Implementando Sistemas Orientados a Objetos para Web usando Servlets Java

Sérgio Soares, Marcelo d'Amorim, Denise Neves,
Marcelo Faro, Luciana Valadares, Gibeon Soares, Antônio Valença

CESAR (Centro de Estudos e Sistemas Avançados do Recife), Recife, Brazil

Abstract

Object-orientation adds many desirable properties in software development as modularity, reuse and maintainability. However, object-oriented programming languages for Web systems, such as Java, have heavyweight solution for user interfaces. This paper presents a solution to integrate lightweight and efficient user interfaces HTML-based with a core system using the Java Servlets technology as a communication layer. This approach uses an architecture which allows changing the communication layer to different technologies such as RMI (Remote Method Invocation), Java IDL (CORBA) or EJB (Enterprise Java Beans). As a case study, it is presented the Vestibular Interativo System, implemented following the approach presented in this work, and the problems encountered during its development.

Keywords: framework, internet solutions, software quality.

Resumo

Este trabalho apresenta uma proposta para desenvolvimento de sistema para a Web que favorecem fatores de qualidade de software. A abordagem de construção de aplicações utilizando o estilo CGI tende a tornar o sistema mal estruturado dificultando sua extensão e manutenção.

Uma sugestão de arquitetura orientada a objetos para a implementação do núcleo do sistema é apresentada. Além disto, é definida uma estratégia de integração entre interfaces leves e eficientes implementada com a linguagem HTML e Javascript, com um sistema orientado a objetos, onde a comunicação entre estas partes é feita utilizando-se Servlets Java.

Um sistema implementado segundo a solução proposta, o Vestibular Interativo, é utilizado como estudo de caso, onde são discutidos problemas encontrados durante a implementação, soluções para estes problemas e estudos que estão sendo realizados para melhorar tais soluções.

1 Introdução

Orientação a objetos proporciona uma transição mais suave entre as fases de requisitos, análise, projeto e implementação, uma vez que a adoção desse paradigma favorece o reuso de software, onde classes podem ser diretamente reusadas ou estendidas em outros projetos, aproveitando esforço gasto em projeto anteriores; além de contribuir para o aumento de modularidade do sistema e conseqüente melhoria da manutenção.

De um modo geral sistemas para a *Web* são distribuídos; desta forma, o uso de orientação a objetos é uma escolha natural uma vez que objetos são naturalmente distribuídos e podem ser executados em paralelo. Ao falar de sistemas para a *Web* e de orientação a objetos, não se pode deixar de mencionar a linguagem de programação Java [1], por ser uma linguagem orientada a objetos e ter sido projetada para o desenvolvimento de programas na Internet.

O uso de *Applets* Java[1] para implementar sistemas na *Web* pode não ser uma escolha viável. Em geral, este tipo de aplicação para ser executada na Internet deve conter, além da interface com o usuário (o *applet*), outras classes do sistema, tornando o tráfego na rede demasiadamente lento devido à necessidade da transferência destes arquivos.

Uma abordagem para desenvolver sistemas relativamente "leves" para a Internet é utilizar uma arquitetura cliente-servidor onde a interface com o usuário é implementada usando páginas HTML [2] juntamente com Javascript[2] [3], linguagem responsável pelas validações realizadas na interface com o usuário. Uma solução no estilo CGI[3] [4] seria responsável pelo sistema do lado do servidor, onde ficariam centralizados os dados. Entretanto, esta abordagem não estimula a adoção de uma solução orientada a objetos, uma vez que a unidade servidora recebe um pedido, processa e dá uma resposta, levando, desta forma, a uma aplicação mal estruturada.

Tais unidades implementam o serviço solicitado em um único bloco monolítico de código, resultando assim, em duplicação de operações complexas na elaboração de uma funcionalidade do sistema pois não há compartilhamento dos serviços.

Este trabalho propõe uma solução que integra o paradigma de orientação a objetos utilizando a linguagem Java com o uso de HTML, JavaScript e Servlets Java[4] [5] objetivando a obtenção de um sistema bem estruturado, reusável, extensível e manutenível e interfaces mais leves e eficientes com o usuário.

Na Seção 2 é feita uma breve explanação sobre a tecnologia de *Servlets*, em seguida, na Seção 3, a arquitetura orientada a objetos utilizada é descrita. Na Seção 4 é apresentada a solução proposta que integra a necessidade por clientes "leves" e sistemas orientados a objetos estruturados, seguida do estudo de caso de um sistema implementado segundo a solução proposta. Finalmente, na Seção 6, a conclusão e trabalhos futuros são discutidos.

