Abstract

Ruby on Rails is a framework that allows the rapid development of web applications while maintaining a well-structured code base. The Ruby on Rails framework produces a Model-Viewer-Controller (MVC) skeleton for a web application. The Model-Viewer-Controller design pattern promotes the development and maintenance of a sensible, efficient, scalable, and easily extensible web application.

Buzzwords like Web 2.0 and Ajax have been around for a while now and are the immerging technologies of choice among web application developers. Ajax is a technology that provides the capacity for asynchronous web page updates. The phrase “asynchronous web page updates” means that individual items on a web page can be updated without transmitting and updating entire web pages. The Ajax support of Ruby on Rails fits seamlessly into the MVC concept of the framework and allows developing Ajax functionality with ease.

This paper concludes with an overview describing how Ruby on Rails supports testing.
Developing applications with Ruby on Rails

Ruby on Rails is an Open Source web application framework written in Ruby for developing web applications with Ruby. It carries the philosophy of Ruby to web development.

Ruby on Rails was created by David Heinemeier Hansson in mid 2004 and is now developed and maintained as an Open Source project. The first stable version (1.0) was released end of 2005. Since the time Rails was released it has collected a lot of attention and the term “Ruby on Rails” has become a buzzword among “Web 2.0”.

Tim O'Reilly states about Rails (compare [14]): “Ruby on Rails is a breakthrough in lowering the barriers of entry to programming. Powerful web applications that formerly might have taken weeks or months to develop can be produced in a matter of days”.

The key principles of Ruby on Rails are as follows:

- *Don't repeat yourself (DRY)* - Keep the code small and do everything only once. Rails favors abstraction and reuse to allow fast development while maintaining understandable and maintainable code.
- *Convention Over Configuration* - Rails forces developers to follow programming conventions that allow the framework to plug together the pieces automatically in the background.

Model-View-Controller Concept

Rails applications are developed with a Model-View-Controller (MVC) concept. Trygve Reenskaug first described the MVC design pattern in 1979. The intent of MVC is to separate code used for data access from code used for the user interface. Nowadays it is the standard for developing applications with a graphical user interface (GUI).

With Rails, a developer is given a MVC skeleton as a starting point, and thus, does not have to create MVC applications scratch nor does he have to have a certain IDE at hand, which will force him to produce a MVC application.

Applying the MVC concept to web applications, following chain of actions comes up:
Figure 1: MVC concept applied to web environment.

1. A web browser sends a request to the controller.
2. The controller then interacts with the model to obtain necessary data.
3. The model queries these data from the database (3) and provides them to the controller (4).
4. The controller then sends it for rendering to the view (5).
5. The view finally generates the output and sends this to the web browser (6).

With Ruby on Rails, the model, the view and the controller each have a representative class from which all models, views, and controllers are inherited. The following three paragraphs handle the classes, namely “ActiveRecord”, “ActionView,” and “ActionController.”

Model - ActiveRecord

Rails uses Object Relational Mapping (ORM) for storing and retrieving data. If the database has for instance a table called products, the model will have a class Product. The multiplicity is important in this case. The Product object will provide methods to get and set attributes of the table products. In addition Rails provides class-level methods that perform operations on the table-level such as Product.find(ID) which returns a certain row or Product.find_all which returns a list of all rows.

The ORM layer of Rails maps tables to classes, rows to objects and columns to object attributes. The difference to other ORM approaches is that there is hardly any effort in configuring this mapping. Following the “Convention Over Configuration” principle, the developer has to follow basic rules and the framework will do the rest. One of these rules is the multiplicity we mentioned above. Table names in the database will get a plural name; the class in the model will be singular. These rules follow simple logic and are hence easy to remember. An object is representing one instance of its class, so the naming convention is a singular noun. The table will hold multiple rows with data items of, for example, a product and so the naming convention here is a plural noun.
The model takes ensures that the inserted data are valid. Whenever a model object gets created or updated, it gets checked against validation statements that the developer has previously defined. Whenever a validation fails, an error message is raised, which then can be processed by the controller, which has invoked the model object creation.

