Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

Rodrigo Alves Keller

Estudo comparativo entre Java e Ruby considerando desenvolvimento de aplicações Web

Belo Horizonte
2008 / 1º semestre

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Curso de Bacharelado em Ciência da Computação

Estudo comparativo entre Java e Ruby considerando desenvolvimento de aplicações Web

por

Rodrigo Alves Keller

Monografia de Projeto Orientado em Computação I

Apresentado como requisito da disciplina de Projeto Orientado em Computação I do Curso de Bacharelado em Ciência da Computação da UFMG

Profa. Mariza Andrade da Silva Bigonha

Orientadora

Belo Horizonte
2008 /1º semestre

RESUMO

O principal objetivo do presente trabalho consiste em comparar o desenvolvimento de aplicações web entre dois ambientes distintos: Java e Ruby. O primeiro, Java, é um ambiente de desenvolvimento web mais consolidado, apesar de não ter sido projetado com foco nesse tipo de aplicação. O segundo, o Ruby, assim como Java, se tornou famoso devido ao seu uso em desenvolvimento web, mais especialmente por um *framework* web criado para o mesmo. Os defensores de Ruby defendem sua praticidade e extrema facilidade de aprendizagem. Ruby é uma linguagem orientada a objetos que contém aspectos dos paradigmas lógico e funcional. Das linguagens surgidas nesse novo milênio, Ruby é sem dúvida a linguagem que mais se cogita a ter fôlego para quebrar o domínio de Java entre as linguagens multiplataforma para desenvolvimento de aplicações web. Como resultado parcial deste estudo tem-se a compilação deste documento, cujo objetivo é auxiliar desenvolvedores na tomada de decisão sobre qual dos dois ambientes utilizar para desenvolvimento de aplicações web.

Palavras-chave:.
Java, Ruby, Rails, Hibernate, Struts.

LISTA DE FIGURAS

FIGURA 1 - MODELO DO BANCO 11
FIGURA 2 – "HELLO WORLD" EM JAVA 13
FIGURA 3 - "HELLO WORLD" EM RUBY 13
FIGURA 4 - CONFIGURANDO O RUBY 14
FIGURA 5 - ESTRUTURA DE DIRETÓRIOS PARA O PROJETO JAVA 15
FIGURA 6 - GERAÇÃO DE CÓDIGO NO RAILS 15

LISTA DE SIGLAS

IDE	Integrated Development Environment
JSP	Java Server Pages
CGI	Commom Gateway Interface
IDE	Integrated Development Environment

SUMÁRIO

RESUMO ..... **III**

LISTA DE FIGURAS ..... **IV**

LISTA DE SIGLAS ..... **V**

1 INTRODUÇÃO **7**

1.1 VISÃO GERAL 7
1.2 MOTIVAÇÃO 7
1.3 OBJETIVOS 7
1.4 ORGANIZAÇÃO DO TEXTO 7

2 REFERENCIAL TEÓRICO **8**

2.1 ASPECTOS GERAIS DE JAVA 8
2.2 ASPECTOS GERAIS DE RUBY 9
2.3 SISTEMA WEB 9
2.4 MODULARIZAÇÃO DE UM SISTEMA WEB 9
2.5 FRAMEWORKS 10
2.6 CONCLUSÃO 10

3 METODOLOGIA **11**

3.1 ESTUDO INICIAL DAS LINGUAGENS 12
3.2 CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO 13
3.2.1 Configurando o Java *13*
3.2.2 Configurando o Ruby *14*
3.2.3 Estudo para desenvolvimento Web *14*
3.2.4 Iniciando o desenvolvimento *15*
3.3 CONCLUSÃO 16

4 RESULTADOS E DISCUSÕES **17**

5 CONCLUSÃO E TRABALHOS FUTUROS **19**

6 BIBLIOGRAFIA CITADA **20**

1 INTRODUÇÃO

1.1 Visão Geral

De tempos em tempos, vemos novas tecnologias surgirem, com o intuito de revolucionarem o mercado em seu segmento. E no desenvolvimento *web* isso também se faz regra.

