

# Experience Report on Ruby on Rails

Florian DECKERT  
Department of Computer Science  
University of British Columbia  
Vancouver, B.C.

December 1, 2006

## **Abstract**

This work covers an evaluation of the web development framework Ruby on Rails. The evaluation criteria are the speed of the development process, ease of use and the reduction of writing redundant code compared to other web technologies. Base for the findings is the development of a web based photo album. Code examples taken from this application are presented to support the argumentation.

# 1 Introduction

In the last couple of years web based applications grew more and more popular. Availability, portability and safe storage of data may be some reasons for the success of the web approach.

Good development tools are essential in software engineering [1]. Of course, a web programmer wants to have a development framework that supports the development process and offers her ease of use. There is a multitude of technologies that aim to support the web programmer in her work developing web applications. A few examples are .NET, Java together with JSP and PHP. Every of these technologies are used for dynamic content generation and provide access to a database backend in order to fetch these contents. Access is done by invoking (self written) SQL queries through functions provided by these frameworks. That means that all CRUD (create, read, update, delete) operations have to be implemented by the programmer including SQL queries.

It is obvious that these CRUD operations are often more or less the same. Thus, it should be possible to automate the generation of the CRUD interface. A lot of tedious, repetitious work was already would have been done by the framework used.

Another rather tedious work is writing configuration files for the mapping between database and their representation within the application.

A recent technology that claims to get rid of a lots of these redundancies in code writing is “Ruby on Rails” [2]. According to the supporters of RoR developing web applications using this framework is therefore faster.

In this report I will try to answer the following questions. Does RoR result in a faster development of web applications? Do we get rid of writing redundant code? And finally, is it really easy to use, or can a web programmer not familiar with RoR get an easy start in this framework?

## 2 Setting

I worked with PHP, SQL, HTML and CSS, so to say the standard technologies for implementing web applications. I have no experience in coding Ruby and using RoR. Thus, I am a total beginner about this technology as most web developers might be.

In order to collect experience about RoR I decided on implementing a

small web application using RoR. A photo album might be a typical web application for nowadays since it needs a database for storing pictures, comments about the pictures, etc. and it can be implemented facilitating dynamic content generators. Thus, this could serve as a sandbox for trying out how well RoR supports the developer in her work.

In my work I used the following software<sup>1</sup> on MS Windows XP:

- Ruby 1.8.4 [5]
- Rails 1.2.5 [6]
- RMagick 1.14.0 [7]
- filecolumn [8]
- MySQL 4.1.9 [9]
- phpMyAdmin 2.6.1 [10]
- Firefox 2.0 [11]

Moreover, I used many resources of the RoR homepage [2] (including the API, FAQ and HowTo's) and a tutorial [12] for doing the first steps.

## 3 Experiences

### 3.1 Ruby

Ruby is a multi-paradigm language designed by Yukihiro Matsumoto [13]. It contains parts of Perl, Smalltalk, Eiffel, Ada and Lisp. Design Principles [14] are consistency: “I’ve tried to follow the principle of least surprise.”, conciseness: “A good servant should do a lot of work with a short oder” and flexibility: “...a language should not restrict human thought.”. Matsumoto aimed to make this language being natural and “fun” for programmers. Thus, the language is supposed to be powerful, behave intuitively and support quick development.

---

<sup>1</sup>At first I used Instantrails 1.3a [3] but then upgraded and added a few components via RubyGems [4].

```
1 # http://www.ruby-lang.org shows a funny example:
2
3 cities = %w[ London Oslo Paris Amsterdam Berlin ]
4 visited = %w[ Berlin Oslo ]
5
6 puts "I still need to visit the following cities:", cities -
    visited
```

My experience is that these claims were not said without a reason. The example above should give the reader an idea that ruby indeed behaves natural (Maybe even close to “do what I mean”?). Moreover, Ruby offers a lot of “syntax fluff” accommodating people coming from almost every other language and giving lots of syntactical freedom. The following example code

```
1 0.upto(4) do |x|
2   print x
3 end
```

does exactly the same as this code:

```
1 i = 0
2 doUntil(i > 3) {
3   print i
4   i += 1
5 }
```

A question is, if so much freedom is good and programmers should stick to coding conventions rather than just “code as they feel”. I think that reading Ruby code is quite easy, at least easier than reading Lisp code. As long as in the code is proper indention and structure this should not result in big problems.

