3
Tecnologias Relacionadas

O objetivo deste capítulo é apresentar um resumo de cada tecnologia relacionada ao processo proposto nesta dissertação, mostrando suas principais características e de que maneira elas estão sendo utilizadas no processo.

3.1.
Model Driven Architecture (MDA)

3.1.1.
Introdução

O Model Driven Architecture [MDA] começou da idéia de se separar a especificação das operações de um sistema, dos detalhes de como as mesmas deveriam ser implementadas em uma determinada plataforma.

O MDA dá suporte à especificação de sistemas independentes de plataforma e à transformação dos mesmos para uma plataforma específica. Fornecendo, com isso, portabilidade, interoperabilidade e reutilização para os sistemas desenvolvidos segundo este paradigma [MDA].

As principais atividades relativas ao processo definido pela abordagem MDA são as seguintes (figura 16):

- Criação do modelo independente da visão computacional;

- Transformação do modelo independente da visão computacional em um modelo independente de plataforma;

- Transformação do modelo independente de plataforma em um modelo específico para uma plataforma;

- Geração de código.

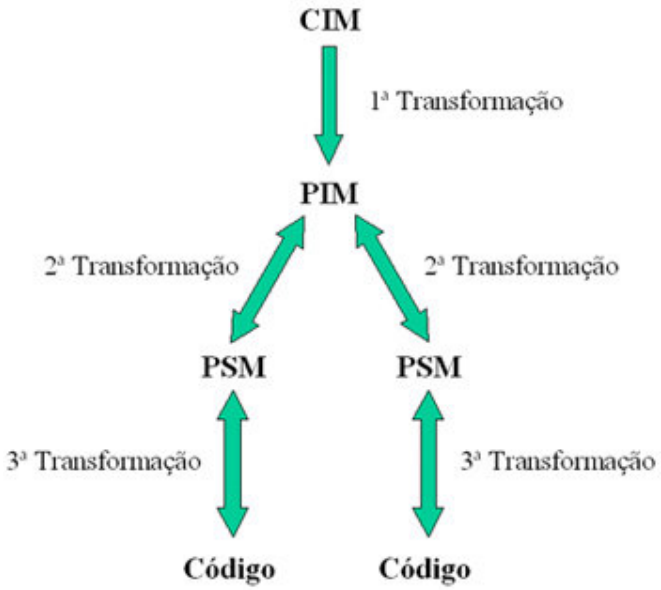

Figura 16 - Model Driven Architecture

3.1.2. CIM (Compution Independent Model)

Este é o primeiro modelo que o MDA define; o modelo de mais alto nível de abstração e independente de qualquer visão computacional. Ele não mostra detalhes da estrutura dos sistemas, tendo um papel importante na definição dos requisitos do sistema.

O processo proposto nesta dissertação não aborda o modelo CIM, tendo como ponto inicial o modelo PIM.

3.1.3. PIM (Plataform Independent Model)

É um modelo com alto nível de abstração e independente de qualquer tecnologia de implementação, mas, diferentemente do CIM, este modelo já possui uma visão computacional. Ele define o sistema de acordo com o negócio, deixando de lado os aspectos relativos à plataforma de implementação.

3.1.4.
PSM (Plataform Specific Model)

O modelo PSM é o resultado da transformação do modelo PIM configurado (com marcações feitas pelo projetista do PIM). Ele é específico para uma plataforma (por exemplo, Java), e pode também ser mais específico ainda, caso possua um subsistema de persistência para a plataforma em questão.

Um PIM pode gerar um ou mais PSMs, de acordo com as plataformas disponíveis. A maioria dos sistemas utilizam, ou no futuro poderão vir a utilizar, mais de uma tecnologia, com isso há a necessidade de se gerar vários modelos PSM, a partir de um único PIM.

3.1.5.
Marcações

As marcações são definidas pelo projetista no modelo PIM, e são elas que informam, na hora de ocorrer as transformações, o que o desenvolvedor deseja que ocorra com o modelo. Elas podem ser de baixo ou alto nível, podendo gerar modelos PSM mais ou menos específicos.

3.1.6.
Transformação do PIM em PSM

A transformação é a etapa mais importante de MDA, pois é nela que um modelo PIM dá origem a um ou mais modelos PSM. O PSM gerado está diretamente ligado às marcações feitas pelo desenvolvedor no modelo PIM, podendo haver ainda alguns detalhes a serem configurados durante a transformação em si.