2 Servlets Java

Nesta seção é feita uma breve introdução sobre a tecnologia de Servlets Java. Servlets Java trazem o poder das aplicações Java para o lado do servidor permitindo que suas aplicações servidoras se beneficiem de todas as vantagens oferecidas pela linguagem Java, como independência de plataforma, gerenciamento de memória, suporte automático a *multithreading*, entre outros.

Servlets são componentes escritos em Java, independentes de protocolo e plataforma, utilizados no servidor, que provêem um framework geral para serviços baseados no paradigma de requisição-resposta. Estes componentes são flexíveis suficiente para suportar serviços padronizados, como servir páginas estáticas ou dinâmicas através dos protocolos HTTP ou HTTPS, serviços proxy, etc.

---

[1] Programa Java designado para executar em web *browsers* possibilitando seu uso na Internet.

[2] Linguagem de programação interpretada geralmente embutida em páginas HTML.

[3] Aplicação que reside no lado servidor que recebe pedidos e gera respostas.

[4] Alternativa Java para programas que executam em webservers (padrão CGI).

Apesar dos servlets serem escritos em Java, seus clientes podem ser escritos em outras linguagens. Quando utilizados em camadas intermediárias (*middle* tiers) de sistemas distribuídos, eles podem ser clientes de outros serviços escritos em outra linguagem, como por exemplo, servlets podem utilizar a JDBC (*Java Database Connectivity*) [6] para acessar bancos de dados relacionais, ou serviços de sistemas legados.

Um servlet é executado num servidor de páginas web, assim como scripts CGI, a diferença é que ao invés de criar um novo processo para cada requisição de serviço, todos as requisições aos servlets são atendidas sob o mesmo processo o qual cria diferentes *threads* [7], uma para cada requisição. Desta forma, o processamento desnecessário para criação de novos processos é poupado, reduzindo o tempo para atendimento aos serviços. Uma outra vantagem do uso de *threads*, e não processos, é a possibilidade de compartilhar dados persistentes entre diferentes requisições. Assim, Servlets Java são superiores em termos de performance e estabilidade em relação a scripts CGI, e oferecem vantagens sobre soluções proprietárias, como independência de plataforma e extensibilidade.

Ao longo deste documento nos referiremos à infra-estrutura necessária para a execução de Servlets Java como ambiente de execução ou apenas ambiente de servlets. Esta infra-estrutura apresenta-se na forma de web *servers* ou *application servers*[5] compatíveis com a especificação de Servlets.

Uma vez que servlets podem fazer uso de todas as APIs de Java, eles podem ser facilmente integrados com outros tipos de componentes. Por exemplo, a maioria dos SGBDs podem ser acessados via JDBC, serviços de diretório através de JNDI [8], e sistemas distribuídos avançados através de RMI [9], Java IDL [10] ou EJB [11].

[5] Produtos que vêm ocupando atualmente uma posição de destaque no mercado por dar suporte a uma série de necessidades enfrentadas no desenvolvimento de aplicações corporativas.

3 Uma Arquitetura Orientada a Objetos

Esta seção apresenta um modelo para estruturar sistemas orientados a objetos, definido em [12]. Este modelo é baseado em camadas divididas em: acesso ao sistema, regras de negócio, persistência e objetos básicos. A figura a seguir mostra como estas camadas estão dispostas entre si:

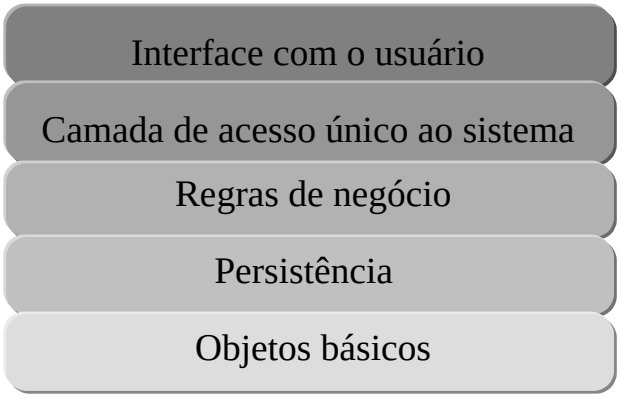

Figura 1: Modelo em camadas orientado a objetos

Na Figura 1, a camada de interface com o usuário está generalizada para dar a idéia de localização das demais camadas em relação ao sistema, na Seção 4 a mesma será especificada. Chamamos de núcleo do sistema desde a camada de objetos básicos até a camada de acesso único ao sistema. Abaixo são descritas as classes presentes em cada camada do núcleo do sistema orientado a objetos:

