RAPHAEL BARRETO PALHARES DE CAMPOS

ANÁLISE COMPARATIVA DE FRAMEWORKS DE PERSISTÊNCIA

LAVRAS – MG

2010

RAPHAEL BARRETO PALHARES DE CAMPOS

ANÁLISE COMPARATIVA DE FRAMEWORKS DE PERSISTÊNCIA

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação

Orientador

Dr. André Vital Saúde

LAVRAS - MG
2010

RAPHAEL BARRETO PALHARES DE CAMPOS

ANÁLISE COMPARATIVA DE FRAMEWORKS DE PERSISTÊNCIA

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação

APROVADA em ____ de _____ de _____

Dra. Ana Paula Piovesan Melchiori – DCC/UFLA

Dr. Antônio Maria Pereira de Resende – DCC/UFLA

_____

Dr. André Vital Saúde

Orientador

LAVRAS - MG
2010

Dedico

A meus pais João Guilherme e Alice.

A minha irmã Marina.

A minhas tias Denise e Cida.

A minhas avós Nita e Aparecida.

AGRADECIMENTOS

A meu pai e minha mãe pelos exemplos de luta, dedicação e honestidade.

Às minhas tias Denise e Cida, pelo apoio incondicional prestado durante toda a minha vida e por serem também minhas mães.

À minha irmã Marina, pela amizade, e também pelas conversas de msn na madrugada! =)

Ao meu orientador e amigo André Saúde, pela orientação e oportunidade de desenvolver este trabalho.

Ao pessoal da Mitah Technologies pelo companheirismo e amizade, em especial ao Ricardo Victório, pela ajuda prestada durante a implementação.

Ao pessoal da Devex Tecnologia e Sistemas, por todos os anos de aprendizado e muito trabalho, e que muito contribuíram para meu crescimento profissional e pessoal.

Às instituições FAPEMIG e CNPq pela oportunidade desenvolver trabalhos de iniciação científica durante boa parte da minha graduação.

Aos meus amigos: o pessoal do metal de Ouro Branco, a turma de 2005/2 da Computação, o povo da minha banda (“Os Internautas”) e ao povo da República Zona51.

Às grandes personalidades que sempre me inspiraram com suas idéias: James Hetfield, Steve Harris, Bruce Dickinson, JRR Tolkien, Douglas Noel Adams, Charles Darwin, Stephen Hawking, Richard Dawkins, dentre vários outros que poderia continuar listando durante um bom tempo.

RESUMO

Em razão da impedância existente entre o modelo Orientado a Objetos, utilizado no desenvolvimento de software, e o modelo Relacional dos Bancos de Dados, surgiram técnicas de mapeamento objeto-relacional (ORM) que permitem uma melhor comunicação entre os dois modelos. Contudo, existem várias especificações e *frameworks* de mapeamento objeto-relacional que implementam as técnicas de ORM, por isso se faz necessário decidir qual implementação utilizar ao desenvolver um software. O presente trabalho tem como objetivo analisar e comparar algumas das soluções existentes para a plataforma Java. Foram escolhidas as especificações: Java Data Objects (JDO) e Java Persistence API (JPA), e os *frameworks* Data Nucleus e Hibernate. As comparações foram feitas com base nas informações coletadas durante a confecção do referencial teórico e segundo critérios gerais para análise de *frameworks* e específicos de mapeamento objeto-relacional existentes na literatura sobre o assunto. Ao final uma combinação de especificação e *framework* foi escolhida para ser utilizada no *framework* Iguassu, desenvolvido pela empresa Mitah Technologies em parceria com a Universidade Federal de Lavras.

Palavras-chave: Frameworks de persistência, Mapeamento Objeto Relacional, Análise Comparativa, ORM, JDO, JPA, Hibernate, Data Nucleus

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API – Application Programming Interface

HQL – Hibernate Query Language

JDO – Java Data Objects

JDOQL – Java Data Objects Query Language

JPA - Java Persistence API

JPQL – Java Persistence Query Language

ORM – Object Relational Mapping (Mapeamento Objeto Relacional)

SOA – Service-Oriented Architecture

SQL – Structured Query Language

TI – Tecnologia da Informação

UML - Unified Modeling Language

XMI – XML Metadata Interchange

XML – eXtensible Markup Language

SUMÁRIO

1. INTRODUÇÃO

A indústria de software é uma área que se encontra em crescimento acelerado, devido principalmente às demandas crescentes do mercado por soluções inovadoras.

A informação é algo de extremo valor para a corporação moderna, os bancos, por exemplo, armazenam os dados financeiros de milhões de pessoas e empresas. A manutenção e o gerenciamento das informações contidas em seus bancos de dados são um dos pilares da área de Sistemas de Informação.

As corporações armazenam grandes volumes de informações em Bancos de Dados Relacionais, e estes, surgidos por volta da década de 70, são baseados em fundamentos matemáticos que garantem uma ótima estruturação e confiabilidade aos dados armazenados. Em contrapartida, na década de 90, a indústria de software viu crescer o paradigma de orientação a objetos, que permitia uma maior abstração aos desenvolvedores de software, e por conta disso diminuía o tempo necessário para a construção de sistemas, que por sua vez eram mais robustos e rentáveis.

Até o final da década de 90, a maioria absoluta de empresas dentro da indústria de software já utilizava do paradigma de orientação a objetos, porém estas empresas desenvolviam software para corporações que, como dito anteriormente, utilizava-se de bancos de dados relacionais para armazenar os dados.

O Modelo Orientado a Objetos (OO) é muito diferente do Modelo Entidade/Relacionamento (ER) utilizado pelos bancos de dados relacionais. Enquanto o primeiro é baseado em técnicas de Engenharia de Software, o outro é fundamentado em princípios matemáticos de relacionamentos entre tabelas. A esta diferença entre os modelos se dá o nome de Impedância Objeto-Relacional (AMBLER, 2003a)

Por causa das diferenças entre os modelos, os desenvolvedores tinham um trabalho maior na hora de extrair dados do modelo ER para o modelo OO, e na hora de armazenar os objetos nas tabelas do modelo ER, sendo necessárias várias linhas de código a mais para tal procedimento, que muitas vezes seriam replicadas em todo o sistema. Essas linhas de código a mais diminuem consideravelmente a manutenibilidade e capacidade de extensão dos sistemas de software (BAUER; KING, 2007).

Para resolver o problema da impedância, surgiram técnicas de mapeamento objeto-relacional (Object Relational Mapping ou ORM) que permitem a tradução dos dados de um modelo para o outro de maneira transparente para o desenvolvedor.

Na plataforma de desenvolvimento Java, o mapeamento objeto-relacional foi padronizado por meio de especificações de persistência, que estabelecem maneiras de realizar o mapeamento em Java. As especificações fornecem as diretrizes para realizar os mapeamentos e estas são disponibilizadas na própria API da especificação ou implementadas em frameworks de persistência.

Com o crescente aumento na utilização de técnicas de mapeamento objeto-relacional, surgiram várias especificações e frameworks para a plataforma Java, por isso determinar qual delas utilizar é uma decisão crucial para empresas de Tecnologia da Informação.

1.1. Objetivos

Este trabalho analisa e compara as soluções para mapeamento objeto-realacional disponibilizadas através dos *frameworks* DataNucleus com Java Data Object e Hibernate com Java Persistence API. As comparações foram feitas tendo por base os dados coletados na confecção do referencial teórico e seguindo

os critérios que são definidos no capítulo de Metodologia. Após a realização das comparações é escolhida uma especificação/*framework* para ser utilizado no *framework* Iguassu.

1.2. Estrutura do Trabalho

A organização do texto é como se segue:

No Capítulo 2 há uma revisão dos conceitos fundamentais de Orientação a Objetos, Banco de Dados, das tecnologias utilizadas e de Mapeamento Objeto-Relacional.

No Capítulo 3 são apresentadas as especificações padrão para persistência e os *frameworks* de persistência Data Nucleus e Hibernate.

O Capítulo 4 discorre a respeito da metodologia.

O Capítulo 5 apresenta os resultados obtidos.

No Capítulo 6 encontra-se a conclusão.

2. REFERENCIAL TEÓRICO

As especificações e *frameworks* de persistência são tecnologias desenvolvidas com base nos conceitos do mapeamento objeto-relacionamento, que por sua vez é um conceito derivado de orientação a objetos e do modelo de entidade-relacional.

Neste capítulo serão abordados os conceitos básicos de Orientação a Objetos, Banco de Dados Relacionais, Mapeamento Objeto-Relacional e das principais tecnologias utilizadas pelas especificações e *frameworks* de persistência que serão apresentados nos próximos capítulos.

2.1. Orientação a Objetos

A orientação a objetos é um paradigma de desenvolvimento de sistemas que abstrai a realidade a ser automatizada em termos de objetos que incorporam tanto informações como procedimentos. (PINHEIRO, 2005).

