Bevier F.F.

Das japanische Juwel

Kapitel 4: Nützliches

bussole InformationsVerlag

bussole InformationsVerlag (bussole IV) versucht mit seinen Veröffentlichungen die Erkenntnis zu untermauern, dass Information keine nachgeordnete, abgeleitete physikalische Größe ist, sondern vielmehr die Generalisierung aller physikalisch relevanten Erscheinungen und damit Basis der Erscheinungen unserer ganzen Welt, bis hin in den soziologischen oder psychologischen Bereich.

Die verblüffende Folgerung: **Information ist messbar, zählbar, wiegbar**.

Bevier F.F.

Das japanische Juwel

Kapitel 4: Nützliches

aktualisiert für Ruby on Rails 2.0

bussole InformationsVerlag

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Hinweise zum Autor

Der Autor hat nach einem Diplom-Studium der Physik an der Universität Karlsruhe (Technische Hochschule) mit stark theoretischem Schwerpunkt und besonderer Vorliebe für Funktionalanalysis 12 Jahre in der Industrie gearbeitet. Er arbeitet u.a. an AS/400 Programmen im betriebswirtschaftlichen Bereich für eine international tätige Softwarefirma.

In seiner eigenen Firma begann er mit der Programmierung einer Kerneldatenbank und entwickelte parallel dazu eine neue Form der Mathematik - die Informationsmathematik.

Rechtliches

Dieser Text ist eine Aktualisierung von Kapitel 4 des Buches „Das japanische Juwel“: ISBN-13 978-3-935031-05-9

Redaktion, Layout, Umschlaggestaltung: Klaus Boos

Inhaltsverzeichnis

Die angeführten Links stammen von Anfang 2008 – eine spätere Gültigkeit kann deshalb nicht gewährleistet werden.

Download:
Die Ruby/Rails-Umgebung sowie die Sourcen der Erstanwendung „FirstRails“ und der Datenbankbeschreibungen können von http://www.bussole.de/Rails/FR.htm herunter geladen werden.

Urheber der freien Software Ruby© ist Yukihiro Matsumoto <matz@netlab.co.jp>, Rails™ und Ruby on Rails™ sind eingetragene Markenzeichen von David Heinemeier Hansson.

4 Nützliches

4.1 Rails-Innereien

4.1.1 Das Rails Verzeichnis

Unter Windows® findet sich das aktuelle Rails-Verzeichnis – mit allen Sourcen – in der Ruby-Umgebung im „lib“-Ordner als Teil des RubyGems-Angebotes. Mit einer Ruby 1.8-Version ist das der Pfad:

```
...\Ruby\lib\ruby\gems\1.8\gems
```

Falls Ruby, Version 1.8, direkt auf dem Verzeichnis „C“ installiert wurde, ist dies also „C:\Ruby\lib\ruby\gems\1.8\gems“.

4.1.2 Handbuch und Tutorials – OpenRoRBook

Umfangreiche Erklärungen, oft mit Beispielen, werden auf dem Rails-Wiki angeboten, das in der Übersichtsseite unter dem Stichwort „Tutorial“ auch einige Anleitungen auflistet. Die HowTos sind zumeist ebenfalls sehr nützlich.

Doch auch auf anderen Seiten lässt sich Interessante über Rails entdecken, so das im Buch verwendete Tutorial Nr. 2 oder das mehrmals erwähnte 4-Tage-Tutorial, beide in einer Aufstellung „Top 12 Ruby on Rails Tutorials“ zu finden so wie zwei weitere, hoch interessante Artikel: „Ajax on Rails“ und „Distributing Rails Applications“.

Ruby on Rails entwickelt sich auch so explosionsartig, dass weiterhin kein Mangel an aktuellen Erläuterungen oder Anleitungen zu befürchten ist – und das auch in Deutsch.

So kam 2007 nicht nur ein kurzes Online-Tutorial ins Internet, auch ein umfangreiches Open-Source-Projekt „OpenRoRBook“, das als Software in Ruby on Rails selbst geschrieben ist, wurde zum Jahresende 2007 durch eine Initialspende des bussole InformationsVerlages aus der Taufe gehoben und beschreibt ausführlich die Installation von Ruby und Rails mit MySQL und Aptana/RadRails – und das sowohl für Windows® als auch für Ubuntu. Zum 01. des Jahres 2008 lag bereits eine grundlegende Einführung „Hello World“

vor, während die zugehörige Software mit den beschreibenden Kapiteln dann ab Februar schrittweise publiziert wurde.

„Hello World“ führt Einsteiger in Ruby on Rails ein, die zwar bereits programmieren können – vorzugsweise in Skriptsprachen – sowie SQL beherrschen, die jedoch bisher noch nichts mit Rails zu tun hatten. Als generell erster Schritt wird die Neuanlage von Projekten vorgestellt: Hier wird die typische Rails-Ordnerstruktur erzeugt, die nicht nur die verschiedenen Elemente einer Anwendung nach ihrem Charakter trennt, sondern auch die grundlegende Architektur von Rails demonstriert. Neben den Generatoren, die Rails sehr umfassend zur Erzeugung seiner Objekte zur Verfügung stellt, werden darüber hinaus die Grundlagen der Dateiverwaltung und des Browser-Handlings offengelegt.

4.1.3 Rails-Informationen und Steuerdateien

Die Rails-Informationen, die im „Welcome“-Fenster „index.html“ unter „About your application’s environment“ angezeigt werden, benutzen die Sourcen, die im aktuellen Basismodul von Rails im Ordner „builtin“ gespeichert sind (beispielsweise Version 2.0.2):

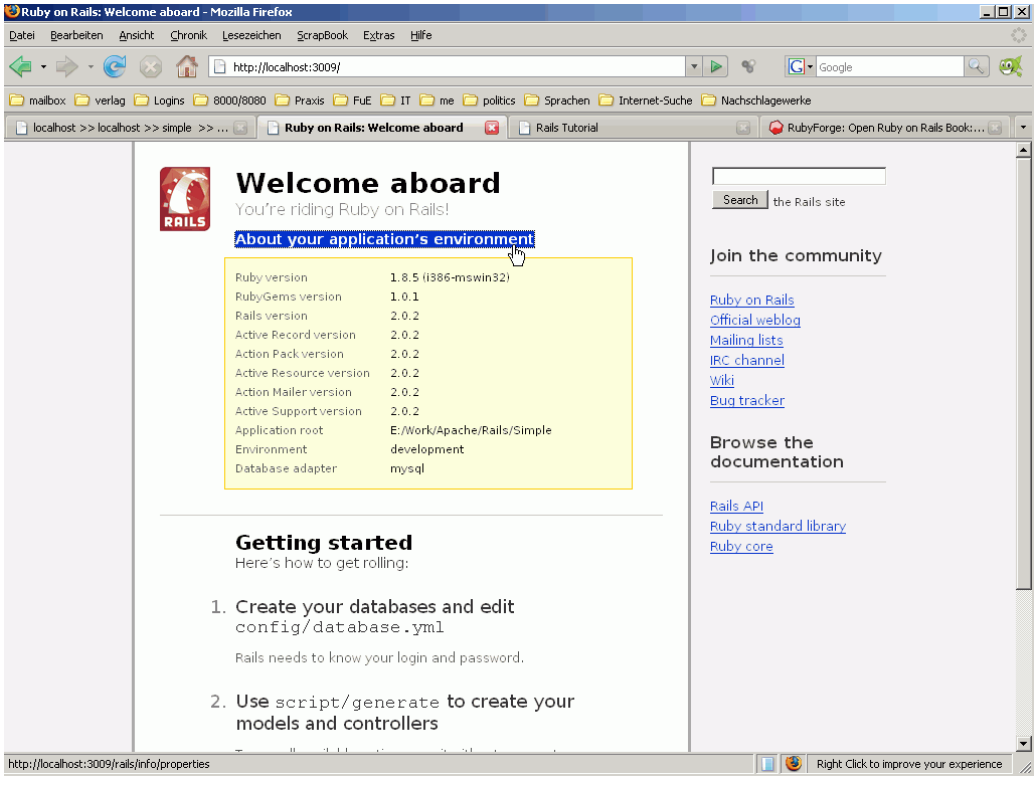
```
...\rails-2.0.2\builtin\rails_info\rails
```

Mit dem Pfad aus „Das Rails Verzeichnis“ ergibt sich demnach „C:\Ruby\lib\ruby\gems\1.8\gems\rails-2.0.2\builtin\rails_info\rails“.

