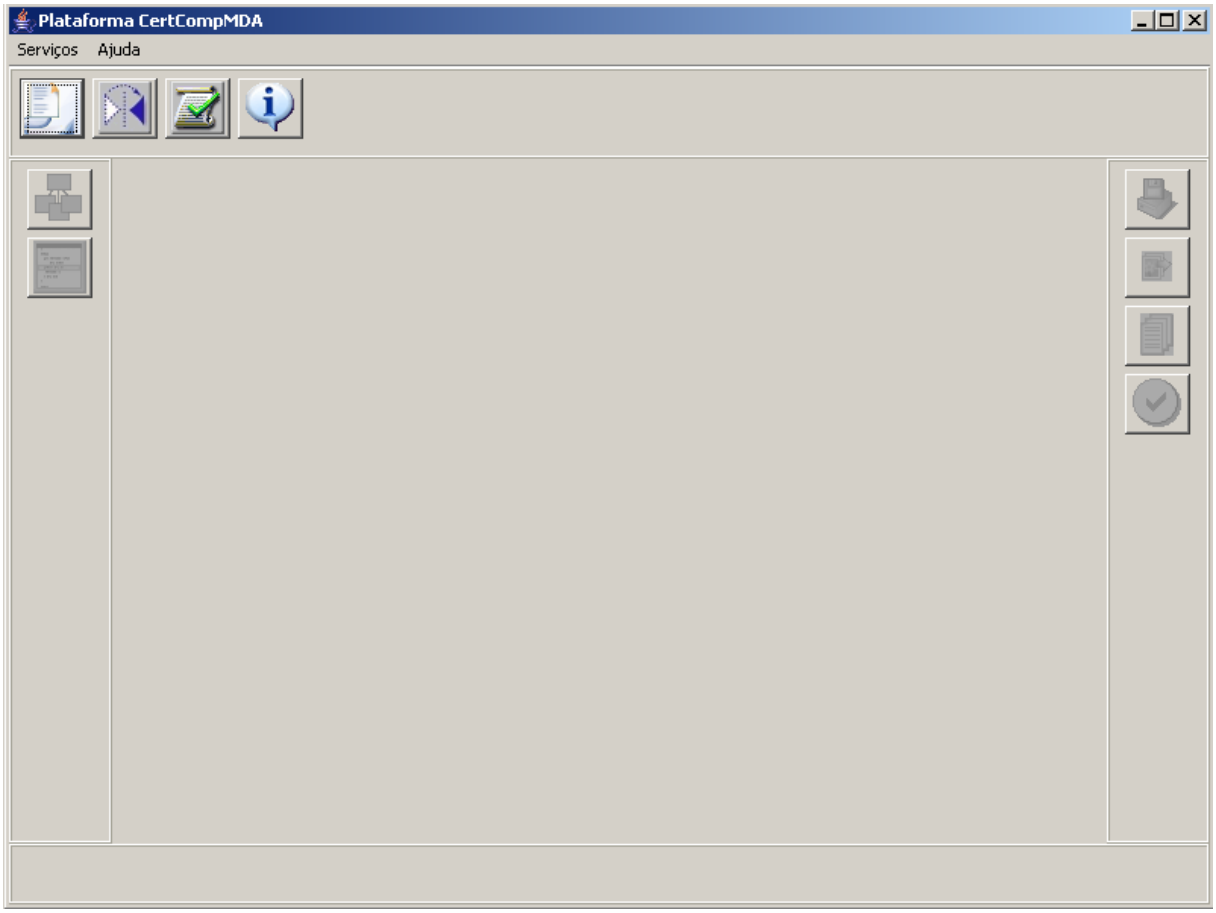


Figura D.4: Barra de ferramentas da aplicação de certificação

Serviço de Administração de Repositório:

Armazenar Modelo: As figuras D.5 e D.6 ilustram a escolha da opção de armazenar modelo do serviço de Administração do Repositório, onde o ícone à esquerda referente a importação de modelos em XMI fica habilitado. Após a importação do modelo, sua estrutura é exibida, com a habilitação do ícone à direita referente a operação de salvar o modelo no repositório. Caso ocorra algum problema nessa operação, como por exemplo, má formação do modelo de representação do padrão com relação ao perfil UML específico, uma mensagem apropriada é exibida informando sua ocorrência, se não é exibida uma mensagem de “Modelo armazenado com Sucesso”.