Segue alguns dos conceitos de orientação a objetos, retirados de Pothu (2008):

- Classe: tipo de organização de dados que define conjuntos de objetos com as mesmas características e comportamentos.
- Objeto: é a instância de uma classe. Um objeto é capaz de armazenar estados através de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos.
- Atributos: estrutura de dados que compõem uma classe. Os valores dos atributos são as características de um determinado objeto.
- Métodos: são as habilidades dos objetos. Em uma classe, um método é apenas uma definição, a execução de um método só ocorre quando ele é invocado a partir de um objeto.

- Mensagem: é uma chamada a um objeto para invocar um de seus métodos, ativando um comportamento descrito por sua classe. Também pode ser direcionada diretamente a uma classe (através de uma invocação a um método estático).
- Herança (ou generalização): mecanismo pelo qual uma classe (sub-classe) pode estender outra classe (super-classe), aproveitando seus comportamentos (métodos) e variáveis possíveis (atributos).
- Associação é o mecanismo pelo qual um objeto utiliza os recursos de outro. Pode tratar-se de uma associação simples "usa um" ou de um acoplamento "parte de".
- Encapsulamento consiste na separação de aspectos internos e externos de um objeto. Este mecanismo é utilizado amplamente para impedir o acesso direto ao estado de um objeto (seus atributos), disponibilizando externamente apenas os métodos que alteram estes estados.
- Abstração é a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais. Em modelagem orientada a objetos, uma classe é uma abstração de entidades existentes no domínio do sistema de software.
- Polimorfismo: mecanismo que permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam..
- Interface é um contrato entre a classe e o mundo externo. Quando uma classe implementa uma interface, ela está comprometida a fornecer o comportamento publicado pela interface.
- Pacotes são referências para organização lógica de classes e interfaces.

2.1.1. Diagramas de Classes

Um diagrama de classes é uma representação gráfica de um modelo orientado a objetos que exibe classes (com atributos e métodos) e os relacionamentos elas.

Basicamente as classes podem se relacionar através de: associação (agregação, composição), dependência ou herança. Macoratti (2009) define os seguintes conceitos sobre os relacionamentos:

- Associação: São relacionamentos que especificam que objetos de uma classe estão ligados a objetos de outras classes.
- Dependência: São relacionamentos de utilização no qual uma mudança na especificação de um elemento pode alterar a especificação do elemento dependente. A dependência entre classes indica que os objetos de uma classe usam serviços dos objetos de outra classe.
- Herança: Relacionamento entre um elemento mais geral e um mais específico. Onde o elemento mais específico herda as propriedades e métodos do elemento mais geral. A relação de generalização também é conhecida como herança no modelo a objetos. Como a relação de dependência, ela existe só entre as classes. Um objeto particular não é um caso geral de um outro objeto, só conceitos (classes no modelo a objetos) são generalização de outros conceitos.
- Agregação Regular: tipo de associação (é parte de , todo/parte) onde o objeto parte é um atributo do todo ; onde os objetos partes somente são criados se o todo ao qual estão agregados seja criado. Pedidos é composto por itens de pedidos.

- Composição: Relacionamento entre um elemento (o todo) e outros elementos (as partes) onde as parte só podem pertencer ao todo e são criadas e destruídas com ele.

A figura 1 mostra como são representadas as associações em diagramas UML.

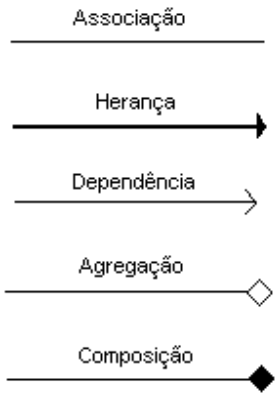

Figura 1 Tipos de relacionamento entre classes no UML

Fonte: MACORATTI, 2009

2.2. Banco de Dados

Um banco de dados é um conjunto de informações organizadas de maneira que permita facilidade de acesso e gerenciamento dos dados (POTHU, 2008).

Os bancos de dados possuem um Sistema de Gerenciamento de Banco de Dados (SGBD) que tem como principal função retirar do aplicativo cliente a

responsabilidade pelo controle de acesso, manipulação e organização de dados. Os fornecedores disponibilizam, através do SGBD, as interfaces necessárias para que os clientes possam acessar a base de dados. No caso dos bancos de dados relacionais a interface são as *APIs* ou *drivers*. Então o SGBD é formado do seguinte conjunto de funcionalidades: (POTHU, 2008)

- Uma linguagem para modelar o esquema das bases de dados.
- Estruturas de dados otimizadas para poder lidar com grandes volumes de dados armazenados.
- Uma linguagem que permita consultas e manipulação dos dados por parte do usuário (no caso SQL).
- Um mecanismo de transações que garanta as propriedades ACID dos bancos de dados (atomicidade, consistência, integridade e durabilidade).

O modelo de dados para se projetar os bancos de dados é o Modelo Entidade/Relacionamento (ER). De acordo com Silberschatz *et al* (2010), o modelo entidade/relacionamento é uma forma de representação do mundo real por meio de entidades e relacionamentos entre elas, propondo uma visualização da realidade por meio de três pontos de vista: Entidade, Atributo e Relacionamento.

As entidades, segundo Pinheiro (2005), são descritas por meio de atributos. Desta forma, as instâncias de uma entidade são identificadas por seus atributos e valores.

Ainda de acordo com Pinheiro (2005), os relacionamentos podem ser definidos como um fato ou acontecimento que liga duas entidades existentes no mundo real. Os relacionamentos possuem como características obrigatoriedade (se é obrigatório ou não) e cardinalidade (um para um, um para muitos e muitos para muitos).

Para implementação do modelo ER no modelo Relacional que é utilizado nos banco de dados, representamos as entidades como tabelas, e os atributos da entidade são as colunas da tabela. As instâncias são os registros presentes na tabela da entidade, cada linha é um registro e deve ser identificada de forma única através de chave primária ou composta. Os relacionamentos podem ser mapeados com o uso de chaves estrangeiras, que referenciam um registro presente em outra tabela da base de dados.

Os bancos de dados devem possuir dados confiáveis, por isso as operações realizadas neles estão em contextos transacionais. Uma transação é uma coleção de operações que são realizadas sobre um conjunto de objetos (PINHEIRO, 2005), a utilização de transações permite que se garanta a consistência dos dados após a realizar operações de inserção, atualização e remoção, de maneira que, se alguma operação dentre as operações definidas para ser executado num escopo transacional falhar, todas as operações realizadas até o momento serão desfeitas (*rollback*). Uma transação só é confirmada (*commit*) se todas as operações dentro de seu escopo forem executadas com sucesso.

2.3. Impedância Objeto-Relacional

Segundo Ambler (2003a), a impedância objeto-relacional existe devido ao fato de que o modelo ER é bem definido matematicamente com dados normalizados em tabelas enquanto o modelo OO é definido com classes, herança e polimorfismo. A Figura 2 mostra a disparidade entre os dois modelos.

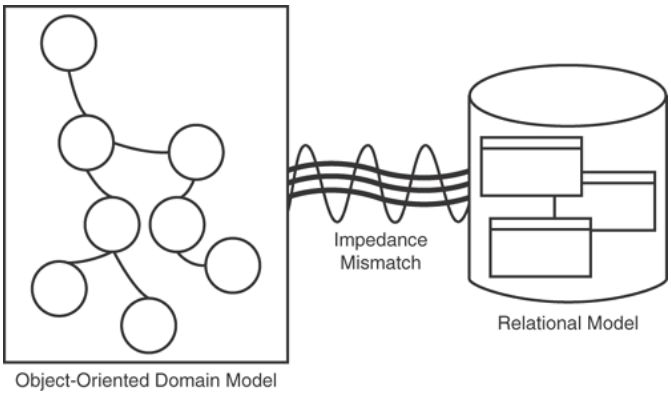

Figura 2 Impedância Objeto Relacional

Fonte: Barcia et al, 2008

Para se obter êxito ao usar objetos e bancos relacionais, é necessário conhecer profundamente ambos os paradigmas, e suas diferenças para tomar decisões baseadas nesses conhecimentos (BAUER; KING, 2007).

2.4. Mapeamento Objeto Relacional

O mapeamento objeto-relacional (ORM) é a base teórica sobre a qual podemos mapear objetos em tabelas de bancos de dados relacionais, visando reduzir a impedância existente entre os dois paradigmas durante o desenvolvimento de software. Quando se mapeia os dados do modelo orientado a objetos para o modelo relacional cria-se o efeito de que o banco de dados seja orientado a objetos, de modo que o programador não precisa se preocupar com a disposição dos dados nas tabelas e se focar na manipulação de objetos e nos problemas de negócio (BAUER; KING; 2007).

Desta forma, uma implementação ORM atua na transformação (reversível) de dados de uma representação para outra de forma transparente

para o usuário, fornecendo uma camada de ligação que deve atender às seguintes proposições (BAUER; KING; 2007):

- Permitir operações CRUD (*create*, *retrieve*, *update*, *delete*) básicas em objetos de classes persistentes.
- Uma linguagem para especificar consultas (*query language*) que se referem às classes ou às propriedades das classes.
- Formas de especificação de mapeamento de classes (metadados)
- Checagem de sujeira (ou *dirty checking*) para determinar se um objeto foi modificado.
- Associação de recuperação preguiçosa (*lazy association fetching*) como forma de otimização para as consultas realizadas.

