PREVALESCENCIA COMO TÉCNICA DE ARMAZENAMENTO DE OBJETOS

I. F. Vieira Junior
Gerência de Tecnologia da Informação e Educacional de Telemática – CEFET-RN
Av. Salgado Filho, 1159 Morro Branco CEP 59.000-000 Natal-RN
E-mail: ninho@interjato.com.br

J. M. B. Vidal
Gerência de Tecnologia da Informação e Educacional de Telemática – CEFET-RN
Av. Salgado Filho, 1159 Morro Branco CEP 59.000-000 Natal-RN
E-mail: jorgiano@cefetrn.br

RESUMO

A utilização do modelo de objetos no desenvolvimento de sistemas tem sido cada vez mais adotada, onde os dados e as operações são agrupadas em entidades programáveis chamadas objetos. Uma questão importante em sistemas computacionais é a maneira que deve ser utilizada para manter e recuperar os dados, mesmo após o término da execução da aplicação, pois os mesmos serão reutilizados em outras execuções do programa. Nas últimas décadas, a maneira mais usada para se garantir a disponibilidade dos dados é a utilização de bancos de dados relacionais, inclusive para o armazenamento persistente de objetos, gerando assim uma impedância entre modelos. Diversas formas de se mapear os modelos de objetos em bancos relacionais têm sido propostas para amenizar tal dificuldade. Como alternativa, apresentamos a prevalescencia, que é uma técnica para manter as informações em sistemas usando um modelo orientado a objeto puro, sem a necessidade da utilização do modelo relacional. Será usado como estudo de caso o *prevayler*, uma biblioteca que implementa um sistema de prevalescencia na linguagem de programação JAVA. A escolha de linguagem JAVA se deve ao fato de sua larga utilização em sistemas reais, que necessitam manter informações, além de ser uma linguagem Orientada a Objetos. O estudo tem por objetivo mostrar a viabilidade da utilização da prevalescencia em sistemas reais, com grande quantidade de dados, realizando uma comparação com o modelo de persistência utilizando bancos de dados relacionais. Um fator determinante para a utilização de banco de dados relacionais, é o seu alto desempenho. Será mostrado nesse artigo, que para uma massa de dados de sistemas de pequeno a médio porte, a utilização de prevalescencia se mostra mais eficiente. Para grandes sistemas, o custo com a quantidade de memória torna-se um fator limitante.

PALAVRAS-CHAVE: programação; persistência; prevalescencia, prevayler, orientação a objetos, prevalência.

1. INTRODUÇÃO

A utilização do modelo de objetos no desenvolvimento de sistemas tem sido cada vez mais adotada, onde os dados e as operações são agrupadas em entidades programáveis denominadas objetos. Uma questão importante em sistemas computacionais é a maneira que deve ser utilizada para manter e recuperar os dados, mesmo após o término da execução da aplicação, pois os mesmos serão reutilizados em futuras execuções. Nas últimas décadas, a maneira mais utilizada para se garantir a disponibilidade dos dados é a utilização de bancos de dados relacionais, inclusive para o armazenamento persistente de objetos. Diversas formas de se mapear os modelos de objetos em bancos relacionais têm sido propostas para amenizar tal dificuldade. Como alternativa, apresentamos a prevalência, que é uma técnica para manter as informações em sistemas usando um modelo orientado a objeto puro, sem a necessidade da utilização do modelo relacional.

O presente artigo tem por objetivo mostrar a viabilidade da utilização da prevalência em sistemas reais, com grande quantidade de dados, realizando uma comparação com o modelo de persistência utilizando bancos de dados relacionais.

Convencionou-se que a persistência é a capacidade de um objeto de sobreviver fora dos limites da aplicação que o criou. Ao se manipular objetos há de se criar uma maneira de se tornar os dados persistentes, atualmente o mapeamento objeto-relacional é a técnica mais utilizada para tornar dados persistentes.

Umas das técnicas mais utilizadas para persistir objetos é o Mapeamento Objeto-Relacional, nessa técnica classes de objetos são mapeadas em tabelas de bancos relacionais, mas com a utilização dessa técnica ocorre o problema de mal-casamento de impedância, uma série de problemas gerados ao se tentar utilizar bancos relacionais para persistir objetos.