- Na camada de objetos básicos ficam as classes Básicas de Negócio que representam os objetos básicos do sistema, como por exemplo prova, aluno, e escola;
- Na camada de persistência ficam as classes Coleção de Dados que são responsáveis pelo armazenamento e recuperação de instâncias das classes Básicas de Negócio;
- Na camada de regras de negócio ficam as classes Coleção de Negócio que além de serem responsáveis pelo armazenamento e recuperação das instâncias das classes Básicas de Negócio, encapsulam as verificações e validações segundo as regras e políticas da aplicação em questão. Um exemplo de regra de negócio consiste em verificar se uma determinada instância já está cadastrada antes de cadastrá-la. As classes Coleção de Negócio utilizam os serviços das classes de Coleção de Dados.
- Na camada de acesso único ao sistema fica a classe Fachada de Negócio (padrão de projeto *Facade* definido em [13]) que provê uma interface unificada para todos os serviços do sistema. Esta classe deve representar os serviços oferecidos pelo sistema e deve ser o único ponto de acesso ao mesmo. Em geral estas classes centralizam todas as instâncias das classes de Coleção de Negócio.

Adaptando a Figura 1 segundo a nomeação dada acima pode-se apresentar a seguinte figura:

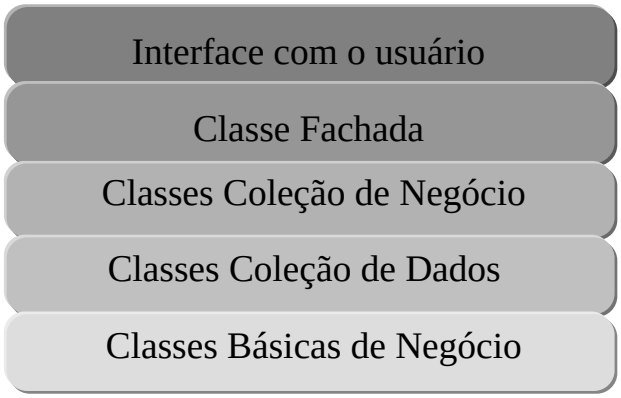

Figura 2: Organização das classe do sistema por camadas

Na camada de persistência (classes Coleção de Dados) devem ser definidas interfaces que abstraem o meio em que os dados (objetos) estão sendo armazenados. Por exemplo, a classe `RepositorioProva` deve ser uma interface com os métodos cadastrar, remover, consultar, atualizar, entre outros. A partir desta interface é que são implementadas classes que especificam em que meio ou estrutura de dados os objetos estão armazenados. Exemplos destas classes seriam a classe `RepositorioProvasArray`, `RepositorioProvasLista` ou ainda `RepositorioProvasBD`. O sistema como um todo deve trabalhar com variáveis do tipo da interface, a qual será instanciada conforme a implementação desejada, tornando o mesmo independente do meio de armazenamento.

Uma abordagem interessante é definir na camada de acesso único ao sistema diferentes interfaces que seriam implementadas pela classe Fachada. Desta forma, diferentes clientes podem ter diferentes visões dos serviços do sistema (fachada), onde, por exemplo, um determinado módulo do sistema trabalha com uma interface que possui apenas métodos de consulta, enquanto outro modulo trabalha com uma interface que, além dos métodos de consulta, também tem métodos de cadastro.

4 Integrando Servlets Java com uma Arquitetura Orientada a Objetos

Desenvolvedores habituados a desenvolver aplicações Web utilizando alguma linguagem de *script* CGI são tentados a escrever servlets utilizando este mesmo estilo. A mudança de paradigma promovida pela adoção de Java é um passo considerável para melhoria da qualidade do código pelas características da linguagem [21] e os recursos que ela oferece; entretanto, é indispensável analisar e projetar o sistema de forma sistematizada, pois, apenas a utilização da linguagem não garante os benefícios almejados.

