Web applications with Ruby and Rails

On the Right Track

Most web libraries make 90 percent of the job simple and the rest impossible. Rails, an open source framework programmed in Ruby, is flexible enough to succeed with that remaining 10 percent.

BY ARMIN RÖHRL AND STEFAN SCHMIEDL

www.photocase.de

Web development is often characterized by quick and dirty hacks with a colorful mixture of program code and HTML. However, an effort is underway to bring a more formal structure to web programming. For example, Rails [1] uses Ruby to implement a classical Model View Controller (MVC) framework, something that the world of smalltalk has used for some time. The MVC framework abstracts the data processing in the model from the GUI-based manipulation code (controller) and the representation code (view), see Figure 1.

This technique of abstracting the data from the data processing methods can be applied to anything from a database table to a workflow model for a large-scale enterprise. The model is partly or completely visualized in the view. Controllers react to user input (among other things) to initiate status changes within the model.

This structure makes the application logic independent of the external representation and user interface. Some development products claim to support MVC, although this support typically turns out to be two thirds hot air on closer inspection.

Software in Practical Applications

Rails includes the Active Record, which provides object relational mapping, to support the model. The Action Pack model subdivides web requests into controller and view-specific components. This design makes Rails really useful for developing database based Web applications, such as the project management software Basecamp [2], which was programmed using Rails. (See the box titled "Interview with the Rails Author.") When you work with Rails, you can tell the design of system comes from a practical background. Rails avoids many of the pitfalls present in more theoretical frameworks.

Rails is based on the DRY concept (Don't Repeat Yourself – in other words, "once and only once.") Repeat code within a program smacks of bad program design.

Rails tends to use reflection and runtime modules to replace configuration files; this removes the need for a typical XML-style configuration marathon. It also has the effect of applying changes immediately without having to go through a time-consuming build process.

Rails also has integrated functionality for general and unit tests. RubyDoc provides documentation. The big advantage is that all three components of the MVC framework were written in Ruby.

If developers stick to the Rails conventions (mapping between database and object structures), so called scaffolding helps to put a project online in next to no time. Functions for adding, modifying, and deleting objects are available at the click of a button.

Populating the Object Database

Our sample application is based on the tutorial from the Ruby homepage. We will create a simple Web application in

THE AUTHORS

Armin Röhrl and Stefan Schmiedl run a company called agile Softwareschmiede Approximity GmbH. They have been using Ruby for five years now, and they still have fun working with Ruby and other hyper-productive project development and management solutions. Productivity is more important to the authors than the latest hype.

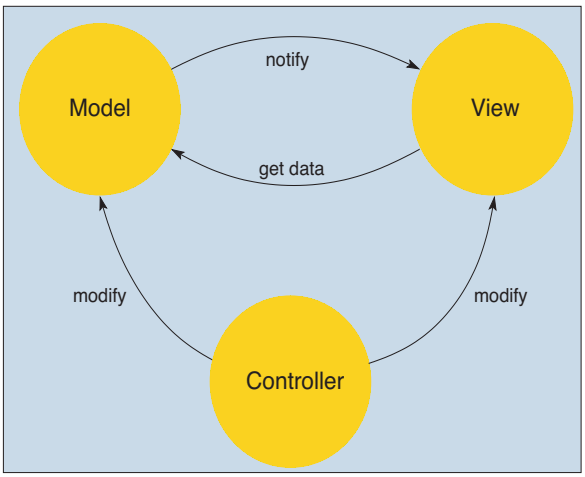

Figure 1: Model View Controller (MVC) schema. Data flow is not identical for all applications and may differ from the data flow shown here.

the folder */var/www/railsdemo*. Rails starts by creating a scaffold containing the required files:

```
rails /var/www/railsdemo
```

This command will overwrite an existing Rails application in the specified directory, so proceed with caution. We need a MySQL database for the sample table; see Listing 1.

The Yaml file *config/database.yml* defines access to the database; the file serializes data in a language-independent format:

```
production:
  adapter: mysql
  database: rails_production
  host: localhost
  username: root
  password:
```

The entry for the test database looks similar to this. In real life, you will not want to run this with root privileges that are without a password.

