Uma reflexão sobre Banco de Dados Orientados a Objetos

Clodis Boscarioli[1], Anderson Bezerra[2], Marcos de Benedicto[2], Gilliard Delmiro[2]

[1]UNIOESTE - Universidade Estadual do Oeste do Paraná

[2]FASP - Faculdades Associadas de São Paulo

boscarioli@unioeste.br, {anderson.bgs, marcos.b01, gilliard.sd}@faspmail.br

Abstract. *The development of Object Oriented Databases has been achieved in order to attend a requeriment related to the storage of complex data types, which can not be well-represented in the relational model. In this paper we propose to provide a survey of this paradigm, presenting concepts and main tools. Finally, we present a discussing some tendencies for the technology.*

Resumo. *Os bancos de dados orientados a objeto começaram a ser desenvolvidos para atender a uma necessidade relacionada ao armazenamento de tipos de dados cada vez mais complexos, de difícil representação no modelo relacional. Esse artigo tem o intuito de prover uma visão geral desse paradigma, apresentando seus conceitos e ferramentas principais, bem como discutindo as tendências para esta área.*

1. Introdução

Até meados dos anos 60, os dados eram mantidos aleatoriamente em arquivos, geralmente como partes integrantes da aplicação. A partir dessa época, surgiram os primeiros Sistemas Gerenciadores de Bancos de Dados (SGBDs) comerciais, provendo armazenamento dos dados de forma independente da aplicação, contudo, sem mecanismos de acesso eficientes.

Codd (1970) propôs a criação de linguagens de alto nível, permitindo manipulação eficiente. A partir dos anos 80 as aplicações computacionais evoluíram, juntamente com o poder de processamento das máquinas, surgindo a necessidade de tratar dados mais complexos, não-convencionais. Devido a esse aumento na complexidade dos dados, surgiu a necessidade de formas mais adequadas de representação e armazenamento, como as bases de dados orientadas a objetos.

Durante a década de 80, como resultado das inovações de hardware, emergiram novas aplicações com utilização intensiva de dados. Para essas aplicações, os modelos de dados tradicionais, baseados no modelo relacional, não eram adequados. Alguns exemplos de aplicações desse tipo incluem sistemas de *design* e produção, como CAD (*Computer-Aided Design*) e CAM (*Computer-Aided Manufacturing*), sistemas para as áreas científica e médica, sistemas de informação geográfica e bases de dados com informações multimídia. Essas aplicações possuem requisitos e características, tais

como dados altamente estruturados, transações longas, dados em multimídia e operações fora do padrão, específicas da aplicação, que as tornam diferentes das aplicações tradicionais (CHAUDRI & ZICARI, 2001, pg. 3).

Silberchatz *et. al.* (2001, pg. 307) afirmam que "À medida que as bases de dados foram sendo utilizadas em um âmbito maior de aplicações, as limitações impostas pelo modelo relacional emergiram como obstáculos. Como resultado, pesquisadores da área de bancos de dados inventaram novos modelos de dados que superassem as restrições do modelo relacional. [...] Nos últimos anos a demanda por maneiras de tratar dados mais complexos tem crescido".

Existe um grande interesse relativo às tecnologias orientadas a objetos na comunidade de desenvolvimento de software, sobretudo no tocante à facilidade de alteração de implementações de acordo com mudanças solicitadas nos requisitos. A capacidade que esse paradigma possui de representar dados complexos uniu-se à tecnologia de banco de dados, gerando os Bancos de Dados Orientados a Objeto (BDOO), que suportam modelagem e criação de dados como objetos (ODBMS, 2006).

O objetivo desse artigo é prover uma visão geral sobre sistema de BDOO, uma tecnologia não tão nova (BANCIIHON, 1988; ATKINSON, *et. al.*, 1989), contudo considerada ainda pouco explorada. Para tal, está organizado como segue:

Conceitos básicos sobre Banco de Dados Relacionais (BDR) e BDOO, são abordados na Seção 2. A Seção 3 diz respeito à modelagem no paradigma orientado a objetos mostrando como um mesmo modelo pode ser usado em ambas as camadas, de banco de dados e da aplicação. Na Seção 4 alguns Sistemas Gerenciadores de Banco de Dados Orientados a Objetos (SGBDOOs) são apresentados, destacando algumas características interessantes dos mesmos. A Seção 5 tratará de algumas tendências interessantes na área. Por fim, a Seção 6 apresenta propostas de trabalhos futuros e as considerações finais dos autores sobre esta investigação, ainda incipiente.