**Controller – ActionController**

The controller is the logical center of the application and will hold all application-level functionality. The controller orchestrates the interfaces between user, view, and model. Most of this orchestration is done in the background by Rails. Responsibilities of the controller are as follows:

- **Routing** - The requests sent by an user are routed to internal actions.
- **Caching** - The model caches as much as possible to prevent unnecessary queries on the model.
- **Session management** - Manages the individual run time information of a certain user of the web application.

What is appealing about the above set of responsibilities is that Rails does it automatically behind the scenes.

**View – ActionView**

The view connects HTML code with the application. Views are HTML files with inline expressions, which can contain arbitrary code. This is called “Embedded Ruby” and is fundamentally the same as the approaches used for PHP and Java's JSP. The view is intended to hold the logic, which is essential to visualize the application. It is not desirable for the view to hold all the logic of the web application. It is the responsibility of the developer to take full advantage of Ruby on Rails by separating the code according to the MVC architecture.

**Code Generation**

Ruby on Rails tries to help the developer set up the components that he needs. Ruby on Rails differs from ASP.NET, where Microsoft tries to provide components that are ready to use, in that Ruby on Rails provides helper scripts, called generators, which produce code skeletons. These skeletons are ready for use, but have no business logic yet.

**Routing**

The address of an example application “http://domain.com/say/hello” can be broken down into three parts: The application identifier, the controller identifier (say) and the action identifier (hello). The following figure illustrates this:
Reconsidering the generic MVC concept introduced before, the MVC concept with Rails terms looks like the following:

The flow is as following:
1. The web browser sends a request.
2. The request gets routed (1a) to a controller (1b).
3. The controller executes the requested action (a class method).
4. The controller has a method, which tells him to return an object of a model class.
5. The controller gets the data from the model (2).
6. The model object queries these data from the database (3) and provides them to the controller (4), which then sends it for rendering to the view component (5).
7. The view component finally processes the object the controller provides, generates the output and sends this to the web browser (6).
Ajax with Ruby on Rails

Ajax: The Hype

The idea of the so-called Web 2.0 was the new buzzword in 2006 to impress potential investors and to raise venture capital.

The term “Web 2.0” itself is not well defined and is often used to express different concepts. These concepts may include both marketing aspects and technical aspects. Unifying is the idea to improve the version one of the “Web”.

The term Asynchronous JavaScript and XML (Ajax, compare [1] for description of origin) is a technique used when talking about “Web 2.0”. A typical “Web 1.0” website loads all elements from top to bottom sequentially. A website is basically a HTML-document which is delivered by a web-server. The following figure illustrates this:

Figure 4: A simplified view on the loading process of a website.

The website in this example consists of nine subparts (see left part of figure 4). If this website is now not only a static document, but dynamically created, every item may be generated individually for the user who is requesting the website. With every user request, the whole website is requested and so regenerated. If for example item 9 (see right part of figure 4) needs to be changed because of a user interaction, items 1-8 will be regenerated as well.

The idea behind Ajax is to break up this process and to do an update only on item 9 while leaving items 1-8 untouched. To enable this, an asynchronous call, encapsulated in a XML message, is done to the web-server. To achieve this, the Javascript-engine of the browser has an object called XMLHttpRequest. The internal representation of a web page in the browser is held in the “Document Object Model” (DOM). This is a structured tree representation of the web page that the browser is currently displaying. An object representation of this tree can be accessed and manipulated by Javascript calls.

In the classic scenario, the user requests a web page. This request is then handled by the web server, which invokes the backend-processing and then delivers the output as HTML
file back to the user's browser. In the Ajax scenario, this process is extended by interaction between the Ajax-Engine and the user-interface. The Ajax-Engine is event-based and manipulates the DOM-object of the browser. If necessary, the Ajax-Engine may also invoke the web-server to retrieve data. This is done by the XMLHttpRequest-object mentioned before.

The term Web 1.0 is used when talking about the classic scenario and Web 2.0 when talking about the Ajax scenario.