Encerramos a década de 90 com sistemas *web* desenvolvidos utilizando tecnologia *CGI* (1) e entramos na década posterior com os *servlets* (2) de *Java* (3). Com *Java*, passamos a produzir páginas dinâmicas com uma riqueza maior de funcionalidade e segurança, baseada em uma linguagem orientada a objeto e difundida pela sua portabilidade e facilidade de aprendizagem.

Com a popularização de Java, *frameworks* (4) para persistência e apresentação de dados surgiram, aumentando a produtividade dos desenvolvedores na criação de aplicações web. No entanto, aspectos como praticidade e dificuldade de manutenção são, para muitos, problemas recorrentes no desenvolvimento *web* utilizando os *frameworks Java* atuais.

David Heinemeier Hansson, não conformado com esses problemas, criou o *Ruby on Rails* (5), um *framework web* criado em *Ruby* (6) com o intuito de favorecer a praticidade e a facilidade no desenvolvimento *web*. O *Rails* popularizou o *Ruby* como linguagem de desenvolvimento *web*, competindo com *Java* nesse segmento de aplicações.

Nesse presente trabalho, vamos comparar as linguagens *Java* e *Ruby* considerando os aspectos mais relevantes do desenvolvimento de aplicações *web*.

1.2 Motivação

Atualmente, no segmento de tecnologia, a produtividade tem cada vez mais se tornado o centro das atenções, capaz de alavancar ou afundar empresas nesse ramo. Assim, desenvolvedores de *softwares* vivem um dilema: produzir *softwares* de forma rápida sem perder a qualidade. Para tanto, tecnologias ditas revolucionárias para criação de programas surgem de tempos em tempos, tecnologias essas que prometem aumentar a produtividade dos desenvolvedores, sem forçar o mesmo a escrever um código sem qualidade. Desse modo, faz se necessários estudos afins de avaliar as vantagens e desvantagens dessas novas tecnologias para sabermos de fato, o tanto que as mesmas podem agregar na produtividade do processo de desenvolvimento de *software*.

1.3 Objetivos

Comparar o desenvolvimento de aplicações *web* entre dois ambientes distintos: *Java* e *Ruby*. Para tanto, será implementado nesse trabalho parte de um sistema *web* comercial previamente especificado, modelado e projetado.

1.4 Organização do Texto

Este texto está organizado da seguinte forma: no Capítulo 2 é feita uma breve descrição de alguns conceitos importantes para o entendimento do trabalho: *aspectos gerais de Java e Ruby*, *sistema web*, *modularização de um sistema web* e *frameworks*. Capítulo 3 é mostrada a metodologia para desenvolvimento desse trabalho. Os resultados e discusões são apresentados no Capítulo 4. Finalmente, o Capítulo 5 apresenta algumas conclusões e os trabalhos ainda a serem realizados.

2 REFERENCIAL TEÓRICO

Apesar de algumas ocorrências na internet focarem as comparações no desenvolvimento *web* entre *Java* e *Ruby*, essas comparações não contém um formalismo necessário para avaliar de forma concisa os prós e contras referentes a esses dois ambientes de desenvolvimento.

Assim, nesse trabalha demos início a um trabalho de estudo comparativo detalhado e metodológico do desenvolvimento *web* em ambientes *Java* e *Ruby*, afim de definir qual desses fornece um ambiente mais factível a produtividade na criação desse segmento de aplicação.

Não se tem notícias, ao menos em língua portuguesa, de nenhum trabalho acadêmico disponível sobre esse assunto.

Para facilitar a compreensão do presente trabalho, serão apresentados termos que serão recorrentes e de grande importância no seu desenvolvimento.

2.1 Aspectos gerais de *Java*

Java é uma linguagem de programação orientada a objeto desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems. Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um "bytecode" que é executado por uma máquina virtual.