2. Bancos de Dados Relacionais *versus* Bancos de Dados Orientados a Objetos

BDRs e BDOOs possuem características distintas mas basicamente servem ao mesmo propósito: persistir dados necessários para a manutenção do negócio para o qual são aplicados, possibilitando a recuperação, comparação e tratamento desses dados a fim de produzir resultados tangíveis.

Em BDR, uma coleção de tabelas, todas com nomes únicos, compõem a base de dados, podendo estar relacionada a uma ou mais tabelas. Conceitos como integridade referencial de dados – que garantem que um dado referenciado em uma tabela esteja presente na tabela que está sendo referenciada – e chaves primárias estão presentes e garantem que um conjunto de informações possa ser representado de maneira consistente, independente da forma de acesso.

Já um BDOO possui três pilares principais: herança, polimorfismo e encapsulamento, discutidos a seguir. Este modelo apresenta maior flexibilidade na manipulação de seu conteúdo e por meio de identificadores de objetos manipula os dados de forma consistente.

Silbeschatz *et. al.* (2001) concluem que as bases de dados tradicionais são bastante eficientes para tarefas ligadas ao processamento de dados. A geração de folhas de pagamento e o gerenciamento de contas correntes, são exemplos. Esse tipo de aplicação trabalha basicamente com tipos de dados simples: numéricos, texto e datas. Além disso, essas aplicações possuem itens de dados que podem ser representados como registros razoavelmente pequenos e campos atômicos.

Apesar do conceito de bancos de dados orientados a objetos ser bastante distinto do modelo relacional, o mesmo resulta da integração entre a orientação a objetos e a tecnologia de banco de dados tradicionais. Enquanto na programação orientada a objetos, os objetos existem apenas enquanto o programa que os criou está em execução, os bancos de dados orientados a objetos podem criar objetos que sejam persistentes e compartilhados entre diferentes aplicativos.

3. Modelagem de um BDOO

O paradigma da orientação a objetos possui uma maneira própria de representar um problema. Essa maneira difere muito da forma tradicional de modelagem de dados utilizada pelos bancos de dados relacionais, apesar de guardar algumas semelhanças, sobretudo, relativas à cardinalidade das relações entre as entidades. A seguir, a abordagem de modelagem utilizada pela orientação a objetos será detalhada.

3.1 Objetos

A modelagem de um banco de dados orientado a objetos está intimamente ligada aos diagramas de classes gerados para o sistema. Para Chaudri & Zicari (2001) uma base de dados orientada a objetos é apenas uma coleção de objetos, enquanto em um sistema orientado a objetos, cada objeto representa cada entidade do mundo real.

Cada objeto possui um estado e comportamentos e é diferenciado por um indicador único (OID ou *object identifier*). O estado do objeto depende do valor de cada uma de suas propriedades e o comportamento é especificado pelas operações que podem ser executadas pelo objeto, possivelmente, modificando o estado do mesmo. As propriedades podem ser atributos ou relações entre esse objeto e outros objetos que façam parte do domínio do problema.

Uma das propriedades mais importantes do objeto é justamente sua identidade. Sem isso, não é possível saber com qual objeto se comunicar, seja para obter ou atualizar dados. Idealmente, um objeto deve assumir uma identidade no momento de sua criação e carregá-la pela sua vida útil.

Embora a noção de identificação de objetos possa parecer semelhante ao conceito de chave primária dos bancos de dados relacionais, esses conceitos possuem algumas diferenças expressivas. Enquanto em um banco de dados relacional, uma chave primária utilizada para identificar unicamente uma tupla pode ter algum de seus componentes alterado por uma mudança de estado da relação, em um banco de dados orientado a objetos dois objetos podem ter estados idênticos, mas terem identidades diferentes. Além disso, uma chave primária identifica unicamente uma tupla em uma relação, enquanto um OID identifica unicamente um objeto na base de dados inteira.