A última seção apresentou uma proposta de organização em camadas com o objetivo de se promover a qualidade no projeto de software. A Seção 2 apresentou o *framework* de desenvolvimento Servlets Java. Será apresentada nesta seção uma estratégia de integração dos conceitos definidos nas duas últimas, permitindo que o sistema desenvolvido esteja fracamente acoplado com a tecnologia de Servlets e estes, por sua vez, estarão desempenhando um papel de canal de informação entre a interface e o sistema.

O alto grau de desacoplamento entre a fachada do sistema e a tecnologia de servlets permite adaptar o núcleo do sistema de forma direta em outras soluções como, por exemplo, a utilização de uma aplicação cliente escrita em Java comunicando-se com um servidor RMI,

representando a fachada do sistema. Um outro exemplo seria a utilização de um *front-end* desenvolvido em uma outra linguagem integrando-se com a fachada do sistema através de uma interface IDL – *Interface Definition Language* de CORBA. Independente dessas vantagens apresentadas, a maior delas refere-se a independência natural entre o código do sistema e o seu *front-end*, representado pelos componentes visuais de uma aplicação cliente ou pela construção de uma página HTML. A Figura 3 ilustra esta arquitetura de implementação.

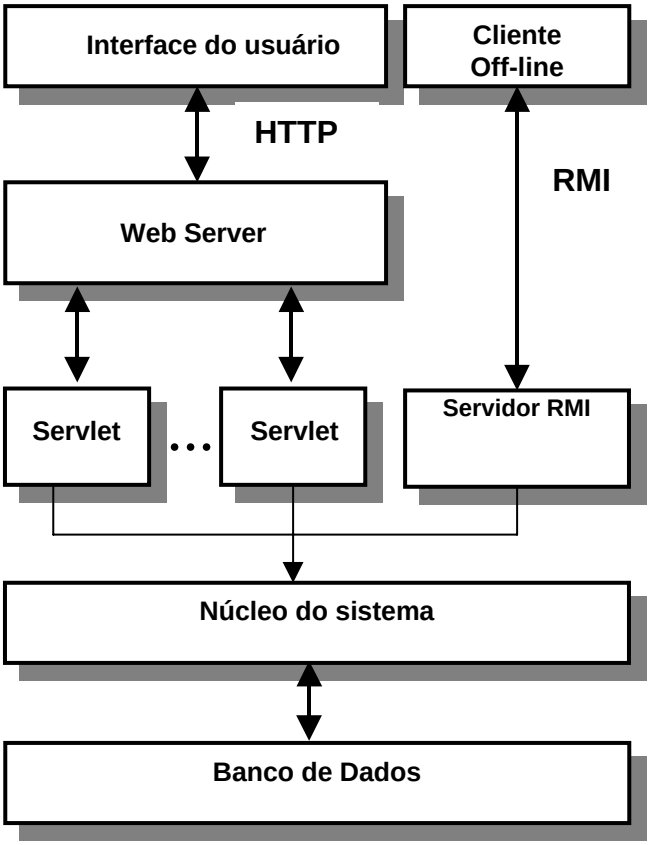

Figura 3: Arquitetura de Sistemas para a Web

A proposta de Servlets Java é de habilitar a criação de páginas dinâmicas HTML utilizando tecnologia Java e, assim deve ser; portanto, apenas a construção de páginas é desenvolvida no código de tais componentes. A elaboração das regras de negócio é função do sistema representado, em última análise, pela sua fachada. Os servlets, por sua vez, precisam comunicar-se com o sistema e dele obter e fornecer informações sem comprometer a proposta da arquitetura.

Com o objetivo de se ter maior controle de acesso ao sistema, apenas uma instância deste é mantida no ambiente de execução dos servlets; isto é garantido pelo uso do padrão de projeto *Singleton* definido em [13], o qual é ilustrado em um trecho de código Java abaixo. Algumas adaptações podem ser feitas ao código diante de situações específicas de alguns projetos como, por exemplo, declarar um atributo de classe representando o número de referências mantidas à instância. Esse valor é necessário caso queira-se verificar a existência de usuários acessando o sistema antes de retirá-lo do ar, por exemplo.

```
public class ExemploSingleton {

    // instância privada administrada pela classe
    private static ExemploSingleton instancia;
    // ...

    // construtor privado da classe ExemploSingleton
    private ExemploSingleton() {
    // inicializações necessárias
    }

    // método estático para recuperar a instância da classe
    public static ExemploSingleton getInstancia() {
        if (instancia == null) {
            instancia = new ExemploSingleton();
        }
        return instancia;
    }
    // ...
}
```

A criação de objetos da classe `ExemploSingleton` é controlada internamente, pois seu método construtor é privado, desta forma, só é possível invocá-lo dentro da definição da classe. É responsabilidade do método público e estático `getInstancia()` retornar a instância única desta classe; caso ainda não tenha sido criada, cria-se em teste de condição realizado no início do método. Esse método deve ser estático, pois, só assim, será possível recuperar uma instância da classe `ExemploSingleton`, da mesma forma, a variável `instancia` é declarada como atributo de classe (estático) pois não está associada ao estado do objeto.

