Instituto de Matemática e Estatística
Universidade de São Paulo

MAC499 – Trabalho de Formatura
Projeto Panda Reloaded

Integrante	**NUSP**	**E-mail**
Gilmar Gimenes Rodrigues	2869156	rgil@linux.ime.usp.br
João Carlos Matsuzaka Costa	2234662	jcosta@linux.ime.usp.br
Paulo Eduardo Azevedo Silveira	2369541	peas@linux.ime.usp.br

Orientador Carlos Hitoshi Morimoto

1. Introdução

Uma faculdade como a Poli tem cerca de 800 alunos de primeiro ano. Cada aluno deve imprimir e entregar seus trabalhos aos professores para que possam ser corrigidos. Supondo que trabalhos de alunos de primeiro ano sejam relativamente pequenos, teremos uma média de 2 a 3 páginas por trabalho, ou seja, cerca de 2000 páginas que após serem corrigidas serão descartadas tornando-se lixo. Juntamente com os impressos serão entregues cerca de 800 disquetes com cópias dos trabalhos. Como todos sabem, disquetes são facilmente danificados, causando na maior parte das vezes a perda de seu conteúdo e obrigando o aluno a fazer outra cópia e entregá-la novamente ao professor.

Os alunos são normalmente divididos em várias turmas, com professores diferentes que, por algum motivo acabam por passar trabalhos diferentes para seus alunos. Isso gera o problema de onde cada turma deve entregar seus trabalho. Alguns são entregues em uma urna, outros na sala do professor ou para o monitor e talvez alguém aceite também por correio. O que acaba causando uma tremenda confusão e às vezes trabalhos acabam sendo perdidos.

Estes e muitos outros problemas fizeram surgir a idéia do sistema Panda.

O Panda foi criado para que os alunos pudessem entregar seus trabalhos sem ter nenhuma das complicações descritas. Eles simplesmente sentariam no computador, fariam o trabalho e o entregariam via internet aos seus respectivos professores. Sem complicações, sem desperdícios. Os trabalhos podem ser sincronizados. Assim suas datas de entrega e assunto podem ser os mesmos.

O Panda quer organizar a entrega dos trabalhos e centralizá-la em um só lugar, assim como disponibilizar as notas e permitir a correção dos mesmos diretamente no sistema, dessa forma criando uma ferramenta para acompanhar o aluno durante a sua graduação e facilitar a interação aluno-professor.

Este documento trata do aspecto técnico do sistema, desde arquitetura até detalhes da implementação.

2. Os Pandas

Não temos muitas informações sobre o que era o Panda antes da versão de 2001. Apenas sabemos que havia uma ferramenta utilizada para a entrega de trabalhos via internet. O administrador cadastrava os professores e suas respectivas matérias no sistema e qualquer pessoa, via internet, poderia se cadastrar e enviar um arquivo para qualquer matéria de qualquer professor. Ou seja, era um sistema muito simples e sem segurança alguma, porém com um enorme potencial.

A pedido dos professores resolveu-se criar um novo sistema de entrega de trabalhos. A nova versão incluiria funcionalidades como uma página na Internet com as informações de cada matéria para os alunos, um sistema de administração mais completo, sistema de correção de trabalhos via Internet, uma lista de discussão para cada matéria, *login* e *senha* para cada aluno fornecidos pelo professor da matéria aumentando dessa forma o nível de segurança e outras funcionalidades menores. Foi assim que surgiu o Panda.

Todo o sistema foi construído para ser usado e administrado via Internet. Um administrador cria contas paras os professores e cada professor cria contas para seus alunos.

Foi nessa época que o Panda começou a ficar famoso. Os professores começaram a dar mais atenção ao sistema e utilizá-lo mais. A divulgação do sistema motivou outros professores a utilizá-lo também, mostrando que o Panda é um bom instrumento para o acompanhamento didático dos alunos. Surgiu então a idéia de introduzir mais funcionalidades no panda para que um dia ele pudesse substituir o webct da usp. Nascia aí o Panda 2.