Em um banco de dados orientado a objetos o estado de um objeto consiste no valor de cada uma das suas propriedades, sejam atributos do próprio objeto ou relações com outros objetos. Os valores de atributos podem ser bastante complexos, tendo a possibilidade de serem, inclusive, outros objetos. Os relacionamentos também influenciam no estado do objeto e podem ser binários, ligando dois objetos, logo, potencialmente influenciando o estado de ambos.

O comportamento de um objeto em um banco de dados orientado a objetos é representado por meio de métodos, que consistem em uma assinatura única dentro do objeto e uma implementação, responsável pela execução da operação. Nos bancos de dados relacionais existe uma grande separação entre dados e operações causando um fenômeno conhecido como incompatibilidade de impedância, gerado pela necessidade de tradução para uma comunicação apropriada entre a linguagem de programação e o banco de dados.

Em um banco de dados orientado a objeto, tanto as informações quanto as operações que atuam sobre essas operações estão encapsuladas em uma entidade única, o objeto. Ou seja, os dados e comportamentos dos objetos são modelados conjuntamente e armazenados no mesmo sistema gerenciador.

3.2 Classes

Classes são agrupamentos de objetos de um mesmo tipo, que possuem comportamentos (operações) e propriedades (atributos e relações) em comum. Os tipos de objetos (classes) podem ser utilizados como domínios para propriedades ou argumentos para operações. Os bancos de dados orientados a objetos possibilitam que novos tipos sejam criados para se adequar aos requerimentos de cada aplicação, podendo ser combinados com os tipos já existentes no sistema e sendo utilizados exatamente da mesma maneira.

Como visto anteriormente, um objeto possui dados (atributos) e operações (métodos) que agem sobre esses dados. Cada objeto possui dados particulares, entretanto as operações são implementadas pela classe a qual esse objeto pertence. Adicionalmente, pode-se considerar uma classe como uma fábrica de objetos, já que é possível gerar instâncias dessa classe de forma estática.

3.3 Herança

Uma classe pode ser derivada de outra classe, em um processo conhecido como especialização. Por exemplo, a classe Veículo exibida na Figura 1 é um molde para quaisquer objetos veículo que se necessite criar. Cada objeto deve ser único e se torna uma instância da classe à qual pertence.

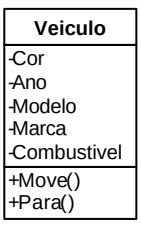

Figura 1 – Classe Veículo.

A herança permite que uma classe seja estendida por outra classe. Tomando o exemplo da classe Veículo, pode-se estendê-la, adicionando atributos e comportamentos para atender ao problema, como uma classe Carro mostrada na Figura 2. A partir desse conceito básico, é possível supor que exista uma infinidade de classes que podem ser derivadas de Veículo, especializando atributos e comportamentos. Essa característica torna-se bastante útil quando da necessidade de expressar um problema que envolva diversos tipos com atributos e/ou operações em comum mas que possuam particularidades inerentes a algumas classes.

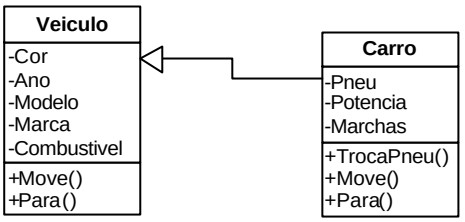

Figura 2 – Classe Carro estendendo a classe Veículo.

3.4 Polimorfismo

Polimorfismo é um princípio que permite um comportamento diferenciado seja implementado por duas classes, desde que sejam especializações de uma mesma classe e o método tenha a mesma assinatura.

A decisão sobre qual comportamento será efetivamente executado é tomada durante a execução do programa, de acordo com o tipo da instância do objeto à qual o comportamento se aplica.

