Impedância objeto relacional
O atrito natural entre os dois mundos

Marcelo Heitor Teixeira Heitor Teixeira

Resumo: A intenção deste artigo é auxiliar na compreensão do atrito natural existente no universo da tecnologia da informação, especificamente em análise e desenvolvimento de software, entre o modelo orientado a objetos e o relacional. Comentar a respeito de tecnologias de mapeamento objeto relacional, usadas na tentativa de reduzir essa discrepância, e que tem como uma de suas principais funções, abstrair diferenças conceituais entre ambos os modelos.
Palavras-chave: Framework Objeto Relacional. Modelo Orientado a Objetos. Hibernate. Java.

Abstract: The intention of this article is to help in understanding the existing natural friction in the universe of information technology, specifically in software analysis and development, between the object - oriented model and the relational model. Commenting on relational object mapping technologies, used in an attempt to reduce this unconformity, has as one of its main functions to abstract conceptual differences between both models.
Keywords: Relational Object framework. Object Oriented Model. Hibernate. Java.

1. Introdução

A velocidade com que o cenário no desenvolvimento de software muda é algo surreal. Novas técnicas e tecnologias quase que despencam sobre nossas cabeças todos os dias. Logo, estar sempre atualizado, ou “andar na moda” como é dito na forma popular, é uma tarefa que tem se tornado cada vez mais difícil de alcançar. Criam-se cada vez mais *Frameworks*, cada vez mais siglas e abordagens que prometem resolver problemas que até ontem não tinham alguma solução, digamos, elegante.

> O avanço frenético nessa área talvez se justifique pela própria necessidade, cada vez maior, por infraestrutura e arquitetura que suporte os novos aparatos tecnológicos, que prometem facilitar a vida das pessoas. Se remontarmos a história recente da tecnologia, veremos uma janela temporal que gira em torno de 50 anos, talvez um pouco mais. Se comparada ao avanço tecnológico atual, desde o advento do micro processador até os dias de hoje, sejamos francos, avançamos muito, em pouco tempo.

Em meio a esse turbilhão, surgiram modelos para análise e desenvolvimento de software e que são amplamente utilizados até hoje. Dentre os modelos, houve o surgimento do modelo orientado a objetos e o modelo relacional, este último para bancos de dados relacionais.

Ambos os modelos se sustentam até os dias de hoje talvez pela eficiência que cada um apresenta. Todavia, são modelos heterogêneos, e que mesmo assim precisam coexistir em milhares de aplicações espalhadas pelo mundo todo. Logo, por razões que serão explicadas neste artigo, um atrito natural é gerado quando há

a utilização agregada de ambas as abordagens.

Portanto o presente artigo descreve o atrito natural existente entre o modelo de dados relacional e o orientado a objetos, além de alternativas que podem ser consideradas no desenvolvimento de um software orientado a objetos.

O objetivo geral é esclarecer o leitor sobre diferenças naturais entre as abordagens: orientada a objetos e a relacional. Registrar ainda que, por existirem tais diferenças, há também um esforço por parte da engenharia de software, em fazer com que ambos os mundos atuem em conjunto por meio de tecnologias que tentam mapear essas diferenças, para com isso reduzirem os atritos existentes. Os objetivos específicos são:

- Compreender a impedância objeto relacional
- Apresentar brevemente uma maneira para reduzir o atrito gerado em aplicações Java
- Apresentar brevemente a especificação JPA – Java *Persistence* API

2. Impedância objeto relacional e como conviver com as diferenças

Impedância é um termo comum na engenharia elétrica que de forma resumida é a oposição que um circuito elétrico faz à passagem de corrente quando é submetido a uma tensão. Todavia, tem sido empregado em software para que se entenda a diferença que existe entre o mundo relacional e o orientado a objetos.

Há certa controvérsia sobre o assunto, todavia autores renomados não contestam a ideia de que a impedância é algo real e que precisa de uma solução: *The object/relational mapping problem is hard. Essentially what you are doing is synchronizing between two quite different representations of data*... (FOWLER, 2012), que por tradução própria: “O problema do mapeamento objeto/relacional é difícil. Essencialmente, o que você está fazendo é sincronizar entre duas representações diferentes de dados”.