Um framework ORM deve então, encapsular a interação entre o aplicativo e o banco de dados relacional, deixando o desenvolvedor livre para se concentrar nos problemas de negócio em questão (POTHU, 2008).

A Figura 3 mostra uma arquitetura genérica para um framework de ORM, é possível enxergar a ligação estabelecida pelo framework entre a Aplicação e o Banco de Dados. Desta forma o *framework* de mapeamento objeto relacional atua na tradução dos objetos persistentes da aplicação para a linguagem do Banco de Dados.

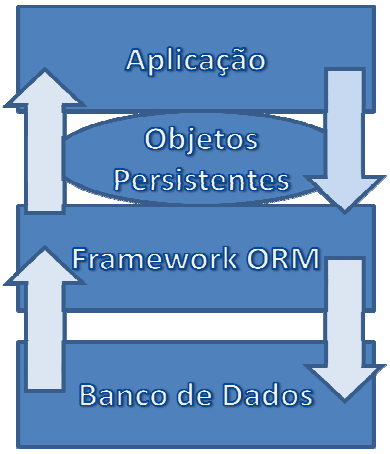

Figura 3 Arquitetura ORM genérica

2.4.1. Sobre os Objetos no Contexto de Persistência

Segundo Bauer e King (2007), o ato persistir objetos consiste em permitir que eles sobrevivam ao processo que os criou, armazenando-os no banco de dados (meio físico) para que depois possam ser restaurados no estado em que estavam para uso em algum ponto no futuro.

É necessário o estabelecimento de um ciclo de vida para os objetos de uma aplicação tendo em vista o contexto de persistência em que estarão inseridos. Basicamente os objetos poderão estar nos seguintes estados:

- Transiente: objeto ainda não presente no banco de dados
- Persistente: objeto que esteja no banco de dados e que possua algum tipo de correspondência (chave) estabelecida entre os dois modelos.

2.4.2. Mapeamento de classes

Os principais tipos de mapeamentos de classes utilizados neste trabalho são: mapeamento de atributos primitivos, mapeamento de atributos complexos e mapeamento de herança. O mapeamento das classes nos *frameworks* ORM é especificado através de metadados que em geral são escritos utilizando tecnologias como o XML ou *Annotations*.

De acordo com Pinheiro (2005), em geral, a relação entre uma classe e uma tabela bem como propriedades e colunas é feita de forma direta. No caso dos relacionamentos entre objetos podem ser representados no banco de dados por meio de chaves estrangeiras e restrições de acordo com a cardinalidade dos relacionamentos. Para as hierarquias de classe, deve-se selecionar uma estratégia de acordo com o modelo orientado a objetos desejado.

A representação de hierarquias de classes não é suportada pelos bancos de dados relacionais, para se contornar este problema se torna necessário mapear a hierarquia do modelo orientado a objetos para o modelo relacional. (PINHEIRO, 2005)

Segundo Ambler (2003b) o mapeamento de herança no modelo relacional pode ser feito das seguintes maneiras:

- Tabela por hierarquia de classes (*One Table Per Class Hierarchy*): toda a hierarquia é mapeada em uma única tabela quem contém os atributos de todas as classes. A sub-classe em questão pode ser diferenciada por meio de uma coluna extra que a identifique.
- Tabela por classe (*Table Per Class*): cada classe possuirá uma tabela própria apenas seus atributos específicos, os atributos herdados são obtidos através de uma chave estrangeira para a tabela de sua super-classe.

- Tabela por classe concreta (*Table Per Concrete Class*): cada classe concreta possuirá uma tabela correspondente contendo todos os atributos próprios e herdados.

Com relação ao mapeamento de relacionamentos entre os objetos, de acordo com Pinheiro (2005), uma associação deve possuir as seguintes informações:

- Relacionamento do Objeto
- Classe Origem
- Classe Destino
- Cardinalidade (1-1, 1-n, n-n)
- Regra de Leitura, Gravação e Inclusão (varia de acordo com o tipo do relacionamento: agregação, composição, associação)
- Tabela que implementa o relacionamento no ambiente de banco de dados
- Coluna que implementa o relacionamento no ambiente de banco de dados
- Atributo da classe origem que implementa o relacionamento

2.5. Outras tecnologias utilizadas

Nesta seção são explicadas as tecnologias Java e BlueBox.

2.5.1. A Plataforma Java

Dentre os vários recursos oferecidos pela plataforma Java, podemos destacar alguns recursos fundamentais para a persistência de dados e que serão

utilizados neste trabalho: as *annotations* e a interface para conexão a banco de dados, o *Java Database Connectivity* (JDBC).

As *annotations* são uma maneira alternativa de se especificar metadados em Java e são representadas pelo símbolo "**@**". O *annotation* é uma informação que pode ser adicionada à declaração de métodos, atributos, que são interpretadas em tempo de execução por bibliotecas e *frameworks* para os quais foram projetadas, não alterando diretamente a semântica da linguagem.

O JDBC é uma interface para programação de aplicativos (API) disponibilizada pela plataforma Java que possibilita a utilização de bancos de dados relacionais nos softwares. Segundo a documentação do JDBC, a API oferece três serviços principais: estabelecimento de conexão com banco de dados, envio de sentenças na linguagem SQL, processamento de resultados e retorno de dados.

2.5.2. BlueBox

Segundo Castro (2010), o BlueBox é uma ferramenta para geração automática de código que utiliza um arquivo XMI como entrada de dados para obter informações de um modelo UML. O código é gerado por intermédio da *engine Velocity* que processa os dados conforme as regras contidas nos *templates Velocity*, que informam como devem ser tratadas as informações do Diagrama de Classes UML. A Figura 4 mostra a arquitetura do BlueBox.

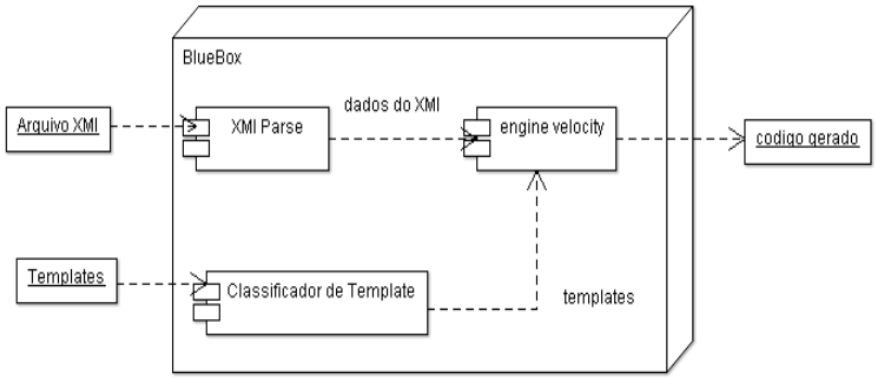

Figura 4 Arquitetura do BlueBox

Fonte CASTRO, 2010

3. ESPECIFICAÇÕES E FRAMEWORKS

Nas próximas seções serão apresentados as especificações e frameworks de persistência comparados neste trabalho.

3.1. Especificações de Persistência

As especificações de persistência são as diretivas aprovadas pela Java Community Process (JCP) e são utilizadas pelos *frameworks* para implementar ORM em Java. A literatura base para as especificações podem ser encontradas nos respectivos sites de seus mantenedores: Apache no caso do JDO e Sun no caso do JPA.

Nas seções seguintes serão apresentadas as duas especificações padrão em Java.

3.1.1. Java Data Objects (JDO)

Java Data Objects (JDO) é uma especificação de persistência que surgiu por volta do ano de 2002 na sua versão 1.0, tendo sido aprovada pela *Java Community Proccess* (JCP) no início de 2005 já em sua versão 2.0 (sua versão mais atual é a 2.1).

A especificação define uma interface para persistir objetos Java (também *Plain Old Java Objects* ou POJO's) em banco de dados, sendo esta indiferente quanto ao tipo do banco de dados utilizado (relacional, orientado a objetos ou mesmo arquivos XML, entre outros).

As relações entre as classes e as tabelas do modelo relacional são estabelecidas utilizando-se de metadados, que podem ser especificados por meio de *annotations* do Java ou arquivos esquemáticos no formato XML.

Os tipos de transações permitidas com o banco de dados são: pessimistas, otimistas, isoladas e sem transações.

Para carregamento de dados o JDO fornece suporte a *queries* SQL nativas e *queries* usando JDOQL (*query language* orientada à objetos padrão do JDO).

Ainda com relação à otimizações para carregamento de dados, o JDO oferece a funcionalidade de *FetchGroups*. Os *fetchGroups* são utilizados de forma a determinar quais propriedades de uma entidade serão carregadas quando é feita uma consulta ao banco de dados de forma a melhorar o desempenho ao retirar propriedades complexas que em geral necessitam de junções entre tabelas diferentes.

O controle da persistência dos objetos é feito utilizando a *PersistenceManagerFactory* e a *PersistenceManager*, ambas são encontradas no pacote *javax.jdo*. A *PersistenceManagerFactory* permite acesso a outras *PersistenceManager* e aos bancos de dados, sendo em geral atribuída uma instância diferente dela para cada banco de dados. O *PersistenceManager* fornece métodos para gerenciar a persistência de objetos, alterando o ciclo de vida deles. Os métodos são detalhados na Tabela 1.