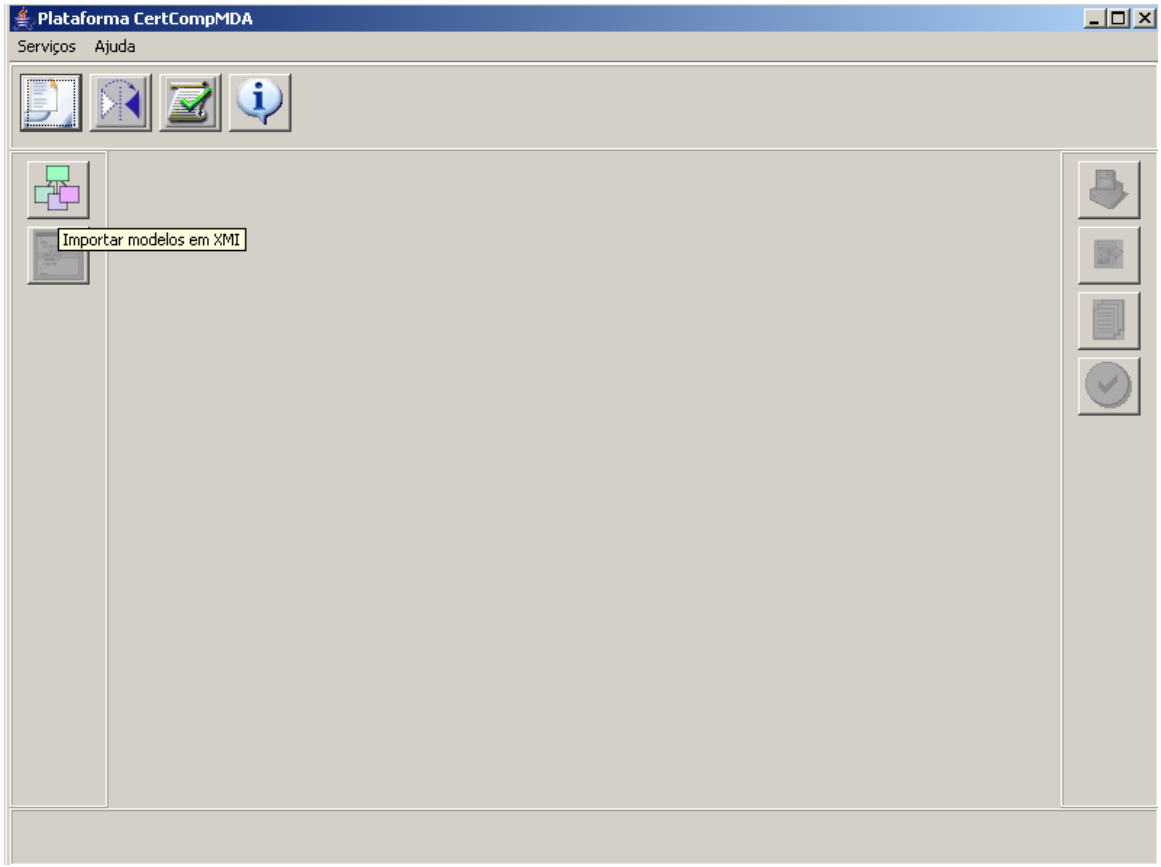


Figura D.5: Importação de modelo em XMI

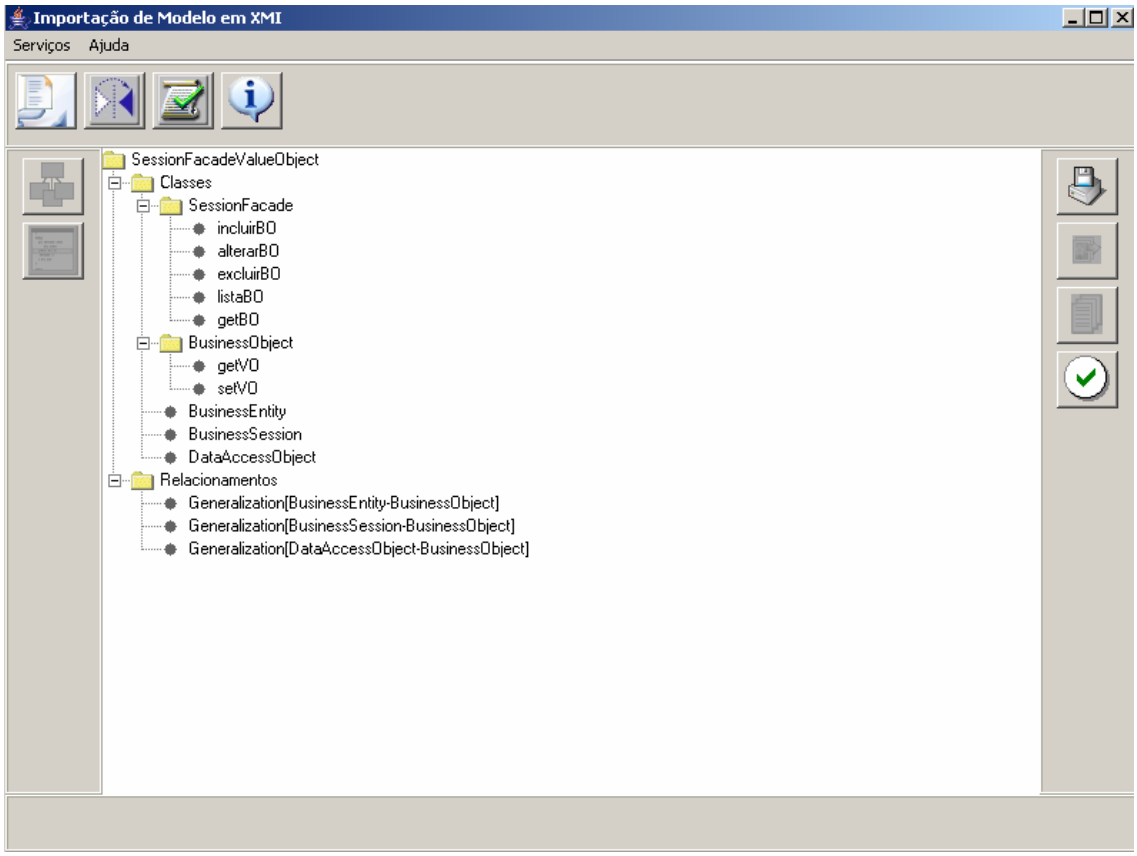


Figura D.6: Estrutura do modelo importado

Consultar o Repositório: A figura D.7 ilustra a escolha da opção de consulta do serviço de Administração do Repositório, com a exibição do conteúdo do repositório do exemplo de aplicação do processo proposto.