The *new_controller* script creates a new controller; its name should start with a capital letter. The views follow the controller name:

```
./script/new_controller ➥
Friends list display new edit
```

Avoid using *process* as a view name, as Rails uses this name for internal purposes. The process for creating a new model is similar:

```
./script/new_model ➥
Person
```

Rails typically discovers the underlying database table automatically. If required, the script will accept a table name after the model name:

```
./script/new_model ➥
Person people
```

To test the configuration of the Rails application, developers should run *rake* (Ruby make) in the application directory. A number of issues can occur depending on your system configuration:

- If *mysql.sock* is not at the usual location, you can either edit *mysql.rb* (in a directory such as */usr/lib/ruby/gems/1.8/gems/active-record-0.9.5/lib/active_record/vendor*) or set the *MYSQL_UNIX_PORT* environmental variable.
- Permissions for the database may not be set correctly.
- The scripts expect */usr/local/bin/ruby* as the path to Ruby. If Ruby is located somewhere else, you can either create a symlink or change the path name in the script files.

To complete the Web application, we now need to provide some content for *app/views/friends/display.rhtml*:

```
01 <html>
02 <body>
03 <h1>Friends#display</h1>
04 <p>Displays a person entry</p>
05 <p>
06 <%= @person.name %><br />
07 <%= @person.street1 %><br />
08 <%= @person.street2 %><br />
09 <%= @person.city %><br />
10 <%= @person.state %><br />
11 <%= @person.zip %><br />
12 </p>
13 </body>
14 </html>
```

This looks similar to Erb, the HTML embedded Ruby variant. The HTML page displays the data that the model provides. The control has to define and create the *@person* object referenced here. This occurs in the *app/controllers/friends_controller.rb* file, which contains the access methods. The *require* keyword adds the model:

```
require 'person'
```

The Web application can now use the Active Record class to access the database. If we now insert the following code for the *display* method, it will get the first entry from the table and write it to the *@person* instance variable for the view class:

```
def display
  @person = Person.find(1)
end
```

The model class definition in *app/models/person.rb* is also worth examining:

```
require 'active_record'
class Person < ➥
ActiveRecord::Base
end
```

Listing 1: SQL for Person

```
01 CREATE DATABASE
   rails_production;
02 USE rails_production;
03
04 CREATE TABLE people (
05   id int(10) unsigned NOT NULL
   auto_increment,
06   name varchar(50) NOT NULL
   default '',
07   street1 varchar(70) NOT NULL
   default '',
08   street2 varchar(70) NOT NULL
   default '',
09   city varchar(70) NOT NULL
   default '',
10   state char(2) NOT NULL
   default '',
11   zip varchar(10) NOT NULL
   default '',
12   PRIMARY KEY (id),
13   KEY name (name)
14 ) TYPE=MyISAM AUTO_INCREMENT=2
   ;
15
16 INSERT INTO people VALUES (1,
   'Superman',
17 '123 Somewhere', '',
   'Smallville', 'KS', '123456')
```

Terms and Conditions

Your subscription will automatically renew each year at the subscription rate in effect at the time of renewal until you cancel your subscription.
You will receive a confirmation notice about 60 days before your subscription expires. Unless you inform us otherwise, we will charge the credit card used with the initial order at the beginning of each new subscription year.
You may cancel your subscription with immediate effect at any time and receive a full refund on all unmailed copies.

If you have any questions please e-mail us at subs@linux-magazine.com.

5 great reasons to subscribe

1 SIGNIFICANT SAVINGS

Save over 30% compared to the cover price – get 12 great issues with DVD for the price of 8!

2 MONTHLY BONUS DVD

The monthly bonus DVD contains complete collections of Open Source packages, full distributions, commercial products, and other special features.

3 GUARANTEED DELIVERY

Never miss an issue again! A Linux Magazine subscription ensures that you'll receive top-quality Linux Know-How delivered to your door every month.