Operação	Método do PersistenceManager
Persistir objeto	makePersistent()
Atualizar objeto	makePersistent()
Remover objeto	deletePersistent()
Obter objeto	getObjectById(), getExtent()

Tabela 1 Métodos do PersistenceManager

3.1.2. Java Persistence API (JPA)

O Java Persistence API (JPA) é uma especificação de persistência padrão aprovada pela JCP em meados de 2006 como parte da especificação EJB3 (Enterprise Java Beans 3).

O JPA, diferentemente da especificação JDO, oferece suporte apenas a banco de dados relacionais; porém também utiliza de interfaces padrão, permitindo aos desenvolvedores a trocar de implementações de persistência sem precisar fazer grandes alterações no código.

Para o mapeamento das entidades, o JPA utiliza-se de metadados ou *annotations*.

Com relação a transações, o JPA permite apenas o uso de transações otimistas ou então sem nenhuma transação que é o modo padrão sobre o qual ele opera.

Para carregamento dos dados, o JPA fornece suporte a *queries* SQL nativas, fornece JPQL e a consulta do tipo *Criteria* por meio de uma API. A JPQL é uma linguagem de consulta orientada à objetos com suporte a funções do ANSI SQL. A API de *Criteria* é totalmente orientada a objetos, e permite criação de objetos exemplo, inserção de expressões e utilização de funções do SQL sem a necessidade de criação de uma *query* específica por parte do usuário.

Os objetos que controlam o funcionamento do JPA, presentes no pacote javax.persistence, são a *EntityManagerFactory* e a *EntityManager*. Da mesma forma que a *PersistenceManagerFactory* do JDO, a *EntityManagerFactory* permite acesso à *EntityManager* e aos bancos de dados, sendo em geral atribuída uma instância diferente dela para cada banco de dados. O *EntityManager* assim como os *PersistenceManager* do JDO fornece métodos para controlar a persistência dos objetos, alterando o ciclo de vida destes. A Tabela 2 mostra os principais métodos do *EntityManager*.

Operação	Método do EntityManager
Persistir objeto	persist(), merge()
Atualizar objeto	update(), merge()
Remover objeto	remove()
Obter objeto	find()

Tabela 2 Métodos do EntityManager

3.2. Frameworks de Persistência

Nas próximas seções serão abordados os *frameworks* estudados neste trabalho, o Data Nucleus e o Hibernate.

3.2.1. Data Nucleus

O DataNucleus é um *framework* de persistência objeto-relacional que anteriormente era conhecido como JPOX, é desenvolvido pela comunidade de software livre e disponibilizado sem custos para ser utilizado no desenvolvimento de aplicações. É um dos frameworks ORM mais flexíveis dentre os disponíveis no mercado devido ao suporte às especificações de persistência JDO e JPA, bancos de dados e linguagens de consulta diferentes como pode ser visto na Fgura 5. (DATANUCLEUS, 2008)

Com relação às especificações o DataNucleus fornece suporte ao JDO e JPA (já abordadas neste trabalho). Sendo que suporta as versões 1.0, 2.0, 2.1 e 2.2 do JDO, a versão 1.0 do JPA além de algumas funcionalidades que estarão presentes na versão 2.0 da especificação (DATANUCLEUS, 2009). O DataNucleus também é conhecido como a implementação de referência para o JDO.

O DataNucleus fornece suporte a variados tipos de banco de dados, dentre eles:

- Bancos Relacionais (RDBMS): Oracle, MySQL, Postgres, entre outros
- Banco Orientado a objeto (OODBMS): db4o
- Outros: Google Big Table, HBase, Arquivos XML, Excel, planilhas OpenDocument

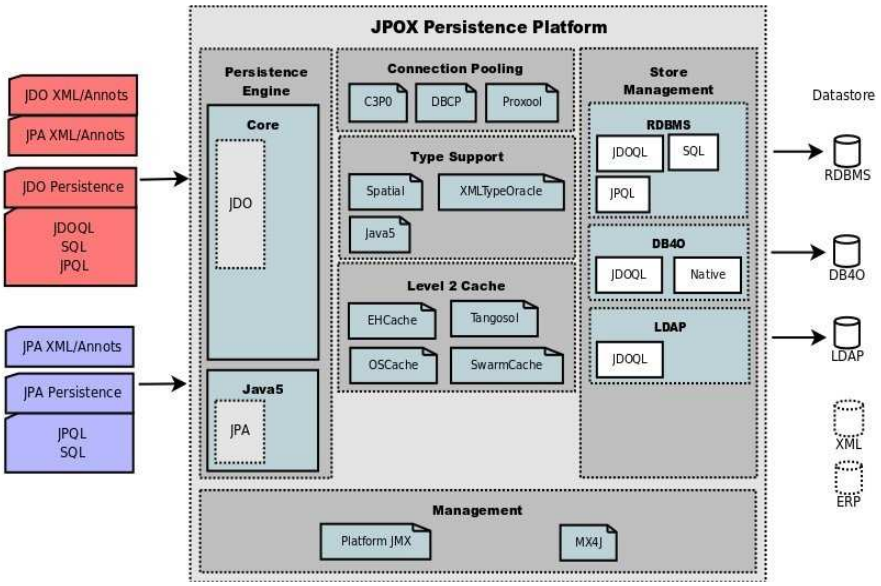

Figura 5 Arquitetura do Data Nucleus

Fonte: DATANUCLEUS, 2008

Além do suporte ao JDO e JPA, e a vários tipos de bancos de dados, o DataNucleus fornece mecanismo de *bytecode-enhancing* para pós-compilação das classes persistíveis de modo a fornecer o mecanismo de persistência transparente.

Na Figura 5, que mostra a arquitetura do DataNucleus (antigo JPOX), podemos identificar os seguintes módulos do *framework:*

- *Engine* de persistência: a *engine* de persistência utilizada pode ser baseada nos padrões JDO ou JPA
- Suporte a tipos: tipos do Java5, Dados Espaciais, etc
- Gerenciamento de Cache: Plugins como EhCache, OSCache, SwarmCache
- Manipulação de banco de dados utilizando JDOQL, SQL e JPQL

3.2.2. Hibernate

O framework de persistência objeto-relacional Hibernate, assim como o Data Nucleus também é desenvolvido pela comunidade de software livre para as plataformas Java e .NET, sendo disponibilizado gratuitamente para seus utilizadores sob a licença LGPL (Lesser GPL).

Diferentemente do Data Nucleus, o Hibernate implementa apenas a especificação JPA e permite conexão apenas com bancos de dados relacionais (RDBMS).

O Hibernate fornece a opção de *bytecode-enhancing* para pós-compilação das classes persistíveis através da ferramenta CGLib, mas isto não é obrigatório como no JDO/DataNucleus. Por padrão o Hibernate utiliza do mecanismo de *Reflection* da plataforma Java para estender as classes do usuário injetando os métodos necessários aos objetos persistentes, mas tudo isto é feito de forma transparente.

O principal módulo do Hibernate é o Hibernate Core, e todos os outros módulos o utilizam como base para suas implementações. Este módulo pode ser utilizado por si só, de forma independente de outros frameworks sob qualquer servidor de aplicações JavaEE/J2EE, sendo necessário apenas configurar sua fonte de dados.

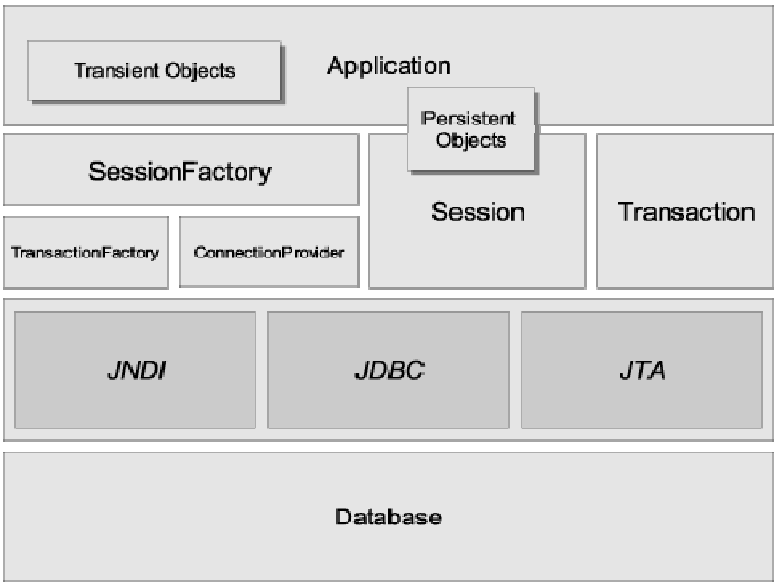

Figura 6 Arquitetura do Hibernate

Fonte: GAVIN ET AL, 2009

A Figura 6 mostra a arquitetura do Hibernate, nela podemos ver as seguintes divisões de funções:

- SessionFactory: é a fabrica de sessões do Hibernate, é instanciada apenas uma fábrica para cada banco de dados utilizado, mas é possível instanciar outras SessionFactory para utilização de bancos de dados diferentes em uma mesma aplicação.
- Session: é responsável pela comunicação com o banco de dados, fornece as operações CRUD, e APIs e linguagens de consulta como a Criteria, JPQL, HQL.

