UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

2006.1

WEB 2.0: EXPLORANDO A WEB COMO PLATAFORMA

TRABALHO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

Aluno – Bruno Pereira da Encarnação, bpe@cin.ufpe.br.
Orientador – Silvio Romero de Lemos Meira, srlm@cin.ufpe.br.

06 de Outubro de 2006

Aos meus familiares,
Aos meus amigos,
Aos meus irmãos

Agradecimentos

Quando se chega a uma etapa como esta na vida, certamente têm-se muito a agradecer.

A DEUS. Não só pelas coisas estranhas que eventualmente permite na sua vida, como nos momentos de piadas sobremodo interessantes.

A minha família. Especialmente a Gildete, Ássima, Edwilson, Wilson, Romilda, Iara, Fabiana, Fábio, Alessandra, Vitor, Abraão, Lizzy e todos que não só toleraram todos estes anos de ausência quanto dispensaram atenção e apoio nos momentos de dificuldade.

A meu orientador, Silvio Meira, pelas horas infindáveis de discussões, aulas fantásticas e conselhos amigáveis. Grande parte do meu aprendizado durante este tempo de graduação certamente se deve a ele.

Aos demais professores do Centro de Informática. Especialmente os professores André Santos, Fábio Silva, Fernando Fonseca e Silvio Melo.

A Adelmário, Lauro, Marcio e Tadeu. Fizemos uma grande equipe durante toda a graduação e certamente são um dos motivos não ter reprovado algumas disciplinas no Centro de Informática. Espero sinceramente que esta equipe continue desenvolvendo coisas relevantes para o mundo.

A meus amigos da graduação. Certamente teria que inserir uma nova seção no trabalho caso fosse citar todos. Entretanto, todos os que de alguma forma contribuíram com a minha caminhada até aqui sabem que podem se sentir orgulhosos ao ler este parágrafo.

A Antônio Carvalho Filho, cujo apoio durante etapas iniciais da minha vida fortaleceu a base que hoje é a minha educação.

A meus amigos da vida toda. Sintam-se orgulhosos por terem vencido mais este desafio comigo. Estejam presentes nos próximos.

A *Laetitia* que acreditou em mim quando disse que não existem impossíveis, apenas improváveis. Todos os improváveis são possíveis.

“If you can't explain it simply, you don't understand it well enough”
Albert Einstein.

Resumo

Este trabalho apresenta uma visão geral do tema web 2.0 abordando de forma conceitual e de forma prática os principais aspectos envolvidos. Por ser uma área incipiente em termos de conhecimentos e publicações, o conceito ainda não está totalmente estabelecido. Entretanto, várias iniciativas estão sendo tomadas no sentido de fornecer embasamento teórico e técnico para a exploração do assunto. Tais iniciativas serão abordadas neste trabalho, além dos conceitos já amplamente aceitos em relação ao tema.

Palavras-chave: web 2.0, Rails, Ajax, ruby, aplicações web.

Abstract

This document presents an overview about web 2.0, exploring the main aspects involved in both conceptual and practical ways. The concept isn't completely estabilished yet because there aren't much information available. However, there is much work being done to develop a theoretical and technical base for further exploration of this subject. These initiatives are going to be discussed on this document, along with the topics widely accepted on this matter.

Keywords: web 2.0, Rails, Ajax, ruby, web applications.

SUMÁRIO

ÍNDICE DE FIGURAS

Resumo

Este trabalho apresenta uma visão geral do tema web 2.0, abordando de forma conceitual e de forma prática os principais aspectos envolvidos. Por ser uma área incipiente em termos de conhecimentos e publicações o conceito ainda não está totalmente estabelecido. Entretanto, várias iniciativas estão sendo tomadas no sentido de fornecer embasamento teórico e técnico para a exploração do assunto. Tais iniciativas serão abordadas neste trabalho, além dos conceitos já amplamente aceitos em relação ao tema.

Palavras-chave: web 2.0, Rails, Ajax, ruby, aplicações web.

Resumo

This document presents an overview about web 2.0, exploring the main aspects involved in both conceptual and practical ways. The concept isn't completely estabilished yet because there aren't much information available. However, there is much work being done to develop a theoretical and technical base for further exploration of this subject. These initiatives are going to be discussed on this document, along with the topics widely accepted on this matter.

Keywords: web 2.0, Rails, Ajax, ruby, web applications.

Objetivos do trabalho

O objetivo deste trabalho é apresentar uma visão geral do tema web 2.0 de forma prática, concisa e simples de forma a possibilitar o entendimento de um conjunto de conceitos considerados importantes em relação ao assunto.

Prover mecanismos teóricos e práticos que facilitem a replicação do experimento de desenvolvimento de uma aplicação web 2.0 abordado neste trabalho.

Muitos aspectos importantes relativos ao desenvolvimento de aplicações web não serão abordados neste trabalho por questões de escopo. Não é objetivo deste estudo se tornar guia de referência completa para o assunto. Todavia, é importante que o mesmo seja suficientemente capaz de elucidar os conceitos básicos envolvidos no desenvolvimento de aplicações web 2.0.

Não é objetivo deste trabalho adentrar nos detalhes de implementação de quaisquer das tecnologias envolvidas. De fato, este estudo se propõe a fornecer conhecimentos que possam servir de base para a incorporação de outros conhecimentos mais específicos da área.

A ausência de documentos que englobam de forma conjunta e concisa os vários conceitos aqui descritos dá ao trabalho um aspecto exploratório. Entretanto, ao longo das várias seções deste trabalho procuraremos embasamento em um conjunto de definições amplamente aceitas e um outro conjunto de conhecimentos adquiridos através de experiências práticas e estudos bibliográficos coletados de várias fontes.

Organização do trabalho

Tendo como objetivo a melhor compreensão e clareza, o trabalho é dividido em três partes principais. Nos primeiros capítulos serão vistos os principais conceitos acerca do termo web 2.0 e aplicações práticas. A leitura destes

capítulos é fundamental para a ambientação com termos e conceitos que serão bastante utilizados ao longo deste estudo. Após este primeiro contato, serão elucidados os principais aspectos relativos à implementação de aplicações web 2.0. Tal parte do trabalho é uma aplicação dos conceitos anteriormente mencionados no desenvolvimento de uma aplicação simples que facilite a compreensão dos mesmos. Por fim, será apresentada uma visão geral de todos os conceitos envolvidos. De fato, um dos objetivos principais do trabalho é se tornar uma referência rápida, acessível e compreensível a respeito de web 2.0 tanto do ponto de vista conceitual quanto do ponto do vista de desenvolvimento.

Metodologia de trabalho

O presente trabalho foi escrito pelo aluno de graduação em Ciência da Computação da Universidade Federal de Pernambuco Bruno Pereira da Encarnação, sob orientação do professor Silvio Romero de Lemos Meira. Este documento é resultado de aproximadamente um ano de estudo e envolvimento com o tema, leitura de várias fontes bibliográficas e o conhecimento adquirido com o desenvolvimento de duas aplicações que contemplam a maioria dos conceitos que serão discutidos ao longo das próximas seções. O conteúdo que será apresentado foi refinado através de reuniões com o orientador e estudos em disciplinas relacionadas ao objeto de estudo deste trabalho. Em particular a disposição dos capítulos e a divisão do trabalho são inspiradas em um artigo sobre web 2.0 [4] onde o autor utiliza a mesma abordagem para discorrer sobre aspectos introdutórios a respeito do conceito em questão.

Relevância

O conceito de web 2.0 é relativamente novo, datando do final do ano de 2004. Existe um conjunto potencial de pessoas ao redor do mundo interessado em entendê-lo tanto sob o ponto de vista de conceitual quanto do ponto de vista

técnico e de novas oportunidades de negócio envolvidas. Esta última visão não faz parte do escopo deste trabalho. Entretanto, a apresentação de uma visão geral do tema com abordagens práticas pode ser muito favorável ao entendimento do conceito, uma vez que a sua definição não é algo de assimilação. Podemos citar como uma contribuição importante deste projeto a integração de vários conceitos que definem web 2.0 e estão dispersos em diversos sites na web e livros em uma fonte somente.

O que é Web 2.0?

O termo surgiu durante uma sessão de *brainstorming* entre representantes da O'reilly Media [1] e da Media Internacional [2], em 2004. Segundo os autores do termo, a expressão servia para denotar um conjunto interessante e rico de aplicações e sites que estavam surgindo naquela época com características comuns. Em outubro do mesmo ano, a O'reilly organizava a primeira conferência a respeito do tema [3].

Desde a sua primeira menção, o termo tem causado polêmica e divisões entre grupos que acreditam que o mesmo está relacionado a manobras puramente ligadas a marketing da própria O'reilly e um segundo grupo que defende que de fato a web estaria passando por um momento de transformação, justificando assim a presença da expressão "2.0", da mesma forma que em versões de software.

Neste contexto, o artigo de Tim O'reilly [4] torna-se crucial para o entendimento do que de fato seria web 2.0. O mencionado artigo procura definir o conceito em termos dos exemplos de aplicações e serviços web aos quais ele estaria relacionado. Neste trabalho tomaremos a mesma direção, apresentando inicialmente um conjunto de aplicações e serviços notadamente pertencentes ao grupo do que seria a nova geração da web. Uma vez estabelecido um arcabouço conceitual, passaremos à segunda parte do trabalho que está vinculada mais a tecnologias em detrimento a conceitos. Por fim, faremos uso de ferramentas que habilitam estes serviços e aplicações.

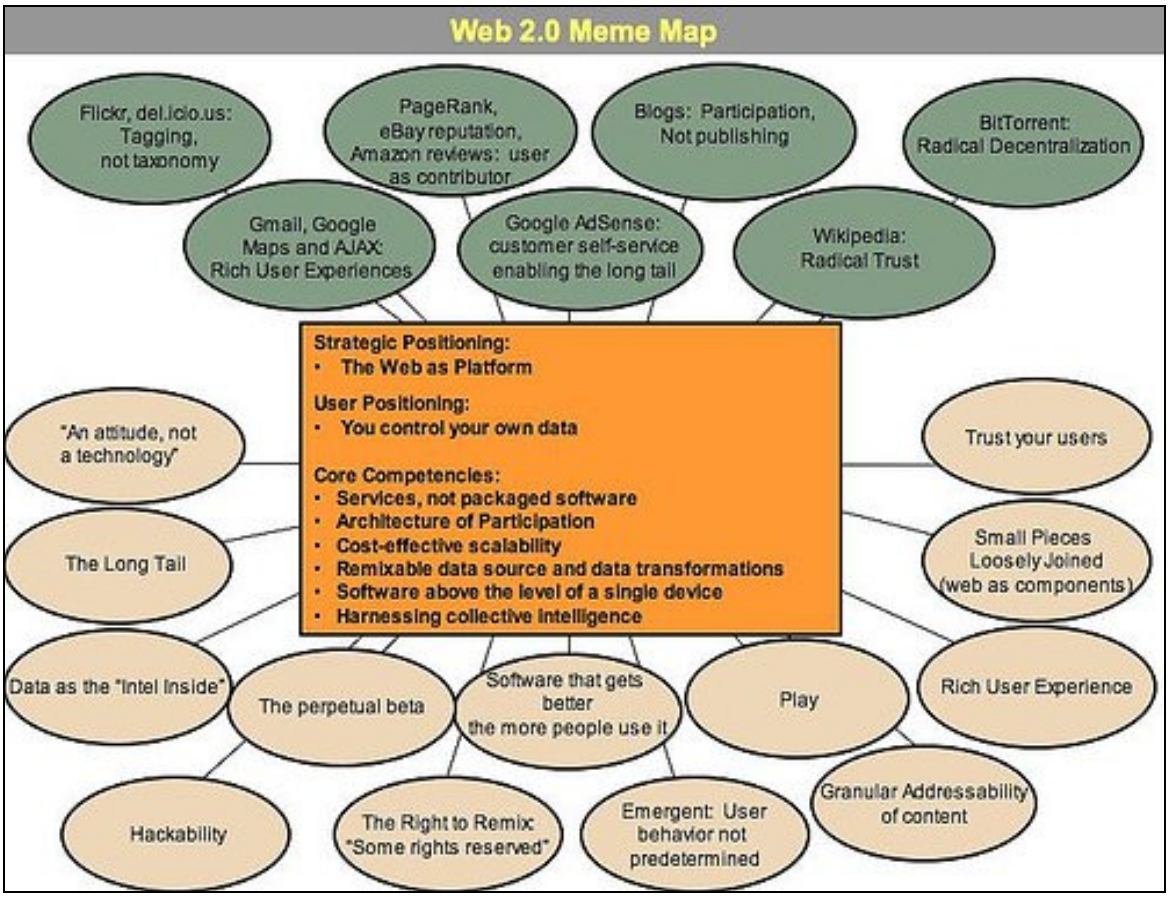


Figura 1. Web 2.0 e os conceitos envolvidos.

Web 2.0 não é um conceito pura e simplesmente. Antes, é um conjunto de conceitos que estão agrupados sob o mesmo nome e possuem maior ou menor relação com duas idéias básicas. A primeira delas é a idéia de que a web deve ser vista como um ambiente de colaboração, onde usuários alimentam um ciclo em que eles consomem conteúdo e também criam e compartilham conteúdo. Tal idéia está intimamente centrada no usuário. A segunda idéia é que a web pode ser vista como uma grande plataforma programável. Este seria um tipo de colaboração a nível de aplicações as quais possuem mecanismos de troca de informações entre si e possibilitam o surgimento de outras aplicações.