In my experience as in others’<sup>2</sup> a programmer can implement a lot of functionality in a small but still easy to understand amount of code. “Duck Typing” may be one factor that reduces the amount of code to be written. For instance, a programmer does not have to care about type casts. A big library containing many feature rich classes that are easy and quite intuitively to use is another reason for getting much functionality out of a small amount of code.

---

<sup>2</sup>See a comparison of programming languages in terms of GZip Bytes [15].

In conclusion, Ruby is indeed intuitive<sup>3</sup>, fun and enables the programmer to come up with results quickly. But it is not simple-programming Ruby still needs good programming skills.

## 3.2 Model Viewer Controller

Since web applications are mostly used to present and modify lots of data it is appropriate to separate data from the user interface. The Model Viewer Controller pattern decouples data and presentation in a model and a view and introduces a controller component. The controller takes care of the business logic and the validation of user input. It uses the viewer component to present the data which is retrieved from the model.

RoR's architecture is based on the MVC. The model is the database and its representation in the application is a set of `ActiveRecords`. Each table that has to be accessed in the application is associated to one `ActiveRecord`. These are specified in the directory `/app/models/` in the application. Setting up one of these is pretty easy since RoR uses naming conventions:

```
1 class Category < ActiveRecord::Base
2   has_and_belongs_to_many :pictures
3 end
```

This code snippet represents the table categories. Note, that RoR is smart enough to do the mapping between plural and singular. There is no configuration file needed in order to define this mapping. The single line in this class provides information about the data relations to RoR. Apparently RoR is not smart enough to deal with foreign keys to notice these relationships. It is possible to access all the fields in the table by simply accessing find-methods as well as the fields of the `ActiveRecord`. For instance, I used this code to create a listing of all categories of pictures:

```
1 catlisting = ""
2 cats = Category.find_all()
3 catlisting << "<ul>"
4 for c in cats do
5   catlisting << "<li>" + link_to(c.name, :action => "show",
6     :id => c.id) + "</li>"
7 end
8 catlisting << "</ul>"
```

---

<sup>3</sup>This does not mean that that it won't be necessary looking up a few things in the manual.

Another CRUD operation that demonstrates the usefulness of the `ActiveRecord` class can be seen from preparing pictures for a slideshow. In each iteration we want a set of pictures assigned to a specific category:

```
1 for category in @pic.categories
2   slideshow = category.pictures
3   # do something with the slideshow...
4 end
```

The code from above is equivalent to this snippet of pseudo code one would have to write without using `ActiveRecords`:

```
1 # Define SQL query.
2 String query_template = "SELECT * FROM pictures JOIN
   categories_pictures ON pictures.id = categories_pictures.
   picture_id AND categories_pictures.category_id = ";
3 String query = null;
4 DataSet slideshow = null;
5
6 for each (category in pic.categories) {
7   query = query_template + category.id;
8   slideshow = SQL.fetch(query);
9   # do something with slideshow...
10 }
```

This example shows that using `ActiveRecord` reduces the amount of code that has to be written. Further, the programmer is released from thinking about the database schema and therefore does not have to care about SQL queries.

The logic for preparing data is done in `ActionController` classes. These can be found in the directory `/app/controllers`. Each model has one controller that is named following naming conventions. These are necessary to associate the controller to the corresponding model. This mapping is done automatically as the mapping between database schema and model. In the controllers' methods models are accessed to retrieve the data for further processing. The results are made accessible for the corresponding views. To each method of the controller is associated one view. For example, the method `PictureController.show` corresponds the view `show.rhtml` that can be found in the directory `/app/views/picture/`. Again, the mapping is done using naming conventions. Views are implemented in HTML with embedded Ruby. This is similar to writing PHP scripts into HTML.