- Transaction: é a interface opcional para controle das transações no Hibernate. É opcional porque a aplicação pode utilizar o *framework* sem transações.
- JNDI, JDBC e JTA: APIs para acesso a banco de dados

4. METODOLOGIA

A pesquisa realizada caracteriza-se por ser de natureza aplicada ou tecnológica, pois ela parte do estudo de teoria de Orientação a Objetos, Banco de Dados e Mapeamento Objeto Relacional para fornecer uma análise de frameworks que são fundamentados sobre estes conceitos.

Com relação aos objetivos este trabalho pode ser definido como uma pesquisa descritiva, pois se baseia na observação, registro e análise dos dados coletados por meio de levantamento bibliográfico e testes realizados em um software.

No que diz respeito aos procedimentos a pesquisa pode ser caracterizada como um estudo de caso, porque foram escolhidas as soluções de ORM a ser utilizadas num software para realização de testes.

Pelo fato de que a utilização dos *frameworks* ter sido feita num software, trata-se de uma pesquisa realizada em laboratório.

A próxima seção explica os critérios utilizados, como foi feita a implementação e a maneira como os testes foram conduzidos.

4.1. Critérios para comparação

Para poder comparar os *frameworks* de persistência analisados nestre trabalho precisamos definir quais critérios são utilizados. A definição dos critérios foi feita tendo por base uma pesquisa bibliográfica sobre o tema, tendo como principais referências os trabalhos de Bauer e King (2007) e Pothu (2008).

4.1.1. Análise dos Frameworks

Segundo Pothu (2008) podemos dividir os critérios em duas seções: critérios gerais para análise de *frameworks* e critérios específicos de ORM. Dentre os critérios gerais foram selecionados os seguintes:

1. Desempenho
2. Existência de boa documentação, fóruns e suporte.
3. Maturidade e longevidade do projeto

Dentre os critérios específicos para mapeamento objeto-relacional que foram estabelecidos por Pothu (2008) foram selecionados os seguintes:

1. Mapeamento de classes: Suporte a tipos, Herança, Relacionamentos.
2. Bancos de Dados suportados.
3. Geração de esquema no banco de dados.

No trabalho de Bauer e King (2007) são definidas algumas questões a que os frameworks devem atender, questões que por sua vez são claramente relacionadas aos critérios específicos de mapeamento objeto-relacional estabelecidos por Pothu (2008). Foram selecionadas as seguintes questões:

1. Como são as classes persistentes?
2. Como o metadado é definido?
3. Como são relacionadas igualdade e identidade do objeto com identidades do banco de dados (chave primária)?
4. Como é o tratamento dos estados dos objetos no contexto de persistência?
5. Quais as facilidades para consultas?
6. Como recuperamos eficientemente dados com associações?
7. Como são tratadas as transações, concorrências?

8. Há gerenciamento de cache?

A Tabela 3 fornece a descrição para cada um dos critérios utilizados neste trabalho.

Critérios Gerais	
Nome	**Descrição**
Documentação e fóruns para discussão	Pesquisa feita nos sites dos projetos
Maturidade e longevidade do projeto	Pesquisa feita na internet, em site de análise de projetos de software livre
Desempenho	(Descrito na Seção 4.2)
Critérios ORM	
Nome	**Descrição**
Forma das classes persistentes	Análise da a Transparência do *framework* e formas de geração/utilização das classes
Definição de metadados	Quais são as abordagens utilizadas para definição de metadados
Identificação de objetos nas tabelas de banco de dados	Verificação de como são mapeadas as instâncias de determinadas classes para linhas particulares de tabelas
Mapeamento de classes	Avaliação das estratégias para mapeamento de tipos, hierarquias, associações

Estados dos objetos no contexto de persistência	Avaliação da forma como os *frameworks* lidam com objetos persistentes/transientes
Facilidades para consultas	Análise das linguagens e APIs de consulta fornecidas pelos *frameworks*
Recuperação eficiente para dados com associações	Verificação de estratégias para recuperação eficiente de dados
Controle de transações e concorrência	Verificação de opções disponíveis para tratamento de transações e concorrência
Gerenciamento de cache	Verificação de opção de gerência de cache
Bancos de dados suportados	Verificação de quais bancos de dados cada *framework* suporta
Geração de esquema de banco de dados a partir das entidades	Verificação se o *framework* oferece geração de esquemas de banco de dados

Tabela 3 Critérios para análise

A partir das respostas aos critérios listados na Tabela 3 é possível contrapor as soluções propostas pelos *frameworks* a fim de se determinar qual deles é a melhor opção para ser utilizado. As respostas para os critérios estabelecidos se encontram no Capítulo 5.

4.2. Sobre os Testes de desempenho

Os testes de desempenho foram realizados no Laboratório de Pesquisa 1 do Departamento de Ciência da Computação da Universidade Federal de Lavras.

O computador utilizado para execução os testes possui a configuração exibida na Tabela 4.

Processador	Intel Core 2 2.4Ghz
Memória RAM	2GB DDR2
Sistema Operacional	Windows XP Professional
IDE Java	Eclipse GanyMede
Versão Java	JDK 1.6
Banco de Dados (SGBD)	PostgreSQL 8.3

Tabela 4 Configuração da máquina utilizada nos testes

A medida coletada foi o tempo gasto para execução das seguintes operações:

- Geração do esquema do banco de dados.
- Tempo de inicialização do sistema.
- Tempo para execução de um teste de entidade.

As operações foram realizadas na plataforma *Trace-On Honey*, também conhecido como Laborapix (RIBEIRO JR. et al; 2007; 2008; 2009; 2010), desenvolvido pela Mitah Technologies em parceria com a UFLA. O sistema possui cerca de 250 entidades que possuem relacionamentos de várias cardinalidades e tipos, além de possuir heranças, com uso de classes abstratas e interfaces.

O teste de entidade é um teste padronizado no sistema Laborapix/Iguassu, e tem como objetivo principal validar as entidades do sistema que são persistentes. Este teste foi escolhido por contemplar as operações CRUD necessárias para avaliação dos *frameworks*.

As operações básicas de persistência (CRUD) implementadas no sistema Laborapix/Iguassu são definidas como se segue:

- Criação de Objetos ou *Create*: Persistência de um objeto transiente.
- Recuperação de objetos de uma classe ou *GetAll*: Recupera todos os objetos de uma determinada classe. No caso do sistema escolhido para testes, existe uma regra para de carregamento que influencia na definição dos *fetchgroups* do JDO ou na determinação do campo com *lazy* ou *eager loading* no caso do *GetAll* serão carregados apenas atributos marcados com a *tag isLookup*.
- Recuperação de um objeto através de chave primária ou *GetById*: Recupera um objeto utilizando buscando por chave primária. Na regra do sistema, a consulta do G*etById* retorna todas as associações de um objeto, sendo que associações de composição deverão estar completas e as de associação comum ou agregação deverão possuir apenas os atributos primitivos.
- Atualização de um objeto (*Update*): Atualização de um objeto desligado.
- Remoção de Objetos (*Delete*): Remove um objeto do banco de dados.

Como se pode notar o modelo de testes contempla todas as operações básicas de persistência (CRUD). Desta forma, podemos afirmar que o

framework de persistência que mais rápido executa o teste de entidade é o que possui o melhor desempenho. Na Tabela 5 são listadas as operações consideradas para medir o desempenho dos *frameworks*.

Medidas de desempenho **Operações**	**Descrição**
Geração do esquema do banco de dados	Tempo gasto para gerar o esquema do banco de dados para todas as entidades do sistema
Inicialização	Tempo gasto para inicializar o sistema. Considerando também o tempo gasto para fazer instrumentação do código fonte
Desempenho	Tempo total gasto para execução de operações relacionadas à persistência de uma entidade.

Tabela 5 Operações consideradas para medida de desempenho

4.2.1. Diagrama de classes

O diagrama de classes do sistema Laborapix é muito extenso, por isso foi selecionada uma de suas entidades para que fossem realizados os testes.

No caso, foi selecionada a entidade Produto, a Figura 7 mostra o diagrama de classes resumido da entidade.