A linguagem Java foi projetada tendo em vista os seguintes objetivos:

- Orientação a objeto - Baseado no modelo de *Smalltalk* e Simula67.
- Portabilidade - Independência de plataforma - "*write once run anywhere*".
- Recursos de Rede - Possui extensa biblioteca de rotinas que facilitam a cooperação com protocolos *TCP/IP*, como *HTTP* e *FTP*.
- Segurança - Pode executar programas via rede com restrições de execução;

Além disso, podem-se destacar outras vantagens apresentadas pela linguagem:

- Sintaxe similar a Linguagem C/C++.
- Facilidades de Internacionalização - Suporta nativamente caracteres *Unicode*.
- Simplicidade na especificação, tanto da linguagem como do "ambiente" de execução (*JVM*).
- É distribuída com um vasto conjunto de bibliotecas (ou *APIs*).
- Possui facilidades para criação de programas distribuídos e multitarefa (múltiplas linhas de execução num mesmo programa).
- Desalocação de memória automática por processo de coletor de lixo (*garbage collector*).
- Carga Dinâmica de Código - Programas em *Java* são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

2.2 Aspectos gerais de *Ruby*

Ruby é uma linguagem de programação interpretada, com tipagem dinâmica e forte, orientada a objetos com vastas semelhanças com *Perl*, *SmallTalk* e *Python*. Projetada tanto para a programação em grande escala quanto para codificação rápida, tem um suporte a orientação a objetos simples e prático. A linguagem foi criada pelo japonês *Yukihiro Matsumoto*, que aproveitou as melhores idéias das outras linguagens da época.

Para manter a praticidade, a linguagem possui algumas características interessantes:

- A sintaxe é enxuta, quase não havendo necessidade de colchetes e outros caracteres.
- Todas as variáveis são objetos, onde até os "tipos primitivos" (tais como inteiro, real, entre outros) são classes.
- Estão disponíveis diversos métodos de geração de código em tempo real, como os "*attribute accessors*".
- Através do *Ruby Gems*, é possível instalar e atualizar bibliotecas com uma linha de comando.
- *Code blocks* (blocos de código), ajudam o programador a passar um trecho de instruções para um método. A idéia é semelhante aos "*callbacks*" do *Java*, mas de uma forma extremamente simples e bem implementada.
- *Mixins*, uma forma de emular a herança múltipla, sem cair nos seus problemas.
- Tipagem dinâmica, mas forte. Isso significa que todas as variáveis devem ter um tipo (fazer parte de uma classe), mas a classe pode ser alterada dinamicamente. Os "atalhos" citados acima, por exemplo, se beneficiam da tipagem dinâmica para criar os métodos de acesso/alteração das propriedades.

2.3 *Sistema* web

Um sistema *web* é composto por um servidor e um cliente, onde o cliente, normalmente de um *browser*, remotamente ou não, realiza requisições a um servidor através do protocolo *HTTP*. O servidor então, trata essa requisição e retorna uma mensagem com o resultado da requisição realizada pelo usuário.

2.4 *Modularização de um Sistema* web

Um sistema *web* modularizado, pode ser dividido em duas camadas:

- A camada de negócio: local onde a informação de interesse do negócio, ao qual o sistema deva abstrair, é persistida, recuperada e atualizada.
- A camada de apresentação: responsável pela interação entre usuário e o sistema.

Na camada de negócio, temos módulos de negócios responsáveis por realizarem todas as requisições necessárias à aplicação ao banco de dados. O objetivo da camada de apresentação é mostrar informações e requisitar ações à camada de negócio.

2.5 Frameworks

Apesar das aplicações envolverem lógicas de negocio distintas, em seu desenvolvimento e possível seguir um padrão, podendo reutilizar código e experiência para um dado tipo de aplicação.

Os *frameworks* são ferramentas que possibilitam reutilizar o código e o comportamento padronizado.

Algumas das vantagens da utilização de *frameworks* para aplicações *web* são:

- Aumento da produtividade: escrevemos menos e utilizamos soluções já padronizadas.
- Reutilização de experiência: *frameworks* são baseados em projetos já consumados, sendo aproveitados os acertos desses projetos e eliminados os erros dos mesmos.
- Foco no Negócio: com o uso do *framework* focamos no negócio a ser modelado, já que a forma como o negócio da aplicação vai ser persistido, recuperado ou atualizado já está implementado e testado no *framework*.

