

Tobias Günther

Webentwicklung auf Schienen

Agil und professionell entwickeln mit Ruby on Rails

Ein verhältnismäßig junges Open-Source-Framework sorgt seit einigen Monaten für Furore in der Webentwicklung: Ruby on Rails bietet mit frischen Konzepten eine Alternative zu bekannten Technologien wie Java und PHP.

Eine kleine Begriffsklärung zu Anfang kann nicht schaden: Ruby ist nicht Rails. Ruby ist eine interpretierte, vollständig objektorientierte Skriptsprache, die 1995 in Japan das Licht der Welt erblickte. Durch die lange Zeit nur spärliche englischsprachige Dokumentation schaffte die Sprache erst um die Jahrtausendwende den Durchbruch in Europa und den USA. Rails hingegen ist ein Webapplication-Framework, das vollständig in Ruby programmiert ist.

Seinen Anfang nahm die Rails-Geschichte 2004, als der Däne David Heinemeier Hansson die Anwendung erstmals als OpenSource-Framework veröffentlichte. Interessant ist, dass Rails nicht im „Labor“ entstand: Es wurde aus einer bestehenden Anwendung (dem Projektmanagement-Tool Basecamp [1]) extrahiert. Dies spiegelt auch den praxisnahen Charakter von Rails wieder. Die Arbeit an Rails wurde belohnt: David Heinemeier Hansson wurde 2005 mit dem von Google und O'Reilly verliehenen Open-Source-Award „Best Hacker“ ausgezeichnet. Außerdem gewann das Buch „Agile Webdevelopment with Rails“ den renommierten Software Development JOLT-Award. Nun aber zur Technik und Arbeitsweise von Rails.

Model View Controller – Architektur

Das Framework Rails nutzt das MVC-Design Pattern (ModelView-Controller). MVC gilt als eine Art Musterlösung, die sich in den letzten Jahren zum Standard (nicht nur) für Webapplikationen entwickelt hat. Die zugrunde liegende Idee ist eine einfache: Der Code wird in mehrere Schichten getrennt, die voneinander isoliert werden. Das bringt gleich mehrere Vorteile mit sich:

- klare Trennung der Verantwortlichkeiten der Code-Teile
- bessere Erweiterbarkeit und Wartbarkeit
- mehr Übersicht durch strukturierten Code
- kurze Einarbeitungszeiten neuer Programmierer in fremden Code

Das Herzstück der MVC-Implementierung in Rails ist die ActiveRecord-Komponente. Sie stellt den Model-Teil dar und ist für die Verbindungen zur Datenbank zuständig. Als O/R-Mapper macht ActiveRecord das Entwicklerleben entschieden einfacher. Hat unsere Datenbanktabelle zum Beispiel eine Spalte mit dem Namen „firstname“, so kann über diesen Bezeichner automatisch auf die Daten zugegriffen werden:

```
RUBY
User.firstname = "Michael"
```

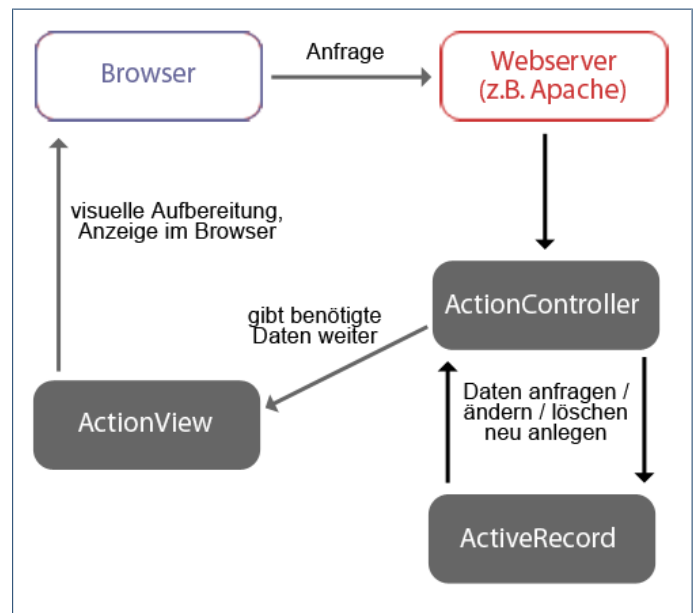
Listing 1

Auch die Suche nach Datensätzen wird stark vereinfacht. Anstatt eines mitunter mühseligen SQL-Befehls kann einfach auf ActiveRecord vertraut werden, der für die einzelnen Spalten automatisch eine find-Methode erstellt:

```
RUBY
```

```
vorname = User.find_by_firstname("Thorsten")
```