My experience about the usage of MVC in the context of RoR is that it helped modularizing the application and it did not constrain my possibilities

at all. The naming conventions are very convenient and also intuitive to use. It saves work because the programmer does not have to specify the mapping between the components e.g. in XML files. Skeptics might argue that using names of the components for the mapping is not safe because names can be ambiguous. I think that names should not be given in an arbitrary manner and not contain any ambiguity. Hence, if there is a well thought naming schema for the components this issue should not be a problem. Besides, mapping files can be error prone and require time for maintenance.

### 3.3 Libraries and Neat Little Features

Ruby offers a lots of libraries available via RubyGems [4]. These were useful to support the development of the application. I used a login system [16] and a helper script simplifying picture uploads [8]. It was fairly easy to integrate them, although the documentation is sometimes not as good as it could be. For image manipulation, like generating thumbnails from the uploaded pictures automatically, ImageMagick [17] and its ruby interface RMagick [7] were pretty useful.

The framework lets developers use helper scripts in order to notify the framework for example about new models and controllers. **rake**, a tool that is a build program similar to **make** is also integrated. I found it very useful for database migrations. Experiences with this tool are described in the following section. My experience is, that these tools work as expected and their usage is fairly simple to learn and well documented.

### 3.4 Database Migrations with rake

A valuable tool integrated in RoR is the database migration tool **rake**. I wanted to find out if the framework has problems when the database schema has to be changed during the development process.

In a web photo album it is useful to get feedback on pictures. Thus, I decided on implementing the possibility of adding comments to each of the pictures. Therefore, I wanted to add a table **comments** to the database.

The documentation suggested using **rake**, some kind of make for Ruby. Migration scripts for rake are stored in **/db/migrate**. These classes are versioned so that **rake** knows in which order the classes have to be executed. Here is the class I used for adding a comments table to my existing database schema:

```

1 class CreateComments < ActiveRecord::Migration
2   def self.up
3     create_table :comments do |table|
4       table.column "text", :string
5       table.column "created_on", :datetime
6       table.column "author", :string
7       table.column "picture_id", :integer
8     end
9
10    execute "ALTER TABLE comments ADD CONSTRAINT
11           fk_comments_pictures FOREIGN KEY ( picture_id )
12           REFERENCES pictures( id )"
13
14    Comment.reset_column_information
15    Picture.find(:all).each do |pic|
16      0.upto(5) do |n|
17        c = Comment.new
18        c.text = "testing comment No. " + n.to_s + " for" +
19              pic.id.to_s + "(" + pic.title + "). Created for
20              testing purposes after migration. Nice pic :-)"
21        c.created_on = Time.now
22        c.author = "migration tool: rake"
23        c.picture_id = pic.id
24        c.save
25      end
26    end
27  end
28 end

```

Note, that you can also add testing data (line 13) using this class. The migration can be executed by calling rake from the command line like this:

```
rake migrate
```

rake will automatically detect which migration has already been executed and start off invoking all migrations after the last one done. It is also possible migrating to a specific version:

```
rake migrate VERSION=3
```

This will result in a database schema regarding all migration classes from 001\_SOMETHING.RB till 003\_ANOTHER\_MIG.RB. Rollbacks are only possible if properly specified in the `self.down` method within a migration. That means, the “undo” operation is not automated and has to be implemented by the programmer.

Another drawback of the migration classes, besides the lack of an automatically “undo”, is that they do not support foreign keys. To resolve this one has to set foreign keys “by hand” as can be seen above.

All in all, the migration process in RoR is very convenient. In other frameworks the programmer has to change the code at many places. In RoR this has only to be done once: in the migration. Moreover, these migrations are database independent, thus it is not necessary to care about the SQL syntax of different database applications (However, there still remains the issue about the foreign keys).