Bauer e King (2007, p. 4) completam este raciocínio dizendo que a persistência tem sido um assunto de muita polêmica entre desenvolvedores Java no mundo todo. Se por um lado existem aqueles que acreditam que o problema da persistência já foi resolvido por meio do advento dos bancos de dados relacionais e seus trechos de código: procedimentos armazenados. Por outro, há quem defenda um modelo de mapeamento objeto-relacional desacoplado do próprio banco.

De acordo com Date (2003, p. 55), no mundo relacional, os dados são mantidos em linhas e colunas (tabelas de dados). Cada linha representa uma tupla, ao passo que cada coluna representa um atributo, ou seja: um dado. À medida que a representação dos dados torna-se mais complexa, criam-se outras estruturas (tabelas) que se relacionam por meio de chaves.

Esse modelo foi proposto na década de 1970 pelo cientista Edgar F. Codd, na IBM. As ideias de Codd foram aceitas de forma quase universal e influenciaram muitos aspectos em diversas áreas da tecnologia da informação. Luckow e Melo (2010, p. 107) atestam que o uso de bancos de dados relacionais, para o armazenamento persistente de dados, faz parte do cotidiano de qualquer desenvolver, pois essa tecnologia está muito bem fundamentada. Ainda sobre Codd – que sem dúvida alguma mereceria muito mais espaço neste singelo artigo – Bauer e King (2007, p. 7) dizem que há quem afirme que o advento de Codd, enquanto conceito relacional para gerenciamento de dados é uma das últimas grandes invenções da computação de todos os tempos.

Com a popularização da orientação a objetos, surge também uma questão que não pode ser desprezada: qual a melhor maneira de representar dados de origem relacional, em modelos de objetos? Antes de tentar responder à pergunta, deve ser observado como um objeto é definido pela engenharia de software.

Em primeiro lugar deve ficar claro que a orientação a objetos não manipula dados por linhas e colunas, conforme o faz o modelo relacional. Na orientação a objetos existe uma representação em memória de um objeto com características e, quase sempre, comportamentos. Sem contar o fato de que objetos são capazes de manter ou não um estado. E, que este estado pode ser recriado em algum ponto no futuro. Bauer e King (2007, p. 8)

Na orientação a objetos, cada registro relacional é representado por uma nova instância do objeto com seus atributos obrigatórios preenchidos. Logo, aqui mostra-se a definição do conceito de estado do objeto, que de acordo com Sommerville (2007, p. 210).

> Um objeto é uma entidade que possui um estado e um conjunto definido de operações definidas para funcionar nesse estado. O estado é representado como um conjunto de atributos de objeto. As operações associadas ao objeto fornecem serviços a outros objetos (clientes) que solicitam esses serviços quando alguma computação é necessária.

Sommerville (2007, p.210) afirma ainda que, objetos são criados obedecendo a suas respectivas especificações. Essas especificações são conhecidas como classes. Geralmente as classes demonstram uma espécie de documentação onde é possível conhecer detalhes do objeto criado. Nas classes, residem os atributos, visibilidade e métodos, além dos tipos de relacionamentos existentes entre os objetos. Ainda em concordância com o texto, Bauer e King (2007, p. 9) confirmam que uma aplicação, com um modelo de domínio, formado por objetos, com ou sem estado e suas respectivas ligações, não funciona diretamente com representação tabular das entidades de negócio. O que eles afirmam é que objetos de domínio de negócio tem as suas próprias representações em uma rede de objetos interconectados.

Estes mesmos autores ainda afirmam que essa disparidade entre os mundos relacional e orientado a objetos fica evidente quando se analisa operações em *SQL*. Bauer e King (2007, p. 9) dizem que,

> Se considerarmos *SQL* e bancos de dados relacionais novamente, conseguimos observar, finalmente, a disparidade entre os dois paradigmas. Operações *SQL* como projeções e junções sempre resultam em uma representação tabular do resultado dos dados. (Isso é conhecido como fechamento transitivo, o resultado das operações sobre relações é sempre uma relação).

E ainda reforçam,

> Isso é um pouco diferente da rede de objetos interconectados utilizada para executar a lógica de negócio na aplicação Java. Esses são modelos fundamentalmente diferentes, não apenas modos diferentes de visualizar o mesmo modelo.