O Panda estava muito bom. Os professores e alunos elogiaram e fizeram muitas críticas construtivas. Muitas idéias surgiram e queríamos colocar todas no panda, mas isso se tornou um problema. O panda tinha sido todo escrito em PHP. Apesar de ser uma boa linguagem e ter recursos poderosos, o PHP tem um problema. O código escrito, por melhor que seja, acaba se tornando muito bagunçado, pois não incorpora técnicas de orientação a objetos. Dessa forma, implementar mais funcionalidades ao panda seria um esforço que não compensaria o resultado. Resolvemos que o melhor a fazer seria reescrever o Panda utilizando uma linguagem mais flexível e mais poderosa, a nossa escolha foi o Java.

Descreveremos a seguir tudo deste novo Panda. Tudo sobre as tecnologias e conceitos utilizados e todo nosso esforço para fazer um sistema melhor e que futuramente possa ser facilmente aprimorado.

3. Metodologia e ferramentas

Desenvolvimento em equipe pode trazer muitos problemas quando as coisas não são bem organizadas. A experiência com o primeiro Panda nos ensinou muito. Tivemos o cuidado de manter uma boa comunicação para não haver falta de sincronismo entre nós, de forma a manter um bom trabalho em equipe.

Para isto, utilizamos o CVS, que é uma ferramenta para o desenvolvimento concorrente e em grupo. Através dele, nós dividíamos as tarefas do dia por mensagens na internet, e depois cada um fazia a sua parte, sempre marcando as mudanças no repositório o mais rápido possível a fim de evitar os conflitos.

Escolhemos a linguagem java, pois queríamos algo muito robusto, orientado a objetos, e que pudesse ser reutilizado por outras pessoas. Alcançamos estes objetivos quando dividimos etapas do desenvolvimento em subprojetos. Inclusive usamos estas ferramentas para o desenvolvimento de outros sistemas.

A ferramenta para programação (IDE) utilizada foi o Eclipse (www.eclipse.org), que além de ser open source, conta com um grande comitê de empresas, tais como Oracle e IBM, que patrocinam seu desenvolvimento, fazendo dela uma ferramenta muito flexível. O eclipse tem tantas características úteis que não poderíamos enumerar todas. Um exemplo é a integração ao cvs, poupando o tempo de usá-lo através de linha de comando.

Queríamos também deixar o novo panda o mais modularizado possível, podendo incluir novos subsistemas sem muitos agravantes e modificações. Para isso, adotamos a arquitetura *Model View Controller (MVC)*, onde existem 3 camadas muito distintas:

Model: trata da comunicação da aplicação com os modelos de dados, que pode ser um banco de dados, arquivos binários, ou qualquer outro tipo de repositório. No caso do panda, o modelo é um banco de dados, e esta camada Model é gerada pelo subprojeto Gerbo, que mantém um mapeamento objeto relacional, para que o SQL fique de fora do código java, gerando independência de banco de dados e orientação a objetos.

View: camada que interage com o usuário, recebe informações e mostra as informações. Para que essa camada possa ser facilmente substituída, ela não deve

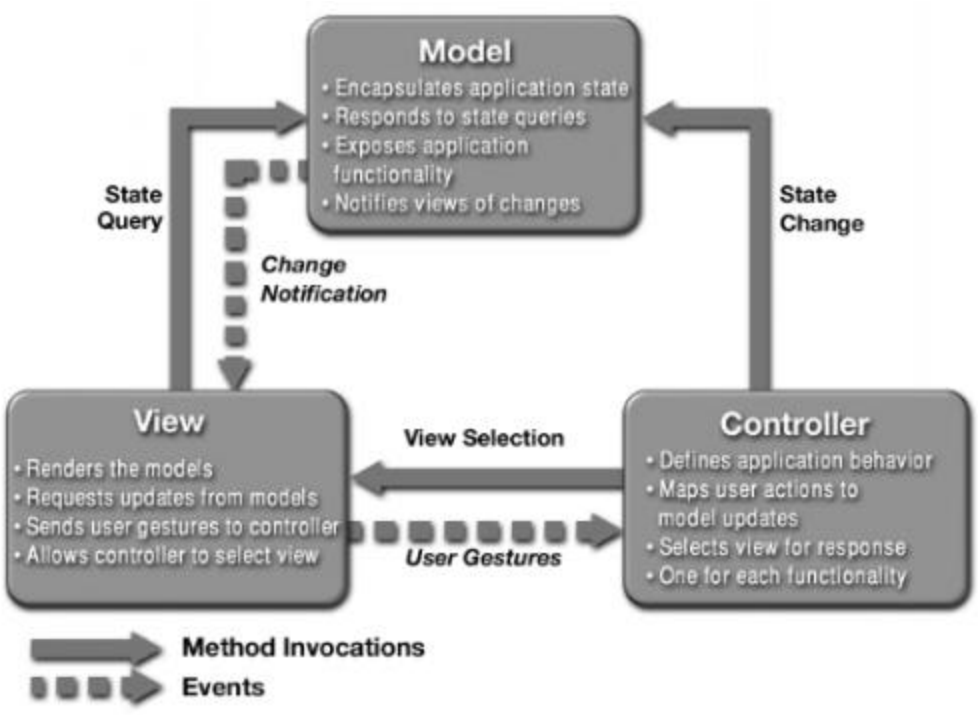
interferir na lógica do sistema (*business logic*). Para esta camada utilizamos o projeto open source Velocity, do grupo Jakarta.