A partir da classe Veículo, considere uma nova classe Barco, como na Figura 3. Note que os comportamentos (métodos) Move e Para são comuns às três classes. Quando o programa estiver em execução, se o método Move for invocado na classe Veículo, as especificidades de cada uma das classes serão utilizadas a fim de causar o efeito desejado, independente de qual seja essa classe, mas pelo tipo da instância do objeto.

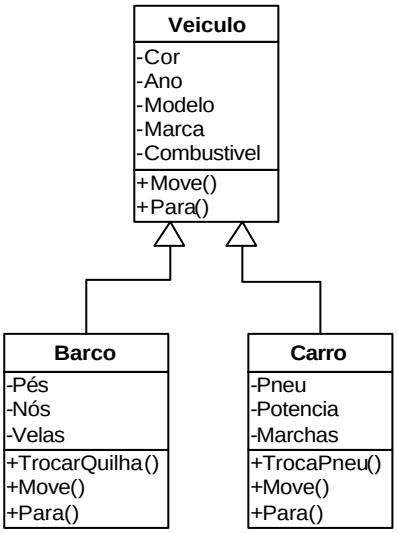

Figura 3 – Classes Carro e Barco especializadas em Veículo.

3.5 Encapsulamento

O conceito de encapsulamento está bastante ligado à abstração de dados, permitindo que apenas as informações que o objeto julgue apropriadas sejam visualizadas externamente, de acordo com o contexto da aplicação.

Esse conceito também pode ser utilizado para acoplar dados com operações comumente executadas, como obter a idade de um Veículo. Ainda, se fosse o caso, seria possível tornar o atributo Ano invisível para os objetos externos e permitir a visualização apenas da operação Idade.

3.6 Utilização de modelos de classes

Ao utilizar um banco de dados orientado a objetos, existe a possibilidade da utilização de um mesmo modelo de classe gerado para o desenvolvimento da aplicação na criação do banco de dados, conforme visto na Figura 4. Geralmente, esse modelo é desenvolvido em UML (*Unified Modeling Language*).

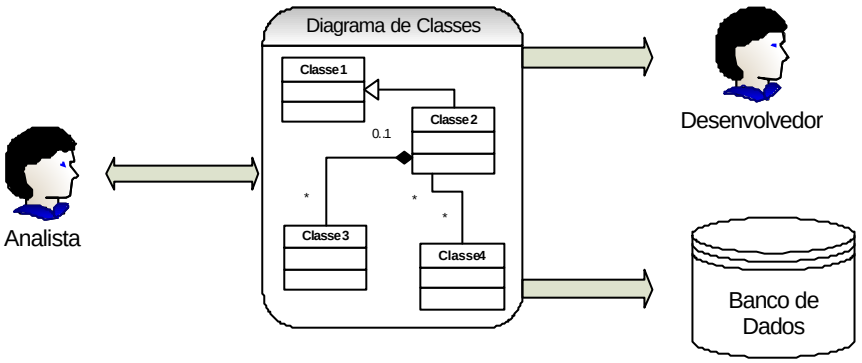

Figura 4 – Modelo de Classes.

Essa abordagem ocasiona vantagens, especialmente, no tocante às alterações de negócio que podem ocorrer durante a análise de um sistema. Segundo Loomis (1992), o projeto de bases de dados relacionais é, em essência, um processo de tentar descobrir como representar objetos do mundo real dentro dos limites de tabelas, de tal maneira que seja obtido um bom desempenho e que a integridade dos dados seja garantida. O projeto de bases de dados orientadas a objeto é um tanto diferente. Na maioria das vezes, o projeto de um BDOO é parte do projeto geral da aplicação. As classes de objetos usadas pelas linguagens de programação são as mesmas usadas no modelo de dados. Devido ao fato de seus modelos serem consistentes, não existe a necessidade de transformar o modelo de objeto para algo que é unicamente utilizado pelo gerenciador da base de dados, como ilustra a Figura 5. Ou seja, nenhum mapeamento, como exposto em (Singh et. al., 2004) se faz necessário.