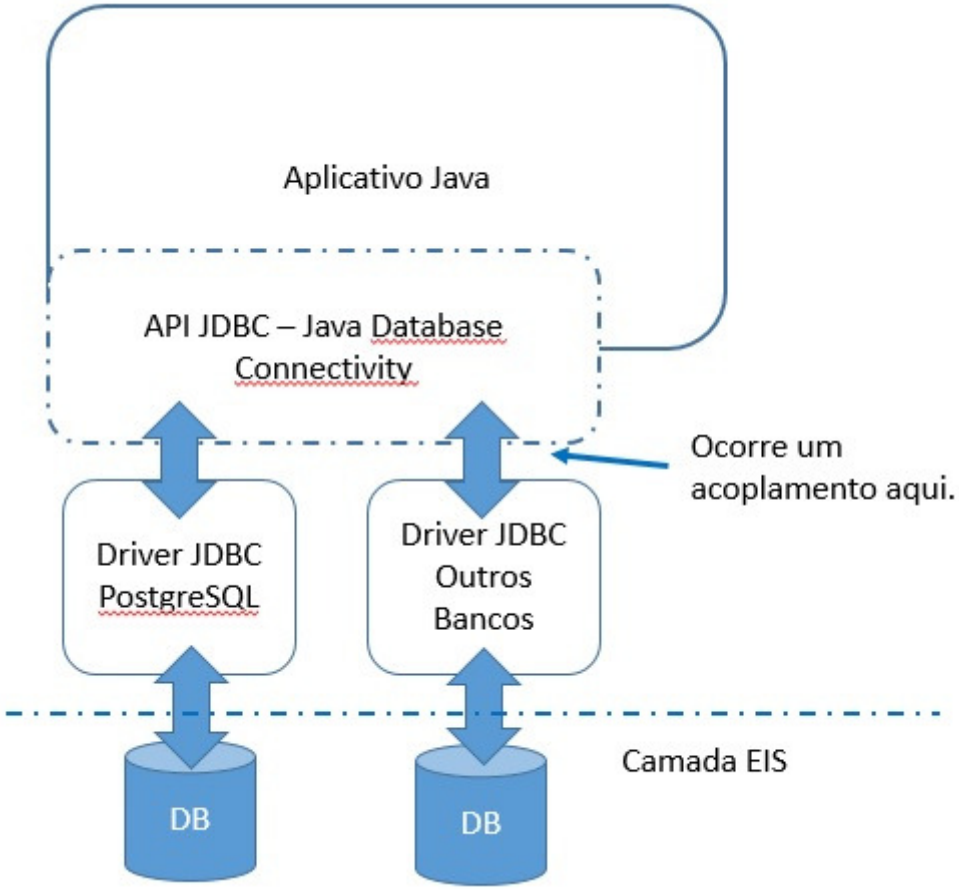
Ainda segundo Bauer e King (2007, 10), neste ponto o leitor já pode perceber o sentido da impedância, ou disparidade do paradigma. Compreender esta diferença ajuda na identificação de problemas que precisam de atenção especial na hora de projetar um software que pretende trabalhar com ambas as abordagens. Problemas como: granulosidade, problema de identidade, relacionados às associações, navegação de dados entre outros.

De fato há um atrito natural causado pela diferença estrutural das abordagens em questão. Seria como tentar encaixar duas formas geométricas diferentes. Logo, Bauer e King (2007, p. 24) dizem que na tentativa de amenizar diferenças surgiram *Frameworks ORM* (*Frameworks* Objeto Relacionais). Um exemplo de *Framework* Objeto Relacional amplamente utilizado em aplicações Java é o *Hibernate ORM.*

Neste momento é importante fazer um adendo registrando que o mecanismo padrão para comunicação com bancos de dados relacionais em Java não é o *Hibernate*, e sim a Java *Database Connectivity - JDBC*. Luckow e Melo (2010, p. 107) afirmam ainda que a *JDBC* é a parte de Java que permite a conexão padrão com bancos de dados relacionais. Todavia, a *JDBC* pura não tem a característica aqui discutida: mapeamento objeto relacional.

A figura a seguir demonstra de forma simplificada a arquitetura tradicional usada por um aplicativo Java, para conexão com bancos de dados relacionais.

Figura 1 – Arquitetura JDBC tradicional.