Controller: camada responsável por delegar a chamada para as ações correspondentes. O projeto aqui utilizado é o Mamute, também criado por nós.

Com essa metodologia MVC, o sistema pode ser feito aos poucos, de maneira incremental, já que um módulo não interfere no outro. Com isto, podemos focar cada período de desenvolvimento em uma das funcionalidades do Panda, sem nos preocuparmos com as integração desta com as demais. Claro que algumas funcionalidades interferem uma nas outras, porém este número diminui significativamente com o MVC.

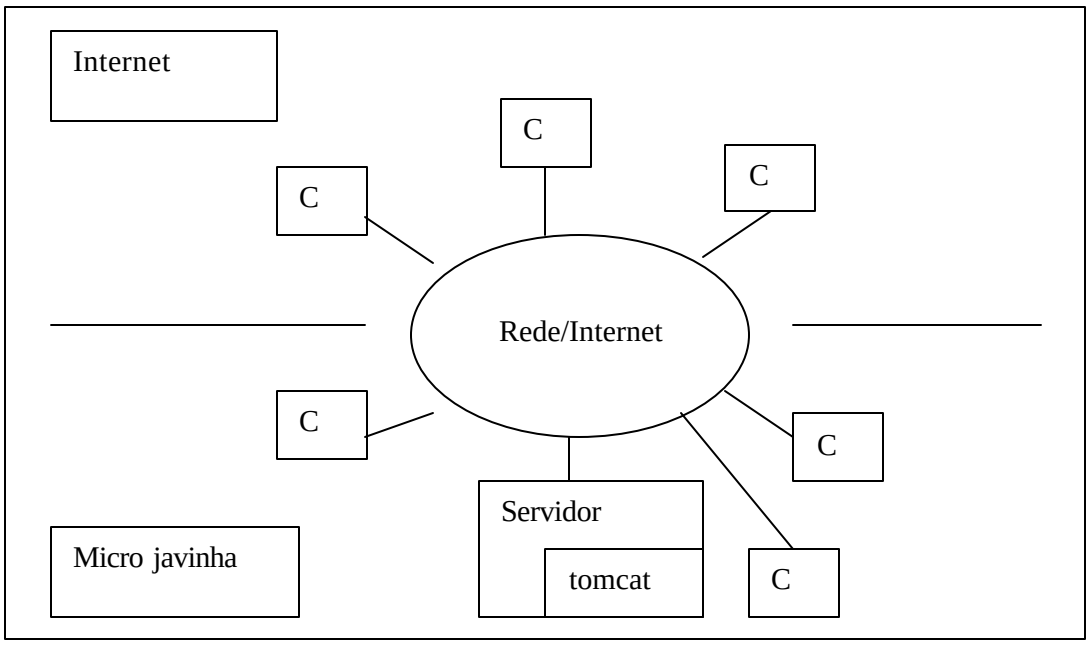
O Panda anterior misturava as 3 camadas: você encontrava código em PHP, SQL e HTML dentro do mesmo arquivo, gerando altíssimo acoplamento, o que dificulta mudanças no sistema. Isso também dificulta qualquer inclusão de novos subsistemas, já que o reaproveitamento de código é nulo.