**Why use Ajax?**

So why make it more complicated to develop web applications? Having code executed at the client and having code executed at the server side will increase the complexity.

- **Better user experience** - The interface reacts directly to changes by the user. Web 1.0 web applications often collect a couple of data items, process these, and give the user a reaction in a batch mode. A Web 2.0 may respond specifically and interactively to each item.
- **Send selective data** - Because reloads of web pages may take a long time and turn the user away from the content of the page, developers try to fit all options the user might access into one page. A Web 2.0 application can focus on what the user actually wants and leave out all he does not want.
- **Make web sites respond quicker** - Since the web page does not load a lot of unneeded content or reload existing content, the response times can decrease significantly. Validation can be pre-handled mostly by the Ajax-Engine and will thus give the feedback immediately without a roundtrip to the web-server.

**The Challenge with Ajax**

- **More possibilities - more responsibilities** - As the number of options in the user interface increases, the importance of selecting an applicable form of interaction increases. It is easy to confuse the user with too many effects or the wrong effects.
- **Server-Load** - While before a user did one request and receive say 100 Kilobyte, he might now do 30 requests and receive 1 Kilobyte per request. This indeed would lower the used bandwidth from 100 kilobytes to 30 kilobytes, but the server has to respond 30 times more often. Current web-servers are known to able to handle about 500-1000 connections at a time. The popular web-servers like the Apache httpd are known to have problems as the number of simultaneous connections increase.
- **Polling** - A typical synchronization problem comes up when the server cannot respond to a request by the client. This result in the client periodically repeating its request until the request is satisfied. The reprocessing of requests for the same service will increase the load on the server.
• **Back-Button problem** - If all the interaction with the user are done by the Ajax-engine, the application has no specific states anymore from the user's point of view. The user cannot use the back button anymore because the back button only switches between the states of normal http requests. The Ajax-Engine and the developer have to keep track of the states and make the roll-back accessible through the back-button.

**Ajax Engines**

The developers of an Ajax application do not have to implement functionality of the XMLHttpRequest themselves. The Ajax Engine is typically a JavaScript-Library, which prevents the developer from implementing Javascript functionality by automatically producing the needed Javascript-code, or makes the use of Javascript easier. An example for the hiding-category is the Google Web Toolkit(GWT, [7]), an example for the “make it easier“-category is Dojo ([8]). The Google Web Toolkit is, as the name implies, is supported by Google, Dojo is supported by among others IBM and SUN. Other popular Ajax Engines are DWR ([9]), Prototype ([10]), and Scriptaculous ([11]). Microsoft is also working on Ajax extensions for the .NET framework, but as of the end of 2006 the names and roadmaps were still changing (compare [12]). The variety of choices is enormous and there is no sure winner yet.

Ruby on Rails uses the libraries Prototype and Scriptaculous. Basically, both are spin-offs of the development of the Ajax support for Ruby on Rails and now their popularity is independent from Ruby on Rails. Prototype provides the functionality to communicate asynchronously via the XMLHttpRequest and to manipulate the “Document Object Model” of the browser. Scriptaculous is an add-on for Prototype and provides “fancy” effects which enhance the user interface. The Ajax support is baked right into the Ruby on Rails framework and is not some minor side technology.

**The Rails way**

With Ruby on Rails 1.0, which was released in December 2005, it was already possible to update a single elements of a website using Ruby code. This approach was limited if several page elements needed to be updated. RJS templates were introduced with Ruby on Rails 1.1. Ruby on Rails 1.1, which was released March 2006, allows one to define templates, which make it easy to update multiple page elements with one request. RJS stands for JavaScriptGenerator, which tells what these templates do: Generate JavaScript.

Ruby on Rails tries to totally hide JavaScript-code from the developer. The framework uses the Prototype- and the Scriptaculous-library mentioned in this section. Rails allows the use of these libraries without writing any line of Javascript code. The Rails way to include Ajax is to hide Javascript from the developer and make the functionality available via Ruby code.
An RJS-template is part of the view component in the MVC-concept and will also be responsible to render a controller method. The JavaScript code generated by the template will communicate directly with the browser and manipulate the DOM element.