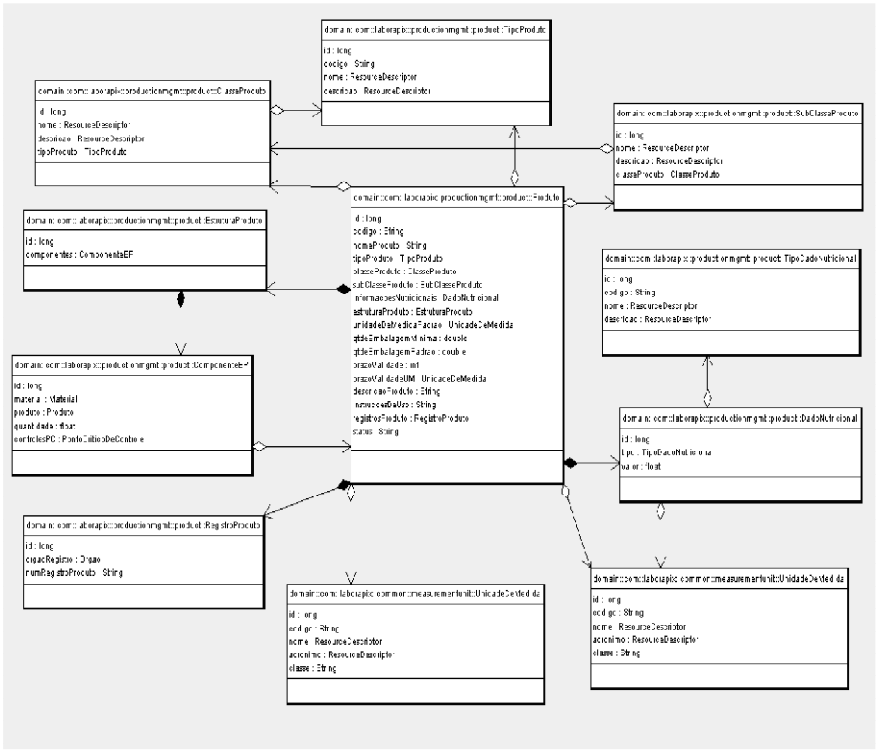

Figura 7 Diagrama de classes da entidade Produto

4.2.2. Geração de Entidades com o BlueBox

As entidades no sistema testado são modeladas como POJOs em diagramas UML. A geração do código fonte em Java para as entidades é feita através da ferramenta BlueBox. Para que o BlueBox possa gerar o código fonte é necessária a criação de um template que especifique as regras para geração do POJOs.

O template das entidades do BlueBox, possuem as regras de persistência nos trechos de declaração das classes e na declaração dos atributos das classes. Na parte de declaração da classe é necessário marcar a classe como persistente e selecionar uma estratégia para herança. Já a parte de declaração dos atributos das classes pode ser separada em: declaração da chave primária (ID), declaração de atributos primitivos e declaração de atributos complexos. Exemplos para declaração de classes para o JDO Data Nucleus e para o JPA Hibernate nos templates do BlueBox são mostrados nas Figuras 8 e 9, respectivamente.

```
83 @PersistenceCapable(identityType = IdentityType.APPLICATION, detachable = "true")
84 @Inheritance(strategy = InheritanceStrategy.NEW_TABLE)
85 ################################################
86 ## Class Declaration
87 ################################################
88 public class ${class.name} $entityUtils.getEntityClassGeneralization($class)
89     $entityUtils.getEntityClassInterface($class)
90 {
```

Figura 8 Declaração de classe para geração de entidades JDO e Data Nucleus

```
53
54##############################################
55## Class Declaration
56##############################################
57@Entity
58@Inheritance(strategy = InheritanceType.JOINED)
59public class ${class.name} $entityUtils.getEntityClassGeneralization($class)
60    $entityUtils.getEntityClassInterface($class)
61{
```

Figura 9 Declaração de classe para geração de entidades JPA e Hibernate

Com relação à seleção da estratégia de herança, foi selecionada as implementações dos *frameworks* para a estratégia de *One Table Per Class* explicada na seção 2.4.2, que segundo Ambler (2003b) é a que suporta melhor a utilização de polimorfismo, oferece melhor visualização no modelo relacional e por necessitar de poucas modificações nas tabelas se for necessário alterar a hierarquia. No caso do JPA foi usada a *annotation* InheritanceType.JOINED e no caso do JDO foi utilizada a *annotation* NEW_TABLE que implementam *One Table per Class*. É possível ver a seleção da estratégia de herança nas Figuras 8 e 9.

A declaração da chave primária nos dois casos é parecida, é preciso informar que o campo é uma chave primária para a classe, alterando o atributo *primaryKey* da annotation @Persistent para true no caso do DataNucleus, e utilizando a *annotation* @Id para o Hibernate. Com relação à estratégia para gerar a chave primária, no caso do DataNucleus foi utilizada a idGeneratorStrategy.INCREMENT, para o Hibernate foi utilizada a GenerationType.TABLE. Ambas as implementações geram as chaves primárias incrementais. As Figuras 10 e 11 mostram a declaração da chave primária para o Data Nucleus e Hibernate, respectivamente.

```
#if (!($class.hasSuperClass) )
    @Persistent(primaryKey="true",
                valueStrategy=IdGeneratorStrategy.INCREMENT,
                defaultFetchGroup="false")
    @Column(name="ID")
    private Long ID;


    public Long getID(){
        return ID;
    }

    public void setID(Long ID){
        this.ID = ID;
    }
#end
```

Figura 10 Trecho de declaração do ID no Data Nucleus

```
#if (!($class.hasSuperClass) )
    @Id
    @GeneratedValue(strategy=GenerationType.TABLE)
    private Long ID;


    public Long getID(){
        return ID;
    }

    public void setID(Long ID){
        this.ID = ID;
    }
#end
```

Figura 11 Trecho de declaração do ID no Hibernate

Na parte de declaração dos atributos primitivos não é necessário fornecer anotações para os campos no Hibernate, que ele já faz o mapeamento do campo para um atributo em uma tabela com o mesmo nome e representação semelhante no banco de dados. Com relação ao Data Nucleus é necessário utilizar a *annotation* @Collumn para especificar o nome que o atributo terá na tabela da entidade. É preciso também tratar os campos transientes de uma classe, com o BlueBox isto é feito verificando se $attribute.ownerScope não é do tipo *instance* (se é um atributo estático), se isso acontecer o campo deve ser

marcado com @Transient no Hibernate e @NotPersistent no DataNucleus. As Figuras 12 e 13 mostram a declaração dos atributos primitivos.

```
123    //primitive
124    #foreach (${attribute} in ${class.primitiveAttributes} )
125        #if ($attribute.ownerScope == 'instance' )
126            #if (($attribute.name != 'id' )
127                @Persistent(defaultFetchGroup = "false")
128                @Column(name = "${attribute.name}")
129                ${attribute.visibility} $entityUtils.getAttributeType($attribute) ${attribute.name};
130
131                #foreach ($tagValue in $attribute.tagsValue )
132                    #if ( $tagValue.name == 'enum' )
133                    public enum $iguassuUtils.upperFirst($attribute.name)
134                    {
135                        $tagValue.value
136                    }
137                    public void set$iguassuUtils.upperFirst($attribute.name)( $iguassuUtils.upperFirst($attribute.name) $attribute.name){
138                        this.$attribute.name = ${attribute.name}.toString();
139                    }
140                    #end
141                #end
142            #end
143        #else
144            #if ( $attribute.changeability == 'changeable')
145                #set ($changeability = 'final')
146            #else
147                #set ($changeability = '')
148            #end
149
150            @NotPersistent
151            public static $changeability ${attribute.nameDataType} ${attribute.name};
152
153            public static $changeability ${attribute.nameDataType} get$iguassuUtils.upperFirst(${attribute.name})() {
154                 return ${attribute.name};
155            }
```

Figura 12 Atributos primitivos do DataNucleus

```
//primitive
#foreach (${attribute} in ${class.primitiveAttributes} )
    #if ($attribute.ownerScope == 'instance' )
        #if (($attribute.name != 'id' )
            ${attribute.visibility} $entityUtils.getAttributeType($attribute) ${attribute.name};
            #foreach ($tagValue in $attribute.tagsValue )
                #if ( $tagValue.name == 'enum' )
                public enum $iguassuUtils.upperFirst($attribute.name)
                {
                    $tagValue.value
                }
                public void set$iguassuUtils.upperFirst($attribute.name)( $iguassuUtils.upperFirst($attribute.name) $attribute.name){
                    this.$attribute.name = ${attribute.name}.toString();
                }
                #end
            #end
        #end
    #else
        #if ( $attribute.changeability == 'changeable')
            #set ($changeability = 'final')
        #else
            #set ($changeability = '')
        #end

        @Transient
        public static $changeability ${attribute.nameDataType} ${attribute.name};
        public static $changeability ${attribute.nameDataType} get$iguassuUtils.upperFirst(${attribute.name})() {
             return ${attribute.name};
        }
```

Figura 13 Atributos primitivos no Hibernate

Com relação ao mapeamento de associações, foi necessário verificar as seguintes informações: objeto do relacionamento, cardinalidade, tipo de associação. No DataNucleus as associações foram mapeadas com as anotações @Element e @Join, que utilizam de tabela auxiliar para implementar os relacionamentos. No Hibernate foi necessário testar a multiplicidade do relacionamento para determinar qual anotação utilizar (no caso 1-1 é @OneToOne, 1-n é @OneToMany, n-1 é @ManyToOne e n-n é @ManyToMany).