Interessant dabei ist, dass die Anzeige des Welcome-Fensters kein vorgefertigtes HTML-File verwendet, sondern ausgehend von dem Befehl der Quellendatei „info_controller.rb“

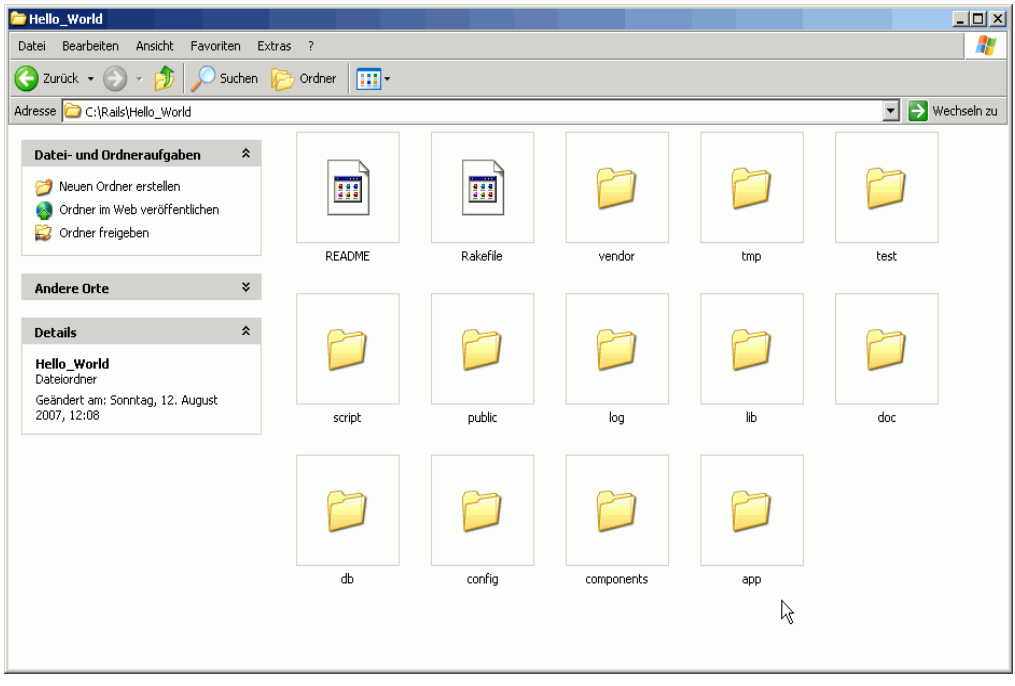
```
render :inline => Rails::Info.to_html
```

die Daten als einfachen Text zusammenfügt, der beispielsweise wie folgt aussehen kann:



Steuerdateien, die für den Ablauf der eigenen Applikation bedeutsam sind, befinden sich in den beiden Rails-Ordnern der jeweiligen Anwendung „public“ und „config“.

Diese Ordner werden im „Skelett“ eines jeden neuen Rails-Projektes automatisch erstellt und mit den erforderlichen Sourcen versehen (siehe „Hello World“):



So ist „.htaccess“ in „public“ für die Anpassung der Verarbeitung von Apache gedacht – wie beispielsweise des „RewriteBase“-Statements zur Anbindung als Alias-Verzeichnis – während der Ordner „config“ Konfigurationsdaten wie die zur Datenbankanbindung offeriert („database.yml“) oder die Integration eigener Source-Verzeichnisse mit „config.load_paths“ („environment“) ermöglicht. Auch die Logik der URL-Aufbereitung, sprich der Zuordnung der URL-Syntax zu ausführbaren Controller-Aktionen und Parameteraufbereitungen („routes“) oder der Umfang der jeweiligen „Umgebungsaktionen“ von Rails kann verändert werden (Ordner

„environments“). Nur die Datei „boots“ sollte nicht angefasst werden, sagt ihr erstes Kommentar-Statement deutlich aus.

Weniger der Steuerung als der Dokumentation dient der Ordner „doc“, der die automatisch erzeugte Dokumentation verwalten und präsentieren kann.

4.1.4 Ruby-HTTP-Server, Caching-Strategien und Klassenvariable

Der WEBrick-Server wird automatisch mit Ruby mitgeliefert und Mongrel lässt sich über RubyGems sehr bequem mit:

```
gem install mongrel
```

in das eigene System integrieren.

Die Vorgaben für den WEBrick-Server wie „port = 3000“ können ebenfalls im Rails-Verzeichnis unter dem Basismodul gefunden werden:

```
...\rails-2.0.2\lib\commands\servers
```

(wie etwa C:\Ruby\lib\ruby\gems\1.8\gems\rails-2.0.2\lib\commands\servers)

Bei Verwendung dieser Ruby-Server ist jedoch zu beachten, dass sie sich hinsichtlich der „in-memory“-Speicherung nicht „CGI“-konform verhalten, wie aus dem Rails-Kommentar von „caching.rb“ im Ordner „action_controller“ ersichtlich ist:

> MemoryStore: Keeps the fragments in memory, which is fine for WEBrick and for FCGI (if you don't care that each FCGI process holds its own fragment store). It's not suitable for CGI as the process is thrown away at the end of each request. It can potentially also take up a lot of memory since each process keeps all the caches in memory.

Das bedeutet, dass beispielsweise mit Apache und CGI alle Klassen aufgrund der immer neuen Prozesse auch immer neu initialisiert werden: Klassenvariablen haben somit generell keinen Bestand – ein Fakt, das ab Ruby 1.8.5/Rails 1.2.x aber auch unter WEBrick bedeutsamer geworden ist, während Mongrel Klassen weiterhin komfortabel handhabt.

Eine kleine Rechnung zeigt jedoch auf, dass die „in-memory“-Speicherung generell mit Vorsicht zu genießen ist, nicht nur im Zusammenhang mit Sessions:

> But remember that 1000 active sessions each storing a 50kb object could lead to a 50MB memory overhead.

Caching-Strategien

Um hier größere Unabhängigkeit zu gewährleisten, bietet Rails deshalb mehrere Caching-Strategien auf Template-Ebene in „caching.rb“ an, die wie üblich ausführlich erläutert werden. So wird das „Page Caching“, das Zwischenspeichern ganzer HTML-Dateien, zwar als sehr schnell bezeichnet, ist jedoch nur für „zustandslose“ Seiten geeignet. „Action Caching“ rettet die Ausgabe einzelner Aktionen und noch „kleinere“ Bestandteile der Applikation können über „Fragment Caching“ gesichert werden, das dazu mehrere Methoden offeriert: Das oben erwähnte „MemoryStore“, dann „FileStore“, das den gewünschten Inhalt im Rails-Ordner „cache“ zwischenlagert, „DrbStore“ mit eigenem, geteilten Drb-Prozess und „MemCacheStore“, das ein externes Tool „Memcache“, über „gem“ installierbar, verwendet.

4.1.5 Active Record

Im 4-Tage-Tutorial findet sich folgende Beschreibung von Sinn und Zweck des Rails-Modells auf Seite 6:

„The Model is where all the data-related rules are stored, including data validation and relational integrity. This means you can define a rule once, and Rails will automatically apply them wherever the data is accessed.“

Sourcen zur Datenbank-anbindung

Die Sourcen zur Datenbankanbindung befinden sich im Rails-Verzeichnis unter dem aktuellen Datenmodul (hier Version 2.0.2):

```
...\activerecord-2.0.2\lib\active_record
```

(also beispielsweise C:\Ruby\lib\ruby\gems\1.8\gems\activerecord-2.0.2)

Die Komponente „base.rb“ dreht sich dabei um Grundsätzliches rund um SQL, „mysql.rb“ fügt dann den letzten Schliff für MySQL hinzu. Die Kommentare

sind wirklich aussagekräftig – sie beschreiben nicht nur im Überblick, was Aufgabe dieses Moduls ist, sondern erklären genauso detailliert die verwendeten Variablen und Methoden.

Auch die Möglichkeit ist interessant, über Rails-Reflektionen („reflection.rb“) an Informationen über Aggregationen und Assoziationen zwischen ActiveRecord-Objekten zu kommen.