4 NO-HASSLE RENEWAL

Unlike other publications, Linux Magazine will not fill your mailbox with renewal reminders and notices. With Linux Magazine, your subscription will automatically renew each year.
You will receive one confirmation roughly 60 days before the end of your subscription – no action required on your part.

5 MONEY BACK GUARANTEE

If for any reason you ever decide to stop reading Linux Magazine, you can cancel your subscription at any time. We will fully refund the purchase price for all issues that have not yet been mailed to you. No questions asked, no cancellation dates to remember, no problems.

For faster service order online: www.linux-magazine.com/Subs

You can also fax this form to **+49 89 9934 1199** or mail it to **Linux Magazine, Stefan-George-Ring 24, 81929 Munich, Germany**

Yes, I want to subscribe to Linux Magazine and receive advanced Linux Know-How every month!

Address

CAPITALS please, required fields are in **bold**

First Name

Last Name

Company

Address 1

Address 2

City

State / Province / Region

Postal Code

Country

Please choose your subscription type and delivery area:

DVD Subscription
12 issues including monthly DVD

☐ United Kingdom	£ 49.90
☐ Other Europe	€ 79.90
☐ Outside Europe – SAL*	US$ 94.90
☐ Outside Europe – Airmail**	US$ 109.90

Standard Subscription
12 issues, no DVD

☐ United Kingdom	£ 39.90
☐ Other Europe	€ 64.90
☐ Outside Europe – SAL*	US$ 74.90
☐ Outside Europe – Airmail**	US$ 84.90

*SAL: roughly 4 weeks delivery
**Airmail: roughly 1 week delivery

Subscription start issue

Issue # ☐☐*

*The issue number that your subscription will start with. If you do not specify any start issue, you will receive the next available issue.

Payment Method

☐ VISA ☐ MasterCard

☐ Cheque enclosed
£ cheque from UK bank only, payable to Linux New Media AG

Expiry Date ☐☐/☐☐

Card Number

Name of Card Holder

E-mail (used only to communicate with you about your subscription)

Signature **Date**

www.linux-magazine.com/Subs

Table 1: Java and Ruby

Application	Java	Ruby
Application GUI	Swing	TK, QT, Fox
Web GUI	JSP, XSLT, XMLC...	Iowa, Rails
Persistent Domain Objects	Entity EJBs, JDO, manual	Kansas
and Transaction Management	generated DAOs &	
RPC Mechanism (Remote Pro-	Session EJBs, Servlets	DRb
cedure Call) for communication		
between front-end and back-end		
Infrastructure for asynchronous communication	Message-Driven EJBs	DRb/Rinda/Tupleserver

The base class *ActiveRecord::Base* does most of the work, as it defines simple methods of access to the database. The table name is the plural of the class name. Interestingly, the plural form is grammatically correct for irregular nouns, for example, *men* for *man* or *people* for person. Refer to the class documentation for more detail on naming rules and conventions [8].

The results of all this work are now visible under *http://localhost/rails/friends/display*. This URL points at the controller (*friends*) and at the action (*display*).

Associated Tables

Rails also has a solution for associating tables. We will use two tables with phone number (see Listing 2).

Box 1: Installation

For users of Mac OS X (version 10.3 or newer), two videos [5] demonstrate the installation process. Of course, they are interesting for Linux users too. Rubygems [6] (version 0.7 or newer), a packet management tool for Ruby, provides a simple approach to installing:

```
gem install rails
```

Whenever a new version of Action Pack or Active Record is released, you can update simply by typing *gem update*. Make sure the database connection to the back-end is available; you can use either MySQL or PostgreSQL. Debian users will need to fulfill a few requirements for the install, as the required components are spread across a large number of packets (see also [7]):

irb1.8, libbigdecimal-ruby1.8, libdbm-ruby1.8, libdl-ruby1.8, libdrb-ruby1.8, liberb-ruby1.8, libgdbm-ruby1.8, libiconv-ruby1.8, libopenssl-ruby1.8, libpty-ruby1.8, libracc-runtime-ruby1.8, libreadline-ruby1.8, librexml-ruby1.8, libruby1.8, libruby1.8-dbg, libsdbm-ruby1.8, libsoap-ruby1.8, libstrscan-ruby1.8, libsyslog-ruby1.8, libtest-unit-ruby1.8, libwebrick-ruby1.8, libxmlrpc-ruby1.8, libyaml-ruby1.8, libzlib-ruby1.8, rdoc1., ri1.8 ruby1.8, ruby1.8-dev, ruby1.8-elisp, and finally, ruby1.8-examples.