Fonte: Elaboração própria 2017

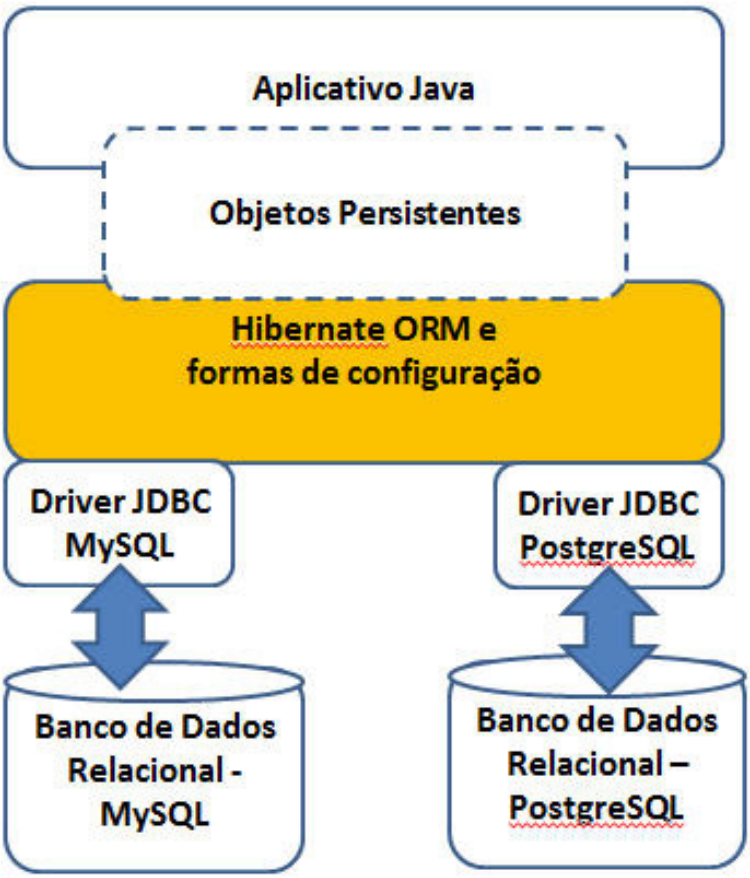
Retomando o texto sobre o *Hibernate ORM*, Luckow e Melo (2010, p. 122) o definem como sendo uma solução *ORM* de mapeamento objeto relacional para a linguagem Java. Gonçalves (2007, p. 512) confirma a essência do *Hibernate* quando sugere que este *Framework* é um projeto audacioso que procura resolver o problema do gerenciamento de dados persistentes em Java. Afirma também que é uma ferramenta que se relaciona de forma objeto-relacional entre o banco e aplicação.

Hibernate tenta auxiliar o mapeamento de atributos entre os dois mundos. Além disso, tem a capacidade de abstrair - em grande parte do código - a necessidade de escrever instruções *SQL* de forma nativa. Segundo Gonçalves (2007, p. 511, 512), essa abstração pode ser vista como benéfica, pois cria a possibilidade do programa não “conversar” diretamente com o banco de dados, o que facilita uma possível portabilidade para outro banco, se necessário.

De acordo com a documentação oficial do produto (*Hibernate*, *hibernate.org*), em sua definição deixa claro que o *Framework* preocupa-se com a persistência e o mapeamento de dados, uma vez que é utilizado para trabalhar com fontes de dados relacionais. Bauer e King (2007, p. 25) explicam que um *Framework* ORM trabalha com a transformação reversível de uma representação de dados, para outra.

A figura a seguir demonstra de forma simplificada e abstraindo em camadas a relação existente entre a aplicação Java e o *Framework Hibernate ORM*.