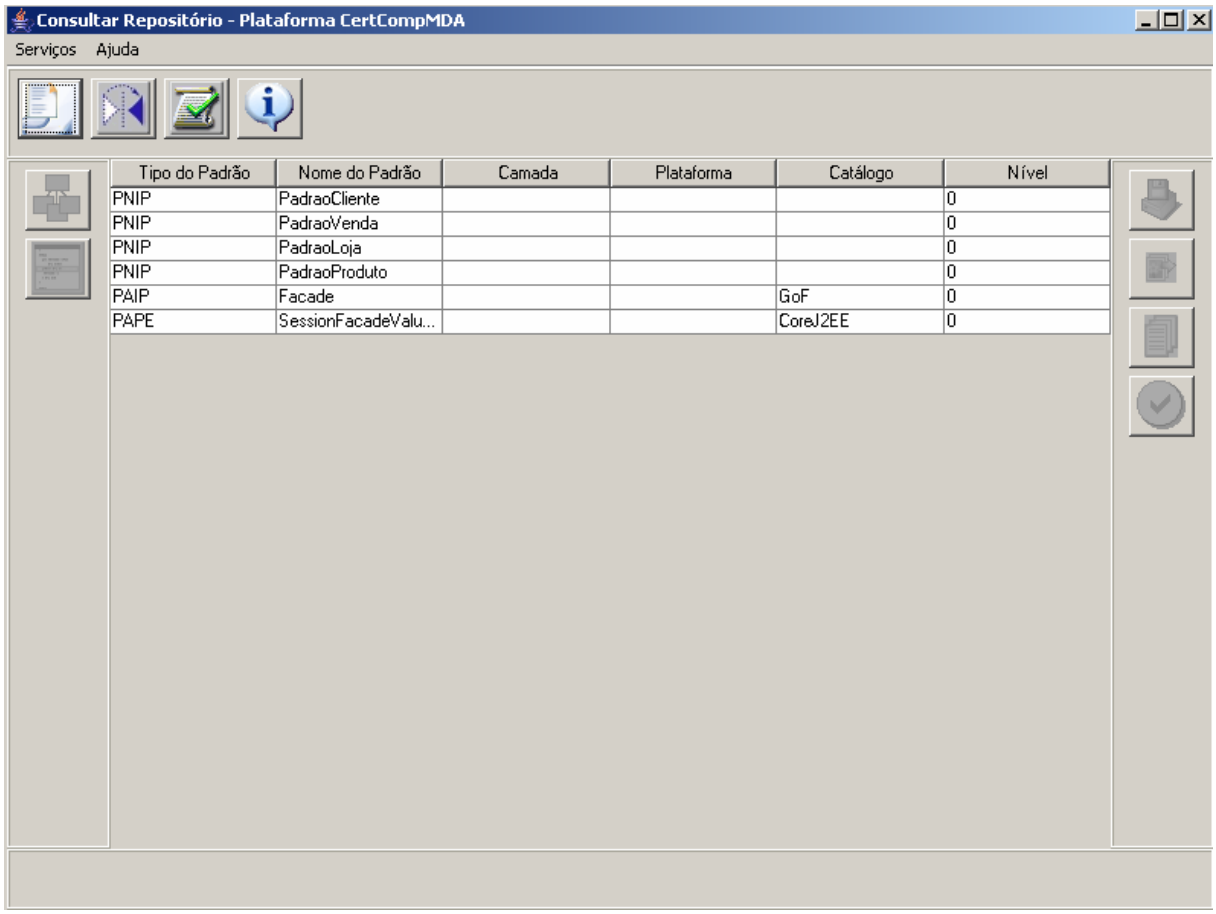


Figura D.7: Conteúdo do repositório

Serviço de Reconstrução de Arquitetura:

As figuras D.8 e D.9 ilustram a escolha do serviço de Reconstrução de Arquitetura, onde o ícone à esquerda referente a importação do código fonte do componente fica habilitado. Após a importação do código fonte, sua estrutura é exibida, com a habilitação do ícone à direita referente a operação de recuperar a arquitetura do componente em UML. Após a recuperação da arquitetura ser executada, sua estrutura é exibida, com a habilitação do ícone à direita referente a operação de exportar modelo em XMI.