É importante ressaltar que web 2.0 não significa necessariamente uma revolução na web em contraste ao que poderia ser chamado de “web 1.0”. O termo está associado à evolução da nossa própria percepção das potencialidades da web e, principalmente, de seus usuários. Neste trabalho estaremos

interessados nos mecanismos conceituais e tecnológicos que estão permitindo esta evolução.

Visão geral de aplicações web 2.0

A exemplo do artigo de Tim O'reilly utilizaremos alguns serviços e aplicações considerados web 2.0 para introduzir alguns conceitos. Esta seção se dedica a enumerar algumas destas aplicações. É de fundamental importância perceber estes conceitos a partir dos exemplos a serem citados, pois dos mesmos depende muito da nossa percepção de uma aplicação web 2.0. Alguns termos como *tags*[1], comunidades e colaboração serão uma constante a partir desta seção. Outra característica em comum dos serviços que serão citados é a produção e compartilhamento de conteúdo. Todos estes conceitos têm a ver com a experiência do usuário. Do ponto de vista de aplicação, os serviços também disponibilizam interfaces que permitem o acesso de aplicações de terceiros.

Flickr

Flickr [5] é um serviço pertencente a Yahoo [6] que permite o compartilhamento, busca e armazenamento de arquivos de imagem na internet. Mais que isto, Flickr permite que usuários classifiquem suas imagens com nomes arbitrários os quais chamamos *tags*. O serviço também possibilita a formação de comunidades que agrupam usuários com interesses comuns. Uma das principais idéias associadas a Flickr, além do compartilhamento de imagens, é a possibilidade que os usuários têm de classificar seus arquivos de forma compreensível a outros usuários através das citadas *tags*. Estas por sua vez são usadas para buscar imagens na base de dados do serviço. Flickr possui uma API[2] que permite que aplicações de terceiros utilizem a sua base de dados.

[1] Palavras-chave arbitrariamente escolhidas para categorizar algo
[2] Conjunto de rotinas de programação

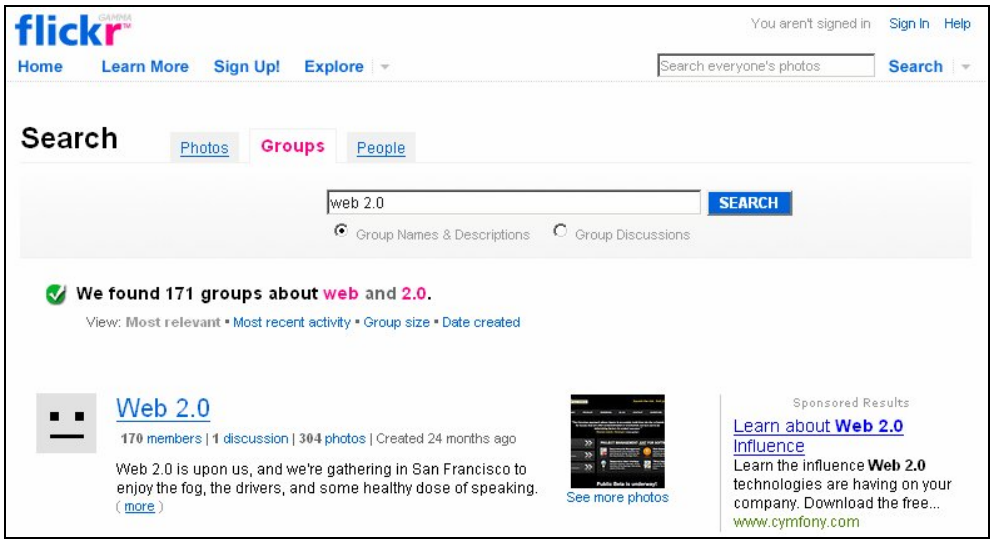


Figura 2. *Screenshot* da interface de Flickr.

Delicious

Este serviço inaugura a nova era de compartilhamento de *bookmarks,* no final de 2003. A maioria das empresas que proviam este tipo de serviço havia fechado as portas devido à bolha da internet. Delicious [7] aplica um modelo semelhante a Flickr de utilização de *tags* para usuários identificarem seus próprios *bookmarks* e também buscarem outros de seu interesse.

Este serviço possui vários aspectos interessantes. O primeiro deles é que se pode entender delicious como um engenho de busca que utiliza uma espécie de "inteligência colaborativa" para filtragem dos resultados. Em geral, os *bookmarks* que são mais associados a um dado termo são de fato interessantes para aquele termo. Outro aspecto interessante é que o sistema pode fazer associações baseadas no conjunto de *tags* de um dado usuário e indicar possíveis *bookmarks* que seriam relevantes para o mesmo.

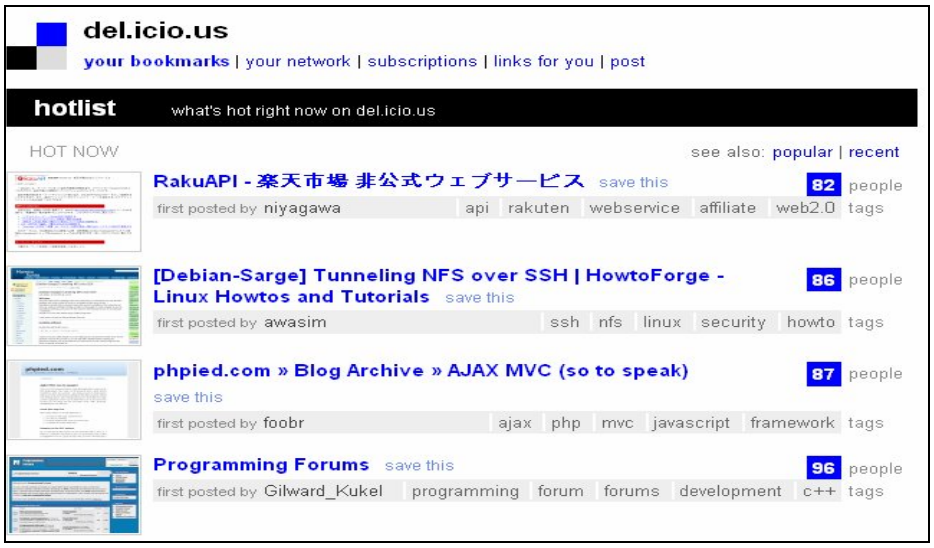


Figura 3. *Screenshot* da interface de delicious.

Desenvolvendo uma aplicação web 2.0

Esta seção tem por objetivo abordar os principais aspectos envolvidos no desenvolvimento de uma aplicação que contemple os conceitos a respeito de web 2.0 anteriormente citados. Em um primeiro momento serão descritos o projeto, as principais decisões envolvidas e todas as outras decisões anteriores à implementação do mesmo. Após este primeiro contato, passaremos aos detalhes técnicos.

Apresentação do projeto

Nos tópicos anteriores vimos alguns exemplos de aplicações web consideradas web 2.0. Procuramos enumerar quais conceitos eram comuns a aplicações deste tipo e desenvolver nossa própria aplicação com o objetivo de validar os conceitos relativos ao tema. Destacamos os conceitos que foram considerados relevantes e deveriam estar presentes em nosso protótipo. São eles: *folksonomy,*[3] colaboração, *web services*, RSS[4] e Ajax.

[3] Categorização não sistemática, em geral usando palavras-chave escolhidas arbitrariamente
[4] Formato de publicação de informação na web baseado em XML

De posse dos conceitos, procuramos desenvolver uma aplicação que fizesse uso dos mesmos. Neste momento do projeto, a 4people2, empresa na época participante do processo de incubação do Recife Beat [8], foi de fundamental importância. O protótipo que será apresentado nos próximos tópicos e o conhecimento a respeito das tecnologias envolvidas foram obtidos com o apoio da empresa.

A aplicação sugerida como protótipo pela 4People2 denomina-se *All My Questions*. Em linhas gerais é uma aplicação semelhante a um fórum web, com algumas ressalvas. Uma delas é o uso de *tags* para classificar as perguntas por assuntos. Outra é a publicação de um *web service*, através do qual é possível manipular alguns dados da aplicação. Por fim, a publicação de informações usando RSS. Parte dos requisitos de *All My Questions* foi extraído com base no exemplo de uma aplicação já existente chamada Yahoo Answers [9].

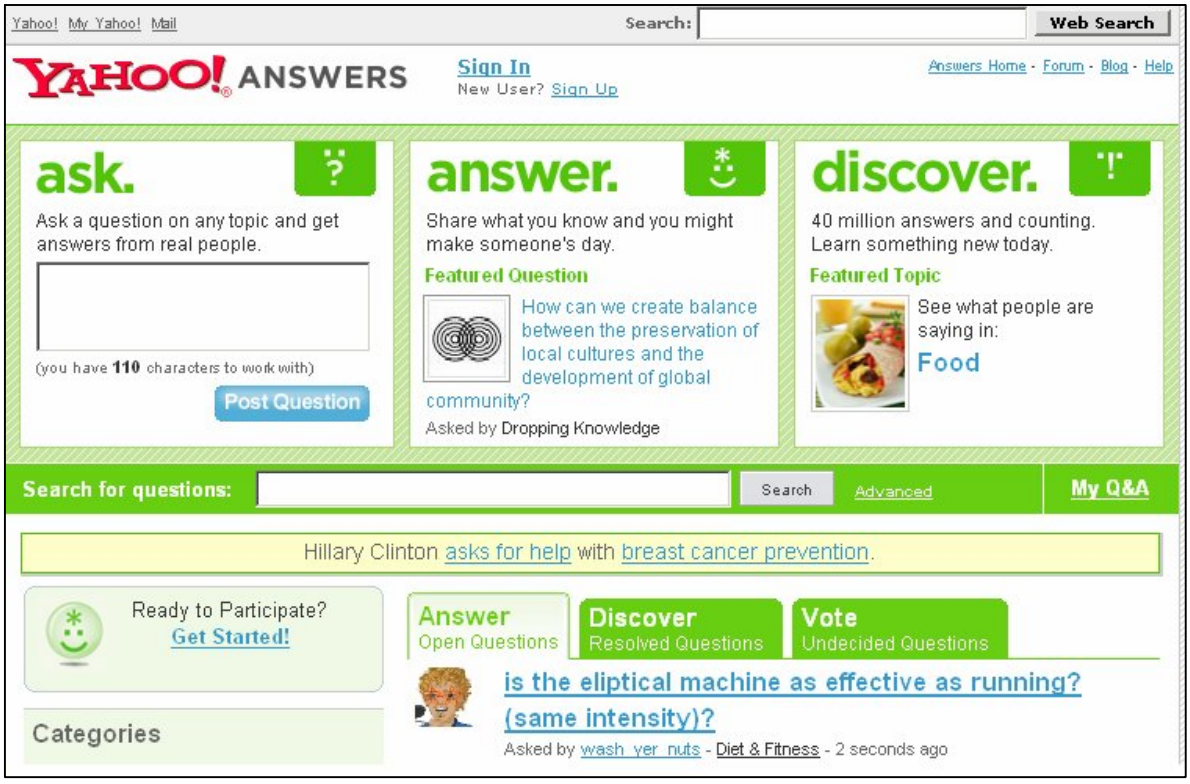


Figura 4. Interface de Yahoo Answers.

Metodologia de desenvolvimento

No projeto *All My Questions* utilizamos uma metodologia desenvolvida juntamente com a 4people2. Tal metodologia foi concebida levando-se em conta os fatores principais: ciclo curto de desenvolvimento, pequeno número de desenvolvedores (precisamente três) e restrições relativas a tempo e escopo do projeto.

Em linhas gerais, a metodologia adotada é orientada a prototipação. Em um dado instante define-se um conjunto de requisitos os quais são rapidamente prototipados e levados a aceitação do cliente (papel desenvolvido neste projeto pela própria 4people2). Deste processo obtêm-se novos requisitos e a aplicação é novamente prototipada em um modelo evolucionário em que a cada ciclo definem-se alguns requisitos a serem adicionados à aplicação, obtém-se a avaliação do cliente e prototipa-se novamente. Este tipo de metodologia adequa-se perfeitamente à natureza do *framework* escolhido para implementação, Rails.

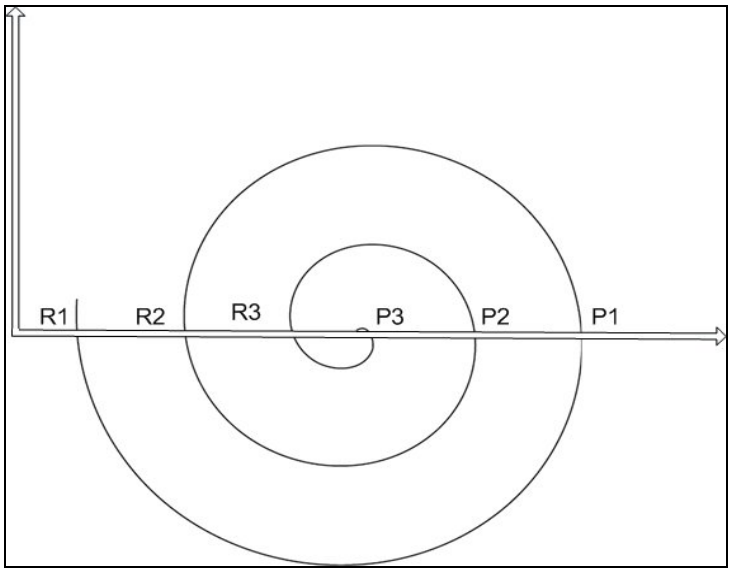


Figura 5. Ilustração da metodologia de desenvolvimento do projeto.