Figura 2 – Arquitetura Hibernate simplificada

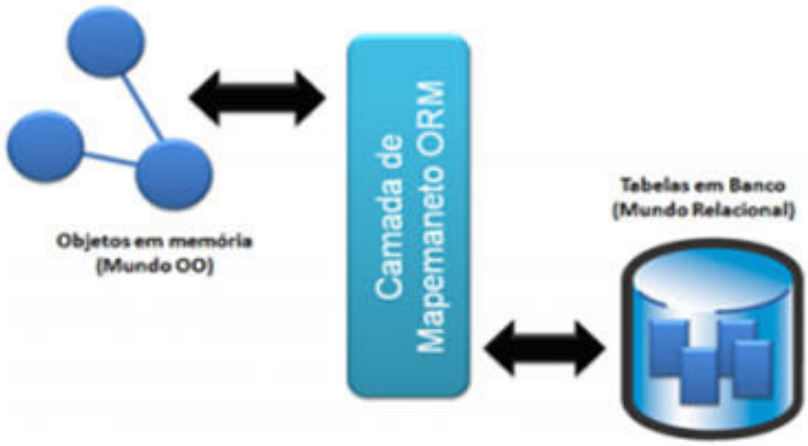


Fonte: Elaboração própria 2017

Ainda de acordo com a documentação oficial disponível em (hibernate.org, hibernate.org/orm/what-is-na-orm) pode ser lido - por tradução própria – que: a impedância objeto-relacional, por vezes chamada de incompatibilidade de paradigmas, é apenas uma maneira elegante de dizer que modelos de objetos e relacionais não funcionam muito bem juntos. Diz ainda que modelos de bancos de dados relacionais representam dados em formato tabular ao passo que linguagens como Java tratam isso de forma interconectada por meio de objetos.

A figura a seguir demonstra a incompatibilidade natural entre as duas abordagens, além de destacar o elemento de camada de mapeamento *ORM*. Neste caso o *Hibernate ORM*.

Figura 3 – Impedância Objeto-Relacional e
Camada de Mapeamento ORM



.
Fonte: Internet 2017.

3. *JPA* – Java *Persistence* API – A API de persistência da plataforma Java

Burke e Monson-Haefel (2007, p.5) descrevem que a plataforma Java *EE* passou por uma profunda transformação para continuar evoluindo. Uma dessas mudanças é assunto desde artigo: a persistência e mecanismos *ORM*. Eles afirmam ainda que a persistência é uma abstração que permite o mapeamento de objetos para o banco de dados, com capacidade de realizar operações de consultas, inserções entre outros, sem a necessidade do *JDBC*, que é sem dúvida, prolixo. Os mesmos autores ainda confirmam que o objetivo da especificação Java *Persistence* é definir um modelo de objeto para mapeamento objeto-relacional (*object to relational mapping - ORM*) e que possa ser portável para qualquer tipo de fornecedor.
Burke e Monson-Haefel (2007, p.5) definem a *JPA* da seguinte maneira,

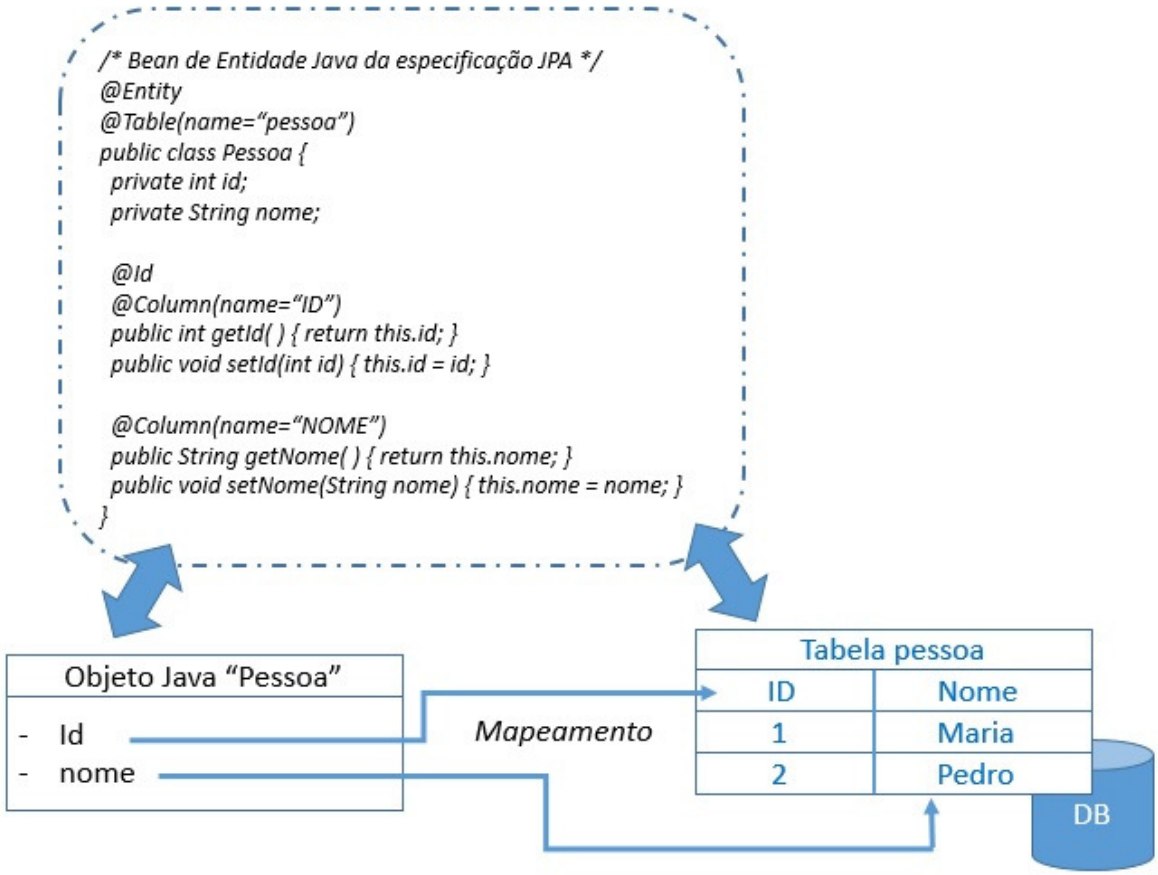
> A Java *Persistence API* define uma maneira de mapear objetos Java simples e comuns (*Plain Old Java Objects - POJOS*) para um banco de dados. Esses objetos Java simples são chamados *Beans* de Entidade. Os *Beans* de Entidade são como qualquer outra classe Java, exceto pelo fato de que eles foram mapeados utilizando metadados da Java *Persistence*, para um banco de dados. Portanto, eles podem ser inseridos e carregados a partir de um banco de dados sem que o desenvolvedor escreva nenhum código *JDBC*. (Burke e Monson-Haefel (2007, p.5)