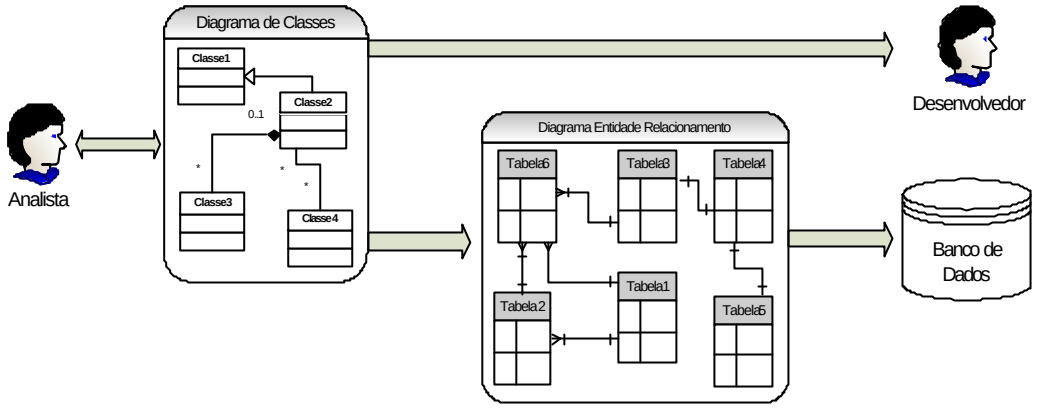

Figura 5 – Utilização do Modelo Relacional.

4. Principais SGBDOOs no Mercado Atual

Os sistemas gerenciadores de bancos de dados orientados a objetos não possuem grandes discrepâncias em termos de funcionalidades exigidas quando relacionados aos SGBDs relacionais. Entretanto, torna-se necessário levantar alguns pontos de diferenciação. Uma das principais exigências de um SGBDOO é o armazenamento de objetos e suas operações de forma a prover uma integração transparente com a aplicação, sem a necessidade de uma camada de tradução dos dados.

A Tabela 1 traz uma comparação entre três grandes SGBDOOs disponíveis no mercado, o Objectivity/DB, o GemStone e o Jasmine, evidenciando que suas funcionalidades não diferem muito com relação aos aspectos mais importantes dos SGBDOOs.

Tabela 1 – Comparativo entre os SGBDOOs Objectivity/DB, GemStone e Jasmine. (Amirbekyan, 1997)

Critério	SGBDOO		
	Objectivity/DB	GemStone*	Jasmine
Definição de dados pelo usuário	SIM	SIM	SIM
Herança Múltipla	SIM	Smalltalk - NÃO Java - SIM	SIM
Valida cardinalidade entre objetos	SIM	NÃO	SIM**
Suporta transações longas	SIM	NÃO	NÃO
Linguagem de definição de atributos	C++, Java, Smalltalk, SQL	Java, Smalltalk	C, C++, ODQL, Java
Armazena os métodos dos objetos no BD	NÃO, os métodos são armazenados no cliente.	SIM	SIM
Suporte a ODL (*Object Definition Language*)	SIM	SIM	NÃO
Suporte a OQL (*Object Query Language*)	SIM	NÃO	SIM
Suporte a SQL	SIM	Smalltalk - SIM	SIM, via

		Java - NÃO	ODBC
Suporta consultas através de interface gráfica	SIM	SIM	SIM
Suporta alterações no esquema através de interface gráfica	SIM	SIM	SIM
Integração com ferramentas CASE	NÃO	NÃO	SIM
Acessa dados de outro SGBD	SIM	NÃO	NÃO
Modifica dados em outro SGBD	SIM	NÃO	NÃO

* Gemstone/S Smalltalk versão 5.1.2 - Gemstone/J Java version 1.1

** Apenas no nível "0, 1 ou muitos".

Para Chaudri & Zicari (2001), a oferta de SGBDOOs tem aumentado, principalmente, como conseqüência do crescimento da utilização da plataforma Java. O mesmo afirmam Dan Lo *et. al.* (2002). Segundo estes autores, o impacto desta linguagem de programação no mundo dos SGBDOOs tem sido considerável e diversos SGBDOOS realizaram modificações para oferecer suporte nativo para o armazenamento de objetos Java. Ferramentas como POET e ObjectStore, inicialmente desenvolvidas para persistência de objetos C++, disponibilizaram versões com capacidade de persistir objetos Java.