2.6 Considerações Finais

Esse capítulo apresentou os principais conceitos recorrentes no trabalho: *aspectos gerais de Java e Ruby*, *sistema web*, *modularização de um sistema web* e *frameworks.* No próximo capítulo mostraremos a metodologia desenvolvida para conclusão desse trabalho.

3 METODOLOGIA

Para o desenvolvimento desse trabalho, está sendo implementado um sistema *web* comercial nos ambientes *Java* e *Ruby*, previamente especificado, modelado e projetado. Cuja implantação ocorrerá na *SuperMoney Financeira e Seguro.* A *Figura 1* apresenta o modelo relacional do sistema.

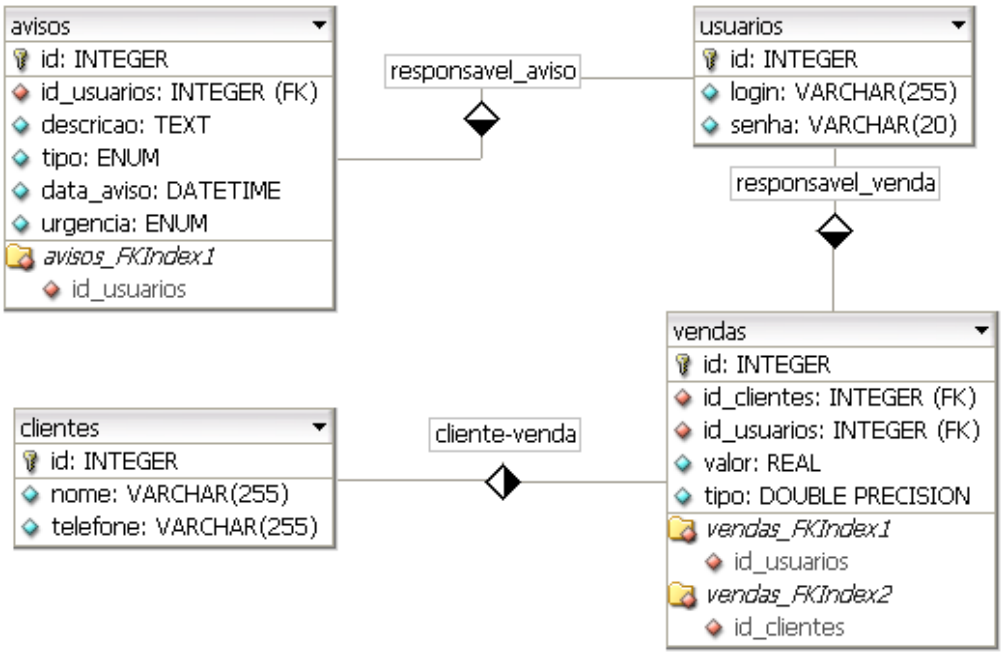

Figura 1 - Modelo do Banco

O sistema será composto de três telas: cadastro de avisos, cadastro de clientes e cadastro de vendas. Apesar de o modelo incluir a entidade *usuários*, essa entidade não será cadastrada no sistema, sendo os usuários sendo cadastro diretamente via script no banco de dados.

No modelo foram inseridas falhas, para que sejam necessárias alterações no mesmo afim de comparar a manutenção no código das tecnologias avaliadas no *Projeto orientado em computação II.*

Criado o modelo do sistema, foram levantados os seguintes quesitos a serem avaliados:

- Estudo inicial das linguagens.
- Configuração do ambiente de desenvolvimento
- Estudo para desenvolvimento *Web*
- Desenvolvimento
- Manutenção
- Implantação

Este presente trabalho visa os quatro primeiros tópicos listados, afim de determinar de maneira concreta, em qual das tecnologias tem-se mais facilidade de ser iniciado um desenvolvimento de qualidade.

A finalização do desenvolvimento, a simulação de manutenção e a implantação serão avaliadas no *Projeto orientado em computação II*.

As subseções a seguir exploram cada um destes itens.

3.1 Estudo inicial das linguagens

Afim de obter um bom resutado, primeiramente foram buscadas referências das linguagens a serem avaliadas: *Java* e *Ruby*. A seguir, alguns pontos que são interessantes de ressaltar nesse quesito.