### 3.5 DRY—entirely true?

“Don’t Repeat Yourself!” is a major guideline in the design of RoR. I already mentioned that RoR is smart enough to prevent the programmer from repeating herself in writing configuration files for the mapping between the database and the data structures in the application. However, the programmer still has to repeat herself in specifying the data relationships in the model classes. But since this can be done by writing a single line of code I do not think that this is a major drawback. It might even help as some kind of short extra documentation. Though, it would be very convenient if that was generated by the framework.

Naming conventions save a lot of work but there are still some issues in which the programmer has to repeat herself. For example logging, authentication, tracing. These “cross cutting concerns” are the classical examples in which Aspect Oriented Programming can help [18].

I decided on trying whether RoR works together with the AspectR [19] library. To find out, I started with a small demo aspect:

```
1 require 'aspectr'
2 include AspectR
3
4 class Rlogger < Aspect
5
6   def initialize
```

```

7     @path = "#{RAILS_ROOT}/log/myLog.log"
8     end
9
10    def tick
11        "#{Time.now.strftime('%Y-%m-%d %X')}"
12    end
13
14    def log_enter(method, object, exitstatus, *args)
15        logFile = File.new @path, "a"
16        logFile.write "#{tick} #{self.class}##{method}: args =
17                        #{args.inspect}"
18        logFile.close
19    end
20
21    def log_exit(method, object, exitstatus, *args)
22        logFile = File.new @path, "a"
23        logFile.write "#{tick} #{self.class}##{method}: exited
24                        "
25        if exitstatus.kind_of?(Array)
26            puts "normally returning #{exitstatus[0].inspect}"
27        elsif exitstatus == true
28            puts "with exception '#{\\$!}'"
29        else
30            puts "normally"
31        end
32        logFile.close
33    end
34 end

```

And instantiated the aspect in the file CONFIG/ENVIRONMENT.RB, a class that contains all initial definitions for a RoR application:

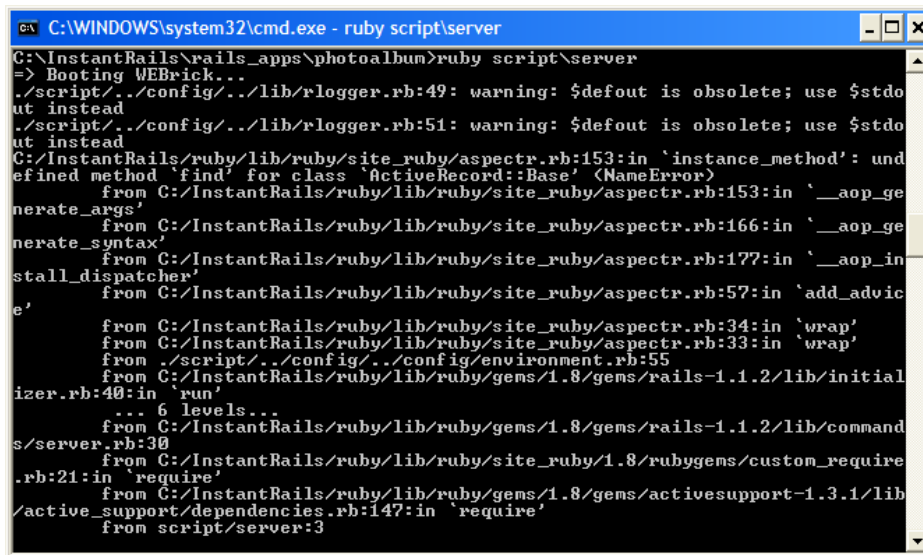
```

1 ...
2 # My Custom Aspect Logger
3 MY_LOGGER = Rlogger.new
4 MY_LOGGER.wrap(ActiveRecord::Base, :log_enter, :log_exit,
5               save)

```

However, I had to realize that this does not work as can be seen in figure 1.