A figura 5 ilustra a metodologia de desenvolvimento do projeto. No lado esquerdo vemos R1, R2 e R3. Cada um deles representa um conjunto de requisitos. No lado direito temos P1, P2, P3. Cada um deles representa um

protótipo resultante do trabalho de implementação dos requisitos daquele ciclo somado à correção de problemas do ciclo anterior. O espiral como um todo representa um modelo em que cada protótipo representa uma evolução em relação ao protótipo anterior.

Por motivos relacionados a escopo, questões ligadas a testes não foram contempladas durante o projeto. Destaca-se o caráter exploratório tanto do projeto quanto da metodologia utilizada, cujos resultados serão vistos a seguir.

Decisões de projeto

Durante os primeiros momentos do projeto algumas decisões importantes foram tomadas. A primeira delas foi a utilização do *framework* Rails [10]. As principais motivações para tal decisão foram: o caráter ágil do desenvolvimento de aplicações web usando Rails e a integração do *framework* com bibliotecas que facilitam a implementação de aplicações que utilizam ajax. Por fim, soma-se às motivações o conhecimento técnico obtido através dos desenvolvedores da 4people2. A segunda definição importante foram os requisitos que dariam origem ao primeiro protótipo.

O projeto foi dividido em quatro ciclos de desenvolvimento. A cada ciclo, excetuando-se o primeiro, são corrigidos eventuais problemas relacionados ao ciclo anterior, adicionados novos requisitos previamente planejados e é implementado um novo protótipo que é validado pelo cliente. Os ciclos serão detalhados nos próximos tópicos com as suas contribuições, principalmente técnicas, para o processo de desenvolvimento de aplicações web 2.0. Segue uma visão geral de cada ciclo

Ciclo um:

- Criação da aplicação web;
- Mapeamento da base de dados;
- Métodos que permitam cadastrar, remover e alterar uma questão;
- Introdução do conceito de usuário na aplicação;
- Primeiro protótipo;

Ciclo dois:

- Adição de módulos que permitam autenticação;
- Pesquisa de questões na base de dados;
- Correção de problemas relacionados ao primeiro protótipo;
- Segundo protótipo com correções e novos requisitos implementados;

Ciclo três:

- Inserção de *tags* para identificar as questões (*folksonomy*)
- Inserção de elementos ligados a Ajax;
- Correção de problemas relacionados ao segundo protótipo;
- Terceiro protótipo com correções e novos requisitos implementados;

Ciclo quatro:

- Adição de RSS e *Web Services* ao projeto;
- Correção de problemas relacionados ao terceiro protótipo;
- Quarto protótipo com correções e novos requisitos implementados;

O desenvolvimento

O objeto principal desta seção é detalhar os principais aspectos relativos à implementação do projeto *All My Questions* e descrever as contribuições técnicas que cada ciclo proporcionou, demonstrando de forma prática tais contribuições. Durante o projeto foram utilizadas diversas bibliotecas de terceiros. É também objetivo desta seção descrever a contribuição destas bibliotecas. Eventualmente, poderá haver problemas com a compreensão dos códigos aqui descritos por parte dos não conhecedores da linguagem ruby e do *framework* Rails. Estas dúvidas poderão ser esclarecidas na segunda parte do trabalho, onde mostraremos o arcabouço conceitual que fundamentou o desenvolvimento da aplicação. É possível que esta seção seja então revisitada e melhor compreendida.

Os objetos principais do projeto *All My Questions* são questões cadastradas por usuários, da mesma forma que em fóruns disponíveis na web. Usuários interagem gerando questões que eventualmente são respondidas por outros usuários. Interessados nas informações podem obter acesso às questões listando-as na página principal do serviço ou utilizando RSS. Eventualmente terceiros podem acessar a aplicação e cadastrar uma nova questão através de um *web service* provido pela aplicação.

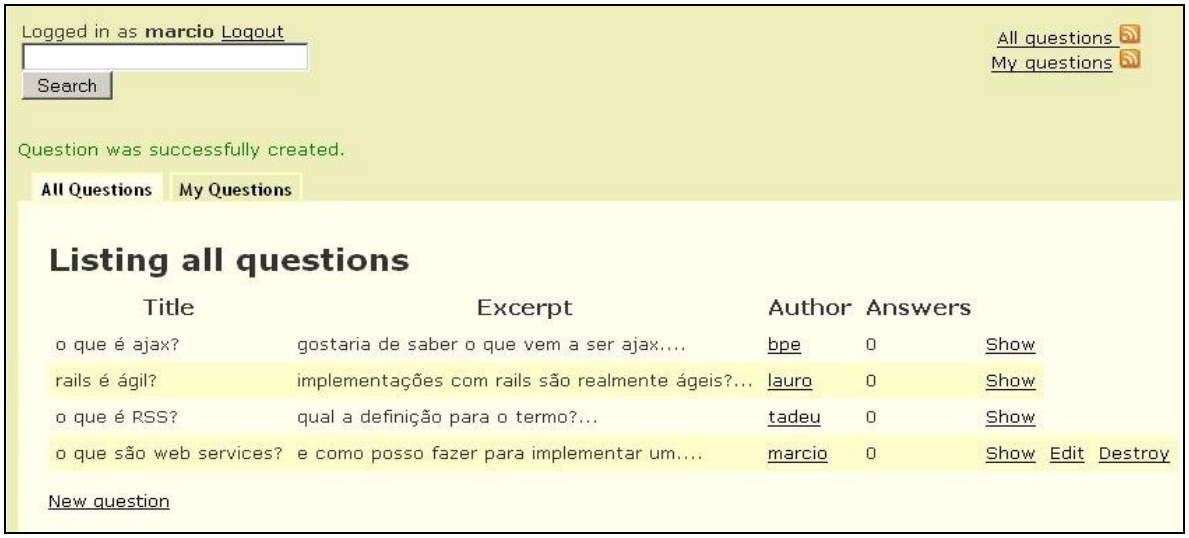


Figura 6. Interface de All My Questions.

Ambiente de desenvolvimento

Durante a fase de implementação deste protótipo, o ambiente de desenvolvimento estava configurado da seguinte forma:

Ambiente de desenvolvimento: Linux / Windows
Linguagem: Ruby 1.8.5
Framework: Rails 1.1.6
Banco de Dados: Mysql 4.0
Ambiente de programação: RadRails [11], um ambiente de programação open source específico para desenvolvimento em Rails.

1ª Ciclo de desenvolvimento

Objetivos: iniciar o projeto, mapear a base de dados e adicionar os métodos necessários a cadastrar, remover e alterar questões.

Todo o núcleo básico da aplicação foi desenvolvido neste ciclo. Basicamente a implementação relativa a este ciclo do projeto se resumiu a manipulação da Base de Dados. O projeto possui um número relativamente pequeno de interfaces com o usuário e estas foram prototipadas em papel. Durante esta fase do projeto foram utilizadas as interfaces geradas por Rails. Paralelamente com a explanação do desenvolvimento do projeto, demonstraremos alguns conceitos básicos relativos ao desenvolvimento de aplicações web usando o *framework* em questão.

Criação do Projeto

Para criar um projeto, o primeiro passo é executar o comando “rails”, que cria a estrutura básica de arquivos e pastas.

```
rails <nome do projeto>
```

Os seguintes diretórios principais são criados*:*

- *<nome do projeto> - Raiz do projeto*
- *<nome do projeto>/app/models – Modelos*
- *<nome do projeto>/app/controllers – Controladores*
- *<nome do projeto>/app/views – Visões*
- *<nome do projeto>/config – Arquivos de configuração*
- *<nome do projeto>/test – Arquivos de teste*
- *<nome do projeto>/script – Scripts auxiliares*
- *<nome do projeto>/public – Arquivos disponíveis para o servidor web*

Definição da estrutura de dados

Para a abstração da integração com o banco de dados Rails utiliza a biblioteca ActiveRecord. Nesse estágio a estrutura das entidades foi definida inicialmente no banco de dados, utilizando arquivos SQL simples, sendo integrado em seguida ao código ruby com a geração dos modelos. Uma das características principais de ActiveRecord é a ausência de arquivos de configuração, criando a estrutura das classes a partir dos mapeamentos correspondentes de acordo com um sistema de inferência de nomes. Ao gerar o código de um modelo, o sistema mapeia o nome do modelo (Inglês, singular) no nome da tabela (Inglês, plural). Por exemplo: O modelo “Question” mapeia a tabela “questions”. As classes são geradas com métodos que garantem transparência no acesso aos dados, não sendo necessário entrar em detalhes em relação ao modo de persistência utilizado. Por exemplo, a seqüência de comandos

```
question.title = “Quem vai ser o presidente?”
question.save
```

atribui um valor ao título de uma questão e salva a modificação na base de dados.

Outro recurso de Rails utilizado foi o *scaffold*, que cria um suporte básico a operações CRUD[5] com apenas uma declaração no modelo ou a utilização do *script* de geração:

```
<raiz do projeto>ruby script/generate scaffold Question Questions
```

O comando acima utiliza o *script* de geração de código para criar o modelo "Question", com as ações CRUD implementadas no controlador "Questions". Este recurso é extremamente útil nas etapas iniciais de desenvolvimento. A seguir um exemplo retirado do controlador principal do projeto, disponível no anexo um:

```
def show
  @question = Question.find(params[:id])
end
```

Este trecho de código faz uma busca na base de dados pela questão cuja chave-primária é igual ao parâmetro "id". Note a ausência de código SQL para este tipo de consulta. Um destaque a ser feito é que cada linha de uma tabela é representada por uma instância do modelo correspondente.

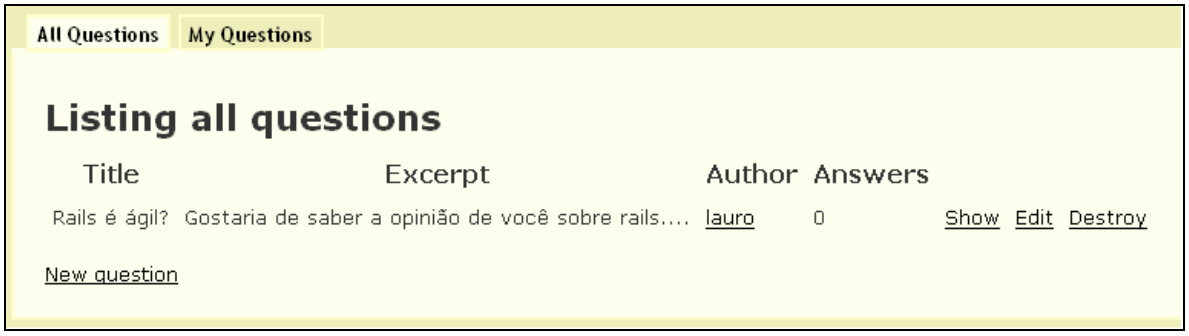


Figura 7. Screenshot da interface de All My Questions com as funções básicas.

Para maior compreensão dos códigos descritos, recomendamos a consulta ao anexo dois (esquema da base de dados).

[5] Rotinas de deleção, atualização, busca e inserção de dados.

2º Ciclo de desenvolvimento

Objetivos: adicionar o processo de autenticação de usuário com os dados *login* e senha. Possibilitar o cadastro de novos usuários. Inserir busca por questões usando como índice títulos e conteúdos. Refinamento da base de dados.

Neste ciclo do projeto a aplicação base começa a ser modificada adicionando-se elementos que não são gerados a partir dos *scripts* básicos de Rails, como autenticação de usuários, busca e *layout*. Fez-se necessária a procura de bibliotecas que suportassem estas implementações.