Setting Up Apache

If you have not done so already, you need to enable the Apache rewrite module:

```
# a2enmod
Which module would you like to
enable?
Your choices are: ... rewrite ...
Module name? rewrite
Module rewrite installed; run
/etc/init.d/apache2 force-reload
to enable.
```

To run a Rails application in */var/www/myapp*, you need to add an entry for the virtual host to your Apache configuration file. The entry would look something like this:

```
01 <VirtualHost *:80>
02     ServerName rails
03     DocumentRoot
   /var/www/railsdemo/public
04     ErrorLog
   /var/www/railsdemo/log/apache.
   log
05
   <Directory/var/www/railsdemo/p
   ublic/>
06       Options ExecCGI
   FollowSymLinks
07             AllowOverride all
08       Order allow,deny
09       Allow from all
10     </Directory>
11 </VirtualHost>
```

Check the *htaccess* file for the Web application for more settings; in our example this is */var/www/railsdemo/public/.htaccess*. Now, working as root, reload your Apache configuration as follows to apply the updates:

```
/etc/init.d/apache2 force-reload
```

Rails uses a suffix (*_id*) to identify external keys, for example *person_id*. As previously mentioned, the table name is the plural form of the model name:

```
./script/new_model Phone
```

The file generated previously, *person.rb* needs two modifications:

```
require 'active_record'
require 'phone'

class Person < ➥
ActiveRecord::Base
    has_many :phones
end
```

The call *has_many :phones* specifies a 1:n relationship. In other words, a person can have more than one phone number. We need to add the following lines to *display.html* to change the display format:

```
<% for phone in ➥
@person.phones %>
<%= phone.phone %><br/>
<% end %>
```

If a model already exists, it is quite easy to use scaffolding techniques in Rails to define display and manipulation methods for the data. Web developers can then fine-tune the basic methods as required.

Automatic Scaffolding

The following example assumes an empty Rails application directory. We will be creating a controller and a model first (*./script/new _controller Friend* and *./script_new _model Person*, see above.) Make sure you do not have an action called *list*. Rails will use the existing database. We simply need to modify the friends controller:

```
scaffold :person
```

This addition creates new methods with default behavior, as specified in the documentation at [9]: *list*, *show*, *destroy*, *new*, *create*, *edit*, and *update*.

This supports scaffolding with the *Person*, but we still need the phone numbers. To avoid name collisions when defining the telephone actions, Rails

Listing 2: SQL for Telephone Numbers

```
01 USE rails_production;
02 CREATE TABLE phones (
03   id int(10) unsigned NOT NULL
   auto_increment,
04   person_id int(10) unsigned
   NOT NULL default '0',
05   phone varchar(15) NOT NULL
   default '',
06   PRIMARY KEY  (id),
07   KEY people_id (person_id),
08   KEY phone (phone)
09 ) TYPE=MyISAM AUTO_INCREMENT=3
   ;
10
11 INSERT INTO phones VALUES (1,
   1, '1234567890');
12 INSERT INTO phones VALUES (2,
   1, '1122334455');
```

adds an action reference to the model name if you set the right flags:

```
scaffold :person, :suffix⇗
 => true
scaffold :phone, :suffix⇗
 => true
```

This gives us the following URLs:

```
http://rails/friends/list_person
http://rails/friends/list_phone
```

As you can see, we have created a structured Web application with a minimum of effort. The application will store and manipulate objects in a relational database. Ruby implements the MVC model neatly, and Ruby even generates the basic code associated with the MVC structure.