I tried wrapping several other classes at several other places (e.g. in controller classes) but every time the interpreter complained that the corresponding methods do not exist. However, in the application these methods are constantly used. My conclusion is that AspectR does not work together with RoR. I assume the reason is that RoR seems to do a lot of reflection in



```
C:\WINDOWS\system32\cmd.exe - ruby script/server
C:\InstantRails\rails_apps\photoalbum>ruby script/server
=> Booting WEBrick...
./script/./config/./lib/rlogger.rb:49: warning: $defout is obsolete; use $stdout instead
./script/./config/./lib/rlogger.rb:51: warning: $defout is obsolete; use $stdout instead
C:/InstantRails/ruby/lib/ruby/site_ruby/aspectr.rb:153:in `instance_method': undeclared method 'find' for class 'ActiveRecord::Base' (NameError)
    from C:/InstantRails/ruby/lib/ruby/site_ruby/aspectr.rb:153:in `__aop_generate_args'
    from C:/InstantRails/ruby/lib/ruby/site_ruby/aspectr.rb:166:in `__aop_generate_syntax'
    from C:/InstantRails/ruby/lib/ruby/site_ruby/aspectr.rb:177:in `__aop_installer'
    from C:/InstantRails/ruby/lib/ruby/site_ruby/aspectr.rb:57:in `add_advice'
    from C:/InstantRails/ruby/lib/ruby/site_ruby/aspectr.rb:34:in `wrap'
    from C:/InstantRails/ruby/lib/ruby/site_ruby/aspectr.rb:33:in `wrap'
    from ./script/./config/./config/environment.rb:55
    from C:/InstantRails/ruby/lib/ruby/gems/1.8/gems/rails-1.1.2/lib/initializer.rb:40:in `run'
    ... 6 levels...
    from C:/InstantRails/ruby/lib/ruby/gems/1.8/gems/rails-1.1.2/lib/command/s/server.rb:30
    from C:/InstantRails/ruby/lib/ruby/site_ruby/1.8/rubygems/custom_require.rb:21:in `require'
    from C:/InstantRails/ruby/lib/ruby/gems/1.8/gems/activesupport-1.3.1/lib/active_support/dependencies.rb:147:in `require'
    from script/server:3
```

Figure 1: RoR does not work together with AspectR.

the background. But since I am not really familiar with the “guts” of RoR and I had time constraints on this work I cannot give a definite answer to this question.

## 4 A Word on Scalability

RoR simplifies lives of programmers of web applications a lot as my experiences might suggest. But ease of use is not the only affordance to a framework. Performance is also very important for web applications since we do not want a web server crunching on every request until the browser on the other side times out.

Many people are discussing about the performance of RoR [20, 21, 22, 23]. Some state that RoR leads to scalability issues and performs in general much slower than other frameworks. I agree on that, but I think these issues are not due to the RoR framework. The reason might be that Ruby itself is quite slow as benchmarks show [24]. It is just reasonable that we have to pay for convenience. “Duck Typing” implies that a function call never really is a function call only. The interpreter has to find out what the object actually is because of the lack of type information. I do not have to mention that

Ruby is slower than byte code or down-compiled languages as a matter of fact since the Ruby code written is interpreted in real time.

However, we have to consider what we want. If we want a framework for developing prototypes quickly and saving development cycles instead of cpu cycles, RoR is in my opinion a really good choice. But I would not “roll” with RoR if I had to develop an enterprise solution in which performance is essential. Though, there is still the possibility of using faster languages than Ruby for bottlenecks in order to improve performance.

## 5 Related Work

In the work on my project I encountered other web development frameworks that are similar to RoR. “CakePHP” [25] is a quite recent implementation of the concepts of RoR in PHP. However, CakePHP does not seem to be supported by a community as large as RoR’s. Thus, it does not have the diversity of plugins, addons and documentation as RoR has.

Another interesting framework is “Hibernate” [26]. It is based on Java technology and works together with application servers. Thus, Hibernate seems to be suitable for enterprise solution. As in RoR, the database model within the application is generated by the framework and migrations are supported by code generation, too.

An interesting question, left open for now, is if CakePHP or Hibernation can overcome the scalability issues of RoR while being as convenient to use.