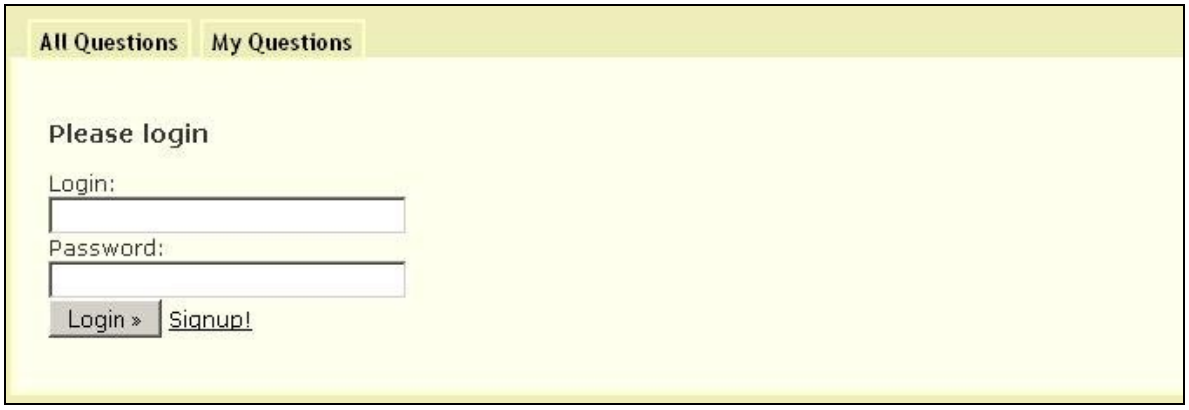


Figura 8. Interface de login de *All My Questions*

Para a autenticação, o sistema *Login Generator* foi utilizado. Ele necessita apenas do uso de *login* e senha, sendo uma alternativa simples para implementação rápida. Uma tabela é criada com os dados necessários e um *script* de geração gera o modelo e controlador correspondentes, com as ações “login”, “signup”e “logout”.

```
gem install login_generator
```

Uma vez instalado, o método *login_required* pode ser utilizado nos outros controladores para certificar que o usuário esteja autenticado ao acessar as

ações. Opcionalmente, algumas ações podem ser acessadas livremente, sem necessidade de autenticação.

```
ruby script/generate login Account
```

Os mecanismos de busca foram desenvolvidos utilizando-se Ferret [12], um implementação em ruby do engenho de busca Lucene, [13] do projeto Apache. A integração com Rails se deu através do *plugin acts_as_ferret* [14], que abstrai grande parte dos detalhes de indexação, sendo apenas necessário declarar o modelo como “acts_as_ferret”. A partir de então, a indexação é automática. Para realizar pesquisas é utilizado o método *find_by_contents* o qual retorna o conjunto de objetos encontrados baseado na consulta.

```
script/plugin install
svn://projects.jkraemer.net/acts_as_ferret/trunk/plugin/acts_as_ferret
```

Detalhes de implementação do controlador de autenticação de usuário do projeto podem ser encontrados no anexo três.

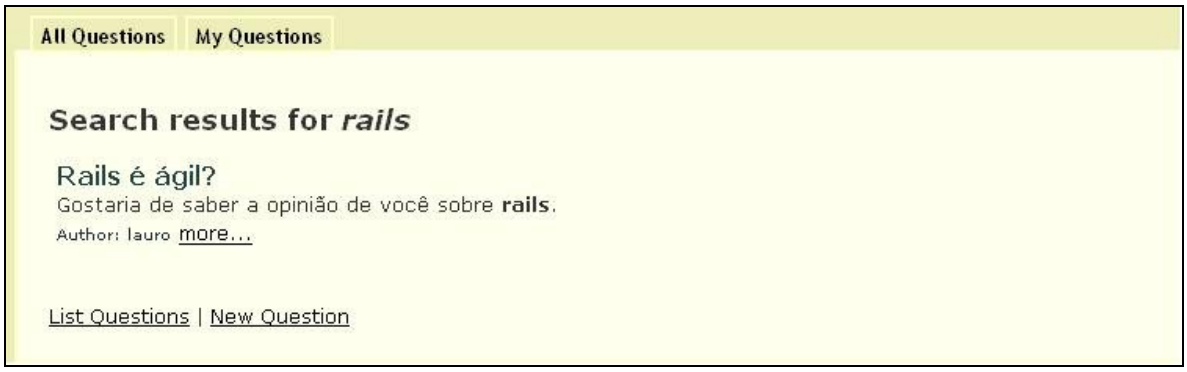


Figura 9. Interface de busca de *All My Questions*.

3º Ciclo de desenvolvimento

Objetivos: adicionar *tags* e componentes que utilizem Ajax.

No terceiro ciclo do projeto os principais meios de manipulação dos dados da aplicação estão implementados. Nesta fase do projeto são inseridas algumas mudanças relativas à interface. Somam-se ao ciclo as implementações de *tags* e elementos que utilizem Ajax.

Figura 10. Interface com tags em *All My Questions*.

Para a implementação de *tags* foi usado o *plugin acts_as_taggable* [14], que funciona nos mesmos moldes do *acts_as_ferret*, sendo declarado no modelo. Com o uso deste *plugin* são incorporados ao modelo os métodos *tag_with* (atribui tags), *find_tagged_with* (procura elementos dado uma *tag*), *tag_list* (retorna todas as *tags*). Estes métodos são básicos para manipulação de tags e fornecem uma abstração importante para desenvolvedores, pois incorporam funcionalidades relativas a relacionamento entre *tags.*

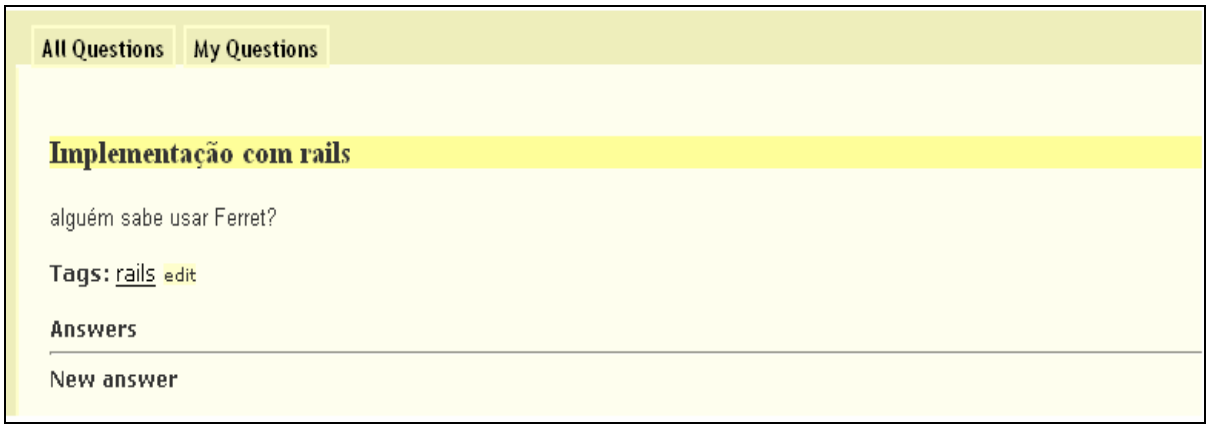


Figura 11. Exemplo de uso de Ajax no projeto.

Elementos relativos a Ajax foram facilmente adicionados ao projeto utilizando-se as bibliotecas já integradas com Rails, no caso Scriptaculous [15]. Foram adicionados ao projeto *InPlace Editors* que facilitaram a edição de títulos e comentários por parte de usuários, sem haver a necessidade de processar novamente toda a página a cada edição.

4º Ciclo de desenvolvimento

Objetivos: adicionar suporte a RSS e *Web Services*. Incrementar a interface com o usuário.

O quarto ciclo do projeto marca o fim da fase evolução do protótipo. Neste momento do projeto são implementadas as funções consideradas de menor risco de desenvolvimento associado. Isto pelo motivo de a adição destas *features* não implicar profundas mudanças estruturais, ou seja, mudanças nos esquemas da base de dados usada.

Ruby possui módulos que facilitam o tratamento de XML. A aplicação deve exportar no formato as questões relativas a um dado usuário e as questões de todos os usuários (gerais). O trecho de código

```
    def all
        @questions = Question.find(:all)
        render_without_layout
        @headers["Content-Type"] = "application/xml; charset=utf-8"
    end
```

disponível no Anexo 4 (quatro) exemplifica o uso de RSS em *All My Questions*. Assim que o método *all* de *RssController* é chamado a aplicação faz uma busca na base de dados retornando todas as questões presentes na base de dados. O método *render_with_layout* se encarrega de utilizar o *template* de RSS (disponível no anexo cinco) para formatar a apresentação.

Rails tem um módulo específico para *Web Services* chamado ActionWebService. Utilizando este módulo foi possível a implementação de um web service usando SOAP que publicasse o serviço de criação de questões. Os detalhes de definição do serviço e implementação da API podem ser vistos no Anexo 6 (seis). O seguinte trecho de código é responsável pela definição da API:

```
class RpcAPI < ActionWebService::API::Base

  #definicao da assinatura do metodo
  api_method :new_question,
    :expects => [{:username => :string},
                 {:title => :string},
                 {:body => :string},
                 {:tags => :string}],
    :returns => [:string]
end
```

Este serviço pode ser então acessado por qualquer aplicação que implemente o protocolo. Como exemplo, temos este trecho de código escrito em ruby que cria uma nova questão via *web service*

```
require 'soap/wsdlDriver'

wsdl_url = " http://localhost:3000/rpc/wsdl "
soap = SOAP::WSDLDriverFactory.new( wsdl_url ).create_rpc_driver
param = {"username" => "tadeu", "title" => "Hello World", "body"=>
"moahahahaha", "tags" => "bueroso maluco" }
result = soap.newQuestion(
param['username'],param['title'],param['body'],param['tags'] )

puts result
```

Web services foi o último requisito previsto para o protótipo. De fato, o desenvolvimento da aplicação foi em muito ajudado por bibliotecas integradas ao *framework* e por módulos de suporte da linguagem ruby, como foi possível se verificar ao longo desta seção. Os maiores problemas encontrados durante esta fase do projeto estão ligados basicamente ao aprendizado da manipulação de APIs e à pouca quantidade de exemplos disponíveis. Ao final desta etapa do projeto foram gerados dois conjuntos de artefatos: os documentos relativos à implementação do projeto (ruby doc) e o código da aplicação como um todo. Estes estão disponíveis juntamente com este documento.

Programando a web

O conceito de *mashups* está intimamente ligado ao conceito de programar a web. No contexto de aplicações web, assim como em música, *mashups* são combinações. No caso particular de aplicações web, os elementos destas combinações são serviços entregues por outras aplicações web. Uma vez combinados, estes serviços resultam em uma aplicação nova. Este conceito foi possível devido à publicação de APIs por parte de desenvolvedores de aplicações web (notadamente os grandes desenvolvedores como Google e Yahoo) permitindo assim o acesso aos serviços providos por estas aplicações através de software escrito por terceiros. Outra motivação foi a ampla aceitação de protocolos web que permitissem a interoperabilidade entre os serviços.

Google Maps é um bom exemplo. Após a publicação da API do serviço [16] por parte de Google, inúmeros *mashups* surgiram utilizando-se os serviços de Google Maps. Um dos mais populares *mashups* do serviço é *Chicago Crime* [17] que combina a base de dados sobre crime na cidade de Chicago e a API de Google Maps. Como resultado temos uma nova aplicação que mapeia visualmente os crimes ocorridos na referida cidade.

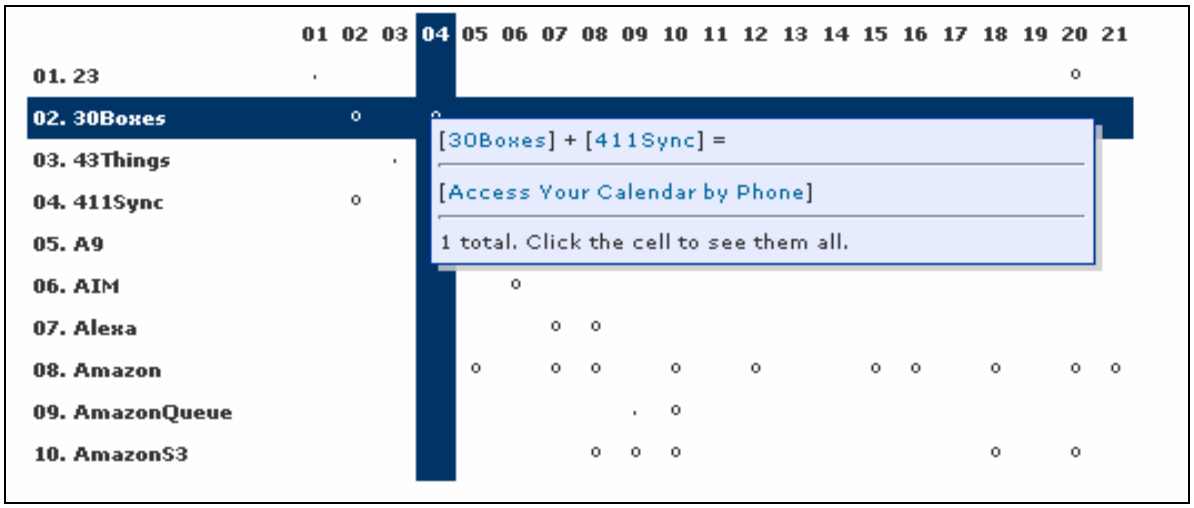


Figura 12. *Mashup matrix* disponível em [18].

Existem vários sites que disponibilizam informações a cerca de *mashups*, APIs públicas e protocolos. O mais conhecido deles é Programmable Web [18]. Quando da elaboração deste trabalho existiam mil e vinte três *mashups* cadastrados e um ritmo de crescimento de dois mashups novos incorporados à base por dia. Do total de mashups, cerca de 30% utilizavam alguma API de localização como Google Maps.

Do ponto de vista de arquitetura, um *mashup* é composto por três elementos distintos:

Provedor do conteúdo

É o responsável pela publicação da API. Sua importância é fundamental na composição da nova aplicação, pois de fato é quem fornece os serviços que servem de base para a construção da nova aplicação web. No caso de *Chicaco Crime*, por exemplo, Google Maps é um provedor de conteúdo, da mesma forma pode ser visto o Departamento de Polícia de Chicago. Com o sentido de facilitar a comunicação, provedores de conteúdo devem publicar seus serviços utilizando protocolos web conhecidos como REST [19] ou SOAP [20].