Como resultado do aumento da popularidade do desenvolvimento de sistema em Java, a segunda versão do padrão ODMG adicionou elementos de ligação Java. Isto também refletiu em algumas mudanças no modelo de objetos, como a noção de interface e a introdução de dois tipos diferentes de hierarquias.

Tabela 2 – Informações básicas sobre os principais SGBDOOs atuais.

Principais SGBDOOs do mercado				
Nome	Open Source	Sistema Operacional	Fabricante	Site Oficial
EnterpriseDB	SIM	Linux, MacOs, Solaris e Windows	EnterpriseDB	http://www.enterprisedb.com/
Objectivity/DB	NÃO	Linux, Windows e Unix Posix	Objectivity Database Systems	http://www.objectivity.com
GemStone	NÃO	Windows, UNIX e Linux	GemStone System Inc.	http://www.gemstone.com
ConteXT	NÃO	Windows e UNIX	Unixspace	http://www.contextsoft.com/
Versant	NÃO	Windows, Linux e Unix	Versant Corp.	http://www.versant.com
Caché	NÃO	Windows, Linux e UNIX	Intersystems Software	http://www.intersystems.com.br
EyeDB	SIM	Linux e UNIX	Sysra Informatique	http://www.eyedb.org/
Jasmine	NÃO	Windows, Linux e UNIX	Computer Associates	http://www3.ca.com/
ORION	SIM	Linux e UNIX	Orion Group (Purdue University)	http://orion.cs.purdue.edu/
ObjectStore	NÃO	Windows, Linux e UNIX	Progress Software	http://www.objectstore.com

De acordo com a Tabela 2, vê-se que em menos de dez anos houve um grande aumento no número de SGBDOO no mercado, todos implementando capacidades de herança, polimorfismo e encapsulamento. Contudo, poucos ainda são *OpenSource*.

5. Tendências

Algumas tendências referentes à orientação a objetos aplicada a bancos de dados estão sendo discutidas há certo tempo, sem terem amadurecido muito nesse período. Existe hoje, a possibilidade de avaliar quais tendências serão de fato concretizadas, levando em consideração o histórico da tecnologia?

Entre as tendências emergentes neste contexto está o armazenamento em memória. Um exemplo de armazenamento em memória é o Prevayler (2006). Produzido independentemente por um grupo de brasileiros, essa ferramenta consiste em um repositório de objetos com desempenho de acesso bastante superior aos bancos de dados tradicionais, principalmente pela característica de manter os objetos em memória no servidor de aplicações. Essa tecnologia ainda tem limitações, principalmente ligadas à recuperação dos dados em caso de falha e aumento da necessidade de memória.

Existem, e bem aceitos no mercado atual, os sistemas de banco de dados objeto-relacionais. Estes podem ser vistos como uma tentativa de estender sistemas de banco de dados relacionais com funcionalidades dos banco de dados orientados a objetos, necessárias para dar suporte a uma classe mais ampla de aplicações e, de certa forma, prover uma ligação entre esses paradigmas.

Além dos sistemas gerenciadores de banco de dados objeto-relacionais, como Oracle (2006), e PostgreSQL (2006), já bastante difundidos no mercado, outra tendência são as interfaces objeto-relacionais que consistem em uma abstração do modelo relacional para uma camada que provê interface de banco de dados orientado a objetos para uma aplicação e pode ser exemplificada por ferramentas como o Hibernate (2006). Essa tecnologia permite que as aplicações persistam os dados de forma transparente em quaisquer tipos de bancos, inclusive arquivos texto, cuidando de toda a complexidade inerente à tarefa. Para a aplicação é como se existisse um banco de dados orientado a objeto servindo de repositório.