Segue um esquema geral do MVC, que a própria SUN usa para a divulgação de suas boas práticas:



4. Projeto Arquitetônico do Sistema

A arquitetura a ser utilizada é baseada em um *thin client*, que pode ser qualquer browser que suporte HTML 4.0. Toda a *business logic* fica no servidor, e só ele tem acesso ao banco de dados. Apenas a camada de visualização (interface) fica exposta e em contato direto com os usuários:



legenda:

S – servidor, pode estar encapsulando um *cluster* de banco de dados, transparente para o sistema (clientes). Nossa implementação é Java puro, então independe de plataforma. E melhor ainda, a solução de mapeamento objeto relacional que utilizamos cria a independência de "vendor" do banco de dados.

C –Cliente, usuário do sistema.

Como o modelo adotado é de *Thin Clients*, tudo pode ser considerado como um único tipo de cliente, deixando o processamento para o servidor, dependendo da autenticação (isto é, se o cliente é aluno, professor, monitor, coordenador, etc..)..

Uma das maiores vantagens deste modelo, é que poderíamos aumentar a quantidade de clientes para este produto. Um PDA ou celular pode facilmente usar o sistema, se tiver suporte a browser.

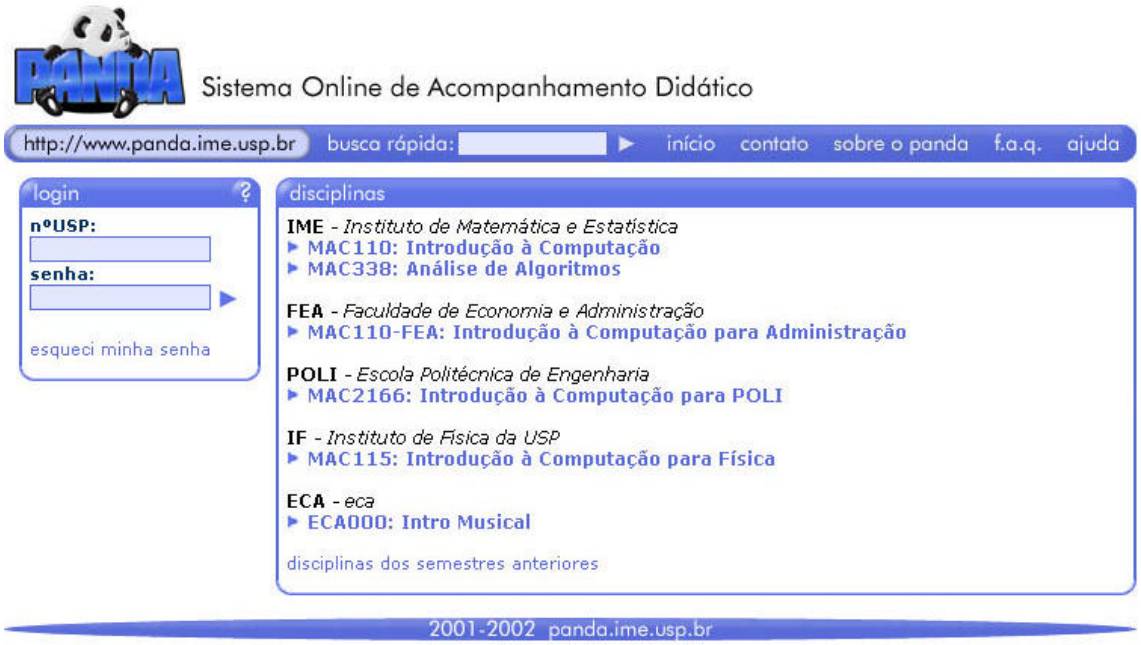
5. Projeto da Interface

Uma das maiores dificuldades que a versão anterior do Panda enfrentou foi a dificuldade de navegação, em especial na parte dos professores, que possuía uma quantidade muito maior de links e opções.

Outra questão importante no desenho das interfaces, foi a reutilização de código HTML. Manter uma similaridade entre as páginas e ainda assim obter uma apresentação satisfatória requer uma atenção especial para cada página a ser desenvolvida. Para agravar, nem sempre durante o desenvolvimento do projeto consegue-se permanecer com o desenho original da interface, o que acarreta em modificações que dificultam a implementação da mesma.