- *Java* por ser mais antiga e difundida, apresenta muito mais referências para consulta do que *Ruby*. Essa, ainda é restrita a um circulo de aficionados pela linguagem, e seu uso comercial ainda é baixo.
- As linguagens *Java* e *Ruby*, são linguagens orientadas a objeto, o que facilita a escrita de código modularizado, facilitando assim a manutenção e reutilização de funcionalidades, duas questões fundamentais na engenharia de *software*. *Ruby*, se destaca nesse quesito por ser orientada a objeto de forma completa e pura, sem exceções. Por exemplo, em *Ruby*, o algarismo 1 é uma instância da classe *Fixnum*.
- Ambas as linguagens fazem uso de *garbage collector* (7), eliminando a necessidade de controle de alocação de memória pelo desenvolvedor.
- *Java* e *Ruby* possuem sistema de *threading* independente do sistema operacional.
- A alta portabilidade está presente nas duas linguagens.
- Ambas as linguagens são interpretadas.
- *Java,* por possuir uma síntese similar a outras linguagens orientadas a objeto, (sua síntese foi inspirada em c++) é mais fácil de ser assimilada. Já *Ruby*, por possuir uma síntese mais enxuta, obriga programadores a escreverem menos código, o que aumenta a produtividade de desenvolvedores experientes na tecnologia. Para efeito de comparação, na *Figura 2* é mostrado um código completo em *Java* responsável por escrever "*Hello World"* dez vezes na tela. Logo depois, é apresentado o similar em *Ruby* na *Figura 3*.

```java
import java.io.*;

public class HelloWorld
{
    public static void main(String args[])
    {
        for(int i=0; i <10; i++)
        {
            System.out.println("Hello World!");
        }
    }
}
```

Figura 2 – "Hello World" em Java

```ruby
10.times{print "Hello World!"}
```

Figura 3 - "Hello World" em Ruby

3.2 Configuração do ambiente de desenvolvimento

A *IDE* escolhida para o desenvolvimento da aplicação a ser desenvolvida, foi o *Eclipse Europa 3.2* (8), que é apontado por desenvolvedores experientes em Java como a melhor *IDE* para essa tecnologia, além de possuir *plugins* que possibilitam desenvolvimento de aplicações em *Ruby*.

Inicialmente, listaremos os passos necessários para configurar o ambiente *Java* afim de possibilitar o desenvolvimento. Em seguida, realizaremos o mesmo para a plataforma *Ruby*.

3.2.1 Configurando o *Java*

Aconfiguração mínima para desenvolver aplicações em *Java* é possuir instalado o *JRE* (9) e o *JDK* (10) *Java*. O *JRE (Java Runtime Environment)* possui os artefatos necessários para execução de aplicações *Java*, já o *JDK (Java Development Kit),* destinado aos desenvolvedores, possui utilitários que possibilitam o desenvolvimento de aplicações para essa plataforma.

Com o *JRE* e o *JDK* instalado, já estávamos aptos a desenvolver aplicações console em *Java*. Porém como nosso interesse é o desenvolvimento de aplicações *web*, ou seja, queremos desenvolver aplicações *J2EE (Java 2 Platform Enterprise Edition),* precisamos de um *container* para publicarmos nossas aplicações,e para isso, nesse projeto foi utilizado o *Tomcat 6* (11), que possui boa integração com a *IDE Eclipse*.

Para publicação de uma aplicação *web* no *Tomcat 6*, é necessário que a aplicação a ser publicada siga uma estrutura pré-definida de diretório além da configuração de dois arquivos: *web.xml* e um arquivo de contexto. A configuração desses arquivos é até relativamente simples, porém para desenvolvedores inexperientes com a tecnologia, essa configuração inicial torna-se algo árduo de ser entendido.

A ultima etapa na preparação do ambiente *Java*, é a integração da *IDE Eclipse* com o servidor de aplicação *Tomcat 6*. Essa etapa é de extrema importância, já que agindo desse modo, podemos executar e depurar as aplicações *web*. O *Eclipse* possui muitas facilidades para integração com o *Tomcat 6*, com a vantagem de, sendo feita de maneira correta, ser transparente para o projeto criado, o *container web* utilizado para publicação da aplicação.