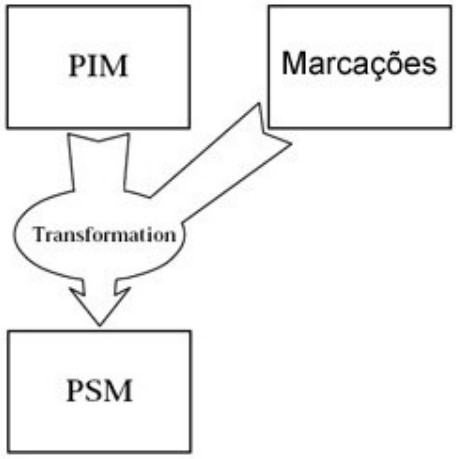

Figura 17 - Transformação do Modelo PIM no Modelo PSM [MDA]

Um exemplo seria a geração de um modelo PIM para a plataforma Java, e descendo mais o nível, podendo gerar subsistemas como de persistência, ou de interface já dentro da plataforma Java.

3.1.7. Geração de Código

A próxima etapa é a de transformação do modelo PSM em código fonte, onde é possível utilizar ferramentas voltadas para MDA que automatizam essa fase. O processo proposto nessa dissertação não aborda a parte de geração de código, sendo o resultado do mesmo um modelo PSM.

3.2. XML Metadata Interchange (XMI)

XMI [XMI] é um formato padrão recomendado pela OMG (Object Management Group), que utiliza a linguagem XML (Extensible Markup Language) [XML] para fazer o intercâmbio de dados, possibilitando o compartilhamento de modelos entre ferramentas de modelagem diferentes. O XMI possibilita a transferência de modelos UML e metamodelos MOF (Meta Objects Facility) [XMI], através do padrão XML DTD (Document Type Definition) [XMI].

O XMI descreve todas as informações de um modelo, incluindo classes, atributos, relacionamentos, heranças e tipos primitivos, entre outros. Ele também descreve toda a parte gráfica do modelo, ou seja, como o mesmo deve ser mostrado em uma ferramenta CASE, envolvendo aspectos como cor, tamanho, altura, largura das classes e tudo o mais relacionado a parte gráfica do modelo.

Hoje os fabricantes de ferramentas de projeto e de banco de dados têm adicionado o XMI aos seus produtos, possibilitando assim a troca de informações com outras ferramentas. O XMI é um sistema aberto a qualquer um que queira usufruir da sua capacidade de troca de informações e com isso evitar formatos proprietários.

A especificação XMI define uma rigorosa abordagem para geração de uma DTD XML a partir de um metamodelo, e para geração de um documento XML como modelo instanciado do metamodelo.

É através do XMI que é feita toda a transformação do modelo PIM no modelo PSM. O PIM é exportado de uma ferramenta CASE como um arquivo XMI, e a ferramenta de apoio utiliza o mesmo para aplicar as transformações. O XMI resultante representa o modelo PSM, que será visualizado após a importação do arquivo por uma ferramenta CASE.

3.3. Reuse Description Language (RDL)

Desenvolvida para instanciação de frameworks, a linguagem RDL [Oliveira05] define formalmente as regras de transformações. Os scripts RDL descrevem as possíveis maneiras de instanciação de um framework.

A estrutura de um script de instanciação RDL é a mesma encontrada na maioria das linguagens de programação, sendo que o termo Cookbook representa um programa e o termo Recipe uma função. Cada Cookbook necessita ter um Recipe chamado main, que indica o ponto de partida da execução do processo de instanciação. A figura 18 mostra um exemplo dessa estrutura.

```
COOBOOK Example
  RECIPE main
    // do something
  END_RECIPE;
END_COOKBOOK
```

Figura 18 - Estrutura de um Script RDL [Oliveira05]

É importante observar que o processo de instanciação é totalmente dependente do domínio da aplicação, e a seqüência de regras do script representa o passo a passo do processo de instanciação.

No processo proposto nesta dissertação os scripts RDL são utilizados para representar cada transformação possível. O desenvolvedor informa a tecnologia que ele deseja e o sistema cuida de selecionar o arquivo RDL correspondente.

Quando os scripts RDL são executados, o processamento das regras contidas no arquivo modificam o arquivo XMI que contém o modelo a ser transformado, normalmente o modelo PIM. Após a execução dos scripts RDL, este arquivo XMI estará atualizado de forma a representar o modelo PSM.