Listing 2



Die MVC-Architektur bringt Struktur in eine Anwendung. Dadurch wird sie einfacher zu warten und besser zu erweitern.

Noch einmal: Weder die Methode „firstname“ noch die Methode „find_by_firstname“ musste vom Entwickler definiert werden. ActiveRecord ist schlau genug, um diese aus den vorhandenen Tabellenspalten herzuleiten.

Der Controller heißt in Rails „ActionController“. Er nimmt die Anfragen aus dem Browser entgegen, holt oder verändert benötigte Datensätze mit Hilfe von ActiveRecord und stellt sie mit Hilfe von ActionView wieder im Browser dar. ActionController ist also sozusagen die Kommandozentrale in Rails und sorgt dafür, dass alle brav zusammenarbeiten.

ActionView bewerkstelligt die Ausgabe an den Browser. Hier werden die vom ActionController zusammengestellten Daten entweder als HTML oder auch sehr problemlos als XML wieder dargestellt. Eine große Stärke des ActionControllers in Rails ist die Modularisierung von Templates. Über verschiedene Typen von Templates können sehr gut wiederverwendbare Komponenten gestrickt werden, die eine unangenehme Duplizierung von Code vermeiden.

Noch eine Sprache? Noch ein Framework?

Denken wir kurz nach: Java, .NET, PHP, Python, Perl und viele andere stehen als Sprachen schon seit einigen Jahren treu zur Verfügung. An Frameworks mangelt es auch nicht: Spring, Struts, Symphony, eZComponents, Prado, ZEND Framework, Django und

im weiteren Sinne auch CM-Frameworks wie TYPO3 und ZOPE. Die berechnete Frage, die sich stellt: Braucht die Welt angesichts dieser Litanei an Frameworks und Sprachen noch weitere? Ein einfaches „Ja“ würde Sie wohl nicht recht überzeugen. Ein Vergleich von Ruby on Rails mit PHP und Java zeigt aber, wo Rails seinen Platz findet.

Ruby on Rails im Vergleich mit Java und PHP

Ich hoffe, mit dieser Gegenüberstellung keinem Anhänger der einen oder anderen Technologie auf die Füße zu treten. Keine der Technologien an sich ist gut oder schlecht. Jede hat ihre Vor- und Nachteile, quasi ihren „Platz im Universum“. Javas unbestreitbarer Vorteil ist seine saubere Architektur. Umfangreiche Objektorientierung, gute Wartbarkeit der Anwendungen und gute Performance vor allem im Enterprise-Bereich sind weitere Pros. Ein klares Kontra ist allerdings die lange Entwicklungszeit. Denn die saubere Architektur hat ihren Preis. Auch die große Menge an Codezeilen, die eine Java-Anwendung meist hat, ist ein Problem.

PHP auf der anderen Seite steht für schnelle Erlernbarkeit, eine riesige Community und leichtgewichtige Anwendungen, die in sehr kurzen Entwicklungszeiten entstehen. Wer aber schon einmal versucht hat, eine langlebige PHP-Anwendung zu erstellen, wird die Schwächen kennen: Wartbarkeit, Erweiterbarkeit und Sauberkeit der Systemarchitektur heißen sie. PHP-Gegner lassen häufig den Begriff „Spaghetti-Code“ fallen, um den Mangel an Struktur in dieser Sprache aufs Korn zu nehmen.