A premissa da prevalência é que os objetos de negócio permaneçam em memória, com isso o tempo de buscas e de inserções reduz drasticamente. Ao se optar pela prevalência não há necessidade de se fazer uma adaptação de dados brutos em objetos, como, por exemplo, ao se utilizar o mapeamento objeto relacional como camada de persistência.

O conceito de prevalência foi inserido por Klaus Wuestefeld, que também desenvolveu um *framework* chamado *Prevayler* implementado em Java que coloca em prática os preceitos da prevalência.

1.1 O modelo orientado a objeto

A orientação a objetos é um modelo de programação e análise que se baseia no relacionamento de entidades programáveis e interoperáveis entre si, denominadas objetos. Os primeiros conceitos deste paradigma de programação surgiu na década de 70 (BOOCH, 1998). O trabalho ainda complementa que este modelo foi concebido na tentativa de gerenciar a complexidade de sistemas, que passaram a ser representados por uma série de componentes, onde estes eram desmembrados em objetos mais simples que interagiam entre si através da troca de mensagens.

O trabalho de (LARMAN, 2000) teve por objetivo propor um conjunto de métodos e técnicas para se criar aplicações orientadas a objetos e com isso conseguir usar os reais benefícios da orientação a objetos. A programação orientada a objetos é baseada em tipos de dados abstratos, conhecidos como TAD. Eles podem ser representados por um encapsulamento que inclui somente a representação de dados de um tipo específico e os subprogramas que fornecem as operações para esse tipo (SEBESTA, 2003).

1.1.1 Classes e Objetos

O conceito-chave do modelo orientado a objetos é o conceito de objeto, segundo (SHALLOWAY; TROTT, 2004). Isto implica que o foco da especificação do *software* está no objeto e não mais em funções, como no modelo procedural.

Objeto, segundo (BOOCH, 1998), é algo que contém comportamento, estado e identidade, com o qual podemos realizar determinadas ações. Objetos com estruturas e comportamentos semelhantes são definidos por uma classe em comum.

O comportamento de um objeto é a forma que ele reage ao receber alguma mensagem de outro objeto, ele pode realizar um ação ou simplesmente ignorá-la, em concordância com (BOOCH, 1998), que define o comportamento de objeto como a forma que ele age e reage a mudanças de estados e ao recebimentos de mensagens de outros objetos.

O estado de um objeto, como descrito no trabalho de (PAGE-JONES, 2001) que diferentemente do modelo procedural, no qual após a invocação de um procedimento, a entidade de *software* que retorna o resultado da invocação morre e ao ser novamente invocada não guarda nenhuma característica de estado anterior, o modelo orientado a objeto é ciente de

seu passado, ele não morre, em caso de invocação futura ele está pronto para repetir os dados. O estado é o conjunto de valores que um objeto consegue manter consigo.

A identidade de um objeto, é uma forma de se distinguir os objetos sem comparar seus atributos, em concordância com o trabalho de (DANTE, 1995), que define que os objetos podem ser distinguidos entre si sem a necessidade de comparar seu valores e seu comportamento. Por sua vez, a referência pode ser utilizada para apontar um objeto único. O trabalho de (BOOCH, 1998) define identidade, como a característica que diferencia ele de outros objetos.
Classe é o modelo, ou o molde, a partir do qual são criados, ou instanciados, objetos (PAGEJONES, 2001). Um determinado conjunto de objetos que possui uma estrutura interna a comportamentos análogos é criado a partir de uma classe em comum e esta classe é denominada de classe pai.

1.2 Conceitos Essenciais da Orientação a Objeto

Em (BOOCH, 1998) e (MEYER, 1997), os autores enfatizam que a capacidade de abstração, o encapsulamento, a modularidade e a hierarquia são fatores altamente relevantes para o modelo orientado a objetos, enquanto que no trabalho de (LEITE; JúNIOR, 2002) é descrito que a abstração, o encapsulamento e a hierarquia são mais relevantes, porém somente em (BOOCH, 1998) considera-se que a falta de um desses conceitos dentro do modelo, faz que este não o contemple em sua magnitude. A seguir, os conceitos de abstração, encapsulamento, modularidade e hierarquia serão descritos sucintamente.