3.2.2 Configurando o *Ruby*

Para preparação do ambiente *Ruby*, inicialmente instalamos o *Ruby 1.86*, versão estável mais recente. A instalação *Ruby* possui um gerenciador de instalação de pacotes, o *RubyGems* que facilita em muito a preparação do ambiente de desenvolvimento.

O desenvolvimento de aplicações *web* do *Ruby* é facilitado pelo *Rails*, *framework* para aplicações web que concentram uma gama de ferramentas para o desenvolvimento desse tipo de aplicação na plataforma *Ruby*. Por exemplo, o *Rails* já vem integrado com um *container web*, o *WEBrick* (12).

Assim, para terminarmos a configuração do ambiente de desenvolvimento *Ruby*, basta executar o comando mostrado na *Figura 4* no *prompt* de comando do sistema operacional.

Figura 4 - Configurando o Ruby

3.2.3 Estudo para desenvolvimento *Web*

Esta foi sem sombra de dúvidas a parte mais trabalhosa desse presente trabalho. O estudo da melhor forma de desenvolver uma aplicação *web* nas tecnologias estudadas, custou um grande esforço, já que para escrevermos bons códigos em uma dada tecnologia, é necessário um conhecimento amplo da mesma.

Começamos nossos estudos desbravando a arquitetura *servlets/jsp* de *Java*. Resumidamente, quando o *container* do servidor *web* recebe uma requisição, o mesmo invoca o *servlet*, uma classe responsável pelo tratamento dessa requisição. Nessa classe, é realizado todo o tratamento de lógica de negócio para a dada requisição. Após o tratamento da mesma, o *servlet* invoca uma página *JSP*, que nada mais é que uma pagina *html* que pode possuir código *Java*, afim de apresentar o resultado da requisição.

Após ambientação da arquitetura *servlets/jsp*, foi iniciado o estudo do *framework* de persistência *Hibernate* para Java. O *Hibernate* é responsável pela persistência e recuperação de informação de um banco de dados qualquer, em nosso caso, do *MySQL*. Esse é o *framework* de persistência *Java* mais difundido devido sua estabilidade, sua simplicidade e sua robustez. O grande problema do *Hibernate* é sua grande quantidade de *xml* para configuração. Para cada entidade que queremos materializar, temos que criar um *xml* para mapeamento da entidade do banco em uma classe *Java*. Para configurar o *Hibernate*, basta adicionar a biblioteca do mesmo no *path* da aplicação. Logo depois, deve-se configurar o arquivo *hibernate.cfg.xml*, onde configuramos a conexão com o banco e o mapeamento das entidades do banco que temos o desejo de materializar. Para cada uma dessas entidades, temos também de criar o arquivo *hbm.xml*, contendo o mapeamento com a classe *ADO Java*.

O estudo no ambiente *Ruby* concentrou-se na arquitetura do *Rails*. O *Rails* utiliza a estratégia *MVC*. A parte correspondente ao *Model* existe nas classes de dados e é implementada pelo *ActiveRecord*. Um segundo componente do *Rails*, o *ActionPack*, implementa o *V* e o *C* que são respectivamente *View* e *Controller*. Um *controller* é uma classe responsável por receber as requisições feitas pela aplicação e executar as ações necessárias para atender essas requisições.

Essas ações, ao serem executadas, provavelmente causarão mudanças no banco que serão efetuadas, por sua vez, por uma ou mais classes de dados. Finalmente, o resultado será exibido por meio de uma *view*, que será retornada ao usuário na forma de *HTML*, imagens, *XML*, ou qualquer outra saída aceitável da aplicação.

3.2.4 Iniciando o desenvolvimento

O ultimo quesito analisado engloba a criação da estrutura da aplicação e a geração de código. Para tal, foi utilizado o modelo do banco já apresentado nesse trabalho. A estrutura de diretórios criada para a arquitetura *Java* é mostrada na *Figura 5*.