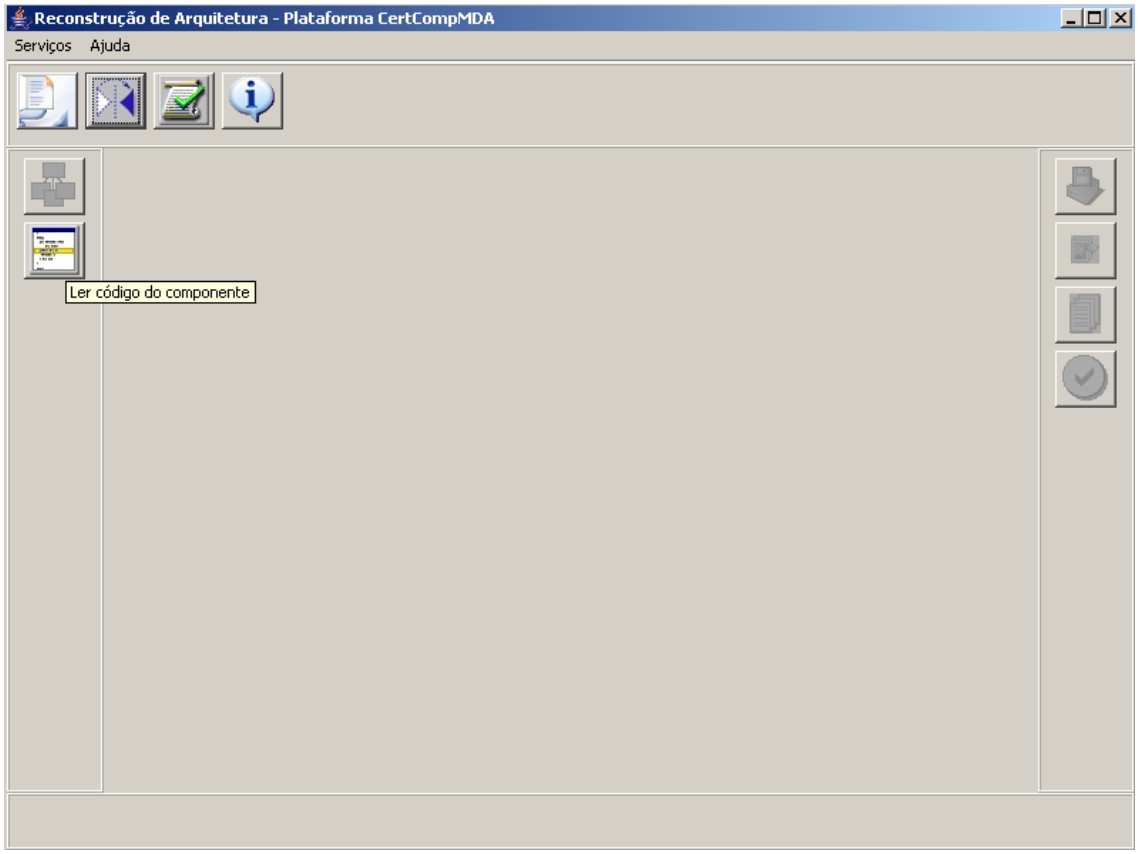


Figura D.8: Importação do código J2EE

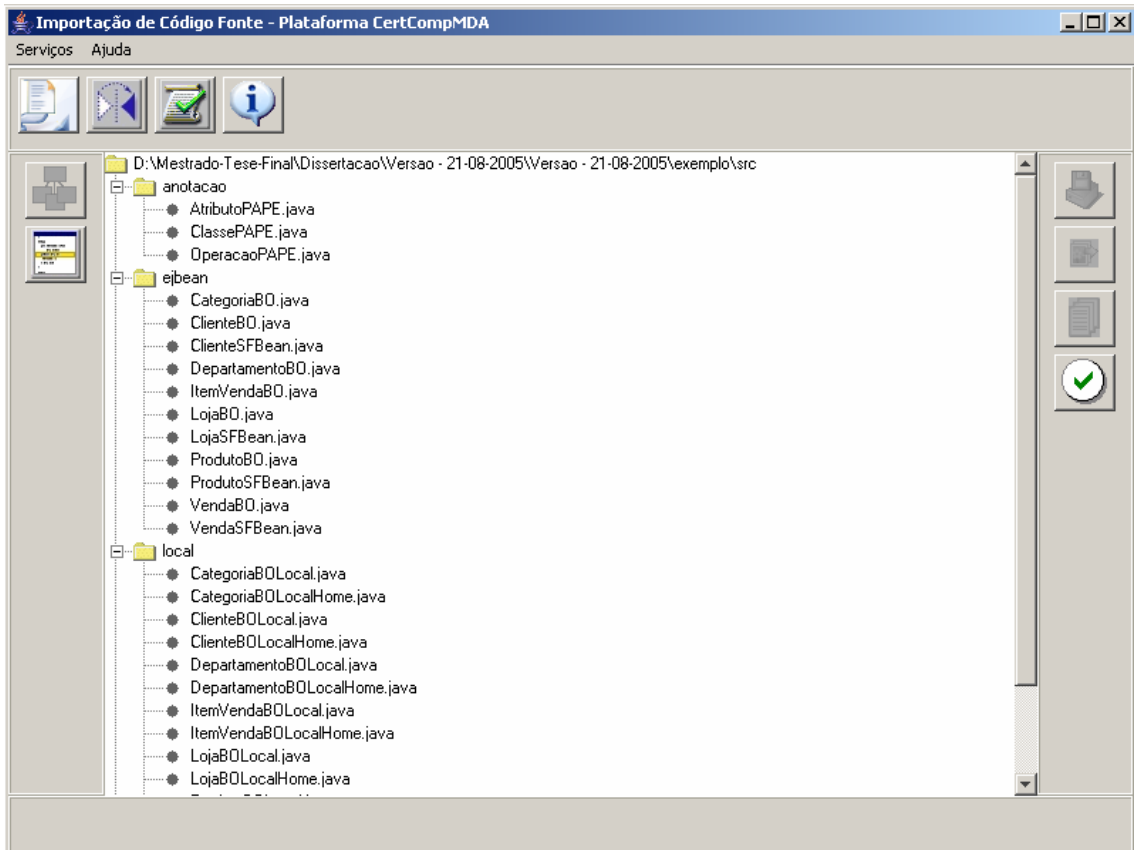


Figura D.9: Código J2EE importado

Serviço de Análise:

A figura D.10 ilustra a escolha da opção de análise de padrões utilizados do serviço de Análise, onde o ícone à esquerda referente a importação do modelo a ser analisado fica habilitado. Após a importação do modelo, sua estrutura é exibida (figura D.11) na mesma forma apresentada na figura D.6, com a habilitação do ícone à direita referente a operação de analisar modelo. Após a execução da análise, o resultado é exibido informando os erros encontrados, as conformidades e os padrões identificados no modelo (figura D.12).