Admitindo-se, neste momento, que a instância da fachada é acessível aos servlets, estes não devem implementar novas funcionalidades não oferecidas pelo sistema. Caso os serviços oferecidos pela fachada não sejam suficientes para implementar uma determinada operação, é uma indicação de que a interface da fachada ainda não está suficientemente estável, precisando assim, ser revisada, corrigida ou modificada para torná-la mais genérica e reutilizável diante de novas situações de uso.

Com os dados acessíveis, resta ao servlet construir a página HTML e retorná-la ao solicitante. É uma boa decisão de projeto separar o conteúdo da forma de apresentação dos dados; para isto, cada página retornada é construída a partir de um modelo onde, além das *tags* padrão de HTML, são inseridas outras representando conceitos do sistema. No momento da elaboração da páginas, as *tags* são avaliadas e substituídas na página conforme é ilustrado na Figura 4. Esta abordagem possui similaridades

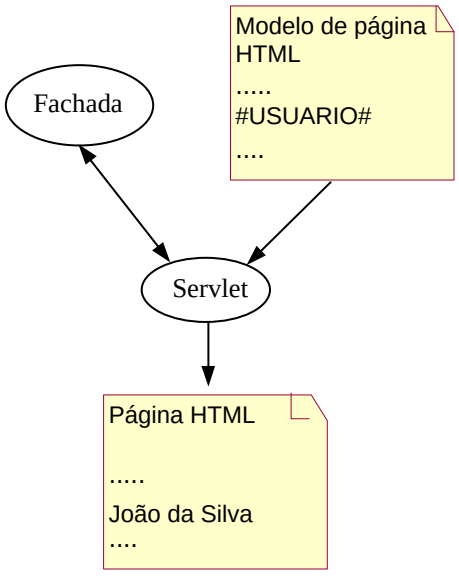

Figura 4: Diagrama ilustrando independência entre forma de apresentação e conteúdo.

com a proposta de JSP [20], tecnologia recém divulgada pela Sun Microsystems, onde o conteúdo dinâmico é avaliado no momento da solicitação da página através de interpretação de código Java. Diferenciando-se porém, pelo fato do código Java estar embutido nos servlets e não nas páginas HTML.

Apresentada na Seção 3, a Figura 1 pode ser expandida para representar duas outras camadas: a de comunicação e a de apresentação, ilustradas na Figura 5. A camada de comunicação trata da infra-estrutura necessária para distribuir e comunicar os diversos componentes do sistema, sendo representada pelos servlets e a camada de apresentação, como seu nome diz, é responsável pela criação, adaptação e disposição dos objetos visuais com os quais será feita a interação com o usuário da aplicação. Apesar desta seção utilizar Servlets Java para implementar a camada de comunicação, nada impede que seja utilizada uma outra tecnologia para tal; assim, a proposta de existência das duas camadas apresentadas são aplicáveis para outras soluções como RMI ou CORBA, por exemplo.

Figura 5: Modelo em camadas estendido

5 Estudo de Caso: Sistema Vestibular Interativo

Esta seção descreve um sistema implementado segundo a solução proposta. Aqui são dados detalhes de como padrões apresentados na seção anterior foram implementados. Por fim, os problemas encontrados são identificados e soluções para estes problemas são discutidas.

Visão Geral

O Vestibular Interativo (VI) (disponível em http://www.vestibularinterativo.com.br) é um sistema que visa o aperfeiçoamento do candidato a estudos universitários. Para isso, o VI disponibiliza, através da Web, o suporte a atividades de ensino, aprendizagem e avaliação, permitindo o acesso on-line a provas, sistemas de tutoramento e acompanhamento de desempenho. O Vestibular Interativo tem como objetivo futuro o desenvolvimento de plataformas e metodologias genericamente aplicáveis de registro e avaliação de indivíduos em processos de recrutamento ou titulação.