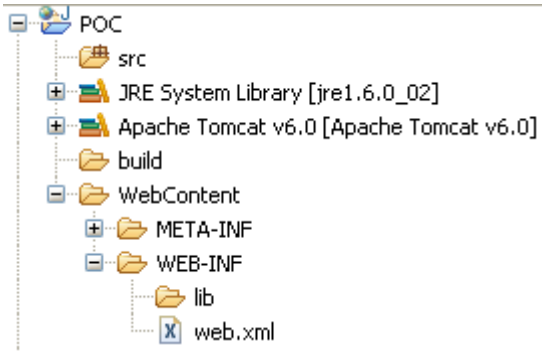

Figura 5 - Estrutura de diretórios para o projeto Java

A geração de código *Java*, para persistência, foi feita utilizando uma ferramenta desenvolvida neste projeto. Essa ferramenta funciona lendo o modelo diretamente do banco *MySQL* e produz um ganho de produtividade imenso, já que os *ADO's*, os *DTO's* e os arquivos *XML* de configuração são todos gerados de forma automática.

Já para o ambiente *Ruby*, o suporte do *Rails* para geração da estrutura de diretórios e geração de código, é fantástica. Para gerar o esqueleto de sua aplicação, basta executar o comando: *Rails "diretório_da_aplicação"*. Com esse comando, é gerado um esqueleto para a aplicação com as configurações mínimas necessárias para publicar a mesma. Um exemplo da execução do comando é mostrado na *Figura 6*.

```
C:\Projetos\Rails>mkdir POC

C:\Projetos\Rails>rails POC
      exists
      create  app/controllers
      create  app/helpers
      create  app/models
      create  app/views/layouts
      create  config/environments
      create  config/initializers
      create  db
      create  doc
      create  lib
```

Figura 6 - Geração de código no Rails

A geração de código do *Rails* também é muito pratica e simples. Para tanto, basta configurar no arquivo *database.yml*, onde definimos informações básicas referente a conexão com o banco

de dados usado. Feito isso, usamos o aplicativo *genarete* pertencente ao *Rails*. Com isso, temos para cada entidade a geração do *CRUD*.

Dessa forma, terminada a geração de código do *Rails*, temos criado um fluxo de tela para criação, edição, remoção e listagem de cada entidade, com seus respectivos atributos.

Terminada a geração de código, para ambas as plataformas, foi criada uma tela de login. Essa tela é tela base do sistema, aberta toda vez que o sistema comercial sendo desenvolvido é iniciado.

3.3 Considerações Finais

Este capítulo apresentou toda a modelagem e implementação feita durante o desenvolvimento desse trabalho, descrevendo detalhadamente cada uma das etapas realizadas. No próximo capítulo, discutiremos os resultados obtidos, ressaltando os prós e contras das linguagens *Java* e *Ruby* no desenvolvimento desse projeto.

4 RESULTADOS E DISCUSÕES

Terminado o trabalho inicial desse estudo, vamos compilar os resultados obtidos, afim de realizar uma avaliação parcial referente às duas tecnologias.

Primeiramente, será apresentado na *Tabela 1* o tempo gasto com cada tarefa desenvolvida durante o andamento desse projeto.

Tarefa	Tempo gasto (Java)	Tempo gasto (Ruby)
Estudo inicial das linguagens	**10h**	**30h**
Configuração do ambiente de desenvolvimento	**10h**	**3h**
Estudo para desenvolvimento Web	**10h**	**5h**
Desenvolvimento Inicial	**3h**	**1h**

Tabela 1 - Tempo gasto com tarefas

Agora, será feita uma breve discusão referente a cada uma das tarefas listadas nesta tabela, focando na comparação entre os ambientes de desenvolvimento *Java* e *Ruby*.