1.2.1 Abstração

A capacidade de abstração, é possibilitar o programador descrever objetos como TAD. O conceito de abstração é bastante utilizado na orientação a objeto. Este conceito se baseia em focar-se no problema e desconsiderar conceitos e dados desnecessários para atingir o objetivo (GOODDRICH; TAMASSIA, 2002).

1.2.2 Encapsulamento

Um outro conceito importante do paradigma em questão, baseia-se na omissão de informações, isto ocorre quando um elemento em questão não deve revelar detalhes de sua implementação. Existe uma interface que será pública onde para os clientes dessa interface não importa como o processamento é feito. Sendo assim, o programador tem total liberdade para implementar o sistema como ele quiser(BOOCH, 1998).

1.2.3 Modularidade

Conceito que se refere a uma estrutura de organização na qual diferentes componentes de um sistema de *software* são divididos em unidades funcionais separadas. A premissa da modularidade é que um componente pode ser reutilizável, ou seja, aproveitado em mais de um projeto(BOOCH, 1998).

1.2.4 Hierarquia

Um objeto pode possuir o mesmo comportamento ou até mesmo parte da estrutura de outro, para isso a orientação a objetos tem o conceito de herança. A herança auxilia para evitar códigos redundantes (GOODDRICH; TAMASSIA, 2002), contribuindo assim para a reutilização de código. O trabalho (DANTE; OLIVEIRA; MENDES, 1993) define hierarquia como o compartilhamento de atributos e operações entre classes e respectivas subclasses. Uma subclasse herda propriedades de sua superclasse e adiciona ao conjunto herdado as suas próprias propriedades. O trabalho de (LEITE; JúNIOR, 2002) ressalta que a aplicação do mecanismo de herança permite ao programador um alto grau de reutilização de código, ainda complementa descrevendo que o seu uso, possibilita a criação de uma nova classe de modo que essa subclasse herda todas as características superclasse; podendo ainda, a classe-filha, possuir propriedades e métodos próprios.

1.3 Técnicas de Persistência Usadas na Orientação a Objeto

Ao se trabalhar em sistemas computacionais, existe a necessidade de se recuperar dados previamente utilizados, para com isso reutilizá-los continuamente. Um dos grandes desafios é como armazenar esses dados em um meio persistente onde seja possível recuperá-los de uma forma simples e segura(MARTINS, 1999).

Em (MARTINS, 1999), a persistência é definida como a capacidade de um objeto sobreviver fora dos limites da aplicação que o criou. Na maioria das vezes, existe a necessidade de que os objetos sejam persistentes, ou seja, que eles permaneçam com o seu estado, mesmo depois que uma aplicação seja finalizada e depois executada novamente.

A definição de Objeto Persistente dada por (FILHO, 2001) é: aquele que sobrevive à execução dos programas, guardado em diversos tipos de mecanismos de armazenamento. Os mecanismos, podem ser a própria memória física, ou memória RAM, um arquivo no disco rígido ou armazenar em algum *software*, como um SGBD. Das técnicas de persistência OO, destacam-se o mapeamento Objeto Relacional, o uso de bancos de dados orientados a objeto e a prevalência.

1.3.1 O Mapeamento Objeto Relacional

Em um mundo dominado pelos Sistemas Gerenciadores de Banco de Dados, uma natural tendência é usar o modelo Relacional para armazenar dados persistentes. Como já citado anteriormente, o modelo orientado a objetos permite conceber aplicações modeladas como objetos que encapsulam tanto comportamento, quanto dados. Por sua vez, o modelo relacional permite o armazenamento de dados em tabelas que poderiam ser manipulados usando uma linguagem própria para este fim (AMBLER, 2003). O trabalho de Ambler ainda afirma que esses dois modelos são usados em conjunto para criar os mais diversos tipos de aplicações. O mundo relacional se baseia em princípios matemáticos, principalmente na teoria dos conjuntos.

Já a orientação a objetos é baseada nos princípios da engenharia de *software* como o acoplamento, coesão e encapsulamento. A orientação a objetos foi concebida para criar aplicações com dados e comportamento, enquanto que o modelo relacional, foi feito para armazenar dados.