![Diagram](http://theflowworld.bernhol.de/say/hello)

**Figure 5: RJS-template in MVC-concept**

### Testing with Ruby on Rails

This section is a short guide through the testing capabilities that Ruby on Rails offers. The framework provides a testing infrastructure, which is ready to use to run the first tests. When a generator creates an application, the infrastructure for testing is also created, too. In addition, whenever a controller or a model is generated, Rails produces the skeleton of testing scripts, which later can be executed. Rails differentiates “units tests”, “functional tests” and “integration tests”.

### Unit testing of Models

A Rails application has three different databases configured: One for production use, one for testing and one for developing. The testing database only exists for automated testing. Whenever the test suite is executed, the so-called *fixtures* get loaded into the testing database. The test-suite is a Ruby script, which is created by the framework for each model. Within this script, methods define *test cases*, which consist of one or more *assertions*. These assertions may test for example if the loaded fixtures are valid with respect to the validation definitions in the model.
Functional Testing of Controllers

A test case that tests a single controller is called functional test. The question is, if a controller method is able to do what it is expected to do. As Ruby on Rails is web development framework, a functional test tests if for example a page was rendered successfully. In this case, the test suite emulates a HTTP GET request. A simple assertion may check if the HTTP response was as expected, a more sophisticated assertion would check if the HTML code was delivered as expected. With a selector, a so-called select assertion, specific html tags can be checked for the existence or a certain content.

Another functional test which is supported by Rails is performance testing. With a dynamic fixture a huge amount of data can be loaded and test cases processing this data can be written. Rails also provides a profiler and a benchmarker to analyze things which only occur in production mode.

Integration Testing of Applications

In contrast to functional tests, integration tests test several controllers at once and try to determine if they cooperate as expected. In contrast to unit and functional tests, integration tests are not created automatically. However, a generator exists to do this. An integration test checks a certain flow of actions. Basically, the test cases in an integration test look similar to the test cases of a functional test, but in the case of integration test a real user interaction is emulated. A functional test only tests if the controller method executes as expected. An integration test also tests if the routing was correct (compare figure 3).

Related work and literature

[2] worked on the facets of Ruby and introduced Ruby on Rails as well as Rubyx OS, a Ruby based Linux distribution. The authors also considered the usefulness of Ruby for teaching object-oriented programming.

[3] introduces Ruby on Rails and compares the framework to Java. The author, Bruce Tate, states an opinion if Java should get Rails, too.

The standard reference about Ruby on Rails is Agile Web Development with Rails ([4]). This book also shortly introduces the scripting language Ruby. A profound overview about Ruby can be found in Programming Ruby: The Pragmatic Programmers’ Guide ([6]).
Conclusion

This paper analyzes the programming language Ruby and the web development framework Ruby on Rails. It introduces Ruby and Ruby on Rails. It presents the philosophy behind Ruby on Rails and describes the MVC concept that is the basis for the framework. The paper introduces and discusses Ajax and shows how implementing Ajax functionality with the web development framework Ruby on Rails works. It introduces testing facilities of Ruby on Rails.

Ruby on Rails eases the set up of web-applications with Ajax functionality. Rails also provides sophisticated testing facilities. Test-driven development with Ruby on Rails seems to be a reasonable way to develop web-applications.

Ruby on Rails has gotten a lot of attention in the past two years. Now, at the end of 2006, the project can still be considered young. The pace of the Rails development is fast and the community is steadily growing. Ruby on Rails turned in a very short time into an established technology for web development. In the meantime, other tools are now being compared against Ruby on Rails (compare [15]).

Ruby on Rails has proven to be a very good environment for developing web based applications. The framework is accomplishing a high degree of perfection in the niche of web development frameworks. The standard work about Ruby on Rails, [4] (The second edition of this book, [5], was first shipped the end of December 2006), was the third-most popular in Amazons “Computer and Internet” section in 2006 (compare [13]). Developers should learn Ruby and work with Ruby on Rails. Decision makers should promote Ruby on Rails and invest into it.

References