## 6 Conclusion

With this self experiment I could show that RoR is indeed a robust and easy to use web development framework. In a couple of days it was possible to familiarize myself with the RoR and program a working prototype for a web based photo album. I conclude that RoR indeed results in a faster development process. There are several reasons that feed this property of the framework. First of all, the programmer can rely on RoR’s automatically mapping between application and database. Therefore is no need for writing configuration files what might not be the favorite job of programmers. Furthermore, fetching data from the database is highly simplified by using `ActiveRecord` classes. Additionally, easy migration of the database schema

supports a faster development process.

Nevertheless, there is still a way to improve RoR in terms of code redundancies. Aspects could help to address cross cutting concerns, that make the programmer repeat herself. Unfortunately the only aspect library for ruby I am aware of does not work together with RoR.

Altogether, I can recommend RoR for developing small and medium sized web applications. However, it is advisable to test RoR on scalability before implementing large web applications since there is a lot of discussion about this issue and there are good arguments for Ruby causing trouble on large scale.

## References

- [1] John Grundy. Software engineering tools. In *Proceedings of the 34th Hawaii International Conference on System Sciences*, 2001.
- [2] Ruby on rails. <http://rubyonrails.org/>, October 2006.
- [3] Instant rails. <http://instantrails.rubyforge.org/>, October 2006.
- [4] Rubygems manuals. <http://rubygems.org/>, November 2006.
- [5] The ruby programming language. <http://ruby-lang.org/>, October 2006.
- [6] rails. <http://rubyforge.org/projects/rails/>, October 2006.
- [7] An interface between the ruby programming language and the imagemagick. <http://rmagick.rubyforge.org/>, October 2006.
- [8] Sebastian Kanthak. Filecolumn - easy handling of file uploads in rails. [http://www.kanthak.net/opensource/file\\_column/index.html](http://www.kanthak.net/opensource/file_column/index.html), October 2006.
- [9] Mysql: The world's most popular open source database. <http://www.mysql.com/>.
- [10] phpmyadmin: Mysql database administration tool. [http://www.phpmyadmin.net/home\\_page/index.php](http://www.phpmyadmin.net/home_page/index.php), October 2006.
- [11] Firefox - rediscover the web. <http://www.mozilla.com/en-US/firefox/>, November 2006.
- [12] Curt Hibbs. Rolling with ruby on rails. <http://www.onlamp.com/pub/a/onlamp/2005/01/20/rails.html>, October 2005.
- [13] Yukihiro matsumoto's diary. <http://www.rubyist.net/~matz/>, December 2006.
- [14] Yukihiro Matsumoto. The ruby programming language. <http://www.informit.com/articles/article.asp?p=18225&rl=1>, November 2000.

- [15] Gzip ranking in the computer language shootout benchmarks. <http://shootout.alioth.debian.org/gp4/benchmark.php?test=all&lang=all>, November 2006.
- [16] Login generator. <http://wiki.rubyonrails.org/rails/pages/LoginGenerator/>, October 2006.
- [17] Image magick. <http://www.imagemagick.org/>, October 2006.
- [18] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Finland, 1997.
- [19] Avi Bryant and Robert Feldt. Aspectr - simple aspect-oriented programming in ruby. <http://aspectr.sourceforge.net/>, November 2006.
- [20] Lasse Koskela. Ruby on rails in the enterprise toolbox. <http://www.javaranch.com/journal/200601/rails.html>, November 2006.
- [21] Joel Spolsky. Language wars. <http://www.joelonsoftware.com/items/2006/09/01.html>, November 2006.
- [22] Rick Bradley. Evaluation: moving from java to ruby on rails for the centernet rewrite. [http://rewrite.rickbradley.com/pages/moving\\_to\\_rails/](http://rewrite.rickbradley.com/pages/moving_to_rails/), November 2006.
- [23] David Heinemeier Hansson. 'arghh, rails on ruby is better than java'. <http://www.loudthinking.com/arc/000280.html>, November 2006.
- [24] The computer language shootout benchmarks. <http://shootout.alioth.debian.org/>, November 2006.
- [25] Cakephp. <http://www.cakephp.org/>, November 2006.
- [26] Hibernate. <http://www.hibernate.org>, November 2006.