Um dos principais problemas ao se utilizar um SGBD em um sistema orientado a objetos é o problema de casamento de impedância. Ele se refere a um conjunto de dificuldades técnicas e conceituais que são encontradas quando um programa escrito em uma linguagem orientada a objetos utiliza um banco de dados relacional para persistir seus dados(AMBLER, 2003).

Em (AMBLER, 2003), o autor ainda complementa que problemas com a identidade do objeto, visibilidade de atributos e objetos, como também a necessidade de normalização de tabelas no modelo relacional contribuam com a impedância entre modelos. Programas orientados a objetos são concebidos com métodos que resultam em objetos encapsulados os quais contém a sua estrutura interna. O trabalho para conseguir representar esses dados internos faz com que a manipulação de dados relacionais se torne complexa(SANTOS; OLIVEIRA, 2003).

1.3.2 Persistência com Prevalescencia

A prevalência, ou prevalescencia, é um modelo de persistência de objetos em memória, que pode ser empregado para construção de um banco de dados (de objetos) em memória para aplicações. Ela difere de outros mecanismos, como linguagens persistentes e Banco de Dados Orientados a Objeto, principalmente pelo fato de que a cópia primária dos objetos residem permanentemente em memória enquanto que um sistema convencional, a cópia primária reside em disco e os blocos são temporariamente trazidos para memória (SANTOS; OLIVEIRA, 2003).

O seu funcionamento se dá através da serialização, o trabalho de (PAMPLONA, 2005) define serialização como uma técnica para colocar os valores que o objeto está utilizando juntamente com suas propriedades de uma forma que fique sequencial, ao se tornar um objeto *Serializable* ele obtém características as quais permitem que ele seja gravado em disco ou enviado via rede.

O trabalho de (SANTOS; OLIVEIRA, 2003) descreve que durante o desenvolvimento de uma aplicação, é necessário ter uma classe raiz, que ao ser instanciada criará o objeto raiz, que também é denominado de base prevalente, essa base é criada a partir da implementação da prevalência. A partir dessa classe todos os objetos de negócio serão referenciados. O trabalho ainda complementa descrevendo que a relação dinâmica a partir do objeto raiz, é um fator determinante para o estado do objeto, ele pode ser transiente ou persistente. A implementação do objeto raiz deve conter estruturas de dados para armazenar da melhor maneira os objetos, implementações em Java normalmente armazena os objetos em *ArrayLists*, tabelas *Hash* ou em *Vectors*. Ao se utilizar dessas estruturas de dados pode-se usufruir de todos os benefícios que essas estruturas de dados oferecem(GOODDRICH; TAMASSIA, 2002). No caso do objeto ser referenciado pelo objeto raiz, ele é denominado de persistente, caso contrário, ele é considerado transiente. O fato do objeto ser transiente faz com que ele não seja persistido, a mudança de estado de um objeto de transiente para persistente se dá no momento de seu referenciamento com o objeto raiz, essa ação é executada durante uma Transação.

Durante o ciclo de vida da aplicação, objetos são instanciados, alterados e finalizados, um grande desafio de qualquer sistema persistente é manter os objetos íntegros e disponíveis ao longo do tempo. Cada alteração na base prevalente deve ser realizada através de uma Transação, o trabalho de (SANTOS; OLIVEIRA, 2003) descreve que no ato que um objeto seja criado, alterado ou removido uma transação deve ser realizada, garantindo assim, a integridade da base prevalente. No momento em que uma destas operações seja realizada um registro é gerado imediatamente para o disco, para que em caso de reinicialização do sistema a base prevalente possa ser reconstruída a partir desses registros serializados em disco.