Ruby setzt mit Rails zwischen diesen beiden Extremen an: Als vollständig objektorientierte Sprache und mit einer guten MVC-Implementierung ausgestattet, sind die Probleme Erweiterbarkeit und Wartbarkeit hervorragend gelöst. Einige Vergleiche zwischen entsprechenden Java-Applikationen zeigen, dass das Rails-Pendant um den Faktor fünf bis zehn weniger Zeilen an Code benötigt – Sie können sich vorstellen, was das für die Themen Entwicklungszeit und Übersichtlichkeit bedeutet. Der Charakter einer dynamisch typisierten, interpretierten Skriptsprache sorgt zusätzlich für kurze Entwicklungszeiten und eine flache Lernkurve.

Einen großen Vorteil bietet die Sprache Ruby durch ihr „principle of least surprise“, dem Prinzip der geringstmöglichen Überraschung. Denn anders als viele Sprachen birgt Ruby kaum Widersprüche in sich. Wer PHP kennt, weiß zum Beispiel um die unterschiedlichen Funktionsnamen: „strtoupper“, aber „str_replace“. Warum dann nicht „str_to_upper“? Oder „strreplace“?

RUBY

```
3.times { print "Ho!" }
# statt langen for-Schleifen - ergibt: "Ho! Ho! Ho!"
print "Ausgabe" unless variable == 1
# eine unless-Abfrage, zusätzlich direkt nach einer Anweisung, macht vieles kürzer
```

Listing 3

Yukihiro Matsumoto entwickelte Ruby als „Programmer's best friend“. Er wollte den Frust, den er selbst mit inkonsistenten Sprachen erlebt hatte, in Ruby vermeiden. Wir wollen fair bleiben und auch bei Ruby on Rails die Nachteile nicht verschweigen: Mit der Performance einer vorkompilierten Java-Anwendung kann Ruby on Rails nicht mithalten. Das kann aber von einer dynamisch interpretierten Skriptsprache auch nicht erwartet werden. Internationalisierung und UTF-8-Unterstützung sind derzeit genauso wie Transaktionen über mehrere Datenbanken hinweg noch nicht ideal gelöst.

DRY – Don't Repeat Yourself

Ein ganz zentraler Gedanke in der Rails-Community ist folgender: „Weder Daten noch Funktionalitäten sollen sich an irgendeiner

Stelle der Applikation wiederholen.“ Redundanter Code erschwert immer und in jedem Fall die Wartung einer Anwendung. Rails selbst wird auch nach dem DRY-Prinzip entwickelt und hilft Entwicklern, DRY zu programmieren – vor allem durch die MVC-Architektur und das flexible ActionView-Templating.

Convention over Configuration

Durch intelligente Konventionen entfällt in den meisten Fällen die Notwendigkeit für umfangreiche Konfigurationen. So simpel diese Idee der sinnvollen Standard-Konfigurationen auch ist, sie ist in Rails im Gegensatz zu unzähligen Frameworks vorbildlich umgesetzt und enorm zeitsparend.

Testing

Testing ist in Rails direkt ins Framework integriert – Alle MVC-Schichten können getestet werden:

- Unit T. testen ein Model
- Functional T. testen eine Controller-Action
- Integration T. prüfen den Fluss durch einen/mehrere Controller

Durch diese feste Integration in das Framework kann nun auch TestDrivenDevelopment betrieben werden – und das ohne die üblichen Kopfschmerzen, die man aus anderen Umgebungen kennt.

AJAX

Ein Web-Framework ohne AJAX-Unterstützung würde heutzutage kaum eines Blickes gewürdigt. Die Unterstützung von AJAX in Rails ist aus zwei Gründen besonders gut gelungen: Da Rails ein recht junges Framework ist, konnte AJAX von Anfang an ins Konzept einfließen. Dies ergab eine ungewöhnlich native Integration. Mit der Integration der AJAX-Library „Prototype“ [2] hat man darüber hinaus von Anfang an auf das richtige Pferd gesetzt. Nach wie vor schießen die AJAX-Bibliotheken wie Pilze aus dem Boden – nur sehr wenige davon sind einen genaueren Blick wert. Prototype in Kombination mit Scriptaculous [3] ist anerkannter Maßen seit langer Zeit eines der besten AJAX-Teams. Seit der Version 1.1 ist zu den HTML- und XML-Templates eine weitere Template-Gattung dazugekommen: RJS-Templates. Sie bieten die Möglichkeit mehrere AJAX-Aktionen gleichzeitig ausführen zu lassen – und das, ohne eine Zeile JavaScript zu schreiben. Denn RJS generiert automatisch den benötigten JavaScript-Code.