- **Estudo inicial das linguagens:** *Java* leva ampla vantagem nesse quesito por ser mais difundida, além de sua síntese ser baseada em outras linguagens orientada a objeto, facilitando o aprendizado.
- **Configuração do ambiente de desenvolvimento:** *Rails* trabalha com a idéia de convenção, ao invés de configuração. Por exemplo, a única configuração necessária após a geração de código do *Rails* é referente ao nome, login e senha do banco. Incrivelmente pratico favorecendo a produtividade.
- **Estudo para desenvolvimento web:** O *Rails* é hoje um dos principais responsáveis pela difusão do Ruby. Com isso, vários facilitadores foram adicionados à linguagem a fim de facilitar o desenvolvimento desse segmento de software. Porém, Java ainda comanda com folga esse segmento oferecendo as melhores ferramentas e documentação para criação de aplicação *web*.
- **Desenvolvimento inicial:** A idéia de convenção, ao invés de configuração, novamente se faz forte nesse quesito. É estimulante, após a geração de código com duas linhas, em *prompt* de comando, rodar e visualizar a aplicação funcionando com criação, edição, remoção e listagem de entidades.

Como esse estudo, ficou claro que é mais fácil iniciar o desenvolvimento uma aplicação em *Ruby* com *Rails* do que no ambiente *Java*. Essa facilidade vem de uma filosofia que acredito seja introduzida pelo *Rails*: convenção ao invés de configuração. Tudo que se tem de configurar, ou gerar em um projeto, ele estabelece um padrão de forma funcional. Por exemplo, ao invés de iniciar uma aplicação com uma tela *"Hello World!"*, o *Rails* gera o *CRUD* para as entidades do banco do sistema.

Uma característica muito interessante, que faltou em *Java*, por seguir o perfil de outras linguagens de programação populares, é *Ruby* ser uma linguagem enxuta e bastante dedutiva. Com isso, se escreve menos e com mais sentido, ganhado em produtividade e legibilidade de código.

Concluindo, terminamos a primeira parte do estudo comparativo, com o ambiente *Ruby* tendo uma vantagem referente ao ambiente *Java* devido ao desenvolvimento com o *Rails* ser bem mais pratico. Essa característica de praticidade é encontrada em outros ambientes, como por exemplo, o *PHP*. Porém essa praticidade geralmente sacrifica a manutenção e o desempenho de código. No próximo projeto orientado em computação, vamos verificar se *Rails* foge desse padrão de linguagens práticas, avaliando principalmente a elaboração e a manutenção de código em *Rails*, e o comparando com *Java*.

5 CONCLUSÃO E TRABALHOS FUTUROS

A proposta desse projeto orientado em computação, era iniciar e desenvolver parte do estudo comparativo entre *Java* e *Ruby*, considerando o desenvolvimento de aplicações web.

Essa primeira etapa do estudo teve seu foco em configuração e geração de código para o produto comercial que está sendo desenvolvido. Analisando os dados coletados do estudo, vimos que o Rails se destaca pela convenção ao invés de configuração. Assim, se gasta menos tempo configurando artefatos, aumentando assim a produtividade.

Também foi notaria o tanto que o desenvolvimento *web Java* é difundido, e bem amparado por ferramentas de bastante qualidade, porém que necessitam de configuração complexa onde o tempo gasto para um leigo na tecnologia tenha algo funcional tende a ser alto.

Como trabalho futuro, no *Projeto orientado em computação II*, realizaremos as seguintes etapas:

- Conclusão do desenvolvimento da aplicação.
- Realização de manutenção no código.
- Implantação do sistema em cliente.

Contribuições do *Projeto orientado em computação I* são:

- Entendimento dos ambientes Java e Ruby.
- Desenvolvimento de parte das ferramentas para geração de código.

6 BIBLIOGRAFIA CITADA

[1] http://pt.wikipedia.org/wiki/CGI

[2] http://pt.wikibooks.org/wiki/J2EE/Servlets

[3] http://pt.wikipedia.org/wiki/Java_(linguagem_de_programação)

[4] http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm

[5] http://www.rubyonrails.org/

[6] http://www.ruby-lang.org/en/

[7] http://en.wikipedia.org/wiki/Garbage_collection_(computer_science)

[8] http://www.eclipse.org/

[9] http://java.sun.com/j2se/desktopjava/jre/

[10] http://www.javafree.org/wiki/JDK

[11] http://tomcat.apache.org/