As novas arquiteturas orientadas a serviço, como a SOA (*Service Oriented Architecture*) (GUPTA, 2005), (COHEN, 2006), também estão forçando os desenvolvedores a produzirem sistemas com um aproveitamento cada vez maior de componentes e, conseqüentemente, com menor acoplamento. Por esse motivo, as bases de dados relacionais, onde até o momento os dados são persistidos, deixam de atender a esse tipo de requerimento, tornando as bases de dados orientadas a objetos uma escolha mais interessante para uma escala ainda maior de aplicações.

Com isso, a possibilidade do surgimento de novas tecnologias ligadas ao paradigma da orientação a objeto, que tenham como missão auxiliar na persistência dos dados de cada um dos objetos, tende a crescer nos próximos anos.

6. Considerações Finais

Os bancos de dados orientados a objetos surgiram como uma alternativa revolucionária para os bancos de dados relacionais. Diversas vantagens são oferecidas por esses sistemas de bancos de dados, como o armazenamento direto de tipos de dados complexos e a representação dos dados e das operações de um objeto. Além disso, um único paradigma (orientação a objeto) é utilizado em diversas fases do desenvolvimento de um sistema, durante a análise, através da modelagem dos dados e no decorrer da codificação.

Apesar de toda a variedade de ferramentas disponíveis no mercado, a utilização da tecnologia de bancos de dados orientados a objeto permanece bastante pequena, quando comparada ao modelo relacional. Para Weidler (2004), algumas das razões para a parca adoção dos bancos de dados relacionais estão ligadas à cultura empresarial: devido ao tempo necessário e custos envolvidos, muitos hesitam na migração para um banco de dados orientado a objetos.

Entretanto, Weidler (2004) reflete também que a falta de familiaridade com um SGBDOO, em comparação com um SGBD relacional, o medo de adotar uma nova tecnologia, o tamanho do domínio da aplicação/empresa a modelar, o tamanho reduzido das equipes de desenvolvimento e o pouco conhecimento em orientação a objetos também são fatores que influenciam na adoção do novo paradigma.

Além disso, nos primeiros SGBDOOs a confiabilidade, o desempenho e a disponibilidade eram bastante deficientes no passado, principalmente devido à falta de investimento em ferramentas de gerenciamento. Essas ferramentas não suportavam SQL (*Structured Query Language*), a linguagem de consultas padrão dos sistemas de bancos de dados.

Outro motivo é a incompatibilidade entre as implementações existentes. Segundo McFarland *et. al.* (1999), existem variações significantes nas capacidades de modelagem das ferramentas existentes. Por exemplo, em alguns SGBDOOs, os relacionamentos são suportados por meio de declarações de alto nível, permitindo a definição de propriedades mantidas pelo próprio desenvolvedor. Em outros, o desenvolvedor deve programar a semântica das relações explicitamente.

Processos comumente efetuados em um banco de dados relacional, como otimização, migração e restauração de dados, ainda carecem de amadurecimento no paradigma orientado a objetos, haja vista esses processos delegarem grande parte do trabalho a um nível mais alto, obrigando a aplicação a se responsabilizar por algumas tarefas que deveriam ser controladas pelo SGBD.

Ainda, tal qual no modelo relacional, existe a necessidade de constantes ajustes e monitoramentos do banco de dados a fim de melhorar o desempenho (*tuning*). A diferença está na quantidade de metodologias e ferramentas de ajuste existentes para cada um dos modelos.

Apesar das vantagens elencadas, vê-se que há grandes obstáculos a serem superados, daí o domínio dos sistemas de bancos de dados relacionais no mercado. Atualmente, BDOO é bastante utilizado em mercados verticais, como telecomunicações, finanças e saúde e mesmo assim, em um número reduzido de aplicações. Contudo, sua utilização como repositório de dados multimídia ou de outros

tipos de dados complexos, tem potencial para elevar o nível de utilização dessa tecnologia.

Como propostas de trabalhos futuros pode-se citar a modelagem comparativa de uma aplicação nos paradigmas de modelagem supracitados, bem como um estudo sobre a linguagem de consultas OQL (*Object Query Language*). Ainda, uma comparação detalhada sobre as ferramentas listadas na Tabela 2 é de interesse.