Zu beachten ist dabei, dass das Modul die angebotene Datenbankkommunikation natürlich nur für diejenigen Dateien durchführen, von denen es Rails-Modelle gibt. Es lassen sich über diese (vorhandenen) Anbindungen (connections) jedoch SQL-Befehle für alle Objekte des Datenbank-Servers ausführen, für die die Berechtigungen dazu vorliegen.

CoC (Configuration over Convention): Hinsichtlich der Namensgebung ist zu berücksichten, ob ein oder mehrere Datensätze in den entsprechenden Instanzenvariablen vorgehalten werden, wenn Rails-Bequemlichkeiten (wie ehemals „autocomplete“) ohne weitere Eingriffe benutzt werden sollen.

Anwendungsbeispiel: Der letzte Schritt von FirstRails

4.1.5.1 Datenbank-Abhängigkeiten

Neben generellen Funktionen für das Datenbank-Management wie „database_statements.rb“ oder „schema_definitions.rb“ – untergebracht im Ordner „connection_adapters/abstract“ – sind auch die typenabhängigen Methoden interessant. So werden beispielsweise die Informationen über die Tabellen („tables“), Spalten („columns“) oder Indizes („indexes“) sowohl von der MySQL-Anbindung („connection_adapters/mysql_adapter.rb“) als auch von der für PostgreSQL („connection_adapters/postgresql_adapter.rb“) angeboten, andere Adapter weisen dagegen nur „columns“ auf.

```
MeinModell.connection.tables
```

liefert deshalb zwar im Falle von MySQL oder PostgreSQL die Auflistung der in der Datenbank vorhandenen Dateien zurück, ansonsten freilich stößt der Befehl auf eine undefinierte Methode. Sollte also nicht klar sein, welcher

Datenbanktypus im Zugriff sein kann, ist die Abfrage entsprechend zu bedingen.

Hinsichtlich der Funktion „indexes“ ist kurz anzumerken, dass der Primärschlüssel von Rails (zumindest für MySQL) nicht dazu gezählt wird.

Anwendungsbeispiel: Der zweite Schritt von FirstRails

4.1.5.2 Instantiierungsverhalten

Die ActiveRecord-Objekte werden pro Datensatz instantiiert, während die Datei selbst der Klasse zugeordnet ist. Betreffen Methoden also die gesamte Datei, sollten sie als Klassenmethoden in dem jeweiligen Modell oder Controller gehandhabt werden. Vielleicht aufgrund dieser Konstruktion bietet Rails auch eine Klassenerweiterung der Ruby-Klasse „Class“ für die Accessoren an.

Bemerkenswert ist hierbei auch, dass Rails für jeden Aufruf der abgeleiteten Modell-Klassen ein neues Klassenobjekt erzeugt, was beispielsweise bedeutet, dass Klassenvariablen (und Instanzvariablen) von Modellen keinen Bestand haben. Um solche Variable (oder andere Applikationselemente) trotzdem dauerhaft verwenden zu können, offeriert Rails verschiedene Caching-Strategien. Das Rails-Wiki bietet darüber hinaus noch Session-Objekte zum „Überwintern“ der zu speichernden Daten an:

```
session[:person] = Person.authenticate(user_name, password)
```

Sie lassen sich wie andere öffentliche Attribute von „session“ dann beispielsweise über:

```
Hello #{session[:person]}
```

wieder zurückholen – zumindest, solange das, was gespeichert werden soll, „marshallierbar“ ist.

Sollen ganze Modell-Objekte auf diese Weise zwischengelagert werden, muss ihre Source laut Beschreibung noch in das Session-Objekt integriert werden:

```
model :person
```

4.1.5.3 Transaktionssteuerung (Commit)

Rails betrachtet üblicherweise jede einzelne Datenbankmanipulation als Transaktion, die im Erfolgsfalle über Commit abgeschlossen wird und zur Kontrolle in einer Protokolldatei des Rails-Ordners „log“ genau vermerkt wird. Der Name der aktuellen Log-Datei entspricht dabei einfach der Rails-Umgebung, in der die Anwendung abläuft wie etwa „development.log“.

Zu beachten ist hier im Testbetrieb, dass phpMyAdmin möglicherweise keine Transaktionssteuerung verwendet – werden also Daten über dieses kleine Tool verändert, sollten sie mit Commit bestätigt werden, um zusammen mit Rails nicht zu sonderbaren Ergebnissen zu kommen.

4.1.6 Action Pack

Das Modul „ActionPack“ im Railsverzeichnis umfasst sowohl die Controller- als auch die View-Elemente (hier Version 2.0.2):

```
...\actionpack-2.0.2\lib\action_controller
...\actionpack-2.0.2\lib\action_view
```

(beispielsweise
C:\Ruby\lib\ruby\gems\1.8\gems\actionpack-2.0.2\lib\action_controller)

Für beide Untermodule findet sich wieder ein „base.rb“ – wie immer ausführlich dokumentiert. Dabei drehen sich die Basisfunktionen der View um die Möglichkeiten, Templates mit Ruby (Dateiendung „html.erb“, ehemals „rhtml“), mit XML (Suffix „builder“, ehemals „rxml“) oder Javascript („rjs“) zu erstellen, während diejenigen der Controller sich darum kümmern, diese Templates über öffentliche Methoden („actions“) mit Leben zu füllen – oder eben auf andere Aktionen umzudirigieren.

Die programmtechnische Verbindung von View und Controller erfolgt einerseits über die Instanzvariable „@template“ des Controllers, mit der auf alle öffentlichen Variablen und Methoden der View inklusive der Helferfunktionen – und damit auch auf deren Klassenvariablen – zugegriffen werden kann, und andererseits über eine Variable „@controller“ im Template.

4.1.6.1 Instantiierungsverhalten

Wie im Fall von ActiveRecord instantiiert Rails bei jedem URL-Aufruf die Controller neu – damit werden alle Instanzvariablen ebenfalls neu initialisiert. Darüber hinaus wird für jede Bildschirmausgabe (via „render“) eine neue CGI-Session aufgebaut (falls CGI genutzt wird).

4.1.6.2 Vererbung

Vererbung wird bei Rails gerne eingesetzt, so etwa bei allen Controllern, die generell von „Application“ abgeleitet werden. Dies verschafft jedem Controller Zugang zu den „Application“-Methoden (und den dort eingebundenen Modulen) sowie deren Instanzvariablen, wobei letztere durch die ständige Initialisierung ebenfalls ständig „neu“ erstellt werden.

4.1.7 Helper

Wie das Kapitel „UnderstandingHelpers“ im Rails-Wiki erklärt, ist dieses besondere Modul von Rails dafür gedacht, die zumeist aufwändige Oberflächenbearbeitung zu verschlanken, um die Quelldateien der Views übersichtlich und lesbar zu halten. Viele solcher „Helferlein“ (Daniel Düsentrieb lässt grüßen) liefert Rails bereits selbst mit – zu finden unter dem aktuellen ActionPack-Ordner in einem eigenen Unterverzeichnis „helper“.

Die benötigten Methoden werden dabei denkbar einfach in die übergeordnete View-Verarbeitung integriert. Das CoC-Prinzip verknüpft Template und Helper einfach über den Namen des betreffenden Objektes – der gleichzeitig der Name des entsprechenden View-Ordners der Rails-Anwendung ist – und ergänzt ihn um den identifizierenden Zusatz „_helper“: beispielsweise „todos_helper“.

Natürlich können auch ganz untypische Namen und Ordner für Helper-Methoden genutzt werden. Sollen Helferlein verwendet werden, die der Rails-Konvention nicht entsprechen, so können diese über das Schlüsselwert „helper“ dem betreffenden Controller gezielt zur Verfügung gestellt werden. Zu beachten ist jedoch, dass Rails die eigenen Bezeichnungen immer um

„helper.rb“ erweitert: Eine Helper-Source „xyz_helper“ wird deshalb nur über „xyz“ angesprochen:

```
helper  :xyz
```

Sollen darüber hinaus auch eigene Ordner für die diversen Helfer benutzt werden, müssen diese der Railsumgebung bekannt gemacht werden. Dafür existiert in der Quelle „environment.rb“ ein Abschnitt:

```
# Add additional load paths for your own custom dirs
```

Im einfachsten Fall, dass der neue Ordner „abc“ im üblichen Rails-Verzeichnissystem untergebracht ist, genügt hier die simple Anweisung

```
config.load_paths += %W(app/helpers/abc)
```

Ein zweiter Weg, Helfermethoden in den Templates der View zu verwenden, ist „helper_method“, das eine Methode des Controllers als „Helfer“ deklariert. Dies funktioniert problemlos auch für eingebundene Module:

```
include MeinModul
helper_method       :modulmethode
```

In der View kann diese Funktion sodann bequem mit

```
<%= modulmethode %>
```

aufgerufen werden.