Segundo (PREVAYLERTEAM, 2004), algumas dessas técnicas de persistência e de garantia de integridade implementadas na prevalência já existiam, um exemplo pode demostrado pelo trabalho de (BIRRELL; JONES;WOBBER, 1987), que propõe uma técnica para a implementação de um pequeno sistema de armazenamento de dados. Algumas das técnicas propostas nesse trabalho foram utilizadas para criar a prevalência. Ao se trabalhar com objetos diretamente na memória, até uma simples busca sequencial, se torna rápido ao utilizar as vantagens em se trabalhar na memória RAM. Esse trabalho foi usado como base para fudamentar o desenvolvimento da prevalência e do desenvolvimento da ferrameta Prevayler. O conceito principal da prevalência é que os objetos de negócio devem permanecer em memória. Na prevalência assim como em sistemas de banco de dados em memória, o armazenamento principal dos objetos é feito em memória. Por este motivo, dois mecanismos devem ser utilizados para garantir a persistência dos objetos: um log de transações e um backup (SANTOS; OLIVEIRA, 2003; BIRRELL; JONES; WOBBER, 1987). O trabalho de (SANTOS; OLIVEIRA, 2003) ainda alerta sobre algumas restrições das classes que devem ser prevalecidas, elas tem que ser serializáveis e deterministas.

2. ESTUDO DE CASO

2.1 O Sistema Gerenciador de Acervo Pessoal

Para este estudo de caso será utilizada uma aplicação desenvolvida na disciplina de Prática em Desenvolvimento de *software* do CEFET-RN. O sistema gerenciador de acervo pessoal tem como objetivo geral disponibilizar aos seus usuários um controle total sobre o acervo pessoal destes, de forma informatizada. Os usuários poderão realizar e monitorar os empréstimos dos livros dos seus acervos, a qualquer momento feitos para os seus contatos como por exemplo amigos, parentes, colegas de trabalho, dentre outros.
Ainda deve permitir ao usuário o controle flexível de seu acervo pessoal de livros, via uma página *web*, um programa *desktop*, ou até mesmo de um dispositivo móvel, como um celular ou o um *Handheld*.

É comum dentro de uma rede de relacionamentos o empréstimo de livros entre amigos, e muitas vezes o dono do livro não mantém um controle direto desses empréstimos, muitas vezes ocasionando um descontrole, ou até mesmo, a perda de parte de seu acervo pelo fato de descontrole.
Para reduzir esse problema o sistema gerenciador de acervo pessoal foi concebido. Ao permitir ao usuário um controle efetivo de seu acervo de livros, o sistema se torna útil, como também permitir a visualização de relatórios para determinar com quem cada livro está.

2.2 Delimitação

Para um maior enfoque desse artigo, algumas classes desenvolvidas foram excluídas, como a parte responsável pela visão para o usuário, ficando assim as classes de negócio, domínio e persistência. A figura 01 demostra as camadas relevantes para este trabalho.

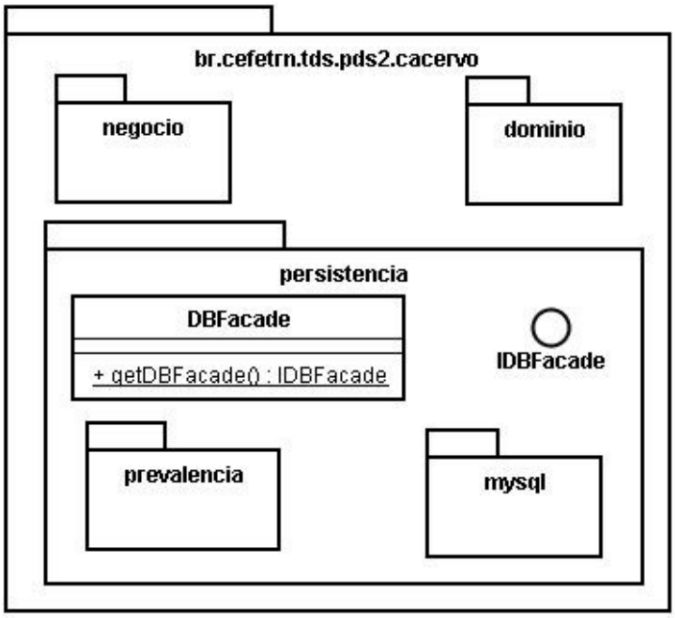

Figura 01: Camadas Relevantes do Sistema de Controle de Acervo

2.3 Análise

O processo de análise do sistema foi iniciado com o levantamento básico de requisitos, inicialmente seis requisitos foram encontrados e estão descritos abaixo. Os diagramas foram modelados de acordo com a UML.