Ainda com relação às associações, é necessário determinar o tipo do relacionamento para definir operações que serão cascateadas. O BlueBox obtém informações sobre os relacionamentos por meio de teste da propriedade *aggregation* existente no *atributo*. A propriedade *aggregation* poderá ter os valores: *none* se for uma associação simples, *aggregate* se for uma agregação e

composite se for uma composição. De posse dessas informações podemos definir as regras de cascateamento.

No Data Nucleus, para mapear o tipo dos relacionamentos é necessário alterar os atributos *deleteAction* e *depentent* da anotação @Element. O atributo *deleteAction* controla a deleção de objetos associados de acordo com o tipo da associação, e o atributo *dependent* serve para determinar se as alterações feitas aos objeto das associações serão propagadas. Se for uma associação simples ou agregação, o atributo *deleteAction* será setado com ForeignKeyAction.NONE que determina que os objetos associados não serão removidos caso o detentor do relacionamento o seja, já o atributo *dependent* será setado com *false*, porque as alterações em objetos associados não deverão ser cascateadas. Se for uma composição, o atributo *deleteAction* será setado com ForeignKeyAction.CASCADE que determina que os objetos associados não serão removidos caso o detentor do relacionamento o seja, já o atributo *dependent* será setado com *true*, porque as alterações em objetos associados sempre são propagadas neste caso. O código do *template* do BlueBox para gerar estas associações se encontra na Figura 14.

```
176  ##############################################
177  ## Associates Attributes Declaration
178  ##############################################
179  //associates
180  #foreach (${attribute} in ${class.associateAttributes} )
181
182      #if ($attribute.ownerScope == 'instance' )
183          @Persistent(defaultFetchGroup = "false")
184          // ${attribute.aggregation}
185          #if ( (${attribute.aggregation} == 'none') || (${attribute.aggregation} == 'aggregate') )
186              @Element(types =${attribute.packageType}.model.entity.${attribute.nameDataType}.class,
187                       dependent = "false", deleteAction = javax.jdo.annotations.ForeignKeyAction.NONE)
188          #elseif ( ${attribute.aggregation} == 'composite' )
189              @Element(types = ${attribute.packageType}.model.entity.${attribute.nameDataType}.class,
190                       dependent = "true", deleteAction = javax.jdo.annotations.ForeignKeyAction.CASCADE)
191          #end
192          @Join
193          ${attribute.visibility} List<${attribute.nameDataType}> ${attribute.name};
```

Figura 14 Associações no DataNucleus

Para definição das regras de mapeamento de acordo com o tipo da associação no Hibernate, precisamos alterar o tipo de cascateamento de operações, e isto é feito definindo os métodos que serão executados em cascata com a anotação @Cascade. Para associações de agregação e associação simples não é necessário definir cascateamento, pois o padrão do Hibernate é para não executar operações em cascata. Já no caso de composições foi definido cascateamento de operações de criação (com CascadeType.SAVE_UPDATE), atualização (com CascadeType.SAVE_UPDATE) e remoção (com CascadeType.DELETE_ORPHAN). A Figura 15 exibe o trecho de código do *template* do BlueBox para gerar as associações das entidades com o mapeamento do JPA e Hibernate.

```
152  ##############################################
153  ## Associates Attributes Declaration
154  ##############################################
155  //associates
156  #foreach (${attribute} in ${class.associateAttributes} )
157
158      #if ($attribute.ownerScope == 'instance' )
159          #if ( $attribute.upperMultiplicity == -1 )
160              @ManyToMany(targetEntity=${attribute.packageType}.model.entity.${attribute.nameDataType}.class)
161          #else
162              @ManyToOne(targetEntity=${attribute.packageType}.model.entity.${attribute.nameDataType}.class)
163          #end
164          //${attribute.aggregation} ${attribute.name}
165          #if ( ${attribute.aggregation} == 'composite' )
166              @Cascade({org.hibernate.annotations.CascadeType.DELETE_ORPHAN,
167                        org.hibernate.annotations.CascadeType.MERGE,
168                        org.hibernate.annotations.CascadeType.SAVE_UPDATE})
169          #end
170          #if ( $attribute.upperMultiplicity == -1 )
171              ${attribute.visibility} Set<${attribute.nameDataType}> ${attribute.name} = new HashSet<${attribute.nameDataType}>();
172          #else
173              ${attribute.visibility} ${attribute.nameDataType} ${attribute.name} ;
174          #end
```

Figura 15 Associações no Hibernate

5. RESULTADOS

A análise comparativa dos *frameworks* de persistência Data Nucleus com Java Data Objects (JDO) e Hibernate com Java Persistence API (JPA) foi feita com base nos critérios de comparação descritos na Tabela 3 do capítulo anterior.

Nas próximas seções são relatadas as comparações dos *frameworks* de acordo com os Critérios Gerais e Critérios Específicos de Mapeamento Objeto-Relacional, por fim há uma listagem que permite determinar qual *framework* foi selecionado como o melhor com relação a cada um dos critérios.

5.1. Critérios Gerais

A comparação dos critérios gerais, que foram definidos na Seção 4.1 está listada na Tabela 6.

Critérios Gerais Critério	Data Nucleus	Hibernate
Documentação e fóruns para discussão	• Possui boa documentação no site. • Possui fórum, porém é pouco ativo.	• Possui boa documentação no site do projeto. • Possui fórum bastante ativo.
Maturidade e longevidade do projeto	Segundo o site de análise de software livre Ohloh (www.ohloh.net) o DataNucleus encontra-se em estado de decréscimo de atividades de desenvolvimento ano a ano. Em contrapartida uma grande oportunidade de crescimento para o DataNucleus é devido ao sua utilização no DataStore da Google AppEngine, por ser compatível com o formato Big Table.	Segundo o site de análise de software livre Ohloh (www.ohloh.net) o Hibernate é um projeto com um código bem estabelecido (maduro), com aumento de atividades de desenvolvimento a cada ano, além de possuir uma grande e ativa equipe de desenvolvimento. Além disso, sua evolução está acontecendo juntamente com a especificação JPA e o Java, que é mantida pela Sun Microsystems

Desempenho	Comparação na seção 5.3	Comparação na seção 5.3

Tabela 6 Comparação de acordo com os critérios gerais de análise

5.2. Critérios de mapeamento objeto-relacional

A comparação segundo os critérios ORM definidos na Seção 4.1 está listada na Tabela 7.

Critérios ORM		
Critério	**Data Nucleus**	**Hibernate**
Forma das classes persistentes	Oferece funcionalidade de *bytecode enhancing* para permitir persistência transparente às classes.	Utiliza o mecanismo de *reflection* de Java para adicionar a persistência transparente. Também oferece integração com ferramentas de *bytecode enhancing* que também permite a transparência.
Definição de metadados	Metadados podem ser definidos via XML ou *annotations*.	Metadados podem ser definidos via XML ou *annotations*
Identificação de objetos nas tabelas de banco de dados	Permite utilização de chaves primárias únicas/compostas e várias estratégias de geração de identidade.	Permite utilização de chaves primárias únicas/compostas e várias estratégias de geração de identidade.
Mapeamento de classes	Permite mapeamento de tipos primitivos de Java,	Permite mapeamento de tipos primitivos de Java,

	hierarquia de classes padrão ORM e associações 1-1, 1-N, N-1, M-N, unidirecionais e bidirecionais.	hierarquia de classes padrão ORM e associações 1-1, 1-N, N-1, M-N, unidirecionais e bidirecionais.
Estados dos objetos no contexto de persistência	Fornece os estados comuns de objetos persistentes/transientes. Possui checagem de sujeira (*dirty-checking*).	Estados comuns de objetos persistentes/transientes. Possui checagem de sujeira (*dirty-checking*) e o método *merge()* que é muito usado em ambientes de desenvolvimento orientado a serviços (SOA) em que não se mantém uma referência fixa ao mesmo objeto em memória, precisando reinstanciá-lo a cada nova requisição.
Facilidades para consultas	Permite execução de consultas em SQL e JDOQL (padrão JDO).	Permite execução de consultas em SQL, JPQL (padrão JPA), HQL (padrão Hibernate). Além de oferecer a API Criteria.
Recuperação eficiente para dados com associações	Permite uso de subconsultas no JDOQL. Não permite alteração de estratégias em chamadas a métodos getByExtent()	Oferece estratégias de SELECT, SUBSELECT e JOIN para as associações em consultas de get. Também oferece

	e getById().	subconsultas em HQL, JPQL
Controle de transações e concorrência	Possui controle de transação com suporte a concorrência, controlados com recursos de *locking* e versionamento.	Possui controle de transação com suporte a concorrência, controlados com recursos de *locking* e versionamento.
Gerenciamento de cache	Permite uso da cache nativa do DataNucleus e de cache disponibilizado por plugins (ex.: EhCache).	Permite uso de cache nativa do Hibernate e de cache disponibilizado por plugins (ex.: EhCache).
Bancos de dados suportados	Suporta uma grande variedade de bancos de dados: relacional, orientado a objetos, planilhas, entre outros.	Apenas Bancos de Dados Relacionais
Geração de esquema de banco de dados a partir das entidades	Oferece ferramentas para geração de esquema de banco de dados.	Oferece ferramentas para geração de esquema de banco de dados.