Für die Templates ist anschließend nur noch zu berücksichtigen, dass eine erforderliche Modul-Instanzvariable „xyz“ mit

```
attr_accessor :xyz
```

zur Verfügung gestellt werden muss. Sie können dann zwar immer noch nicht mit „@xyz“ in der View abgerufen werden, sind aber über

```
controller.xyz
```

ansprechbar.

Kleine Nebenbemerkung: Rails hat überhaupt keine Probleme damit, wenn gleiche Funktionsnamen sowohl im Controller als auch im dazugehörigen Helfer vorliegen.

4.1.8 Initialisierung der Rails-Anwendung

Die Initialisierung durch „initializer.rb“ erfolgt naheliegenderweise im Rails-Basismodul (für Version 2.0.2):

```
...\rails-2.0.2\lib\
```

(beispielsweise C:\Ruby\lib\ruby\gems\1.8\gems\rails-2.0.2\lib\)

Hier wird die vorhandene Umgebung wie die Ruby-Version oder die installierten Rails-Module überprüft und nach Bedarf angebunden. Aufgerufen wird die Initialisierung „Rails::Initializer.run“ dabei von den beiden Sourcen „boot.rb“ und „environment.rb“ im „config“-Ordner der eigenen Rails-Anwendung.

4.1.9 Parameterübergabe

Solange die Kommunikation nicht über Pfade verläuft, die von und zum Browser gehen, funktioniert die normale Strategie: Methoden werden mit Parameterlisten definiert, die je nach Bedarf mit Unterlassungswerten vorbelegt werden können. Bei Ruby dürfen sie sogar in der Anzahl variieren oder gar ganze Funktionen sein (*, &).

Betrifft es dagegen Methoden, die parametrisiert über URLs aufgerufen werden, stellt Rails die Instanzvariable „@params“ für die Controller zur Verfügung. Wird also die Methode „abc“ mit den Parametern „x“ und „y“ aufgerufen wie etwa:

```
http://localhost:3000/application/abc?x='Parameter z'
        &y=true
```

oder über einen Link_to-Befehl wie

```
<%= link_to "Text des Links", :action => "abc",
        :params => {"x" = 'Parameter z', "y" => true} %>
```

so liefert Rails in der Hash-Variablen „@params“ den „geplätteten“ Wert „xParameter zactiondatabasecontrollerapplicationytrue“ mit der einfachen Möglichkeit, über

```
y = @params['y']
x = @params['x']
```

an die Werte der Parameter zu gelangen.

Natürlich ist es besser, nicht direkt auf diese Instanzvariable zuzugreifen, sondern nur über die zugehörige öffentliche Methode:

```
y = params[:y]
x = params[:x]
```

Zu beachten ist, dass „@params“ während der Initialisierung (Routine „initialize“) der Controller noch nicht definiert ist. Auch andere Browser-Kommunikationselemente (wie „flash“ oder „render“) schlagen an dieser Stelle noch fehl.

Und last but not least sollte nicht übersehen werden, dass Parameter, die per URL übergeben werden, immer vom Typ „String“ sind.

Anwendungsbeispiel: Der zweite Schritt von FirstRails

Ein besonderer Parameter innerhalb „@params“ ist das Hash, das alle vorhandenen, eingabefähigen Datenfelder eines Formular-Tags <form> mit ihren jeweiligen Datensätzen – via „attributes“ – korreliert. Es weist dabei den Namen der Instanzvariablen auf, die das ActiveRecord-Objekt des zu verarbeitenden Satzes enthält. Heißt das Modell beispielsweise „Eigenschafts“ und wird

```
@eigens = Eigenschafts.find(params[:id]
```

in einem Formularelement über

```
<%= text_field 'eigens', 'beschreibung' %>
<%= text_area 'eigens', 'anmerkung', 'rows' => 5 %>
```

verwendet, so finden sich die von den Benutzern eingegebenen Werte für die Dateifelder „Beschreibung“ und „Anmerkung“ im Parameter „:eigens“ wieder – einem Hash, das diese Feldnamen mit den aktuellen Eingaben der Anwender enthält.

Beachtenswert ist, dass zu den eingabefähigen Feldern, die Rails als Parameter akzeptiert, auch „versteckte Felder“ gehören. Soll also beispielsweise ein Wert, dessen Klartext über Ajax verändert wurde, passend zur Anzeige dem Programm mitgeteilt werden (um beim Speichern die Übergabe aktueller Inhalte zu gewährleisten), so genügt es, diesen Wert in in einem „Hidden“-Feld zu fixieren.

Wichtig ist hierbei, dass Rails seine Feldwerte über die Namen bestimmt, weshalb dem „Hidden“-Feld neben seiner DOM-Id auch ein eindeutiger Name vergeben werden muss – im Übergabeparameter „@params“ wird dann der Inhalt aus dem „value“-Attribut dieses Feldes übermittelt.

Für das lokale Feld mit der lokalen Variable „mem“ sowohl als DOM-Id als auch als Name sowie dem Wert „value“ der lokalen Variable „show_id“:

```
<input id="<%= mem %>" name="<%= mem %>"
       type="hidden" value="<%= show_id %>" />
```

überreicht die Anweisung (im Programm):

```
x = params[:mem]
```

demnach den aktuellen Wert der lokalen Variablen „show_id“.

Rails bietet dafür zwei verschiedene Hidden-Tags an. Der erste erwartet zwei Bestandteile, die Datei/Modell und Feld angeben müssen, der zweite dagegen ermöglicht eine datei-unabhängige Definition. In beiden Fällen wird daraus ein HTML-Tag erstellt, das neben der DOM-Id noch das Attribut „name“ aufweist

um sicherzustellen, dass die Parameterübergabe mit „@params“ den Inhalt dieses Hidden-Feldes mit berücksichtigt.

Die Parametervariable „@params“ darf natürlich auch manuell korrigiert werden, um etwa die komfortable Rails-Verarbeitung hinsichtlich des „Validate“ und „Save“ zu nutzen, die diese Variable ebenfalls kennen und damit in Betracht ziehen können.

Anwendungsbeispiel: Der dritte Schritt von FirstRails

Was auch noch sehr praktisch ist: Die Kommunikation via „@params“ funktioniert nicht nur als Einbahnstraße vom Browser zu Rails, sondern auch umgekehrt. Im Programm kann diese Instanzvariable also vor einem „render“-Befehl verändert und dann im Template benutzt werden – solange eben Datentypen verwendet werden, die „marshallierbar“ sind.

Anwendungsbeispiel: Der vierte Schritt von FirstRails

4.1.10 Die Ruby/Rails-Dokumentation mit Rake

Wie viele andere moderne Entwicklungsumgebungen erlaubt es Ruby, die mit „#“ gekennzeichneten Source-Kommentare zur Dokumentation zu verwenden – was Rails ausgiebig nutzt: Im Anwendungsordner „doc“ führt die Datei „README_FOR_APP“ den passenden Rake-Befehl auf, der einfach nur:

```
rake appdoc
```

lautet. Dieser kann wieder (unter Windows®) in einer bat-Datei untergebracht werden und bearbeitet daraufhin diejenige Rails-Anwendung, in deren Pfad er sich befindet. Dazu erzeugt er einen Unterordner „app“ von „doc“, in welchem die Dokumentation erstellt oder bei Bedarf aktualisiert wird.

Die Datei „README_FOR_APP“ ist dabei selbst Bestandteil der Railsdokumentation – wird sie umbenannt, erfolgt keinerlei Aktualisierung mehr. Das deutet auch auf einen weiteren, jedoch nicht sonderlich tragischen Nachteil dieser Art hochaktueller Dokumentation hin: Die dokumentationseigenen Bezeichnungen liegen natürlich in Englisch vor.

Zur „README_FOR_APP“ ist noch anzumerken, dass der Text, den sie enthält, in die erzeugte Startseite „index.html“ der Dokumentation einfließt und somit als anwendungsübergreifende Einführung dienen kann.