- F-001: Controlar Acervo - Inserção, exclusão, atualização e recuperação de livros.
- F-002: Controlar Contatos - Inserção, exclusão, atualização e recuperação de contatos.
- F-003: Emprestar Livros - Realizar o empréstimo para algum contato cadastrado.
- F-004: Baixa de Livro - O sistema deve dar baixa no acervo, no ato do empréstimo.
- F-005: Devolver Livro - Realizar a devolução de um ou mais livros.
- F-006: Entrar no Sistema - O sistema deve realizar a autenticação dos usuários.

A partir destes requisitos foi elaborado um modelo de domínio. Na figura 02 subseqüente é demostrado o modelo de caso de uso do sistema, onde o foco do estudo de caso será na realização dos cadastros, já que estes criam e alteram significativamente objetos de domínio.

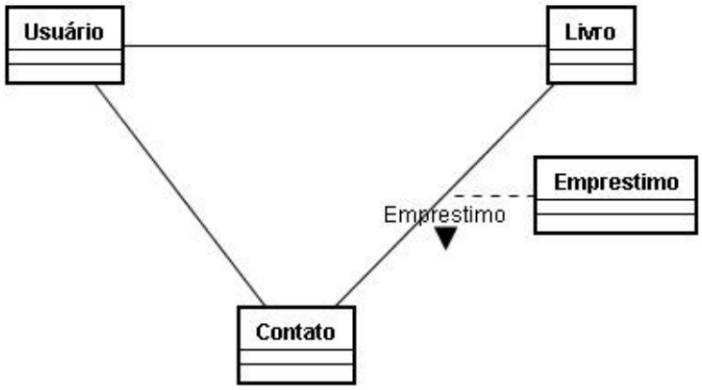

Figura 02: Modelo de Domínio do Sistema de Controle de Acervo

Foi usado o modelo em camadas para o desenvolvimento do sistema, com isso fica mais simples e independente a implementação da camada de persistência. O uso de padrões de projeto como *singleton*, *facade* e *factory* ajudaram para garantir a alta coesão entre as partes. O *software* foi organizado em pacotes *Java*, onde cada pacote é uma unidade funcional, que fornece serviços para outra. No domínio utilizamos o padrão *Java Beans*. O nosso foco é na camada de persistência, onde existe uma interface em comum denominada *IDBFacade*.

A interface *IDBFacade* é o contrato base, que deve ser implementado por as opções de persistência a serem oferecidas pelo sistema. A Figura 03 mostra o trecho de código e denomina os métodos a serem implementados.

```
package br.cefetrn.tds.pds2.cacervo.persistencia;
import java.util.Collection;
/**
 * Interface que cada metodo de persistencia deve implementar
 * @author Ivanilson Júnior
 */
public interface IDBFacade {
    public boolean save(Object o);
    public boolean update(Object o);
    public boolean delete(Object o);
    public Collection query(Object o);
    public Collection query(Object o, String query);
    public Collection query(Object o, Object o2);
    public Collection query(String consulta);
}
```

Figura 03: "Interface que deve ser implementada pelo tipo de persistência"

Para garantir o baixo acoplamento da camada de persistência com o resto da aplicação, durante a implementação, alguns padrões de projeto foram utilizados. A classe *DBFacade* é a classe responsável pelo retorno do método de persistência selecionado, a partir de um parâmetro definido no ato da execução do programa a classe retorna a implementação de persistência que será utilizada.

A utilização do padrão de projeto *singleton* se deu pelo fato da garantia de uma instância única do banco, a garantia se da pelo único acesso através do método estático getDBFacade sem parâmetros de argumento. Aliado ao *singleton* o padrão *Factory Method* auxilia na criação de objetos do tipo persistência, aliada a interface *IDBFacade*. (Figura 04)

```
private static IDBFacade getInstance() {
    if (singleton == null) {
        String classe = System.getProperty("DBFacade");
        if (classe != null && !classe.equals("")) {
            try {
                System.out.println("Persistência Selecionada: " + classe);
                singleton = (IDBFacade)Class.forName(classe).newInstance();
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("Persistência Selecionada: DBFacadePrevayler");
            singleton = DBFacadePrevayler.getInstance();        // dbfacade default*/
        }
    }
    return singleton;
}
```

Figura 04: "Método que retorna o tipo de persistência"

A classe *DBFacade* identifica o método de persistência a ser usado através de uma propriedade chamada *DBFacade*, esta propriedade é identificada pelo método System.getProperty("DBFacade"), que retorna o valor das variáveis. Essa variável é definida no ato da execução do programa pelo parâmetro -DDBFacade=DBFacadePersistencia.