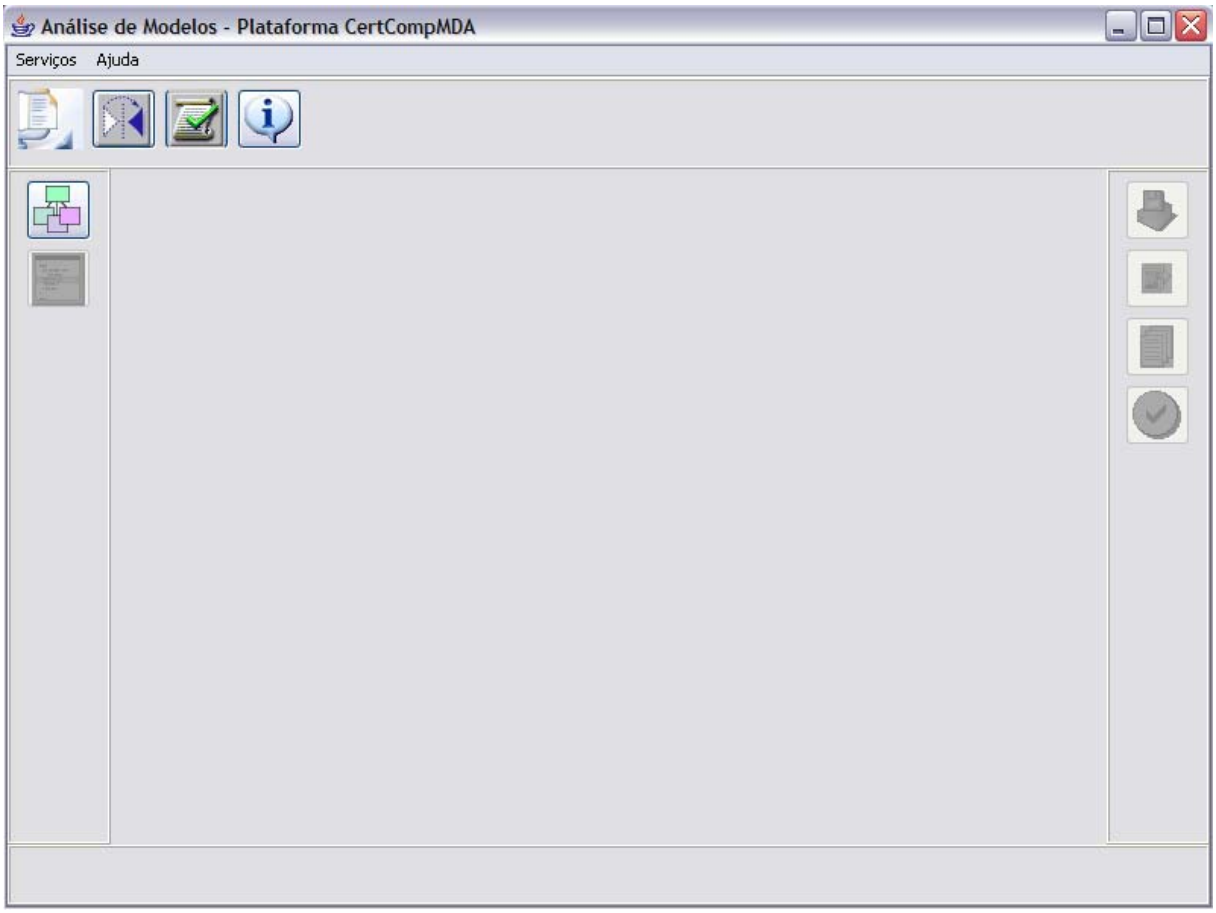


Figura D.10: Escolha da opção Análise de Padrões Utilizados

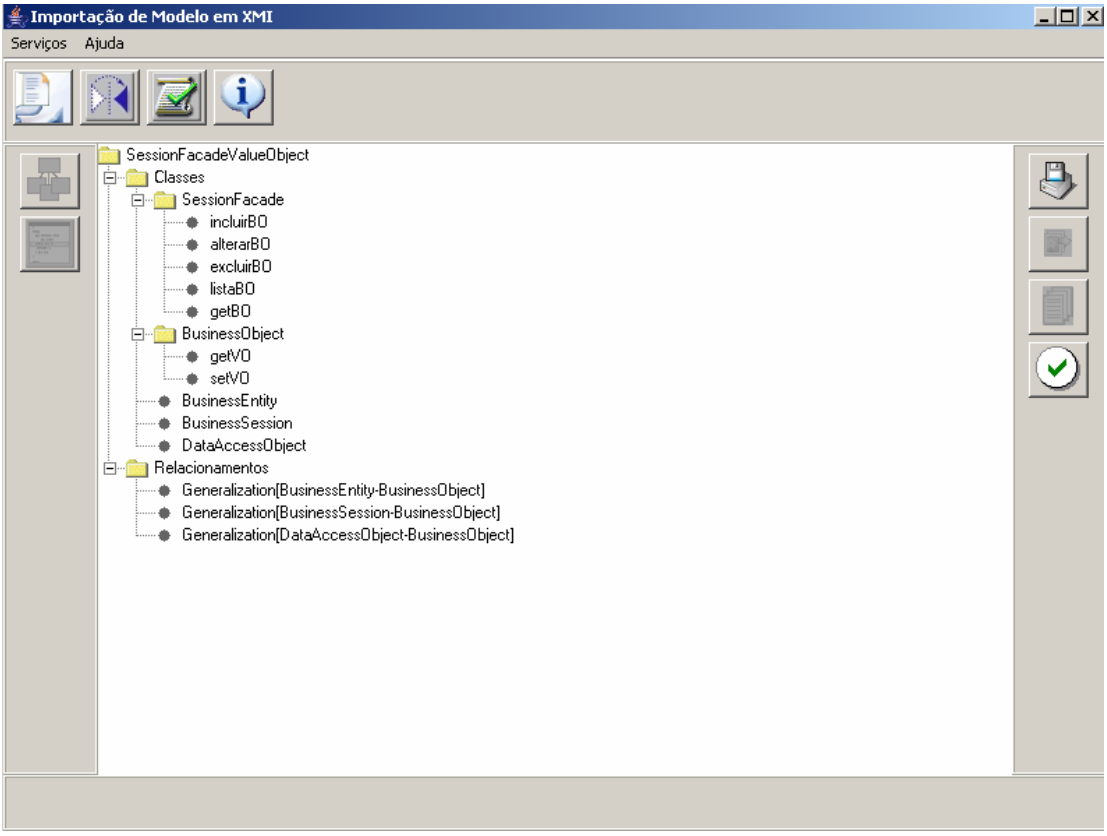