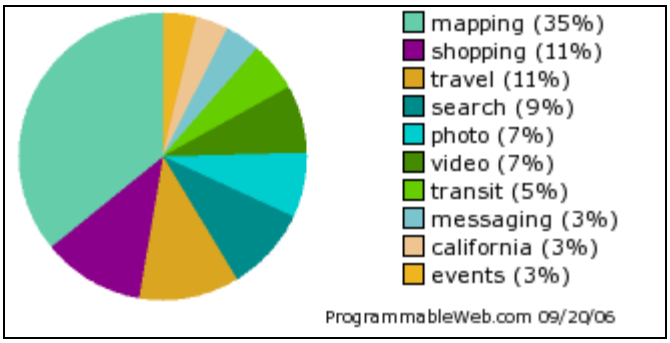


Figura 13. Mashups agrupados por tags.

O mashup

É onde a aplicação está hospedada. Representa a aplicação propriamente dita. É quem de fato implementa a lógica da aplicação web e faz a integração entre os diversos utilizados.

Navegador web

Responsável pela renderização da aplicação e fornece os mecanismos que tornam possível a interação com o usuário.

Protocolos web e mashups

Nesta seção serão introduzidos os conceitos relativos aos principais protocolos utilizados para comunicação entre os diversos serviços que podem compor os *mashups*. É importante que estes protocolos sejam independentes de plataforma para garantir a operabilidade entre os serviços.

SOAP

SOAP é um acrônimo para *Service-Oriented Acess Protocol*. É uma das principais tecnologias que compõe o Paradigma de *Web Services.* Anteriormente, o protocolo denominava-se *Simple Object Acess Protocol*, porém foi atualizado com a mudança de foco de sistemas baseados em objetos para interoperabilidade de sistemas. SOAP usa XML [22] para encapsular as mensagens que são trocadas em geral via protocolo HTTP [23]. SOAP marca uma evolução de um outro protocolo chamado XML-RPC [24]. Google é um bom exemplo de empresas que publicam seus serviços usando SOAP.

```
class CodeController < ApplicationController
  def googletest
     yourkey = 'YOUR GOOGLE DEVELOPER KEY'
     @yourquery = 'SEARCH TEXT'
     XSD::Charset.encoding = 'UTF8'
     googleurl = "http://api.google.com/search/beta2"
     urn = "urn:GoogleSearch"
     driver = SOAP::RPC::Driver.new(googleurl, urn)
     driver.add_method('doGoogleSearch', 'key', 'q',
        'start', 'maxResults', 'filter', 'restrict',
        'safeSearch', 'lr', 'ie', 'oe')
     @result = driver.doGoogleSearch(yourkey,
        @yourquery, 0, 3, false, '', false, '', '', '')
  end
```

Figura 14. Exemplo de acesso SOAP à API de Google em Rails.

REST

REST é considerada por muitos a arquitetura de web services mais simples. A palavra é um acrônimo para *Representational State Transfer*. As requisições do protocolo se parecem bastante com requisições HTTP simples. Aplicações que usam o protocolo fazem uma consulta via HTTP passando os parâmetros na URL [25] e recebem como resposta da parte do servidor um documento, em geral XML. As interfaces de REST estão limitadas a requisições HTTP, como *get()* e *post()*. Em geral os serviços disponibilizados por Yahoo usam REST.

```
class CodeController < ApplicationController
  def yahootest
    query = CGI.escape("SEARCH TEXT")
    yahookey = "YOUR YAHOO DEVELOPER KEYs
    url = "http://api.search.yahoo.com/" +
      "WebSearchService/V1/webSearch?" +
      "appid=#{yahookey}&query=#{query}" +
        "&results=3&start=1"
    result = Net::HTTP.get(URI(url))
    @doc = REXML::Document.new result

  end
end
```

Figura 15. Exemplo de acesso à API de Yahoo usando Rails.

Políticas de mashups

Em geral as empresas que publicam APIs utilizam de mecanismos para se proteger do uso indevido dos serviços que as mesmas disponibilizam. Estas restrições podem comprometer o desenvolvimento de *mashups* do ponto de vista de negócio para terceiros.

Os principais mecanismos utilizados por provedores de conteúdo para evitar uso abusivo de seus serviços são: identificação única da aplicação, restrição de tráfego, restrições de certas APIs para uso comercial, obrigatoriedade de autenticação dos usuários do *mashup* no sistema da empresa provedora de conteúdo.

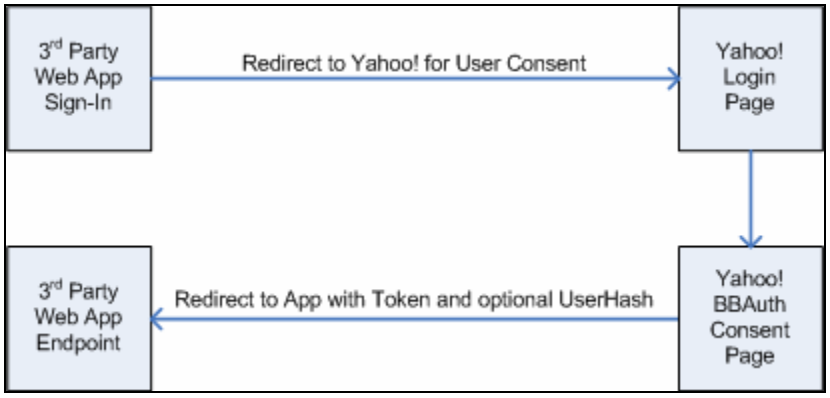


Figura 16. Autenticação de *mashups* construídos com serviços Yahoo.

Tomemos como exemplo de desenvolvimento de um *mashup* usando os serviços de Yahoo. Inicialmente é necessário que o desenvolvedor cadastre sua aplicação na base de dados da empresa. Uma vez cadastrada, a aplicação recebe um identificador que passar a fazer parte do processo de autenticação em algumas requisições de uso de API. Em alguns casos, Yahoo também limita acessos em relação a *mashups* considerados comerciais. Estas regras e restrições em geral são impostas por todos os grandes provedores de conteúdo como Yahoo, Google e Microsoft[38]. Na época que este trabalho foi concebido os acessos à recém-criada API de busca do Yahoo eram limitados a cinco mil por dia, por exemplo.

Um caso de sucesso recente na web em termos de *mashups* é Salesforce [26]. A empresa desenvolve sistemas de gerenciamento de relacionamento com o cliente, chamados de CRM. Possui concorrentes de renome como Microsoft [27], Oracle [28] e SAP [29]. Salesforce conseguiu um diferencial em relação a seus concorrentes transformando seus produtos em uma grande plataforma programável de CRM. A empresa abriu esta plataforma para o desenvolvimento de aplicações de terceiros e na época em que este trabalho foi escrito já contava com mais de 350 soluções relacionadas em seu portifólio. As aplicações são categorizadas e possuem uma vitrine virtual [30], através da qual clientes podem assinar os serviços da empresa. Atualmente, Salesforce se especializou em soluções *on demand*, com baixo custo de implantação e manutenção comparado aos concorrentes mais fortes, além de adaptação a necessidades exclusivas de cada cliente.

Tecnologias que habilitam a web 2.0

Nesta seção serão descritas as principais tecnologias que foram utilizadas na implementação da aplicação *All My Questions*. Entretanto, isto não nos permite dizer que são as únicas tecnologias que habilitam o desenvolvimento de aplicações web 2.0, seja no nível de linguagem, ferramentas ou *frameworks*. É interessante enfatizar que web 2.0 é algo muito mais conceitual que tecnológico propriamente dito. No entanto, o conhecimento tecnológico é tão importante quanto o conhecimento teórico a respeito do assunto.

Ajax

Em Fevereiro do ano de 2005, Jesse James Garrett escreve o artigo que é considerado a primeira citação para o termo *Ajax* [31]. No citado artigo, Garret defende que aplicações web estavam caminhando rumo a fornecer uma experiência mais rica para usuários. Experiência esta semelhante à obtida através

de aplicações desktop. Um dos responsáveis por este fato seria um conjunto de tecnologias que ele agrupou sob o nome de ajax.

Na época em que Garrett escreveu seu artigo, Google já havia lançado algumas aplicações web que faziam uso intenso do que ele chamara de Ajax. As aplicações eram Google Suggest [32] e Google Maps [33] são dois exemplos. De fato, Garret as usa para exemplificar esta nova experiência a qual defende.

Figura 17. Interface de Google Suggest.

A definição de Ajax não é simples. Isto porque Ajax em si não é uma tecnologia e sim um conjunto de tecnologias agrupadas em uma “definição guarda-chuva”, poderíamos assim dizer. Do ponto de vista tecnológico Ajax é composto pelas seguintes tecnologias:

- XHTML ou HTML juntamente com CSS para formatação;
- DOM [63] acessado pelo cliente via linguagem de script,
- XML para manipulação e troca de dados;

É importante destacar que todo o ferramental tecnológico necessário ao surgimento de Ajax como grupo de tecnologias já estava presente na web há algum tempo. O objeto XMLHttpRequest já estava disponível nos navegadores da

Microsoft desde 1999, por exemplo. Há também outros modelos de comunicação assíncrona com servidor. Entretanto, foi a aceitação dos padrões já existentes que habilitou o desenvolvimento de aplicações *ajax enabled* de forma escalável. Outro destaque a ser feito é que o maior impacto trazido por Ajax não foi no modelo de comunicação, porém no modelo de navegação do usuário.

AJAX = *Asynchronous JavaScript* + XML

O modelo clássico de navegação pode ser esquematizado da seguinte forma: o usuário interage de alguma forma com a página a qual gera uma requisição para o servidor; este processa os dados e retorna o conteúdo para o cliente que o renderiza. Em termos práticos, este esquema prevê a necessidade de processamento de toda a página HTML mesmo para pequenas requisições. Com o uso de Ajax, é inserido o conceito de micro transações que são elementos assíncronos. Estas conseguem melhorar a interatividade com o usuário, pois permitem que o usuário continue interagindo com a página enquanto uma entidade intermediária chamada *Ajax Engine* cuida que estas transações sejam terminadas com êxito (figura 18). A atuação desta entidade é de fundamental importância. Os eventos gerados pelo usuário que não necessitam de processamento por parte do servidor, a exemplo de atuação de um dado em memória, são resolvidos pelo *engine*. Caso haja a necessidade de resposta por parte do servidor, é então feita uma requisição assíncrona usando XMLHttpRequest e XML que é tratada assim que o servidor transmitir a resposta. Todo este processo é feito sem o conhecimento do usuário, permitindo ao mesmo experimentar uma experiência mais rica em termos de navegação que o modelo clássico que impõe muitas interrupções, uma vez que o usuário necessita aguardar a resposta do servidor na maioria das vezes que uma transação é feita.

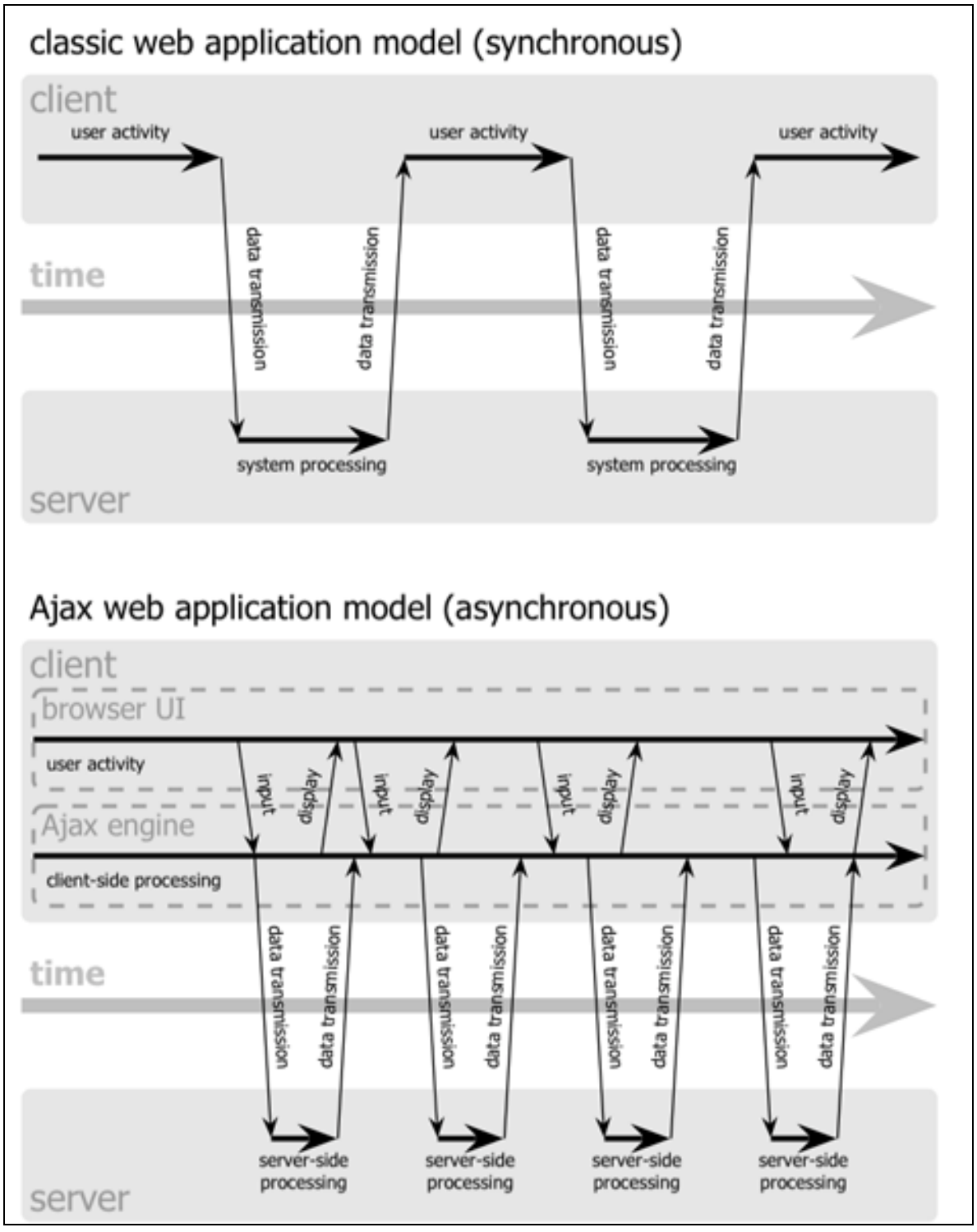


Figura 18. Comparativo entre modelos de aplicação web clássico e Ajax.