RUBY

```
page.replace_html 'thoughts', :partial => 'thought'
page.visual_effect :highlight, 'thoughts'
page.insert_html(:after, 'thoughts', '<p>Neuer Inhalt</p>')
page.form.reset 'thought-form'
```

Listing 4

Die erste Zeile aus obigem Code ersetzt den HTML-Inhalt des Elements mit der ID „thoughts“ durch ein beliebiges HTML-Template – in diesem Fall eines namens „thought“. Der Vorteil: Das gleiche HTML-Template kann sowohl bei der ursprünglichen Zusammenstellung der Seite verwendet werden, als auch bei einem AJAX-Update. Das bedeutet weniger Code. Die zweite Zeile generiert Code, der die Effekt-Bibliothek „Scriptaculous“ in Schwung bringen wird. Das HTML-Element „thoughts“ wird durch den Effekt „highlight“ kurz aufleuchten und dann wieder normal dargestellt. Zeile drei dürfte Sie nicht mehr überraschen: Über insert_html wird ein neues HTML-Element hinzugefügt (im Gegensatz zu replace_html, wodurch ein vorhandenes ersetzt wird). In unserem Fall wird es nach (:after) dem Element „thoughts“ eingefügt und besteht aus dem Inhalt „<p>Neuer Inhalt</p>“. Die runden Klammern

mern sind bei einem Methoden-Aufruf in Ruby meist freiwillig, die Benutzung steht dem Programmierer frei. In Zeile vier schließlich wird einfach nur das Formular mit der ID „thought-form“ zurückgesetzt und damit eventuelle Einträge darin gelöscht.

Was noch?

Ein ausgeklügeltes E-Mail-System macht Versand und sogar Empfang von E-Mails einfach. Auch aus ActionMailer kann das bekannt flexible Templating verwendet werden.

WebServices sind ebenfalls fest in Rails integriert. Rails erhebt hier keinen Anspruch auf Vollständigkeit. Das heißt, dass standardmäßig nicht sämtliche denkbaren Features aus SOAP oder WSDL zur Verfügung stehen. Vielmehr steht ein ansehnliches Arsenal an Funktionen zur Verfügung, das für die meisten Web-Service-Aufgaben ausreicht.

Code-Generatoren sind eine Spezialität von Rails. Soll zum Beispiel ein neuer Controller angelegt werden, erstellt der Kommandozeilen-Befehl „ruby script/generate controller Name“ die komplette notwendige Dateistruktur samt Codegerüst.

Rails in freier Wildbahn

Bisher war alles nur graue Theorie. Doch obwohl Ruby on Rails noch ein verhältnismäßig junges Framework ist, hat es seine Feuertaufe bereits in zahllosen Web-Projekten bestanden:

www.eins.de

Das Online-Community-Network muss tagtäglich die beachtliche Menge von ca. 1,2 Millionen Seiten ausliefern.

www.basecamphq.com

Business Week zeichnete die Site mit „Best of the Web 2005“ aus.

www.niketorino.com

Nike hatte anlässlich der Olympischen Winterspiele 2006 in Turin eine Themen-Site in über 30 Sprachen erstellt.

Zahllose weitere

- www.calendarhub.com
- www.campfirenow.com
- www.fluxiom.com
- www.odeo.com

Im Rails-Wiki [2] werden viele weitere Projekte vorgestellt.

Links und Literatur

[1] Das Projekt-Tool Basecamp: <http://www.basecamphq.com>

[2] Weitere Rails-Projekte: <http://wiki.rubyonrails.com/rails/pages/RealWorldUsagePage1>

DER AUTOR



Tobias Günther leitet seit zweieinhalb Jahren die Web-Agentur puremedia in Stuttgart (www.puremedia-online.de). Schwerpunkte seiner Arbeit sind TYPO3-Extension-Entwicklung, AJAX und Ruby on Rails.