Tabela 7 Comparação de acordo com os critérios específicos ORM

5.3. Desempenho dos frameworks

Os resultados para o teste de desempenho dos *frameworks* são listados na Tabela 8

Medidas de desempenho Operações	Data Nucleus	Hibernate
Geração do esquema do banco de dados	25 segundos	19 segundos
Inicialização	8,68 segundos	6,762segundos
Execução de operações . (Tempo Total)	9,4 segundos	6,754 segundos

Tabela 8 Resultados das medidas de desempenho

A medida de execução de operações se refere ao tempo total gasto para execução das operações executadas no sistema Laborapix utilizando os dois *frameworks* comparados. O tempo de execução para cada operação em separado é listado na Tabela 9.

Tempo de execução Operações	Data Nucleus	Hibernate
Create	104 milisegundos	177 milisegundos
GetByID	83.3 milisegundos	75.5 milisegundos
GetByExample	102 milisegundos	15 milisegundos
GetAll	300 milisegundos	125 milisegundos
Delete	179 milisegundos	266 milisegundos
Update	172 milisegundos	16 milisegundos

Tabela 9 Tempo de execução de operações de persistência

5.4. Resultados da comparação

Para escolher um dos *frameworks* é necessário analisar as respostas dadas a cada critério na seção anterior. Na Tabela 10 estão listadas as escolhas de *frameworks* com as justificativas para decisão de acordo com os resultados da análise.

Critérios Gerais	
Nome	**Framework Escolhido**
Documentação e fóruns para discussão	Hibernate. Por possuir um fórum mais ativo.
Maturidade e longevidade do projeto	Hibernate. Porque possui um projeto mais maduro e com maiores possibilidades de evoluir.
Desempenho	Hibernate. Por sido superior em todos os testes de desempenho.
Critérios ORM	
Nome	**Framework Escolhido**
Forma das classes persistentes	Hibernate. Por permitir mais opções para transparência.
Definição de metadados	Ambas fornecem alteranativas equivalentes.
Identificação de objetos nas tabelas de banco de dados	Ambas fornecem alteranativas equivalentes.
Mapeamento de classes	Ambas fornecem alteranativas equivalentes.
Estados dos objetos no contexto de persistência	Hibernate. Pelo fato do método merge trazer muitas vantagens ao desenvolvedor.
Facilidades para consultas	Hibernate. Por fornecer facilidades como a API

	Criteria.
Recuperação eficiente para dados com associações	Hibernate. Permite mais opções para otimização no carregamento.
Controle de transações e concorrência	Ambas fornecem alteranativas equivalentes.
Gerenciamento de cache	Ambas fornecem alteranativas equivalentes.
Bancos de dados suportados	Data Nucleus. Fornece grande variedade de bancos de dados.
Geração de esquema de banco de dados a partir das entidades	Ambas fornecem alteranativas equivalentes.

Tabela 10 Resultados da análise comparativa

Com base nos resultados listados e justificados na Tabela 10, o conjunto especificação/*framework* escolhido foi o Hibernate com JPA.

6. CONCLUSÕES

Ao analisar os *frameworks* Data Nucleus e Hibernate, foi possível notar que ambas as soluções são muito parecidas, sendo que os critérios que levou a selecionar uma delas são recursos adicionais simples, mas que muito auxiliam ao desenvolvedor.

O Hibernate com a especificação JPA apresenta alguns recursos que muito auxiliaram durante a fase de implementação do software utilizado no estudo de caso, como a API de Criteria, estratégias adicionais para carregamento associações, entre outros.

O DataNucleus não foi escolhido na análise presente neste trabalho, mas não deve ser totalmente descartado porque foi equivalente ao Hibernate em vários aspectos, além de oferecer o suporte a variados tipos de bancos de dados que podem vir a ser soluções concorrentes aos bancos de dados relacionais no futuro. O declínio de atividade no desenvolvimento do Data Nucleus é algo que afeta muito este *framework*, mas, em um cenário em que exista a necessidade de utilizar bancos de dados diferentes dos relacionais, este *framework* pode se tornar uma opção viável.

No cenário atual, em que bancos de dados relacionais são os mais utilizados, o Hibernate se apresenta superior ao Data Nucleus, principalmente levando-se em conta todos os recursos adicionais que apresentados, pode-se dizer que estes foram determinantes para decidir a respeito da utilização deste *framework* no sistema Laborapix/Iguassu. Além disso, outros fatores relevantes foram o desempenho apresentado pelo Hibernate, que em todos os casos testados foi superior ao Data Nucleus e as perspectivas apresentadas pelo projeto do Hibernate, que, ao contrário do Java Data Objects e do Data Nucleus encontra-se em franco crescimento tanto em número de utilizadores quanto na quantidade de evolução da implementação e da qualidade dos recursos oferecidos.

Os resultados foram obtidos durante o estágio na empresa Mitah Technologies, conveniada à Universidade Federal de Lavras – MG, e atualmente todos os colaboradores estão desenvolvendo o software Laborapix utilizando como *framework* de persistência o Hibernate.

7. REFERÊNCIAS BIBLIOGRÁFICAS

Ambler, S.W. **The Object-Relational ImpedanceMismatch,** Ambysoft Inc., 2003a, Capturado em Agosto de 2009. On-Line. Disponível na Internet no endereço http://www.agiledata.org/essays/impedanceMismatch.html

Ambler, S. W. **Mapping Objects to Relational Databases: O/R Mapping In Detail,** Ambysoft Inc., 2003b, Capturado em Agosto de 2009. OnLine. Disponível na Internet no endereço http://www.agiledata.org/essays/mappingObjects.html

Pothu, S. A **Comparative Analysis of Object-relational mappings for Java,** MSc. Thesis, University of Applied Sciences at Braunschweig/Wolfenbuettel, 2008

Bauer, C.; King, G. **Java Persistence com Hibernate,** 1ª Edição, Editora Ciência Moderna Ltda.,2007.

Datanucleus Team. **Data Nucleus Project Documentation v.1.0.0.m4,** DataNucleus Access Plataform, 2008, Capturado em Agosto de 2009. OnLine. Disponível na Internet no endereço http://www.datanucleus.org/

King, G.; Bauer, C.; Andersen, M. R.; Bernard, E.; Ebersole, E.; **Hibernate Reference Documentation 3.3.2.GA,** JBoss, 2009, Capturado em Agosto de 2009. OnLine. Disponível na Internet no endereço http://www.hibernate.org/

Barcia, R.; Hambrick, G.; Brown, K.; Peterson, R.; Bhogal, K. S. **Persistence in the Enterprise: A Guide to Persistence Technologies**, 1ª Edição, IBM Press, 2008.

Biswas, R.; Ort, E. The Java **Persistence API - A Simpler Programming Model for Entity Persistence,** Capturado em Agosto de 2009. OnLine. Disponível na Internet no endereço http://java.sun.com/developer/technicalArticles/J2EE/jpa/

Sun Microsystems, **Documentação do JDBC**, Capturado em Outubro de 2009. Online. Disponível na Internet no endereço http://java.sun.com/products/jdbc/overview.html

Silberschatz, A.; Korth, H. F.; Sudarshan, S.; **Database System Concepts**, 6ª Edição, McGraw-Hill, 2010

Apache JDO. **Documentação do Java Data Objects**, Capturado em Outubro de 2009. OnLine. Disponível na Internet no endereço http://db.apache.org/jdo/

Jendrock, E.; **The Java EE 5 Tutorial,** Capturado em Agosto de 2009. OnLine. Disponível na Internet no endereço http://java.sun.com/javaee/5/docs/tutorial/doc/?wp406143&PersistenceIntro.html#wp78460

Mahmoud, Q. H.; **Getting Started With Java Data Objects (JDO): A Standard Mechanism for Persisting Plain Java Technology Objects,** 2005, Capturado em Novembro de 2009. OnLine. Disponível na Internet no endereço http://java.sun.com/developer/technicalArticles/J2SE/jdo/

Ezzio, D.; **Using and Understanding Java Data Objects,** Capturado em Fevereiro de 2010. OnLine. Disponível na Internet no endereço http://java-jdo.info/Apress-Using.and.Understanding.Java.Data.Objects/tindex.htm

Macoratti, J. C..; **UML - Diagrama de Classes e Objetos,** Capturado em Fevereiro de 2010. OnLine. Disponível na Internet no endereço http://www.macoratti.net/net_uml1.htm

Pinheiro, J. F. V.; **Um Framework para Persistência de objetos em Banco de Dados Relacionais;** Msc. Thesis; Universidade Federal Fluminense, 2005

Castro, Lucas de Luca. **Procedimentos de Modelagem e uma Ferramenta de Geração Automática de Código.** 2010. 59f. Monografia (Curso de Ciência da Computação) – Universidade Federal de Lavras, Lavras, 2010.

Ribeiro Jr., J. C., Victorio, R. A. S. S., Saúde, A. V., Marcucci, M. C., ET AL. **Sistema de rastreabilidade aplicado às atividades de produção rural, industrialização e comercialização de produtos apícolas**, Patente: n.PI0701450-3 (2007), PCT-BR2008-00112 (2008), MX/A/2009/011274 (2009), US 12/596,612 (2010).