Referências Bibliográficas

AMIRBEKYAN, V. *Comparison of O2, Objectivity, ObjectStore, Versant*. Disponível na URL: http://galaxy.uci.agh.edu.pl/~vahe/products.htm. Data de último acesso: 01/05/2006.

ATKINSON, M.; BANCILHON, F.; DeWITT, D.; DITTRICH, K.; MAIER, D.; ZDONIK, S. *The Object-oriented Database System Manifesto.* Proceedings of the International Conference on Deductive and Object-Oriented Databases *(DOOD '89),* Kyoto, Japan, pp. 223-240. Dezembro, 1989.

BANCIIHON, F. *Object-oriented Database Systems.* Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. Austin, Texas, United States. Pages: 152 – 162, 1988.

CODD, T. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM,* Vol. 13, No. 6, June 1970.

COHEN, F. *Ready To Test SOA, Web Services, ESBs, and BI? Discussion.* The Enterprise Java Community. Maio, 2006. Disponível na URL: http://www.theserverside.com/tt/articles/article.tss?l=ReadytoTest. Data de último acesso: 20/06/2006.

CHAUDRI, A. B; ZICARI, R. (2001). *Succeeding with Object Databases: A Practical Look at Today's Implementations with Java and XML*, Willey Computer Publishing, 1st edition.

DAN LO, C. T; CHANG, M.; FRIEDER, O.; GROSSMAN, D. *The Object Behavior of Java Object-Oriented Database Management Systems.* Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC™02), 2002.

GUPTA, S. *Service Oriented Architecture*. Disponível na URL: http://javaboutique.internet.com/tutorials/serv_orient/. Data de último acesso: 30/06/2006.

HIBERNATE (2006). Site Oficial. Disponível na URL: http://www.hibernate.org. Data de último acesso: 05/05/2006.

KUENG, P. *Comparison of Ten ooDBMS*. Institute of Computer Science, University of Fribourg, Switzerland, June 1994. Disponível na URL: ftp://ftp-iiuf.unifr.ch/pub/Information_Systems/Kueng94a.ps. Data de último acesso: 13/03/2006.

LOOMIS, Mary E.S. ODBMS *vs*. Relational. *Journal of Object-Oriented Programming Focus On ODBMS*, pg. 35, 1992.

McFARLAND, G.; RUDMIK, A.; LANGE, D. *Object-Oriented Database Management Systems Revisited*. 1999. Disponível na URL: http://www.dacs.dtic.mil/techs/oodbms2/oodbms-toc.shtml. Data do último acesso: 01/04/2006.

ODBMS. Site do Portal ODBMS.org. Disponível na URL: http://www.odbms.org/. Data de último acesso: 10/06/2006.

ORACLE CORPORATION. Site Oficial. Disponível na URL: http://www.oracle.com. Data de último acesso: 20/05/2006.

POSTGRESQL. Site Oficial. Disponível na URL: http://www.postgresql.org/ Data de último acesso: 20/05/2006.

PREVAYLER. Site oficial. Disponível na URL: http://www.prevayler.org. Data de último acesso: 24/05/2006.

SILBERSCHATZ, A.; KORTH, H. F. and SUDARSHAN, S. (2001). *Database System Concepts*, McGraw-Hill, 4th edition.

SINGH, A.; KAHLON, K.S.; SINGH, J.; SINGH, R.; SHARMA, S.; KAUR, K. *Mapping Relational Database Schema to Object-Oriented Database Schema.* Transactions on Engineering, Computing and Technology, v1, Dezembro, 2004. Disponível na URL: http://www.enformatika.org/data/v1/v1-38.pdf. Data de último acesso: 13/06/2006.

ULLMAN, J. D.; WIDOM, J. *A First Course in Database Systems.* Prentice Hall, 1st edition, 1997.

WEIDLER, V. (2004). *Database Comparisons: An Overview*. Disponível na URL: http://www.psu.edu/courses/its_training/free_seminars/Database/DatabaseComparisons_Overview/. Data de último acesso: 13/04/2006.