Infelizmente as interfaces não ficam elegantes em todos os browsers já que a diversidade nas versões destes e a falta de padronização do modo de interpretação de HTML tornam a utilização de muitos dos recursos disponíveis difícil e por vezes impossível. O principal exemplo disso são os bugs de navegação apresentados no Netscape 4.7 (note que esse browser já esta ultrapassado), que foram resolvidos com um tratamento especial na *folha de estilos*.

Aqui podemos observar um exemplo da interface do Panda 2:


Sistema Online de Acompanhamento Didático
http://www.panda.ime.usp.br
busca rápida:
início
contato
sobre o panda
f.a.q.
ajuda
login
nºUSP:
senha:
esqueci minha senha
disciplinas
IME - Instituto de Matemática e Estatística
MAC110: Introdução à Computação
MAC338: Análise de Algoritmos
FEA - Faculdade de Economia e Administração
MAC110-FEA: Introdução à Computação para Administração
POLI - Escola Politécnica de Engenharia
MAC2166: Introdução à Computação para POLI
IF - Instituto de Física da USP
MAC115: Introdução à Computação para Física
ECA - eca
ECA000: Intro Musical
disciplinas dos semestres anteriores
2001-2002 panda.ime.usp.br

6. Banco de dados

Uma das partes mais interessantes do projeto é a independência de banco de dados. Para atingir este objetivo, criamos um subprojeto chamado Gerbo, que a partir de um arquivo XML, gera arquivos java que fazem o **mapeamento objeto relacional**. Também conhecido como camada de persistência.

O que é este mapeamento? Normalmente, quando você lida com SQL em java, você tem chamadas para a API JDBC dentro do código java, algo que deixa seu código bastante ilegível, não orientado a objetos, e "hardcoded". O mapeamento muda tudo isso. Você passa a trabalhar exclusivamente com objetos, você não tem mais SQLs pelo seu código fonte, em vez disso, eles ficam em um arquivo de configuração.

Para ilustrar, está aqui um exemplo técnico. Normalmente uma inserção no banco de dados utilizando código Java seria algo do tipo:

```
Connection c = DriverManager.getConnection(connectionString);
Statement s = c.createStatement();
s.execute("INSERT INTO usuarios ..." + "Neo");
```

Utilizando o subprojeto gerbo:

```
Usuario user = new Usuario();
user.setName("Neo");
user.save();
```

Se houver a necessidade de mudança de banco de dados, basta editar o arquivo de configuração (Database.properties) e colocar um arquivo com as SQLs apropriadas. O Panda já foi testando, com sucesso, utilizando o MySQL 3.23, MySQL 4.x e o PostGreSQL 7.2, sem alterar uma única linha de código java.

Também foram utilizados *PreparedStatements* em vez de *Statements*, o que delega o "escaping" das strings e binários ao driver de conexão. Um *ConnectionPool* (classe DatabaseBroker) foi utilizado para evitar o estresse de operações I/O com o banco de dados, já que o java sempre se utiliza de sockets para fazer a conexão ao banco. Ele faz o compartilhamento de N conexões (que pode ser configurado

através de um arquivo) e fica dependendo da thread que pegou a conexão para devolvê-la a pool, caso contrário fica dependendo do garbage collector.

7. Projeto de classes do Sistema

As principais classes, que mapeiam os objetos Java para linhas da tabela do banco de dados, foram geradas através de um XML que é processado pelo projeto Gerbo, já supracitado quando falamos sobre o banco de dados. Essas classes implementam o pattern Data Access Objects, que faz com que os objetos tenham o controle da sua própria persistência, porém sem mostrar isto diretamente ao usuário (ausência de transparência).

Estas classes estão no pacote br.usp.ime.arca.panda.model.dao.

O outro grande grupo de classes são as “ações” que o subprojeto Mamute dispara quando uma chamada ao servidor web é realizada. Um conjunto de ações forma a lógica do sistema (business logic), que tem acesso direto a camada de modelo do MVC, e envia os resultados para a camada de visualização.

Além destas existem classes de suporte: a piscina de conexões, classes para comprimir e descomprimir os trabalhos, classes de métodos auxiliares e outras menos importantes.

8. O servidor

O panda está hospedado em um athlon 1.4 Mhz com 512 Mb RAM que roda o sistema operacional Linux Debian 3.0. A escolha do S.O. não foi ao acaso, mas sim ao fato de ser o sistema mais difundido entre os alunos do IME e portanto o que mais conhecemos internamente.