A figura 19 a seguir mostra um exemplo de um diagrama de classes contendo uma única classe e a mesma sem atributos e métodos. Esse modelo será a base para efetuarmos algumas transformações utilizando um script RDL.

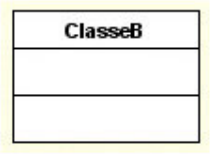

Figura 19 - Exemplo de um Modelo Simples

Abaixo, na figura 20, temos o exemplo de um script RDL que tem a finalidade de adicionar uma nova classe, e adicionar na mesma um método e um atributo. Esse script espera também que o modelo contenha uma classe denominada ClasseB, na qual ele irá adicionar um método e um atributo.

```
COOKBOOK Example
  RECIPE main
    NEW_CLASS (ClasseA);
    NEW_METHOD (ClasseA, metodoA);
    NEW_ATTRIBUTE (ClasseA, atributoA);
    NEW_METHOD (ClasseB, metodoB);
    NEW_ATTRIBUTE (ClasseB, atributoB);
  END_RECIPE;
END_COOKBOOK
```

Figura 20 - Exemplo de um Script RDL

É possível visualizar na figura 21 o novo modelo contendo as transformações definidas no script RDL da figura 20. A classe já existente no mesmo foi transformada através da adição de um método e de um atributo, e uma nova classe também foi adicionada contendo também um método e um atributo.

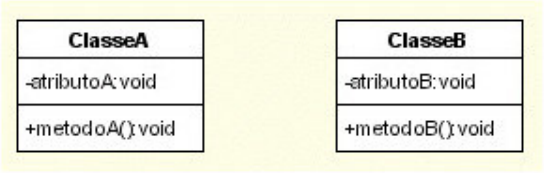

Figura 21 - Modelo Atualizado

No processo proposto nesta dissertação, a linguagem RDL será usada para descrever as regras de transformações das tecnologias envolvidas na persistência de objetos. Serão elas as responsáveis pelas transformações nos modelos a serem adaptados.

3.4. Frameworks de Persistência

Um framework de persistência tem como objetivo implementar componentes que executam a construção e atualização de objetos em algum mecanismo de persistência de forma transparente, diminuindo, dessa maneira, o impacto causado pela troca do mecanismo de persistência.

3.4.1. Castor

O Castor [Castor05] é um framework voltado para a ligação de dados (databinding). Ele foi proposto para ser a ligação entre objetos Java, documentos XML, diretórios LDAP e dados SQL, promovendo mapeamentos entre todas estas estruturas de representação de objetos.

A API do Framework Castor específica para a persistência em bancos de dados relacionais é a JDO, uma implementação inspirada no padrão Java Data Objects da Sun. A API provê também integração com ambientes J2EE.

A figura 22 a seguir mostra que o conceito central da arquitetura do JDO está em implementar um sistema de cache, onde todo o acesso a banco fica transparente para o desenvolvedor. Esse sistema visa também diminuir o acesso ao banco, deixando o trabalho de quando e como acessar o banco sob a responsabilidade do framework.

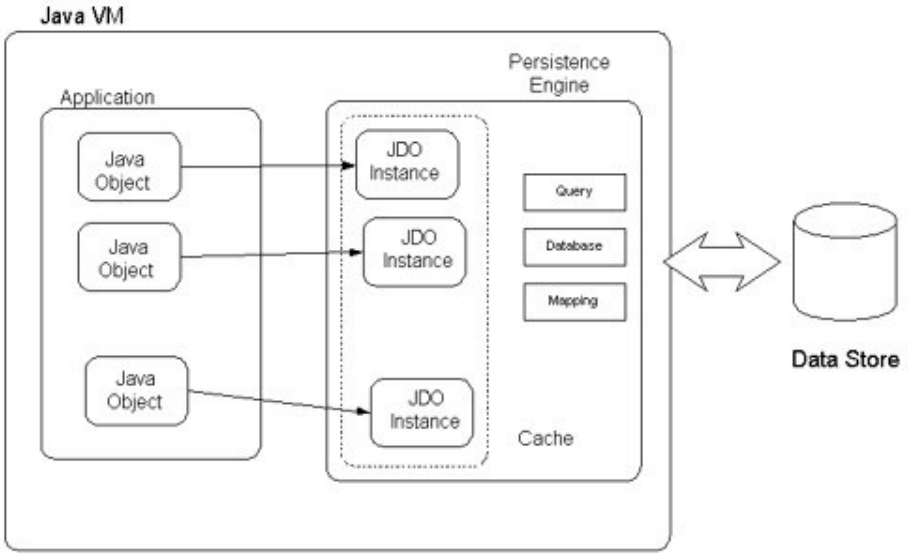

Figura 22 - Arquitetura do Framework Castor JDO [Castor05]