Ajax provê um consumo racional de banda caso a abordagem seja aplicada com cautela. O uso desmedido de micro transações em geral acarreta sobrecarga no lado do servidor, aumentando o tempo de resposta por parte do servidor e, às vezes, inviabilizando temporariamente o serviço. Muitos exemplos de aplicações de Ajax em páginas, informações sobre arquiteturas e padrões de projeto podem ser encontrados em Ajax Patterns [34].

A interação conseguida com Ajax se deve em grande parte a um objeto chamado XMLHttpRequest, inicialmente o implementado pela Microsoft para a versão 5.0 do seu navegador, o *Internet Explorer*. Atualmente este já possui implementação nativa nos principais navegadores disponíveis na internet. O padrão da W3C para o objeto [35] está disponível desde Abril do ano de 2006 e tem como objetivo melhorar a interoperabilidade, visto que as implementações correntes do objeto nos diversos navegadores ainda não operam totalmente entre si.

Uma tecnologia importante dentro do contexto de Ajax é Javascript. A linguagem de script pode ser considerada a “cola” que integra as diversas tecnologias que compõe Ajax. Por ser considerada por muitos rebuscada e de difícil manutenção, iniciativas surgiram buscando minimizar o trabalho de desenvolvedores e empresas no desenvolvimento de aplicações *Ajax enabled*. A seguir estão listadas algumas destas iniciativas, juntamente com uma breve descrição do foco.

Prototype

Prototype[36] é um *framework* javascript distribuído sob a *MIT License* [37]. O objetivo do projeto é fornecer ferramentas que facilitem a criação de páginas web dinâmicas. Quando da construção este trabalho, o *framework* estava na versão 1.4.0, tendo sido incorporado a Rails. Prototype também serve de base para a implementação de outros *frameworks* javascript, como Scriptaculous.

Dojo Toolkit

Semelhante a Prototype, Dojo é um toolkit javascript que visa auxiliar desenvolvedores na programação visual de páginas web. Dojo, do ponto de vista de componentes, é mais voltado para *widgets*[6] que Prototype. Existem editores de texto simples, formulários, calendários e menus que podem ser utilizados de forma simples. O *framework* é distribuído sob a Licença BSD [39].

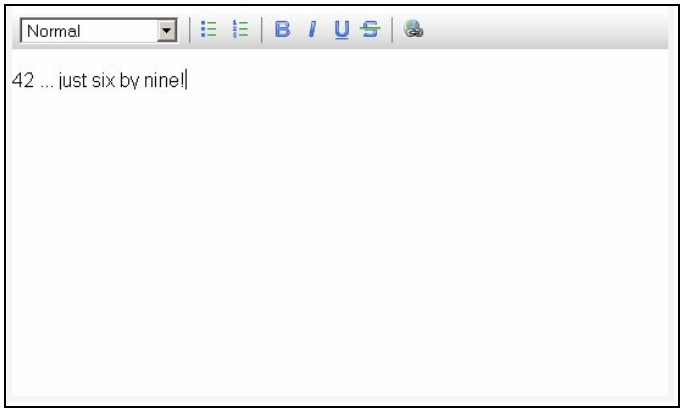


Figura 19. Componente de edição de texto de Dojo.

Qooxdoo

Qooxdoo [40] é um *framework* javascript cujo objetivo é aproximar a experiência do usuário de aplicações web à experiência conseguida com softwares d*esktop*. Para isto fornece uma série de widgets e efeitos visuais que se assemelham ao sistema operacional da Microsoft, Windows. Visualmente, qooxdoo é mais rico que Prototype e Dojo, entretanto as aplicações em geral são mais custosas do ponto de vista de desempenho. Qooxdoo é um projeto *open source*.

[6] Componentes de interface gráfica

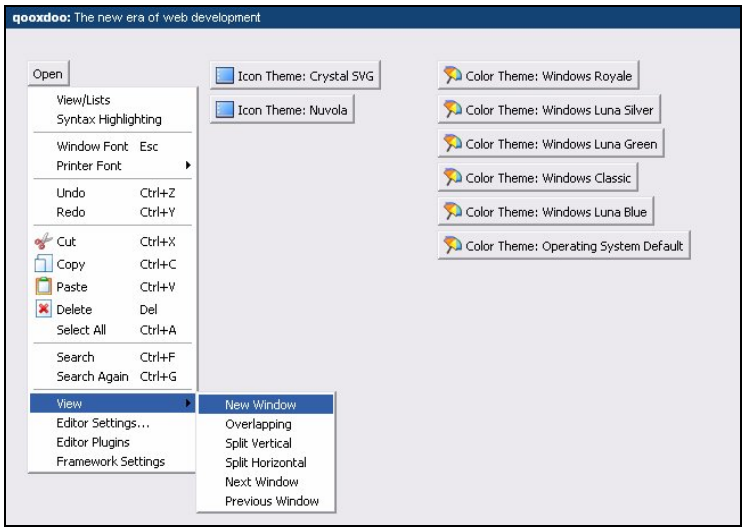


Figura 20. Interface de menu de Qooxdoo.

Ruby

Ruby é uma linguagem interpretada e orientada a objetos [41]. Em 1995, Yukihiro Matsumoto , criador da linguagem, fez o primeiro release de ruby. Desde então, a linguagem vem crescendo rapidamente no mundo. Mais recentemente, este crescimento acelerou-se grande parte devido à popularização de Rails [42] que é o principal software escrito na linguagem. Atualmente a comunidade da linguagem possui um grupo muito ativo com listas de discussão, blogs e grupos de usuários.

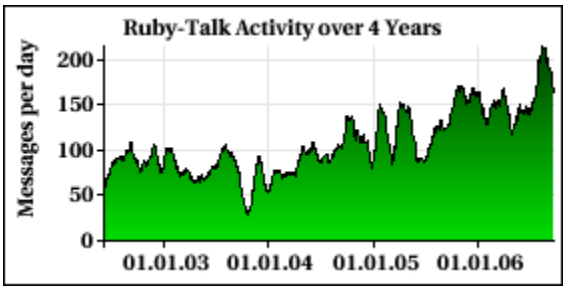


Figura 21. Crescimento da lista de discussão de ruby

Nesta seção procuraremos abordar ruby de uma forma que seja de fácil entendimento para programadores Java. Há mais diferenças que semelhanças, o

que não torna necessariamente o entendimento de ruby confuso. Não é objetivo desta seção tratar questões como desempenho e produtividade comparando-se as duas linguagens, antes fornecer informações para o entendimento dos elementos básicos da linguagem.

Similiaridades:

- A memória em ruby é gerenciada por um *Garbage Collector*;
- Existem métodos *public*, *protected* e *private*;
- Objetos são fortemente tipados;
- Existe uma ferramenta para gerar documentação de ruby que é bem semelhante à Javadoc chamada RDoc;

Diferenças:

- Ruby é uma linguagem interpretada, não havendo a necessidade de compilar-se o código;
- Interfaces gráficas podem ser utilizadas através de *GUI Toolkits*, como WxRuby [43], FXRuby [44] ou Ruby-GNOME2 [45];
- Parênteses são opcionais e geralmente são omitidos;
- As chaves utilizadas na declaração de métodos ou classes são substituídas pela palavra-chave *end*;
- Ruby é dinamicamente tipada;
- Ruby não possui tipos primitivos, tudo em ruby é encarado como objeto. Por exemplo, números como 42 são objetos em ruby;
- Nomes de variáveis em ruby são somente *labels*, não existem tipos associados a eles;
- O construtor sempre é um método chamado *initialize*, diferindo de Java onde este é o nome da classe a qual ele pertence;

Rails

A definição clássica de Rails pode ser encontrada em seu livro oficial [46]. Em poucas linhas, Rails é um *framework open source* desenvolvido em ruby que tem como objetivo auxiliar o desenvolvimento de aplicações web que fazem uso ostensivo de Banco de Dados e utilizam o modelo MVC [47].

Rails é uma iniciativa de um dinamarquês chamado David Heinemeier Hansson[6] que lançou a primeira versão do projeto em Julho de 2004. Entretanto, somente em fevereiro de 2005 Hansson decidiu compartilhá-lo com desenvolvedores, sob a *MIT License* [48]. Tal tipo de licenciamento de software oferece, entre outras, a possibilidade de se copiar, distribuir e modificar o seu código fonte. A partir de então, Rails é mantido por um grupo de contribuidores que participam ativamente de um *core team.*

O desenvolvimento em Rails é ágil. Em [46] David Hansson afirma que Rails é projetado para ajudar desenvolvedores a responder rapidamente a mudanças. Desta forma, o *framework* também tem como objetivo possibilitar que se possa prototipar e evoluir o sistema de maneira eficiente. Responder rapidamente a mudanças implica um processo de desenvolvimento em que há maior interação entre desenvolvedores e clientes. Outro importante destaque é a integração do framework com bibliotecas que facilitam o uso de Ajax e *templates* HTML.

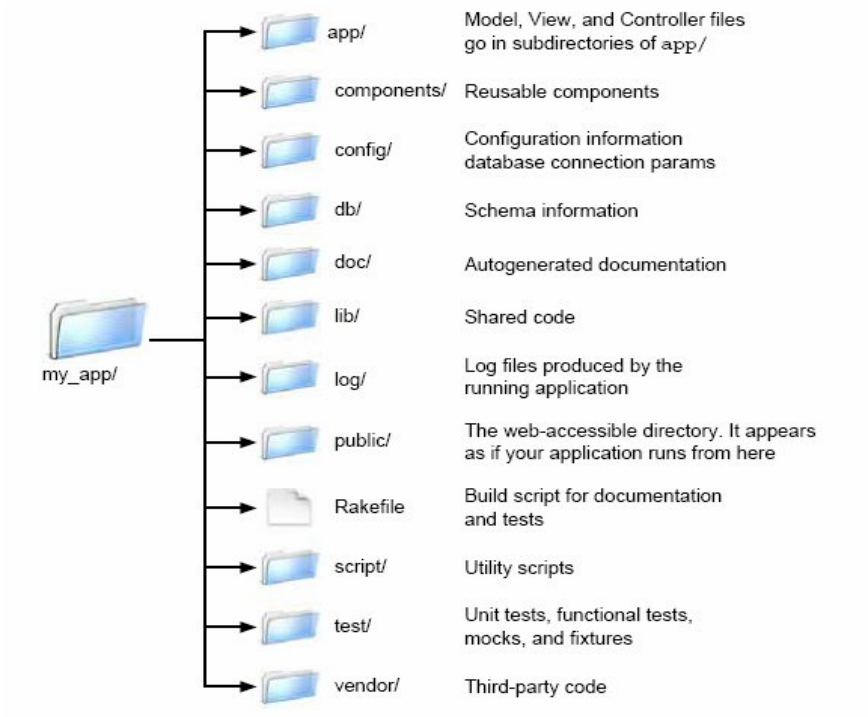


Figura 22. Estrutura de diretórios de uma aplicação Rails

Arquitetura das aplicações Rails

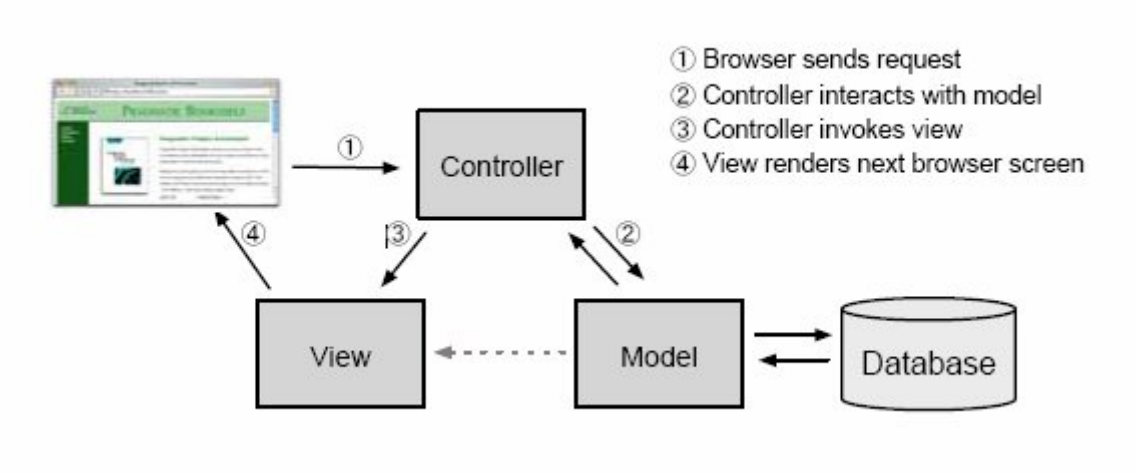


Figura 23. O modelo MVC e Rails

Aplicações desenvolvidas utilizando-se Rails baseiam-se no modelo MVC por imposição do próprio *framework*. A abreviatura corresponde à expressão em

inglês *Model-View-Controller*. É um modelo bastante eficiente e conhecido para desenvolver aplicações web que divide a aplicação em três camadas principais onde cada parte desempenha um papel específico. São elas:

Model

Responsável pela manutenção do estado da aplicação íntegro. Esta camada responde pela persistência dos dados tanto em transações transientes quanto em transações que utilizam acessos a bases de dados. Parte da validação dos dados da aplicação é feita nesta camada, para reforçar a consistência das regras de negócio. Esta camada está associada principalmente à persistência dos dados.