O panda conta com um sistema de RAID. Dois discos rígidos de 60 Gb cada fazem parte desta máquina. Dessa forma os dados são espelhados nos dois discos rígidos e se ocorrer qualquer problema com um deles todas as informações estarão seguras e disponíveis no outro disco rígidos.

Além disso basta substituir o disco rígidos danificado por outro para que tudo volte ao normal. O Debian usa a versão 2.4 do kernel que foi compilado com suporte a RAID.

O apache é o servidor web utilizado no panda. Ele faz o redirecionamento para o Tomcat, que é o servlet container. Qualquer versão que suporte servlets 2.3 pode ser usado.

O panda usa o banco de dados mysql versão 4.0.4. Apesar de ser uma versão beta, ele apresenta uma reformulação enorme comparado com a versão 3 e muitas funcionalidades novas. Além de já estar bem estável.

O panda conta com a versão 1.4.1 do java, baixada e instalada a partir do site do java seguindo as instruções da própria Sun.

9. Boas práticas utilizadas

Estes tópicos já foram abordados anteriormente, mas gostaríamos de salientá-los novamente. E gostaríamos de deixar claro que todas essas boas práticas foram utilizadas pela necessidade de fazer um projeto escalável e que não gerasse grande interdependência. Todos estes itens foram estudados durante o projeto, sendo que não tínhamos conhecimento prévio deles. Até mesmo java foi aprendido no decorrer do ano.

- Utilização do CVS para a programação em grupo. Todo o histórico dos arquivos pode ser encontrado em http://cvs.arca.ime.usp.br/cgi-bin/viewcvs.cgi/panda/
- Classes de mapeamento objeto/relacional foram geradas a partir de um descritor XML, utilizado pelo projeto Gerbo, criado pelo Arca – IME e encontrado em http://cvs.arca.ime.usp.br/cgi-bin/viewcvs.cgi/gerbo/
- Sistema amplamente baseado na arquitetura/padrão *Model View Controller* (MVC), através da utilização da *template engine Velocity* http://jakarta.apache.org/velocity, e da controladora Mamute, criada por alunos da Arca – IME e encontrado em http://cvs.arca.ime.usp.br/cgi-bin/viewcvs.cgi/mamute/
- *Business Logic* completamente plugável com "*actions*" do Mamute.
- *Design* do *website* pode ser modificado facilmente por qualquer *webdesigner*, já que o código da *business logic* não está no implementado no meio do *design*, por causa do *Velocity* e arquitetura MVC.
- Utilização de *design patterns* como *Direct Access Objects*, *Singletons*, *Pools* e outros.
- Independência de plataforma e de banco de dados.
- SQL separado do código java, e também separado do HTML, muito diferente do que acontecia no panda anterior.
- Código fonte segue os padrões (*coding standards*) estabelecidos pela SUN: http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

10. Documentação

Como o novo panda ainda não foi testado, ainda não foi criado um manual do usuário, o qual consideramos fundamental para o projeto. Mas o projeto já conta com extensa documentação do código fonte, através de comentários dentro do código, e também através do javadoc, que pode ser encontrado em:

http://www.arca.ime.usp.br/pandadoc/

Este trabalho pode ser utilizado como documentação do sistema no geral.

11. Bibliografia para aprendizado

Durante o projeto, aprendemos muito, desde java a design patterns. Aqui estão alguns dos principais links e livros utilizados:

- Model View Controller with Java, http://www.exciton.cs.rice.edu/JavaResources/DesignPatterns/MVC.htm
- Model View Controller Java Blueprint, http://java.sun.com/blueprints/patterns/MVC.html
- Data Access Object Patter, http://java.sun.com/blueprints/patterns/DAO.html
- Tomcat user guide, http://jakarta.apache.org/tomcat/tomcat-4.0-doc/index.html
- Ant user guide, http://jakarta.apache.org/ant/manual/index.html
- Velocity user guide, http://jakarta.apache.org/velocity/user-guide.html
- Java JDBC tutorial, http://java.sun.com/docs/books/tutorial/jdbc/basics/
- Servlet tutorial, http://java.sun.com/webservices/docs/1.0/tutorial/doc/Servlets.html
- Design Patterns, Erich Gamma et al.