O sistema atende a três principais grupos de interesse envolvidos em concursos vestibulares: alunos em preparação para o exame; professores da rede de ensino médio; escolas de ensino médio e cursos preparatórios para exames vestibulares.

Abaixo estão listadas as principais funcionalidades implementadas no sistema:

- cadastro de alunos, professores e escolas;
- resolução de provas on-line para alunos, com medição de tempo;
- avaliação de desempenho de alunos;
- avaliação de desempenho de escolas, por disciplina e por assunto (dentro de uma disciplina);
- estatísticas comparativas de desempenho de alunos e/ou escolas em relação a vestibulares reais e ao próprio Vestibular Interativo;

Algumas funcionalidades em desenvolvimento incluem:

- criação de novas provas (através de um banco de questões) com base em critérios fornecidos pelo usuário, tais como assunto e grau de dificuldade;
- resolução de questões com visualização e manipulação dinâmica de dados (uso de multimídia);
- sala virtual para debates com professores e especialistas em conteúdos específicos.

Como o Vestibular Interativo tem como requisito a construção de uma aplicação para que o aluno possa resolver uma prova off-line [14], a arquitetura de desenvolvimento adotada facilita a implementação deste requisito, uma vez que o núcleo do mesmo pode ser reusado em uma aplicação Java que pode ser instalada na máquina do usuário e através de RMI se comunicar com o núcleo do sistema.

Números do Sistema

O Sistema Vestibular Interativo foi implantado em agosto de 1999. No final de janeiro de 2000 o sistema contava com 4.741 alunos, 1.849 professores e 2.266 escolas cadastradas. O VI já teve cerca de 4.500 acessos num único dia. De setembro a dezembro de 1999 o sistema teve mais de 47.600 acessos. Estes números auxiliam a avaliação da solução adotada no tocante a usuários simultâneos e à robustez do sistema.

Implementação e Problemas Encontrados

O Vestibular Interativo é uma aplicação desenvolvida para a Web e segue a solução proposta neste artigo. As linguagens HTML e Javascript foram utilizadas para a construção da interface com o usuário, definindo formulários, validações de preenchimento, máscaras de edição, etc. Servlets Java respondem às solicitações do usuário através dessa interface, comunicam-se com o núcleo do sistema e, por fim, elaboram uma resposta para o usuário.

Para se construir servlets sobre o protocolo HTTP, utilizado na Web, é necessário estender a classe HttpServlet. Seus principais métodos são o `doGet()`, `doPost()` e o `init()`. Os dois primeiros são responsáveis por receber uma solicitação do usuário e monta uma página HTML como resultado. Esse métodos, apesar de possuírem o mesmo objetivo, trabalham de forma ligeiramente diferente, fugindo do escopo deste artigo uma explicação mais detalhada. A inicialização de um servlet é feita através de seu método `init()` de forma transparente para o desenvolvedor. Para uma explicação mais detalhada sobre Servlets Java, consulte [5].

A classe `VestibularInterativo`, representando a fachada do sistema, implementa o padrão de projeto *Singleton*, conforme a classe `ExemploSingleton` apresentada na seção anterior. Todos os servlets que necessitam de algum serviço do sistema inicializam uma variável do tipo da fachada conforme ilustrado abaixo:

```
public class ServletExemplo extends HttpServlet {

    private VestibularInterativo sistema;

    public void init(ServletConfig config) throws ServletException {
        sistema = VestibularInterativo.getInstancia();
    }
    // ...
}
```

Durante a fase de testes, o sistema apresentou instabilidade causada no acesso concorrente às conexões do driver JDBC. Até onde foi avaliado, o driver não suportou o acesso simultâneo para uma mesma conexão com o banco de dados e não portou-se de forma robusta diante do problema, sendo necessário esforço considerável para apontar o problema com precisão. Diante deste problema, o controle de concorrência foi feito utilizando-se os mecanismos de Java para contorná-lo [7], assim evitava-se a disputa da conexão na própria aplicação.

É necessário melhorar a arquitetura proposta para que alguns problemas detectados em fases adiantadas sejam solucionados, como por exemplo a falta de um controle de sessão do usuário nas aplicações. A implementação atual do sistema não possui estado tornando necessário que a informações do usuário corrente sejam armazenadas na página HTML, funcionando como identificadores para as próximas interações com o sistema. A implantação de controle de sessões visa melhorar o desempenho do sistema, uma vez que os dados mais freqüentemente utilizados são mantidos na sessão do usuário.