Burke e Monson-Haefel (2007, p.28) dizem algo que corrobora com este estudo, quando afirmam que o foco da persistência objeto-relacional está em envolver elementos de dois universos estruturalmente diferentes: objetos Java (na forma de *beans* de entidade), e tabelas e colunas de bancos de dados relacionais.

A figura abaixo pretende demonstrar um exemplo do mapeamento proposto pela especificação Java *Persistence*, e do mecanismo *ORM*. Por meio desta imagem pode ser facilmente identificado que o principal objeto de um mecanismo *ORM* é integrar as abordagens orientada a objetos com a relacional da forma menos

traumática possível.

Figura 4 – Mapeamento da Especificação JPA



Fonte: Elaboração Própria 2017.

4. Conclusão

Desde que comecei com Java, em meados de 2001, tive a oportunidade de atuar primeiramente em desenvolvimento de sistemas. Por volta de 2007, direcionei o foco para atuar em áreas de infraestrutura de aplicações Java. A pouco, por volta de 2015, estou de volta ao desenvolvimento.

Concluir este ciclo me permitiu desenvolver uma visão mais acurada a respeito do conteúdo do artigo em questão. Logo, gostaria de esclarecer que a temática aqui não está avaliando a tecnologia *Hibernate* tecnicamente. Ou seja, se é melhor ou pior que outra abordagem. Este seria um aspecto que sozinho já renderia um artigo dedicado somente para este propósito.

O que quero dizer é que como programador, consigo ver algumas facilidades (comodidades) que o *Hibernate* oferece no lado do desenvolvimento em Java. Por outro lado, como analista de infraestrutura, cuidando da "saúde" de servidores de aplicação, consigo ver – também – o quanto custa caro para o servidor - caso o *Framework* e determinadas configurações não estiverem muito bem ajustadas. Quando digo "muito bem ajustadas", me refiro a maioria dos programadores

iniciantes, que – ainda por falta de maturidade – só conseguem ver o *Hibernate* como uma tecnologia “da moda”; e que o usam porque “todo mundo usa”. Ainda quando questionados sobre o motivo de adotarem essa tecnologia em seus projetos, já ouvi dezenas de respostas vagas como: “...não preciso conhecer SQL. É por isso que eu uso!”.

Se o leitor se enquadra no parágrafo acima, gostaria de dar um conselho: pare um instante de desenvolver seu aplicativo. Leia e estude sobre o Hibernate. Melhor, leia e estude sobre mecanismos de persistência e mapeamento objeto relacional. Procure entender se realmente sua aplicação necessita disso.