Für die weitere Dokumentation wird danach alles, was mit „#“ in den Sourcen gekennzeichnet ist, ausgegeben – dazu gehören auch Befehlszeilen, die zum „späteren Gebrauch“ nicht gelöscht, sondern auf diese Weise nur deaktiviert wurden.

Interessant sind dabei die Möglichkeiten, den Text aufzubereiten: Die üblichen HTML-Tags funktionieren prächtig und über

```
# ....blablabla (#xzy) blablabla
```

wird sogar ein Link innerhalb der Dokumentation auf die Erläuterungen zur Methode „xyz“ eingerichtet. Die Schriftgröße eines Wortes „abc“ kann mit

```
# ....blablabla +abc+ blablabla
```

verringert werden und eine Aufzählung „Punkt 1, Punkt2“ in Listenform wird erstellt mit:

```
# * Punkt 1
# * Punkt 2
```

Zu berücksichtigen ist hier nur, dass nicht versehentlich die Schreibweise „# *Punkt 2“, also ohne trennendes Leerzeichen, benutzt wird – dann nämlich wird die Eingabe wie ganz normaler Kommentartext behandelt.

Das Ergebnis liegt anschließend browsertauglich vor und bietet eine Übersicht aller Source-Dateien an mit all ihren öffentlichen Klassen und Methoden, wobei praktischerweise noch die Möglichkeit besteht, sich den Source-Code derselben anzusehen. Schön ist auch die automatische Verlinkung auf bestehende Klassen, solange der Klassenname in der exakten Schreibweise im Text aufgeführt worden ist.

Die Quellen zur Ruby-Dokumentation finden sich im „rdoc“-Ordner der aktuellen Ruby-Quelldateien (nicht Rails!), die die unterschiedlichen Möglichkeiten, den Text aufzubereiten, in der Source-Datei

```
simple_markup.rb
```

im dortigen Unterordner „markup“ ausführlich beschreiben. Liegt Ruby direkt auf dem Wurzelverzeichnis, ist der vollständige Pfad dieses Verzeichnisses unter Windows® also:

```
C:\Ruby\lib\ruby\1.8\rdoc\markup
```

4.1.11 Ajax mit Rails

Wie unglaublich einfach Rails die Verwendung von Ajax (Asynchronous Javascript und XML) macht, führt der Artikel „Ajax on Rails“ vor. Die üblichen Kommunikationsbefehle im Browser-Template „link_to“ und „form_tag“ werden dafür durch „link_to_remote“ und „form_remote_tag“ ersetzt. Diese Variante ermöglicht es, neben der Ausführung von Programmteilen („:action“) auch den Identifikator des jeweiligen DOM-Objektes mitzugeben, das gezielt bearbeitet und verändert werden soll (und beispielsweise bequem über den DOM-Inspektor des Firefox begutachtet werden kann):

Dazu muss in den HTML-Tags die Option „id“ verwendet werden wie etwa:

```
<div id="xyz">
```

Dann kann über

```
<%= link_to_remote( "blablabla", :update => "xyz",
        :url =>{ :action => :methodname }) %>
```

die Ajax-Engine aktiviert werden, wobei die Dokumentation weitere Optionen aufführt – so kann unterschiedliches Verhalten beim Aktualisieren erreicht werden, je nachdem, ob das Ganze erfolgreich verlief oder schief ging:

```
link_to_remote "Delete this post",
        :url => { :action => "destroy", :id => post.id },
        :update => { :success => "posts", :failure => "error" }
```

4.1.11.1 Javascript

Wie das Rails-Wiki in „UnderstandingAjax“ erklärt, wird dazu die Javascript-Prototype-Library „prototype.js“ benutzt, die jeder Rails-Anwendung im Ordner „public/javascripts “ mitgegeben wird mit so kleinen Bequemlichkeiten wie „$()“, das das Javascript-Element für eine mitgegebene „id“ zurückgibt oder „$F()“, das Feldwerte von Formelementen zur Verfügung stellt:

```
var d = $('myDiv');
var divs = $('myDiv','myOtherDiv');$F('userName')
```

Auch „Try.these“ ist im ungewissen Milieu der Browser nicht ungeschickt, da es mehrere Methoden ausprobiert, bis endlich eine funktioniert. Ob Fehler im Javascript auftreten, ist nämlich nicht ganz so einfach festzustellen, denn meistens rührt sich der Browser in dem Fall einfach nicht – doch auch hierfür bietet der Firefox eine Kontrolle (unter „Extras“): die Fehler-Konsole.

Bei Javascript die Fehlerkonsole im Auge behalten!

Der folgende Befehl lässt sich indessen schon per Augenschein kontrollieren:

```
<%= javascript_tag "$('xyz').scrollTo()"%>
```

dies erlaubt es, per Rails-Anweisung „javascript_tag“ die Funktion „scrollTo()“ zu verwenden, die den Browser auf das besagte Element „xyz“ ausrichtet – doch das nur als kleine Kostprobe.

Um jene Funktionalitäten in Rails zu integrieren, wird der Helper „prototype_helper.rb“ mit wieder erfreulich viel Dokumentation angeboten. Aktiviert werden die jeweiligen Javascript-Bibliotheken im Template über

```
<%= define_javascript_functions %>
```

das den gesamten vorhandenen Skript-Code in das von Rails erzeugte HTML-File kopiert oder gezielt über

```
<%= javascript_include_tag 'prototype' %>
```

das die Javascript-Funktionalität, hier von „prototype.js“, über Links zur Verfügung stellt.

Unter den Javascripts und den Helferlein finden sich natürlich noch mehr Möglichkeiten – und genauso natürlich bietet Rails auch die Möglichkeit, eigenen Code im Javascript-Tag unterzubringen.

Anwendungsbeispiel: Der dritte Schritt von FirstRails

Der Javascript-Tag selbst wird in der Source-Datei „javascript_helper.rb“ definiert, wo auch folgendes Beispiel einer Hinweismeldung zu finden ist:

```
<%= javascript_tag "alert('All is good')" %>
```

Dies installiert den Javascript-Befehl für das Meldungsfenster mit dem Ausgabetext „All is good“.

Doch „alert“ kann noch mehr. So lassen sich etwa Feldattribute der DOM-Id „xyz“ mit folgenden Varianten begutachten:

```
<%= javascript_tag "alert($('xyz'))"%>
<%= javascript_tag "alert($('xyz').id)"%>
<%= javascript_tag "alert($('xyz').value)"%>
<%= javascript_tag "alert($F('xyz'))"%>
```

Existiert das Objekt (noch) nicht, wird der Hinweis ausgegeben, dass das Gesuchte nicht definiert ist.

Weitere praktische Anwendungen des Javascript-Tags sind die Anzeige von Informationen über den benutzten Browser per Navigator-Objekt:

```
<%= javascript_tag "document.write
      (navigator.appCodeName)" %>
```

oder das Neuladen der Seite, das den Browser auf den Ausgangszustand bringt, bei dem die angezeigten Werte wieder mit den abgespeicherten Werten in der Datenbank übereinstimmen:

```
<%= javascript_tag "self.location.reload()" %>
<%= javascript_tag "location.reload()" %>
```

Zuordnungen lassen sich mit den „Abkürzungen“ von „prototype.js“ ebenfalls bequem durchführen, wobei sich in der vorliegenden Konfiguration diese Schreibweise als erfolgreich erwies:

```
<%= javascript_tag "$('xyxy').value=$F('abab')"%>
```

mit den beiden lokalen „Hidden“-Feldern

```
<input id="xyxy" type="hidden"
        value="" />
<input id="abab" type="hidden"
        value="<%= testvariable_Rails%>" />
```

Warum die vorliegende Version und die Schreibweise so betont wurde? Weil der Versuch <%= javascript_tag "$F('xyxy')=$F('abab')"%> merkwürdigerweise erfolglos blieb. Der erwähnte „alert“-Befehl kann dabei wieder auf allereinfachste Weise zeigen, wie sich die Werte über Javascript manipulieren lassen.

Javascript kann gelegentlich den Kontrollfluss verlieren

Er zeigt zum Beispiel aber auch, dass manchmal sogar Javascript sich nicht ganz so verhält wie erwartet. So verlor es nach dem folgenden Statement in „_dropdrag.rhtml“:

```
<%= javascript_tag "if($F('#{change_allowed}') == 0)
        {$('#{drop_icon}').style.display = 'none'}"%>
```

den Fokus – ein nachfolgender „alert“-Befehl wurde schlicht nicht mehr ausgeführt. Nachdem dieses Statement entfernt wurde, weil es dank Rails durch ein CSS-Format „hide“ ersetzbar war, lief alles wieder wie erwartet.