2.4 Testes de Carga

O teste será realizando usando o componente gerado, ele vai avaliar o tempo de determinadas operações no sistema, como criar um usuário, inserir livro, inserir contato e remover usuário. O código fonte abaixo descreve como será feito os testes. A unidade de tempo utilizada pelo estudo comparativo será o milisegundo, recuperada pelo método estático System.currentTimeMillis(). Cada teste será executado por três vezes para se manter uma média de cada operação.

2.4.1 Arquitetura de Testes

A plataforma utilizada para os teste é a IBM-PC, todos os testes serão realizados na mesma máquina, um *Intel Celeron* 1.3 Ghz com 1GB de memória RAM e HD de 40GB com 12GB de espaço livre. O *MySQL* foi instalado da forma *default* e o tipo de tabela usada foi o *InnoDB*, na sua versão 5.0.27-community-nt.

3. RESULTADOS

O teste inicia com a criação de usuários, a aplicação gera de forma aleatória uma quantidade definida de usuários, esta prova foi realizada primeiramente com 100 usuários e depois mais 900 usuários, os tempos são medidos em milisegundos. A prova foi repetida por 3 vezes, onde os resultados da tabela correspondem a média aritmética. Após a execução do programa, os seguintes resultados foram obtidos:

Tabela I – Teste de inserção

Resultados(ms)	***100 Usuários***	***900 Usuários***
MySQL	4641	43703
Prevayler	3562	33938

Analisando a tabela I pode notar que a prevalência é mais rápida do que o MySQL, quando inserimos 100 usuários, onde temos em média de uma inserção a cada 35,62ms. No MySQL temos uma média de uma inserção a cada 46,41ms. Com a inserção de 900 usuários o tempo médio do MySQL foi para 48,55ms, enquanto a prevalência ficou em 37,70ms.

No próximo teste foi medido os tempos de recuperação de todos os Usuários cadastrados, o primeiro teste mediu-se o tempo de recuperação de 100 Usuários e no segundo o tempo de recuperação de 1000 Usuários

Tabela II – Teste de recuperação

Resultados(ms)	***100 Usuários***	***1000 Usuários***
MySQL	162	968
Prevayler	0	0

Analisando a tabela II é possível perceber a real vantagem da prevalência, como os objetos consultados ficam permanentemente na memória o tempo de consulta é nulo, já os objetos estão instanciados, ao contrário do MySQL que ainda tem que consultar a base de dados e instanciar cada objeto para poder retorna-lo.

No próximo teste foi medido os tempos de remoção de todos os Usuários cadastrados, o primeiro teste mediu-se o tempo de remoção de 100 Usuários e no segundo o tempo de remoção de 1000 Usuários

Tabela III – Teste de remoção

Resultados(ms)	***100 Usuários***	***1000 Usuários***
MySQL	5390	48187
Prevayler	4562	41344

Analisando a tabela III é possível perceber que na remoção a prevalência tem uma vantagem, mas não é tão significativa quanto na inserção ou na recuperação.

4. CONCLUSÕES

O paradigma de desenvolvimento orientado a objeto, é atualmente um padrão para construção de *software*. Aliado a práticas e processos que aumentam significativamente a produtividade, como práticas ágeis, contribuem de forma significativa para aumentar a qualidade do *software*, como também a satisfação dos seus usuários e desenvolvedores.

O fato da prevalência trabalhar totalmente orientada a objetos já reduz o esforço do programador em ter que realizar o mapeamento das classes de seu sistema. Uma camada de persistência bem projetada permite ao programador ter total liberdade de qual técnica de persistência usará em sei sistema.