3.4.2. Hibernate

Hibernate [Hibernate] é um mecanismo bem simples e poderoso que permite a persistência de objetos em banco de dados relacionais de maneira transparente e para qualquer tipo de aplicação Java (Web ou baseada em Swing).

Sua grande vantagem está em evitar que comandos SQLs sejam misturados com o código da aplicação. Também não há necessidade de se mapear o resultado de suas consultas para os objetos, pois isso é feito automaticamente.

Para se utilizar os recursos do Hibernate são necessários seguir 5 passos. São eles [Hibernate]:

- Criação das tabelas no banco de dados onde os objetos vão persistir;
- Criação do objeto cujo estado vai ser persistido;
- Criação de um XML que relaciona as propriedades do objeto aos campos da tabela;
- Criação de um arquivo contendo as propriedades para que o Hibernate se conecte ao bando de dados;
- Criação da classe DAO que vai persistir o objeto.

A figura 23 mostra todo o trabalho que o framework faz no lugar do desenvolvedor. Desde a gerência do pool de conexões, passando pela tradução dos arquivos XMLs de mapeamento tabela-classe em cláusulas SQL, o manuseamento de dados em cache evitando o acesso desnecessário ao banco, e, por último, as restrições de integridade existentes no banco.

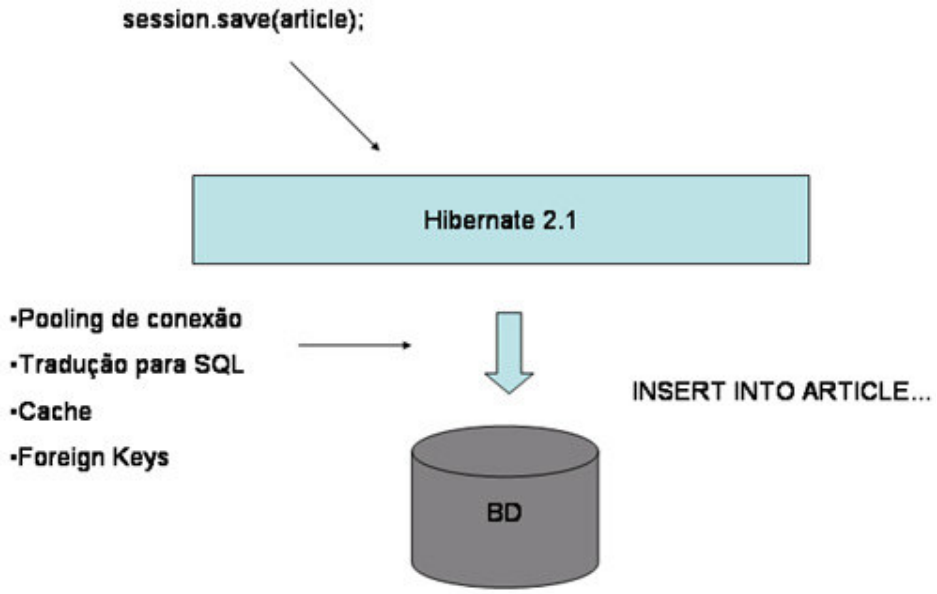

Figura 23 - Exemplo do Funcionamento do Framework Hibernate [Hibernate]

3.4.3. iBatis

O iBATIS Data Mapper framework [iBATIS] foi criado para facilitar o uso de um banco de dados relacional em aplicações Java e .NET. Ele encapsula os objetos através de stored procedures ou cláusulas SQL, utilizando arquivos XML para descrevê-los [iBATIS]. Ele provê parâmetros tanto no formato de objetos, quanto no de tipos primitivos. Esses parâmetros são utilizados em tempo de execução para montar as stored procedures ou cláusulas SQL.

A figura 24, a seguir, dá uma visão geral do funcionamento do framework de persistência iBATIS. Nela é possível ver que o mesmo armazena as cláusulas SQL e as stored procedures em arquivos XML, independentes de plataforma. Logo, se houver necessidade de mudar de plataforma, esses arquivos XML não necessitarão ser modificados.