Anwendungsbeispiel: Der dritte Schritt von FirstRails

Bei Javascript die Fehlerkonsole im Auge behalten!

Über welches interne Problem Javascript hier stolperte, war seinerzeit nicht herauszufinden, denn wenn Javascript auf Fehler stößt, wird es einfach ignoriert – nur die Fehlerkonsole kann dann unter Umständen erklären,

warum Befehle sang- und klanglos untergehen. Bei dem nächsten „Fokus-Verlust“ lag es dadurch jedoch rasch nahe, woran es liegen könnte – es handelte sich um den Javascript-Befehl „delete“:

```
<%= javascript_tag "if($('xyz_id')){delete $('xyz_id')} "%>
```

Das DOM-Objekt, das dadurch gelöscht wurde, fehlte wohl dem „prototype“-Framework später irgendwo: Vorsicht also damit, DOM-Objekte zu entfernen.

Auch hier – der einfache „alert“-Befehl demonstrierte klar, dass es so nicht ging.

Diese simple, doch vielsagende Funktionalität führt auch deutlich vor, wie differenziert Javascript mit den diversen Instanzen derselben HTML-Seite umgeht. Diese Fähigkeit von Javascript wird besonders dann bedeutsam, wenn es darum geht, das, was die Anwender sehen, mit dem, was das Programm weiß, abzugleichen. Das kann gelegentlich recht unterschiedlich ausfallen und verlangt deshalb eine sehr präzise Vorgehensweise im Umgang mit den HTML-Instanzen.

Beim Blättern die Abstimmung von Ajax-Werten im Browser und im Programm kontrollieren!

Das Vor- und Zurückblättern beispielsweise ist eine Sache des Browsers, von dem das „ferne“ Ruby nicht wirklich etwas erfährt – dies wird dann gefährlich, wenn (wie es bei Ajax-veränderten Werten vormals geschah) die ursprünglichen Feldwerte nach dem Blättervorgang in der Browseransicht wieder zum Vorschein kommen können. In solch einem Fall würde das Programm auf einen unangenehmen Konflikt stoßen, wenn es sich die bisherigen Arbeitsschritte separat „gemerkt“ hätte: Die Benutzer sehen etwas anderes, als das Programm zu wissen glaubt. Versuchen sie dann das, was ihre Augen ihnen verkünden, auch zu speichern, könnte ohne weitere Kontrolle also etwas anderes in der Datenbank ankommen, als die Anwender erwarten würden – denn die Daten werden immer noch vom Programm und nicht vom Browser beherrscht.

Cachen im Zusammenhang mit dem Browser ist mit Vorsicht zu genießen

Merke: Im Zusammenhang mit Ajax ist die „Zwischenlagerung“ von Arbeitsschritten im Programm sehr genau zu kontrollieren, weil hier (fast) nur Javascript den Abgleich zwischen dem Programm und dem, was der Browser

anzeigt, ermöglichen kann, denn Javascript bemerkt jedes Vor und Zurück und unterscheidet dabei sehr genau die Reihenfolge der „Blätter-Instanzen“.

Auch wenn der Blättervorgang selbst aktuell noch nicht abgefragt werden kann, durchläuft Javascript seine Befehlskette häufig erneut, wenn der Browser sein Bild frisch aufbaut – wieder einfach nachprüfbar über das simple „alert“. Diese Tatsache lässt sich über „remote_function“ ausnutzen, das erforderliche Werte aus Browserelementen (die das Attribut „value“ aufweisen müssen) zur Abstimmung oder Kontrolle übertragen kann:

```
<%= javascript_tag(remote_function(
        :url => { :action => "js", :controller => "application",
            :params => {:idnr => idnr, :cont => cont } },
        :with   => "'mem='+ $F('#{mem}')" )) %>
```

Der obige Befehl ruft die Methode „js“ im Controller „application“ mit den „normalen“ Parametern „idnr“ und „cont“ (beides lokale Variable) auf und übergibt dann noch mit „:with“ den über Javascript bestimmten Wert eines Feldes, dessen DOM-Id durch eine Variable „mem“ vorgegeben ist.

Zu beachten ist hier, dass kein Ziel für ein „Update“ vorgegeben ist, da das Ganze nur als Einbahnstraße vom Browser zu Ruby gedacht ist (s. „render :nothing => true“ in Methode „js“).

Anwendungsbeispiel: Der vierte Schritt von FirstRails

Die entsprechende Methode von „application“ (oder einem der angedockten Module) in Rails würde dann wie folgt aussehen:

```
def js
        x = params[:mem]
        y = params[:idnr]
        z = params[:cont]
        ...
        render :nothing => true
                # keine Ausgabe an den Browser veranlassen
end
```

Ein weiterer Weg, die Abstimmung durchzuführen, kann „huckepack“ beim Abspeichern selbst geschehen, zumindest dann, wenn dies über die Ajax-Variante „form_remote_tag“ geschieht und der Wert nur beim Speichern selbst benötigt wird. Über die Option „onsubmit“ von „form_remote_tag“ lässt sich nämlich ebenfalls eine Remote-Funktion aufrufen, der per Javascript Werte mitgegeben werden können.

Soll ein steuernder Wert, dessen Klartext über Ajax verändert wurde, jedoch nur – passend zur Anzeige – dem Programm mitgeteilt werden, um beim Speichern korrekte Inhalte zu verwenden, so genügt es auch, diesen Wert in einem „Hidden“-Feld zu fixieren. Rails führt daraufhin alles Notwendige selbst aus.

Nebenbemerkung: Ganz außen vor soll hier die Frage bleiben, was denn mit Anwendern geschieht, die überhaupt kein Javascript nutzen – denn dann taugt natürlich die schönste interaktive Webseite nicht mehr viel. Deshalb ist es eine der grundlegendsten Entscheidungen hinsichtlich der Oberfläche, ob Ajax verwendet wird oder nicht: Ob Drag & Drop oder die anderen Ajax-Nettigkeiten benutzt werden dürfen, beeinflusst schon sehr den Komfort einer Anwendung, zwingt dann aber auch dazu, von den Benutzer Javascript zu verlangen.

Was zum Schluss noch erwähnenswert ist, sind Verhaltensunterschiede zwischen „Original-HTML“ und Javascript wie in folgendem Fall, der nur beim Firefox (Version 2.0.0.3) auftrat:

Firefox: Javascript-Reset und HTML-Reset können unterschiedliche Ergebnisse liefern

Das Problem fiel im Zusammenhang mit der Reset-Funktion

```
<%= javascript_tag "form.reset()"%>
<%= javascript_tag "$('formID').reset()"%>
<%= javascript_tag "document.reset()"%>
```

auf: Wurden Textfelder geändert und mit dem Javascript-Reset bereinigt, so tauchte deren Wert nach einem Blättervorgang und sogar nach der Speicherung wie aus dem Nichts wieder auf, obwohl – nach der Speicherung – sogar der Quelltext die richtigen Inhalte angab. Wurde die Bereinigung dagegen über das HTML-Reset

```
<input type=reset>
```

ausgeführt, benahm sich Firefox korrekt in Bezug auf die Textfelder.

Hinsichtlich der Ajax-kontrollierten Felder wie den Listen oder Drag & Drop verhielt es sich dagegen umgekehrt.

4.1.11.2 Verbleibende Grenzen gegenüber internen GUIs

Was deshalb trotz Ajax und allem Rails-Komfort gerade für Programmierer, die bisher nur mit internen Oberflächen arbeiteten, bedauernswert bleiben dürfte, ist die Tatsache, dass die grundsätzliche Zerlegung von Programm und Darstellung einem völlig freien Datenaustausch zwischen beiden natürlich im Wege steht. Während der Rails-Precompiler große Freiheiten hinsichtlich der Übergabe von Ruby-Variablen an den Browser gewährt, ist bei der Entgegennahme von HTML-Werten selbstverständlich eine entsprechende Transaktion des Browsers erforderlich. Dabei ist immer zu berücksichtigen, dass das Ganze über Internet und möglicherweise sogar über langsame Leitungen erfolgen könnte, sodass diese Vorgänge soweit als möglich „komprimiert“ erfolgen sollten – das war schließlich mit ein Grund für den Siegeszug der Ajax-Technologie, die nur noch „Teile“ des darzustellenden Ganzen bearbeitet. Das erlaubt es ja auch, „Leitung zu sparen“ bei dem erhöhten Verkehrsaufkommen, das steigender Bedienungskomfort so mit sich bringt.