Figura D.11: Estrutura do modelo importado para análise

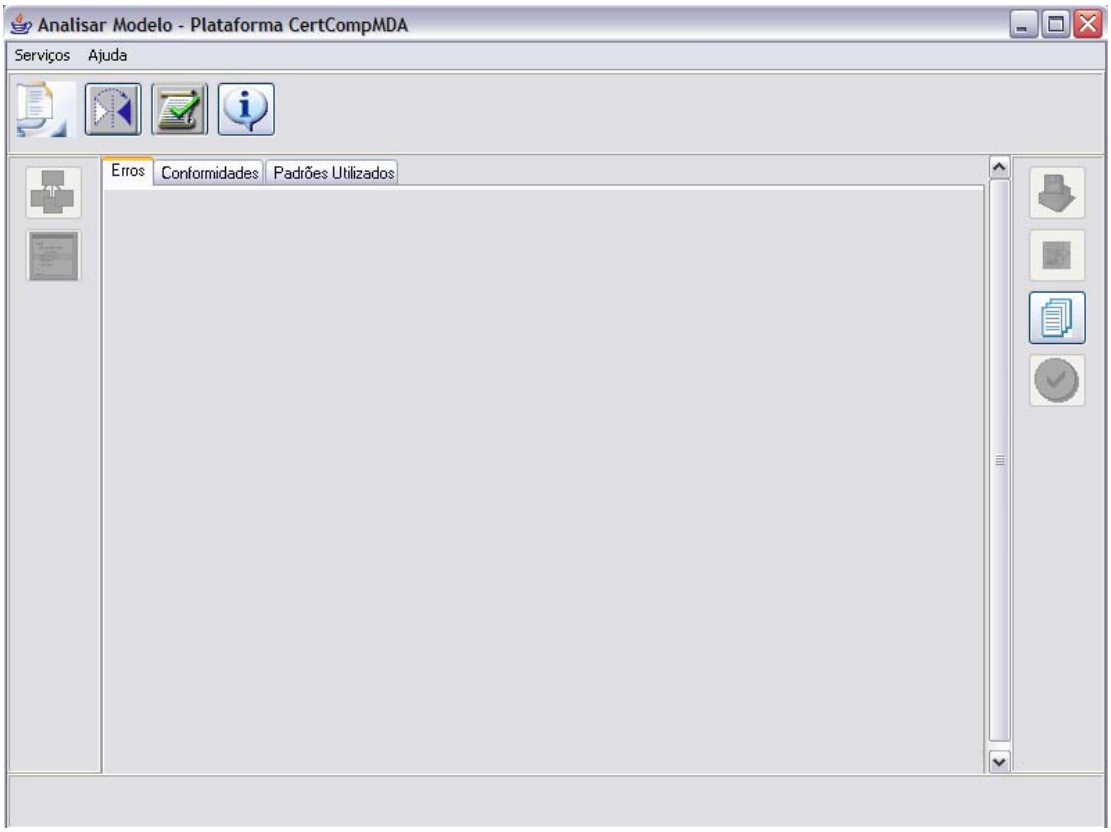


Figura D.12: Resultado da Análise de Padrões Utilizados