6 Conclusão e Trabalhos Futuros

Neste trabalho é proposta a integração de orientação a objetos com HTML, Javascript e Servlets Java. Esta integração deve auxiliar a implementação de um sistema com interfaces leves e eficientes que utilizam Servlets Java para se comunicar com o núcleo do sistema, orientado a objetos.

Uma das vantagens de utilizar a linguagem Java tanto na aplicação servidora (servlets) quando no núcleo do sistema é a possibilidade de uso de uma infinidade de APIs, como JDBC, RMI e Java IDL por exemplo. Além disso, pelo fato da linguagem Java estar em crescente utilização, o uso da mesma na implementação do núcleo do sistema favorece o reuso do sistema em futuros projetos que venham a ser implementados nesta linguagem. A simples utilização de uma linguagem orientada a objetos permite o uso de conceitos importantes que contribuem para a melhoria melhoria da qualidade do software (manutenção, reuso, etc.).

Apesar dos problemas que surgiram estarem solucionados, meios mais eficientes de resolvê-los são ainda matéria de estudos. Um exemplo é o problema de acessos concorrentes a conexões com o banco de dados que foi resolvido com o uso dos mecanismos de controle de concorrência disponíveis na linguagem Java [7]. Uma possibilidade de melhoria para este problema seria a adoção de um gerenciador de

transações e de um *pool* de conexões [15], onde há um controle sobre as conexões e as transações realizadas com o banco de dados existentes de forma transparente para o desenvolvedor.

A solução proposta neste artigo permite que o sistema seja desenvolvido usando de forma plena todas as vantagens do paradigma de orientação a objetos com o requisito de construção de interfaces leves, tornando assim viável a construção de sistemas orientados a objetos para a Web utilizando Java.

Agradecimentos

Agradecemos a equipe Vestibular Interativo do Centro de Estudos e Sistemas Avançados do Recife (CESAR). O CESAR[6] foi concebido para suprir a necessidade de uma maior interação entre o meio acadêmico, o meio empresarial e a sociedade em geral. A instituição posiciona-se como uma entidade autônoma, ágil e flexível, que conhece e apoia a pesquisa e atende, simultaneamente, às necessidades do mercado.

7 Referências

[1] James Gosling, Bill Joy, and Guy Steele. The Java Language Specification. Addison-Wesley, 1996.
[2] Ian S. Graham. The HTML Sourcebook. Wiley Computer Publishing, second edition, 1996.
[3] David Flanagan. JavaScript The Definitive Guide. O'Reilly & Associates, Inc., second edition, 1997.
[4] Shishir Gundavaram. CGI Programming on the World Wide Web. O'Reilly & Associates, Inc., first edition, 1996.
[5] Jason Hunter and Willian Crawford. Java Servlet Programming. O'Reilly & Associates, Inc., first edition, 1998.
[6] JDBC Page. http://java.sun.com/jdbc/
[7] Doug Lea. Concurrent Programming in Java. Addison-Wesley, 1997.
[8] JNDI Page. http://java.sun.com/products/jndi/
[9] RMI Page. http://java.sun.com/products/jdk/rmi/
[10] Java IDL Page. http://java.sun.com/products/jdk/idl/
[11] EJB Page. http://java.sun.com/products/ejb/
[12] Euricélia Viana and Paulo Borba. Integrando Java com Bancos de Dados Relacionais. III Simpósio Brasileiro de Linguagens de Programação, 1999.
[13] Erich Gamma et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
[14] Sistema Vestibular Interativo. Relatório Técnico, CESAR, Março 1999.
[15] Hans Bergsten. Improved Performance with a Connection Pool. http://www.webdeve7lopersjournal.com/columns/connection_pool.html
[16] Hans Bergsten. An Introduction to Java Servlets. http://webdevelopersjournal.com/articles/intro_to_servlets.html

[6] http://www.cesar.org.br

[17] Hans Bergsten. Servlets Are for Real! http://webdevelopersjournal.com/columns/java_servlets.html
[18] The Java Servlet API White Paper http://java.sun.com/products/servlet/whitepaper.html
[19] JDK Documentation Page. http://java.sun.com/products/jdk/1.1/docs/
[20] JSP Page. http://www.javasoft.com/products/jsp/.
[21] Gosling J. et al, The Java Language Environment – A White Paper, 1996.