Warum überhaupt Werte des Browsers interessant sein können? Beispielsweise für die Frage, welches Listenelement von den Anwendern ausgewählt wurde oder auch, auf welcher Blätter-Instanz die Benutzer gerade Halt gemacht haben. Schließlich erlaubt ein Browser häufig, dass derselbe Vorgang beliebig oft durchgeführt wird: Die Anwender können beispielsweise über Blättern oder Links beliebig oft ein und denselben Satz zur Verarbeitung auswählen – und dann zwischen den einzelnen aufgeblätterten Datensätzen hin- und herhüpfen. Das verarbeitende „ferne“ Programm kann hier unmöglich die Kontrolle darüber behalten, welcher Satz den nun tatsächlich derjenige ist, den die Anwender gerade angesprungen haben. Es muss also schon prüfen, was genau

die Benutzer gerade im Griff haben – und dafür braucht es ein HTML-Element, das mit den bearbeiteten Sätzen eindeutig korreliert ist, mit dem also gewährleistet werden kann, dass ein Satz auch eindeutig identifiziert werden kann.

Deshalb ist im Zusammenhang mit Ajax Javascript eigentlich immer die erste Wahl für eine weitergehende Bearbeitung solcher Browser-Instanzen, da es sich „innerhalb" der betrachteten Instanzen bewegt, während die „entfernten" Ruby-Programme prinzipiell das Problem haben, die jeweilige Instanz (oder zumindest deren Werte) korrekt bestimmen zu müssen. Dass dies nicht immer ganz leicht ist, zeigte der Fall (aus früheren Releases), als sich die Anzeige nach dem Droppen als nicht sehr dauerhaft beim Blättern erwiesen hatte.

Es drohen freilich Missverständnisse und Dateninkonsistenz, wenn die Anwender etwas anderes als das Programm sehen. Dabei ist es unbeachtlich, dass das Blättern im Browser von Javascript (bisher) noch nicht abfragbar ist (es gibt aktuell noch kein Event „OnBack" oder „OnForward") – das Programm hat trotzdem irgendwie Klarheit zu schaffen.

Dass Javascript die Kontrolle nicht verliert und die diversen Verarbeitungsschritte genau auseinander halten kann, ist dafür unerlässlich. Werden die maßgeblichen Werte im HTML-Code des aktuellen Blattes gespeichert und später per Javascript abgefragt, besteht wenigstens die Sicherheit, die Daten genauso erfahren zu können, wie sie auch der Browser den Anwendern anzeigt. Hierbei ist freilich zu beachten, dass der Zugriff von Rails, mit dem HTML-Werte manipuliert werden, ausschließlich über die „render"-Funktionalität erfolgt. Wird also ein Update des Ajax-Vorgangs auf ein anderes, größeres DOM-Element ausgeführt als „gerendert" wird, so kann per Rails trotzdem nur auf das letztere Element zugegriffen werden.

Cachen im Zusammenhang mit dem Browser ist mit Vorsicht zu genießen

Merke: „Cachen", hier im Sinne von programmseitigem Vermerken von Feldinhalten, ist also auch in diesem Zusammenhang eine Sache, die in jedem einzelnen Fall auf Korrektheit überprüft werden muss – sicher der Grund, warum Rails seine mit dem Browser kommunizierenden Klassen ständig neu initialisiert.

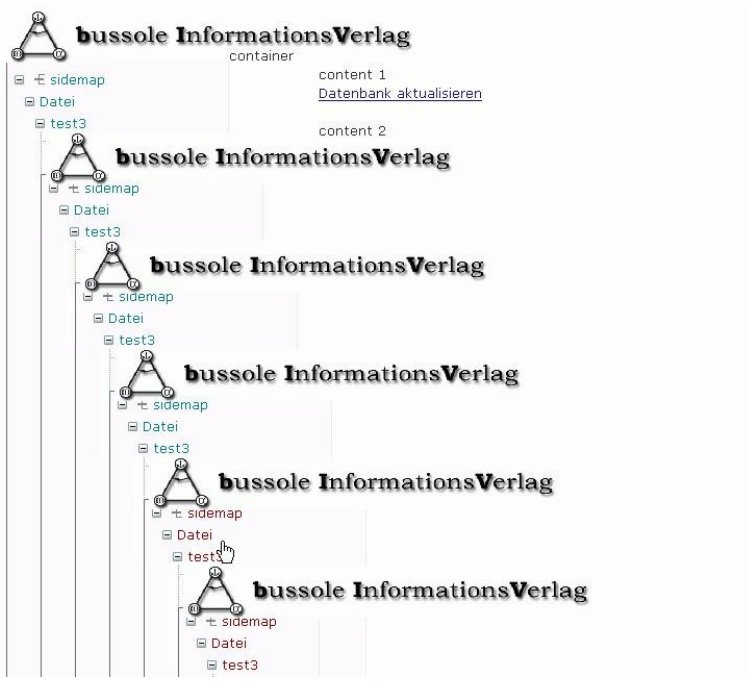
Anwendungsbeispiel: Der dritte Schritt und der vierte Schritt von FirstRails

4.1.11.3 Wie es nicht gemacht werden soll

Da Ajax eine elementgenaue Bearbeitung ermöglicht, verlangt es auch eine elementgenaue Präzision in der Verarbeitung. Partials sind hier ein sehr gutes Mittel, diese Präzision zu erreichen, doch trotz ihrer Einfachheit erlauben auch sie Fehler.

So beispielsweise sieht es aus, wenn eine Treelist in einem Partial untergebracht ist, im Controller bei der aufgerufenen Verarbeitung jedoch das gesamte Layout aufgerufen wird mit:

```
render :action => 'index'
```



Im obigen Fall wurden die einzelnen Texte der Baumstruktur (hier „test3“) mit „link_to_remote“ versehen und die zu aktualisierende DOM-ID dieses Textes zur Aktualisierung angegeben – mit dem Parameter „:update“. Bei jedem Klick auf den betreffenden Text wurde also die im Link angegebene Aktion aufgerufen, in der sich aber der folgende Render-Befehl befand:

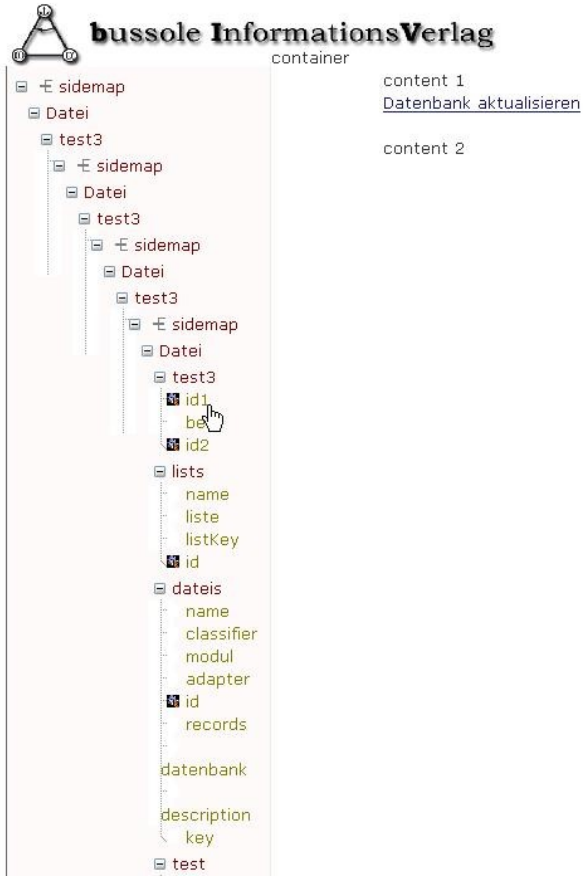
```
render :action => 'index'
```

Das bedeutet, dass Ajax zwar nur den einzelnen Text „optisch“ austauschen sollte – das aber immer mit dem gesamten Layout.