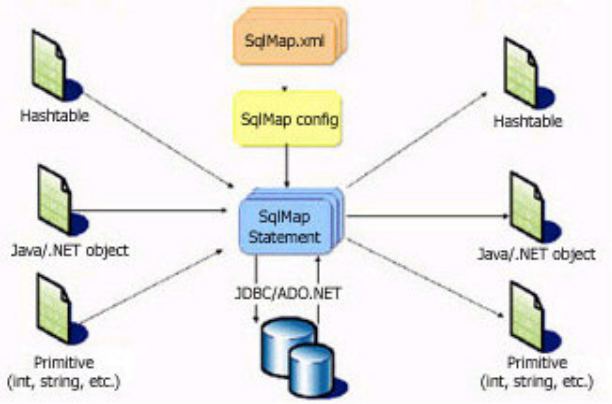

Figura 24 - Visão Geral do Framework iBatis [iBATIS]

A grande vantagem de se utilizar o framework de persistência iBATIS, é que o mesmo trata automaticamente de alguns passos que o desenvolvedor teria que tratar sem ele. São eles:

- Montar a cláusula SQL ou a stored procedure com as informações dos objetos passados como parâmetro;
- Executar as mesmas;
- Montar novos objetos com os dados que retornaram após a execução das mesmas;

No lugar dos passos acima, o desenvolvedor só precisa montar o XML que descreve a cláusula SQL ou a stored procedure, de maneira a receber determinados objetos como parâmetro e indicar o que deverá ser retornado após a execução do mesmo.

3.4.4. OJB

O Object-Relational Java Bridge [OJB05] é um projeto do grupo Apache [Apache] para prover uma implementação open-source dos padrões de mapeamento de objetos ODMG [ODMG] e JDO [JDO]. Ele permite que objetos sejam manipulados sem a necessidade de implementar alguma interface em especial ou estender uma classe específica.

A biblioteca dá suporte a cenários cliente-servidor (aplicações distribuídas) ou standalone, de forma que é possível utilizar a API OJB para persistência de objetos mesmo em ambientes J2EE [J2EE].

O OJB também dá suporte à configuração de esquemas em tempo de execução - geração de tabelas para mapear uma hierarquia de classes ou classes relativas a um conjunto de tabelas -, e implementa uma série de elementos que visam melhorar a performance da Camada de Persistência.

A figura 25 mostra as camadas do Framework OJB. Nela é possível ver a interação dos padrões de mapeamento JDO e ODMG com o PersistenceBroker [PB] (kernel do Framework). A camada Object Transaction Manager [OTM] ainda está sendo desenvolvida e terá como principal objetivo, a implementação de uma API de alto nível, contendo as funcionalidades em comum do JDO e ODMG.

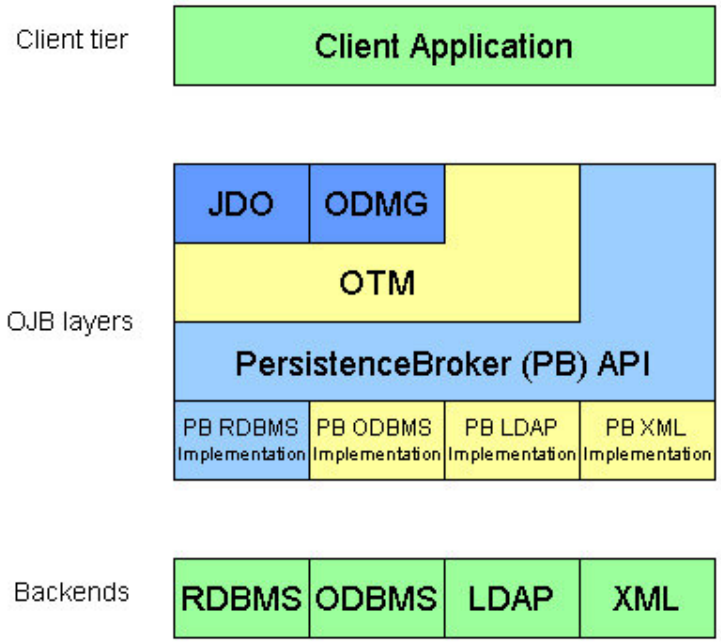

Figura 25 - Camadas do Framework OJB [OJB05]