Logout

Rails makes it easier to develop basic applications if you stick to the rules. The system provides an extremely useful introduction to programming web applications. If you need more help getting started with Ruby, check out the Ruby Manual. ■

INFO

[1] Ruby on Rails: *http://www.rubyonrails.org*

[2] Basecamp: *http://www.basecamphq.com*

[3] Apache Struts: *http://struts.apache.org*

[4] Struts Action Invocation Framework (SAIF): *http://struts.sourceforge.net/saif/*

[5] Tutorial videos: *http://www.rubyonrails.org/show/HomePage*

[6] Rubygems: *http://rubygems.rubyforge.org/wiki/wiki.pl*

[7] Rails for Debian: *http://www.rubyonrails.org/show/RailsOnDebian*

[8] Rules for plural names: *http://api.rubyonrails.org/classes/ActiveRecord/Base.html*

[9] Scaffolding methods: *http://api.rubyonrails.org/classes/ActionController/Scaffolding/ClassMethods.html*

[10] The Rail author's weblog: *http://loudthinking.com*

Box 2: An Interview with Rails Developer, David Heinemeier Hansson

LM Why did you develop Rails?

DH I had been programming in PHP for a few years while keeping a close eye on all the web-related projects in Java. I loved PHP for its immediacy and was very inspired by the frameworks in Java. So I didn't want to give up the quick turn-around from PHP, but neither was I willing to suffer under the problems it had with large, object-oriented systems.

Right around the height of my annoyance with PHP, and not long after a disillusioning meeting with Java (worked in a J2EE shop for 6-7 months), I read about how Martin Fowler and the Pragmatic Programmers were really into this niche language. That language was of course Ruby, and once I climbed the initial stumbling blocks, it was just love, love, love from there on.

Putting all my inspirations from Java into a framework carried by Ruby proved to be a very natural path for me. Oh, and I needed to do it anyway in order to build Basecamp as quickly as we did (in two man-months).

LM What is the biggest strength of Rails?

DH Clean immediacy. You can create applications incredibly quickly that won't result in a code mess. Build one quickly to keep. A strong focus on proven patterns, such as test- and domain-driven development, along with an MVC division really helps here.

LM What is the biggest weakness of Rails?

DH IIt can be hard to convince prospective clients (or your manager) that they should let you be productive in a relatively unknown language when all they read about is Java and C#.

LM What is the future of Rails?

DH Hopefully, a 1.0 release within the year 2004. I think 2005 could really be The Year of Ruby, and I'm hoping the release of Rails will help ensure that. But otherwise, I have pretty humble goals for the further development of the framework. Rails is currently around 2,500 lines of code. I'd like it not to grow much larger than that.

LM Please tell us 2 paragraphs on Basecamp. Maybe you could also mention the business model.

DH Basecamp is a web-based project management application built around weblogs, milestones, and todo lists. It's a deceivingly simple application that has had a huge impact on how many people manage their projects. It's a hosted application that firms pay between $19 and $59 a month to use. There's even a lifetime free 1-project plan, if you just want to try it out.

LM What is the future of basecamp?

DH We're continuously improving the application. Just last week we launched a whole slew of new features to better manage the responsibilities on a project. Also, I think we've just tapped a tiny percentage of a huge market. Project management is everywhere. Besides the obvious uses in businesses, we're seeing people use Basecamp to plan everything from home improvement initiatives to weddings. Basecamp is a great fit for such a wide range of projects because of its simplicity. Managing the project is not a project in itself.

LM Where do you think web-based programming is leading?

DH I hope it's away from the intense complexity that J2EE, .NET, and other heavy-handed environments have somehow convinced people they need. Dynamic languages such as Ruby and the frameworks built around them, like Rails, are going to sweep the scene in the coming years. Customers and developers will begin to realize that most web-applications doesn't need the firepower of an Imperial star destroyer, but rather the flexibility of an X-wing (yeah, I just watched Star Wars on DVD).