View

Camada que implementa a interface com o usuário, normalmente baseando-se nos dados obtidos na modelo da aplicação. Tal camada torna possível a formatação dos dados presentes do modelo de forma a ser acessível ao usuário do sistema. Camada associada principalmente à interface com o usuário.

Controller

O controlador é responsável por orquestrar a aplicação. Recebe uma entrada (geralmente do usuário), interage com o modelo para o processamento dos dados e depois com a camada de apresentação para a visualização da informação gerada neste processo.

Rails é intencionalmente um framework MVC. A tríade Model-View-Controller evita que desenvolvedores misturem as camadas de aceso aos dados, apresentação e negócio. Este tipo de mistura acaba por tornar o código confuso e de difícil manutenção, pois não existem divisões visíveis de funcionalidades entre as várias partes do sistema. Agilizar o desenvolvimento é um dos objetivos de Rails, juntamente com a clareza e concisão das aplicações desenvolvidas. A arquitetura base do framework demonstra claramente este objetivo.

Web 2.0 e desenvolvedores

Esta seção demonstra as iniciativas que estão sendo tomadas por grandes empresas de desenvolvimento de aplicações web que possibilitam a implementação de aplicações por parte de terceiros. Do ponto de vista de desenvolvimento, estas iniciativas são em geral ferramentas, *frameworks*, padrões de interface e especificações para implementação de *mashups*. De fato, com o advento da idéia de programar a web, muitas empresas começaram a incentivar terceiros a programarem utilizando suas plataformas. Aplicações inovadoras e interessantes têm surgido a partir deste suporte provido por tais empresas. A seguir são mencionadas algumas delas, destacando-s que estas são apenas exemplos, não correspondendo à totalidade das empresas envolvidas com tal idéia.

Yahoo Developer

Yahoo foi uma das primeiras empresas a incentivar o uso de seus serviços como uma plataforma programável. A empresa possui um portal para desenvolvedores interessados em utilizar tal plataforma para programar seus próprios serviços disponibilizando tutoriais e padrões de projeto. Yahoo também disponibiliza no referido portal as informações necessárias para o desenvolvimento de *mashups*. As duas principais contribuições de Yahoo para os desenvolvedores certamente são a sua ampla biblioteca de padrões de interface que estão sendo usados por vários desenvolvedores e sua biblioteca de interface chamada Yahoo UI [49].

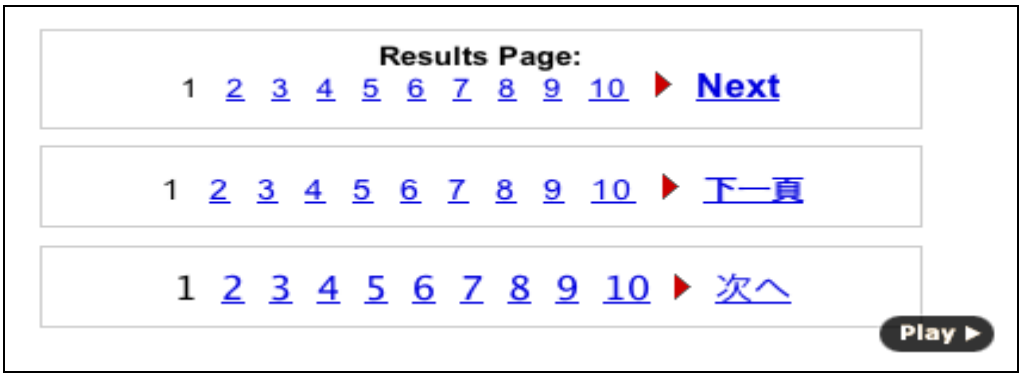


Figura 24. Exemplo de padrão de interface da biblioteca de Yahoo.

A biblioteca de padrões de interface de Yahoo auxilia desenvolvedores a lidarem com problemas comuns em termos de implementação de interface levando-se em conta enriquecer a experiência com o usuário e melhorar a usabilidade da aplicação desenvolvida. Os principais problemas listados na biblioteca são questões relacionadas a paginação em buscas, paginação com itens e disposição de componentes na página. Outros padrões são disponibilizados como efeitos de transição, por exemplo.

```
view plain | print | ?
1 <script>
2 var transaction = YAHOO.util.Connect.asyncRequest('GET', sUrl, callback, null);
3 </script>
```

Figura 25. Exemplo de utilização de um *YUI Manager*.

Yahoo UI é um conjunto de utilitários escritos em javascript e disponibilizados por Yahoo sob a licença BSD [50]. A biblioteca possui dois grupos de componentes: *YUI Controls* e *YUI Utilities*. Os *controls* em geral são elementos de interface com o usuário a exemplo de menus e calendários. Os *utilities* por sua vez são facilitadores para tratamento de eventos e requisições. Com o auxílio destes componentes desenvolvedores podem criar aplicações ricas do ponto de vista de interface e com um nível de trabalho menor que se lidassem diretamente com as nuances de javascript.

Google Developer

Google Developer [51] é o portal de Google para desenvolvedores. De fato, a empresa tem um papel importante quando se trata de aplicações web com experiência rica para usuários, visto seu pioneirismo em aplicações como Gmail [52] que provê uma experiência de e-mail integrada com o messenger da empresa, Gtalk [53]. Sem o suporte de Ajax aplicações como Gmail se tornariam inviáveis no presente momento em termos de tecnologia.

Da mesma forma que Yahoo, Google possui um conjunto de ferramentas para auxiliar o desenvolvimento de aplicações web ajax enabled.chamado Google Web Toolkit [54]. O framework é totalmente desenvolvido em Java e distribuído sob a Apache License 2.0. Os objetivos principais do toolkit são abstrair dos desenvolvedores problemas com compatibilidade da aplicação com os diversos navegadores do mercado, detalhes de implementação relacionadas a javascript, além de fornecer um conjunto de componentes reusáveis e prover uma suíte de testes. A popularidade deste framework se deve justamente ao fato de prover mecanismos que permitem que desenvolvedores criem interfaces ricas utilizando uma linguagem de alto nível, no caso Java. O compilador chamado GWT Compiler se responsabiliza por compilar o código em Java para javascript. Google Web Toolkit é compatível com praticamente toda a semântica de Java 1.4.

Conclusões e trabalhos futuros

Principais contribuições

Este trabalho tem duas vertentes. A primeira ligada à parte teórica, relativa aos principais conceitos relacionados ao termo “web 2.0”. A segunda ligada à parte tecnológica, que foi o desenvolvimento de uma aplicação utilizando o *framework* rails que validasse tais conceitos.

Do ponto de vista conceitual, a principal contribuição dada por este trabalho foi a descrição dos principais conceitos relacionados a web 2.0 com aplicações práticas. O conceito ainda não é bem entendido de forma que um estudo que contemple boa parte das idéias relacionadas ao termo é bastante importante.

Sob a perspectiva de desenvolvimento, o projeto tem várias contribuições. A primeira delas é uma aplicação totalmente funcional que pode servir como base para a implementação de inúmeras aplicações web 2.0. Outra contribuição importante é o agrupamento de várias bibliotecas e rotinas que auxiliam o desenvolvimento destas aplicações.

Localmente, o trabalho pode se tornar referência para os vários integrantes do grupo de discussão de Web 2.0 do Centro de Informática da Universidade Federal de Pernambuco.

Dificuldades encontradas

Todo projeto audacioso tem um conjunto grande de riscos associados.

Uma dificuldade encontrada foi a definição de um escopo razoável para o projeto que, além de facilitar a compreensão dos principais aspectos relativos ao tema do trabalho, pudesse contribuir de forma prática na formação do corpo de conhecimentos a respeito do assunto. A ausência de informações sobre o tema na literatura clássica de Computação também foi algo digno de preocupação. O fato de o conhecimento a respeito das potencialidades de web 2.0 ser ainda incipiente

dificultou bastante o desenvolvimento deste projeto. De uma forma geral, a ausência de informações a respeito do assunto que fossem dignas de crédito foi um fator complicador. A filtragem destas informações consumiu grande parte do tempo disponível para compilação deste trabalho.

Trabalhos futuros

É inegável o caráter exploratório deste projeto. Vários aspectos importantes relativos aos assuntos aqui listados não foram apresentados por motivos de escopo. Existe um conjunto substancial de problemas a serem solucionados em relação ao tema, tanto na área de desenvolvimento, quanto na área conceitual e de negócios. Abordar individualmente cada um destes temas é um desafio.

A definição de uma metodologia de desenvolvimento para aplicações que contemplem os conceitos de web 2.0 é um exemplo de trabalho futuro. Com metodologias bem definidas pode-se ganhar tempo precioso durante o ciclo de desenvolvimento.

Estudos relacionados a testes, escalabilidade de aplicações web e otimização de uso de recursos como processamento e banda são também trabalhos interessantes para um futuro próximo.

A evolução deste documento de forma a abranger os conceitos que certamente surgirão à medida que exploramos o universo de web 2.0 é um trabalho absolutamente imprescindível.

Considerações finais

Este trabalho apresenta o tema web 2.0 em linhas gerais. É importante destacar que existem outras tecnologias e conceitos citados ao longo deste documento que contribuem para a compreensão do mesmo. Alguns esforços foram desprendidos no sentido de desvincular o conceito de Ajax do conceito de web 2.0. O mesmo ocorreu com Rails. A abordagem prática presente no trabalho pode auxiliar bastante o desenvolvimento de uma aplicação web 2.0, entretanto aplicações comerciais necessitam de estudo mais amplo, contemplando outros

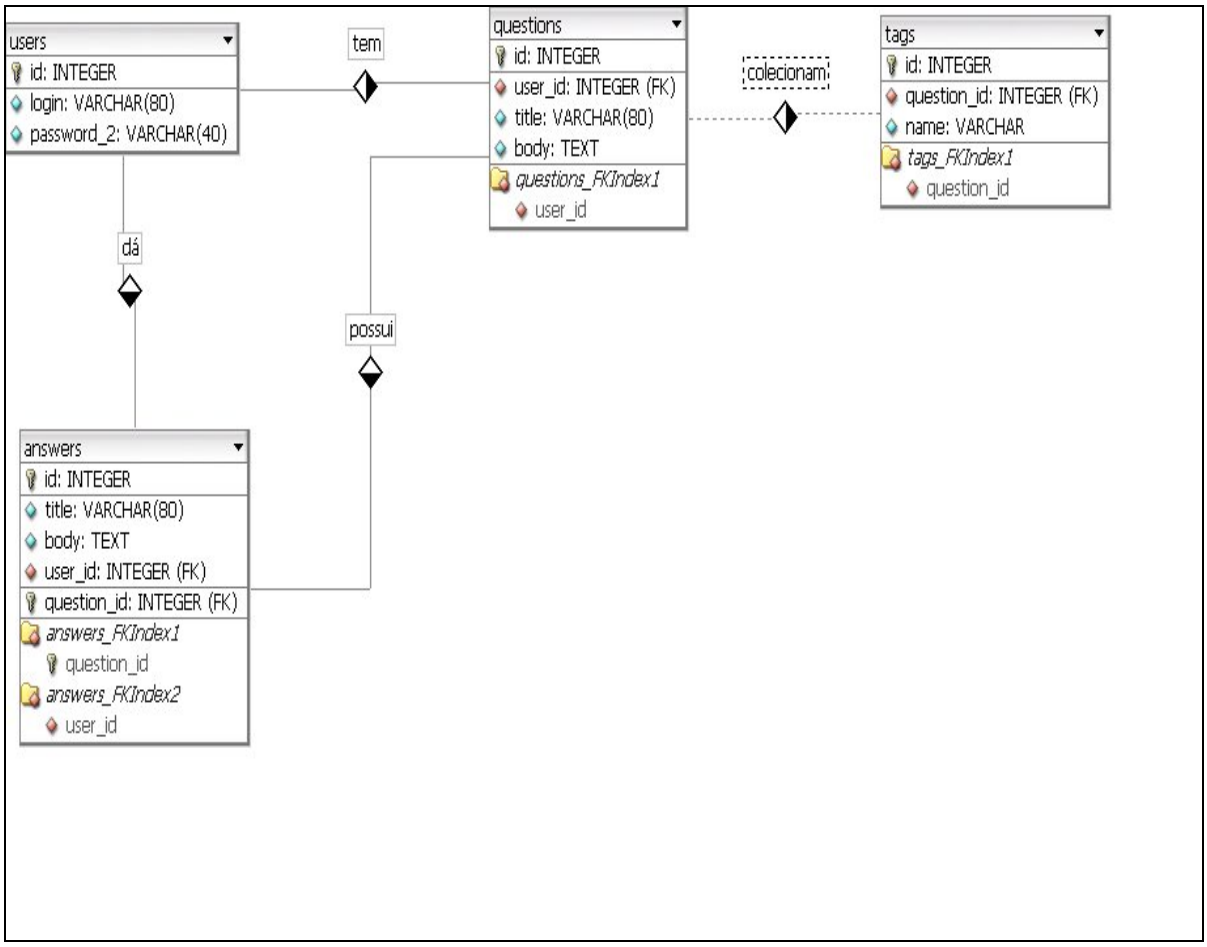
aspectos não abordados diretamente neste trabalho. O ferramental tecnológico apresentado ao longo do projeto não é a única abordagem possível para o desenvolvimento de aplicações web 2.0, assim como também as bibliotecas citadas. É um fato relevante que não é a tecnologia em si que define se uma aplicação é web 2.0 ou não, mas os conceitos que tal aplicação utiliza e como estes conceitos se relacionam.