O uso da prevalência ainda permite tempos de consultas muito reduzidos, pois como utiliza plenamente o modelo orientado a objeto, trabalha somente com a referência dos objetos, conseguindo assim, tempos muito curtos de acesso a objetos, como demonstrado no estudo de caso.

Ao se trabalhar com a inserção de objetos a prevalência se tornou uma opção interessante, pois conseguiu tempos inferiores a implementação do mapeamento objeto relacional, utilizando o banco *MySQL*. Ao se remover objetos, a prevalência teve um desempenho também melhor que a outra implementação.

Com a crescente necessidade dos sistemas de ter um tempo de resposta cada vez menor, a prevalência pode ser utilizada para se alcançar esse requisito não funcional, cada vez mais importante para o desenvolvimento. Aliado a necessidade de se ter um ambiente totalmente orientado a objeto, faz com que o programador tenha um trabalho de desenvolvimento intuitivo e também reduz o esforço de programação, pois a prevalência reduz a necessidade de códigos SQL dentro da aplicação.

5. REFERÊNCIAS BIBLIOGRÁFICAS

AMBLER, S. W. **Agile Database Techniques**. [S.l.]: Wiley, 2003.

BIRRELL, A. D.; JONES, M. B.; WOBBER, E. P. **A simple and efficient implementation for small databases**. digital systems Research Center, v. 1, p. 13, 1987.

BOOCH, G. **Object-oriented analysis and design with applications**. 2. ed. [S.l.]: Addison-Wesley, 1998.

DANTE, C. A.; OLIVEIRA, P. A. F. de; MENDES, R. R. da F. **Orientação a Objetos - continuação**. Outubro 1993. Disponível em: <http://www.pr.gov.br/batebyte/edicoes/1993/bb27/orientacao.htm>. Acesso em: 27 jun. 2006.

DANTE, C. A. A **Identidade de um Objeto**. Setembro 1995. Disponível em: <http://www.pr.gov.br/batebyte/edicoes/1995/bb46/objeto.htm>. Acesso em: 27 jun. 2006.

FILHO,W. de P. **Engenharia de software: Fundamentos, métodos e padrões**. 1. ed. [S.l.]: LTC, 2001. 604 p.

GOODDRICH, M. T.; TAMASSIA, R. **Estruturas de Dados e Algoritmos** em Java. [S.l.]: Bookman, 2002.

LARMAN, C. **Utilizando UML e padrões: uma introdução à análise e ao projeto orientado a objetos**. 1. ed. [S.l.]: Bookman, 2000.

LEITE, M.; JúNIOR, N. A. S. R. **Programação Orientada a Objeto: uma abordagem didática**. [S.l.], Agosto 2002. Disponível em: <http://www.ccuec.unicamp.br/revista/infotec/artivos/leiterahal.html>.

MARTINS, V. **Persistência de Objetos**. [S.l.], Setembro 1999. Disponível em: <http://www.pr.gov.br/batebyte/edicoes/1999/bb90/objetos.htm>.

MEYER, B. **Object-Oriented Software Construction**. 2. ed. [S.l.]: Prentice Hall Professional, 1997.

PAGE-JONES, M. **Fundamentos do desenho orientado a objeto com UML**. 1. ed. [S.l.]: Pearson education do brasil, 2001.

PAMPLONA, V. F. **Prevayler: Serialização e Snapshot**. Fevereiro 2005. Disponível em: <http://www.javafree.org/content/view.jf?idContent=15>. Acesso em: 10 jul. 2006.

PREVAYLERTEAM. **Prevalence skeptical FAQ.** 2004. Disponível em: <http://www.prevayler.org/wiki.jsp?topic=PrevalenceSkepticalFAQ>. Acesso em: 10 mai. 2006.

SANTOS, C. A. L. G. dos; OLIVEIRA, R. de. **Persistência de Objetos em Sistemas Prevalentes**. Dissertação (Mestrado) — UNIVERSIDADE PRESBITERIANA MACKENZIE, 2003.

SEBESTA, R. W. **Conceitos de linguagens de programação**. 5. ed. [S.l.]: Bookman, 2003.

SHALLOWAY, A.; TROTT, J. R. **Explicando Padrões de Projeto**. 1. ed. [S.l.]: Bookman, 2004.