Faça testes comparando a *performance* de operações em massa de sua aplicação, com e sem o Hibernate. Tenha em mente que haverão situações em que será mais eficiente não usar um mecanismo ORM; em outras, certamente será melhor usar. Enfim. Estude, reflita e então faça! Quando eu trabalhava na *IBM*, uma grande multinacional, gostava muito de uma palavra que tinha uma conotação de frase. Eles diziam simplesmente: Pense!

Com relação à especificação *JPA* – Java *Persistence* API, o pessoal da antiga *Sun Microsystems*, desde 2009 da *Oracle*, percebeu que precisava reformular diversos pontos da especificação *EJB*, pois da forma atual na época, certamente logo seria deixado de lado pela própria comunidade. Então a resposta veio então por meio do *EJB* 3.0, que inclusive investiu pesado na reconstrução do mecanismo de persistência e *ORM*.

Até atrevo-me a dizer que não seria coincidência pensar que a especificação *JPA* fez uso da experiência e maturidade que o *Hibernate* já dispunha na época, para ser então definida.
O leitor deve ficar a par que existem outras abordagens que prometem evitar o tal atrito gerado entre os universos: relacional e orientado a objetos. Uma opção seria projetar a aplicação com um banco de dados compatível à linguagem. Estou falando em usar, por exemplo, um banco de dados orientado a objetos, caso a aplicação também use tal abordagem. Porém, isso pode - digo pode - funcionar melhor para um projeto novo. Ou seja, migrar de um banco relacional para outro modelo em um projeto, já em uso, certamente causará grandes dores de cabeça.

O mais comum é que isso quase nunca acontece em um ambiente corporativo. Ou seja, ninguém que precisa de resultados imediatos investe tempo, migrando seus sistemas apenas para satisfazer uma vontade por exemplo. Isso geralmente se justifica pelo fato das empresas já possuírem legados com toneladas de dados na forma relacional. Portanto, talvez este trecho justifique o motivo da escolha pelos *Frameworks* como o *Hibernate*.

Não posso encerrar sem comentar que outra abordagem é criar o seu próprio mecanismo de mapeamento e persistência. Mas, sejamos francos, com a correria e a cobrança por produtividade e entrega que vivemos atualmente, não parece razoável investir um tempo precioso criando um *ORM* para suas aplicações.

O tempo investido em desenvolver um mecanismo ORM próprio poderia ser bem melhor aproveitado se fosse investido em códigos de negócio, que são o ponto central da aplicação. Criar o próprio mecanismo exige muito tempo e esforço, sem contar que é muito difícil prever todos os erros que podem ocorrer, antes que ocorram.

Por fim, ainda hoje, a opção mais usada para tentar reduzir o atrito entre modelos relacionais e o mundo orientado a objetos, é fazer uso dos *Frameworks* para mapeamento objeto relacional. Mesmo porque os bancos de dados relacionais

ainda dominam o universo do armazenamento de informações. E acredito que se um dia, por exemplo, os bancos de dados orientados a objeto ultrapassarem os relacionais, o legado deste último ainda assim será muito grande para acabar do dia para a noite.

Referências

Bauer, C. King, G. **Java Persistence com Hibernate**, Rio de Janeiro: Editora Ciência Moderna. 2007.

Burke, B. Monson-Haefel, R. **Enterprise Java Beans 3.**0. São Paulo: Pearson Prentice Hall, 2007.

Date, C. J. **Introdução a Sistemas de Bancos de Dados**, 8ª Ed. Rio de Janeiro: Elsevier, 2003.

Fowler, Martin. OrmHate. Disponível em <https://martinfowler.com/bliki/OrmHate.html>. Acesso em 18 de maio de 2017.

Gonçalves, E. **Desenvolvendo Aplicações Web com JSP Servlets, Java Server Faces, Hibernate, EJB 3 Persistence e Ajax**. Rio de Janeiro. Editora Ciência Moderna. 2007.

Hibernate ORM. Disponível em <http://www.hibernate.org>. Acesso em 19 de maio de 2017.

Luckow, D. H. Melo, A. A**. Programação Java para a Web**. Editora Novatec. São Paulo. 2010.

Sommerville, I. **Engenharia de Software**, 8ª Ed. São Paulo: Pearson Addison-Wesley, 2007.