Anexo 1 – questions_controller.rb

```
class QuestionsController < ApplicationController

  layout "questions", :except => [:update_tags, :update_question]
  before_filter :login_required, :except => [:show, :index ,:list]

# ------------------------------------------------------------------------
  def check_user(action = "show")
    if   @question   and   @question.user   !=   @session['user']   and
@session['user'].id != 1
      flash[:notice] = 'Question invalid or user not authorized'
      redirect_to :controller => "questions", :action => action
    end
  end
# ------------------------------------------------------------------------
  def index
    list
    render :action => 'list'
  end
# ------------------------------------------------------------------------
  verify :method => :post, :only => [ :destroy, :create, :update ],
         :redirect_to => { :action => :list }
#  ------------------------------------------------------------------------
def list
    @question_pages, @questions = paginate :questions, :per_page => 10
  end
#  ------------------------------------------------------------------------
def show
    @question = Question.find(params[:id])
  end
#  ------------------------------------------------------------------------
def new
    @question = Question.new
  end
# ------------------------------------------------------------------------
  def create
    @question = Question.new(params[:question])
    @question.user = @session['user']
    if @question.save
      flash[:notice] = 'Question was successfully created.'
      redirect_to :action => 'list'
    else
      render :action => 'new'
    end
  end
# ------------------------------------------------------------------------
  def edit
    @question = Question.find(params[:id])
    check_user
  end
```

```
 def destroy
    check_user
    Question.find(params[:id]).destroy
    redirect_to :action => 'list'
  end
# ----------------------------------------------------------------------------
  def index
    redirect_to :action => 'list'
  end
# ----------------------------------------------------------------------------
  def my_questions
    if @session['user'] == nil
      flash[:notice] = 'Invalid user'
      redirect_to :action => 'list'
    end
    @results = Question.find_by_user(@session['user'].login)
  end
# ----------------------------------------------------------------------------
  def questions_by_user
    @results = Question.find_by_user(params[:login])
  end
# ----------------------------------------------------------------------------
  def update_tags
    @question = Question.find(params[:id])
# ----------------------------------------------------------------------------
    check_user("show")

    @question.tag_with params[:value]
    @tag_string = params[:value]
  end
# ----------------------------------------------------------------------------
  def update_question
    @question = Question.find(params[:id])
    check_user("show")
    if params[:title] then # title?
      @question.title = params[:value]
    else
      @question.body = params[:value]
    end
    @question.save
    @string = params[:value]
  end
end
```

Anexo 2 – Esquema da base de dados de All My Questions



Anexo 3 – AccountControler.rb

```
class AccountController < ApplicationController
  model   :user
  layout "questions"

  def login
    case @request.method
      when :post
        if @session['user'] = User.authenticate(@params['user_login'],
@params['user_password'])

          flash['notice']  = "Login successful"
          redirect_back_or_default :action => "welcome"
        else
          @login    = @params['user_login']
          @message  = "Login unsuccessful"
      end
    end
  end

  def signup
    case @request.method
      when :post
        @user = User.new(@params['user'])

        if @user.save
            @session['user'] = User.authenticate(@user.login,
            @params['user']['password'])

      flash['notice']  = "Signup successful"
          redirect_back_or_default :action => "welcome"
        end
      when :get
        @user = User.new
    end
  end

  def delete
      if @params['id'] and @session['user'] and @session['user'].id ==
      @params['id']

      @user = User.find(@params['id'])
      @user.destroy
    end
    redirect_back_or_default :action => "welcome"
  end

  def logout
    @session['user'] = nil
  end

  def welcome
    redirect_to :action => "list", :controller => "questions"
  end
```

Anexo 4 – RssController.rb

```
class RssController < ApplicationController
    def all
        @questions = Question.find(:all)
        render_without_layout
        @headers["Content-Type"] = "application/xml; charset=utf-8"
    end

    def mine
        user = @session['user']
        params[:user] = user.login unless user == nil
        foruser unless user == nil
        render :text => "User not logged in" unless user != nil
    end

    def user
      @questions = Question.find_by_user(params[:id])
      render_without_layout
      @headers["Content-Type"] = "application/xml; charset=utf-8"
    end

end
```

Anexo 5 – Template para RSS

```
xml.instruct!

xml.rss "version" => "2.0", "xmlns:dc" =>
"http://purl.org/dc/elements/1.1/" do
 xml.channel do

   xml.title       "All Questions RSS"
   xml.link        url_for(:only_path => false, :controller => 'rss')
   xml.description "rss for questions"

   @questions.each do |q|
     xml.item do
       xml.title       q.title
       xml.link        url_for(:only_path => false, :controller =>
'questions', :action => 'show', :id => q.id)
       xml.description q.body
       xml.guid        url_for(:only_path => false, :controller =>
'questions', :action => 'show', :id => q.id)
     end
   end

 end
end
```

Anexo 6 – Web Services em Rails

```
#Definicao da API  ( app/apis/rpc_api.rb )

class RpcAPI < ActionWebService::API::Base

  #definicao da assinatura do metodo
  api_method :new_question,
    :expects => [{:username => :string},
                 {:title => :string},
                 {:body => :string},
                 {:tags => :string}],
    :returns => [:string]
end




#Implementação API ( app/controllers/rpc_controller.rb )
# This class is the controller in charge of handling remote procedure
calls.

class RpcController < ApplicationController
  wsdl_service_name 'Deepthough'
  web_service_api RpcAPI

  # Creates a new question. The question owner will be the user with the
given username.

def new_question(username, title, body, tags)
    question = Question.new
    question.title = title
    question.body = body
    question.tags = tags
    user = User.find(:first, :conditions => ["login = ?", username])
    question.user = user
    question.save
    "ok"
  end
end
```

Referências

[1] *O'reilly Media*. Portal da O'relly. http://www.oreilly.com/, acessado em Outubro de 2006.

[2] *Media Internacional*. http://cmp.com/, acessado em Outubro de 2006.

[3] *Web 2.0 Conference.* Conferência Mundial sobre web 2.0. http://www.web2con.com/web2con/, acessado em Outubro de 2006.

[4] *What is the web 2.0.* Artigo de Tim O'reilly sobre web 2.0. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html, acessado em Outubro de 2006.

[5] *Flickr.* Serviço de Compartilhamento de imagens. http://www.flickr.com/ , acessado em Outubro de 2006.

[6] *Yahoo.* http://www.yahoo.com/, acessado em Outubro de 2006.

[7] *Delicious.* Serviço de *Social Bookmarking*. http://del.icio.us/, acessado em Outubro de 2006.

[8] *Recife Beat.* Pré-incubadora ligada ao Centro de Informática da Universidade Federal de Pernambuco. http://www.cin.ufpe.br/~beat/, acessado em Outubro de 2006

[9] *Yahoo Answers.* Serviço de perguntas e respostas de Yahoo. http://answers.yahoo.com/ , acessado em Outubro de 2006

[10] *Ruby On R*ails. Framework web. http://www.rubyonrails.org/, acessado em Outubro de 2006.

[11] *Rad Rails*. IDE para Rails. http://www.radrails.org/, acessado em Outubro de 2006.

[12] *Ferret.* Engenho de busca. http://ferret.davebalmain.com/trac, acessado em Outubro de 2006.

[13] *Lucene.* Engenho de busca. http://lucene.apache.org/ , acessado em Outubro de 2006.

[14] *Acts_as_ferret.* http://projects.jkraemer.net/acts_as_ferret/, acessado em Outubro de 2006.

[15] *Scriptaculous.* Biblioteca Javascript. http://script.aculo.us/, acessado em Outubro de 2006.

[16] *Api de Google Maps.* http://www.google.com/apis/maps/, acessado em Outubro de 2006.

[17] *Chicago Crime.* http://www.chicagocrime.org, acessado em Outubro de 2006.

[18] *Programmable web.* Portal com informações sobre mashups. http://www.programmableweb.com/, acessado em Outubro de 2006.

[19] MARSHALL, K. *Web Services on Rails*. O'Reilly, 2006.

[20] MARSHALL, K. *Web Services on Rails*. O'Reilly, 2006.

[21] *Mashups: The new breed of Web app*. Uma introdução a mashups. http://www-128.ibm.com/developerworks/library/x-mashups.html , acessado em Outubro de 2006.

[22] *XML Specification*. http://www.w3.org/XML/, acessado em Outubro de 2006.

[23] *HTTP Specification*. http://www.w3.org/Protocols/, acessado em Outubro de 2006.

[24] *XML-RPC.* http://www.xmlrpc.com/spec, acessado em Outubro de 2006.

[25] *URL*. http://www.w3.org/Addressing/, acessado em Outubro de 2006.

[26] *Salesforce*. http://www.salesforce.com/, acessado em Outubro de 2006.

[27] *Microsoft*. http://www.microsoft.com, acessado em Outubro de 2006.

[28] *Oracle.* http://www.oracle.com/, acessado em Outubro de 2006.

[29] *SAP.* Sofware de Gestão de Relacionamento com cliente. http://www.sap.com/solutions/business-suite/crm/index.epxm , acessado em Outubro de 2006,

[30] *AppExchange.* Portal de aplicações de Salesforce. http://www.salesforce.com/appexchange/, acessado em Outubro de 2006.

[31] Tate, B. *From Java To Ruby: Things Every Manager Should Know*. Pragmatic Bookshelf, 2006.

[32] *Google Suggest*. http://www.google.com/webhp?complete=1&hl=en, acessado em Outubro de 2006.

[33] *Google Maps.* http://maps.google.com/, acessado em Outubro de 2006.

[34] *Ajax Patterns.* Padrões Ajax. http://ajaxpatterns.org/, acessado em Outubro de 2006.

[35] *Xml RPC Specification*. http://www.w3.org/2000/03/15-XML-protocol-Viewpoints, acessado em Outubro de 2006.

[36] *Prototype.* Framework javascript. http://prototype.conio.net/, acessado em Outubro de 2006.

[37] *MIT License*. http://www.opensource.org/licenses/mit-license.php, acessado em Outubro de 2006.

[38] Fowler, C. *Rails Recipes*. Pragmatic Bookshelf, 2006.

[39] *Licença Dojo Toolkit*. http://dojotoolkit.org/foundation/#license, acessado em Outubro de 2006.

[40] *Qooxdoo*. Framework javascript. http://qooxdoo.org/, acessado em outubro de 2006.

[41] THOMAS, D.; FOWLER, C.; HUNT, H. *Programming Ruby: The Pragmatic* Programmer's Guide. 2a. Edição. Pragmatic Bookshelf, 2004.

[42] THOMAS, D., et al. *Agile Web Development with Rails: A Pragmatic Guide*. Pragmatic Bookshelf, 2005.

[43] *WxRuby Home. Toolkit* para desenvolvimento de interfaces gráficas em ruby. http://wxruby.rubyforge.org/wiki/wiki.pl, acessado em Outubro de 2006.

[44] *FxRubu Home.* http://www.fxruby.org/, acessado em Outubro de 2006.

[45] *FrontPage - Ruby-GNOME2 Project Website*. http://ruby-gnome2.sourceforge.jp/, acessado em Outubro de 2006.

[46] THOMAS, D., et al. *Agile Web Development with Rails: A Pragmatic Guide*. Pragmatic Bookshelf, 2005.

[47] THOMAS, D., et al. *Agile Web Development with Rails: A Pragmatic Guide*. Pragmatic Bookshelf, 2005 p. 9.

[48] *Licença do framework Rails*. http://www.opensource.org/licenses/mit-license.php, acessado em Outubro de 2006.

[49] *Yahoo UI.* http://developer.yahoo.com/yui/ *,* acessado em Outubro de 2006*.*

[50] *Licença Yahoo,.* http://developer.yahoo.com/yui/license.txt*,* acessado em Outubro de 2006.

[51] *Google Developer Portal*. http://code.google.com/, acessado em Outubro de 2006.

[52] *Gmail*. Serviço de e-mail do google. http://gmail.com/, acessado em Outubro de 2006.

[53] *Gtalk*. Comunicador instantâneo de Google. http://www.google.com/talk/, acessado em Outubro de 2006.

[54] *Google Web Toolkit.* Framework web desenvolvido por Google. http://code.google.com/webtoolkit/, acessado em Outubro de 2006.