Und so sieht derselbe Fall aus, wenn in der ausführenden Aktion statt des gesamten Layouts nur die komplette Treelist (wiederum an der Stelle des einzelnen Text-Elementes „test3“)

```
render :partial  => 'treelist', :locals => { :name => "sidemap" }
```

angezeigt werden soll:



Im Prinzip liegt hier natürlich immer noch der gleiche Fehler vor: Von Ajax wird lediglich die Aktualisierung eines bestimmten HTML-Elements verlangt,

doch das über „render“ ausgegebene Partial liefert viel zuviel an – im obigen Fall beispielsweise den gesamten Baum.

Die einfachste Lösung ist hier, nicht ein einzelnes Text-Element anzusprechen, sondern die gesamte Treelist im „:update“-Parameter anzugeben. Stimmt dann noch die Definition im CSS-File, reagiert die Treelist – wenigstens html-technisch – wie gewünscht.

Manchmal muss freilich ein wenig nachgeholfen werden, um die HTML-Seite dazu zu überreden, das Gewünschte anzuzeigen.

4.1.11.4 Ruby-Fehlermeldungen: Stichwort „Scheinbare Reaktionslosigkeit“

Es kann vorkommen, dass während der asynchronen Verarbeitung Rubys Fehlermeldungen nicht mehr am Bildschirm sichtbar werden, sondern nur noch in den Protokollen im Rails-Anwendungsordner „log“ aufzufinden sind. Diese „scheinbare Reaktionslosigkeit“ rührt schlicht daher, dass Ajax gerade nicht mehr den gesamten Bildschirm aktualisiert, sondern nur noch das angeforderte DOM-Element. Bietet dieses jedoch nicht genug „Anzeigefläche“ oder ist es sogar zur erforderlichen Ausgabe überhaupt nicht fähig, verschwindet die anfallende Fehlermeldung sang- und klanglos im Nirvana der Protokolldatei. Im Verlauf dieser Verarbeitung eine Ausnahmebearbeitung („exceptions“, „raise“) direkt ausgeben zu wollen, macht deshalb wenig Sinn, da das Ergebnis nur in einer Log-Datei begraben wird.

4.1.11.5 Aktuelle Prüfungen von Benutzereingaben: observe_field, observe_form

Mit Hilfe von Ajax wird es möglich, Feldinhalte nicht erst beim Abspeichern zu prüfen, sondern direkt nach der Eingabe. Dies erfolgt über die beiden Befehle

```
observe_form
observe_field
```

über die Rails per Javascript die modifzierten Feldinhalte abfragt und an eine angegebene ausführende Einheit versendet, die entweder über eine URL

bestimmt wird oder eine Funktion sein muss. Genaueres findet sich in der ausführlichen Dokumentation der Befehle in „prototype.rb“ (ActionView-Komplex im Ordner „helpers“).

Was wohl, zumindest aktuell, nicht funktioniert, ist der Parameter „:on“ – laut Handbuch soll er den Auslöser des „observe“- Befehls von „change“ auf einen beliebigen Event umleiten können. Während also üblicherweise der „observe“-Befehl aktiviert wird, wenn das betrachtete Feld eine Änderung erfährt, sollte dieser Parameter ihn dazu befähigen, auch auf andere Ereignisse zu reagieren, beispielsweise wenn das Feld angeklickt würde. Im Moment scheint jedoch die Javascript-Prototype-Library „prototype.js“ gar nichts mit diesem Parameter anfangen zu können.

Zu beachten ist wiederum für die Programmierung, dass Fehlermeldungen im Ajax-Zyklus nicht wie üblich am Bildschirm sichtbar sind, wenn das betreffende Feld nur ein Eingabefeld ist, sondern nur in der Protokolldatei im Ordner „log“.

Und ebenfalls beachtenswert ist die „zweigleisige“ Parametrisierung. So lässt sich die übliche Rails-Methodik „:params =>“ neben der Javascript-Übergabe „:with“ verwenden. „.params“ übergibt wie immer die betrachteten Felder als Schlüssel mit ihren aktuellen Inhalten „value“. Statt über den Namen des HTML-Objektes kann mit

```
:with => "'input='+$('xyz').value"
:with => "'input='+$F('xyz')"
```

auch über die ID zugegriffen werden. Im obigen Fall enthält die empfangende Methode zusätzlich den Schlüssel „:input“ in ihrem Parameterfeld „:params“ – mit dem Wert desjenigen Feldes, das die DOM-Id „xyz“ vorweist.

Selbstverständlich ist die Wahl der zu aktualisierenden DOM-Id abhängig von der speziellen Aufgabenstellung, die von den „observe_“-Befehlen angestoßen wird. So macht es vermutlich wenig Sinn, das betrachtete Feld selbst zu aktualisieren, wenn es ein einfaches Textfeld ist. Warum? Weil dies bedeutet, dass von irgendwoher ein „besserer Text“ ermittelt wurde, der in dieses Feld

eingestellen werden könnte – das aber wird vermutlich besser über „...auto_complete...“ gelöst.

Deshalb sollte der Parameter „:update“ in „observe_...“ nicht einfach unterlassen werden, denn er wird mit dem auslösenden Feld vorbelegt. Wenn also, wie im Beispielfall, eine Fehlermeldung ausgegeben werden sollte, dann muss auch dasjenige HTML-Element, das diese Fehlermeldung anzeigen kann, über diesen „:update“-Parameter kenntlich gemacht werden – denn, wie gesagt, es macht wohl wenig Sinn, die Fehlermeldung im zu bearbeitenden Textfeld anzuschauen.

Anwendungsbeispiel: Der zweite Schritt von FirstRails

Was mit ein wenig mehr Aufwand verbunden ist, sind eventuelle Konsequenzen einer fehlerhaften Bearbeitung. Wird von den Anwendern eine problematische Auswahl getroffen, und soll diese nicht nur gemeldet, sondern auch rückgängig gemacht werden – und das in allen Browsereinstellungen, sogar nach etwaigem Vor- und Zurückblättern ohne „Serverkontakt“ – so braucht es manchmal noch ein wenig Javascript, um das gewünschte Ergebnis zu erhalten.

Anwendungsbeispiel: Der dritte Schritt von FirstRails

4.1.11.6 Texteingabe: auto_complete und Rails 2.0

Vorbemerkung: „auto_complete“ ist seit Rails 2.0 nicht mehr nativer Bestandteil von Rails, wird aber weiterhin als Plugin zur Verfügung gestellt.

Eine vereinfachte Textvervollständigung in „nativem Rails“ – unter Verwendung der Autocomplete-Funktion aus dem „script.aculo.us“-Modul „controls.js“ analog der (ehemaligen) Rails-Funktionalität „auto_complete.rb“ – findet sich in Kapitel 4 („Elemente und Contents: Vom Umgang mit Inhalten“), Sektion „Datei „links“ – Textvervollständigung: autocomplete“ des Open-Source-Tutorials „OpenRorBook: Einfache Tutorial-Verwaltung mit Rails“, gespendet vom bussole InformationsVerlag.

Das Plugin „auto_complete“ unterscheidet sich sich jedoch nicht von der vorherigen Funktionalität. Die Funktion „text_field_with_auto_complete“ – nun im Ordner „lib“ in der Source „auto_complete_macros_helper.rb“ des Plugins untergebracht – erzeugt weiterhin die notwendigen HTML- und Javascript-Befehle mit Hilfe der „Autocomplete“-Funktion aus dem „script.aculo.us“-Modul „controls.js“, um den Anwendern eine Liste von Auswahlmöglichkeiten anzubieten – falls sie passende Eingaben gemacht haben:

```
<%= text_field_with_auto_complete 'obj', 'meth' %>
```

Die dafür erforderliche Funktion „auto_complete_for_...“ kann jedoch im Falle, dass keine besonderen Wünsche an die Ausgestaltung der Liste vorliegen, schlicht über die Anweisung

```
auto_complete_for :obj, :meth
```

im Controller aktiviert werden. Die Source „auto_complete.rb“ des Plugins zeigt dann, dass diese Methode dem Controller über „define_method“ automatisch eingefügt wird und dass sie per SQL diejenigen Datensätze zusammensucht, die die von den Anwendern eingegebenen Buchstaben im besagten Datenfeld enthalten. Zuletzt wird die so aufgefundene Liste noch mit „auto_complete_result“ aus „auto_complete_macros_helper.rb“ aufbereitet und zur Auswahl angeboten.

Dafür braucht es im einfachsten Fall tatsächlich nur die beiden Anweisungen „text_field_with_auto_complete“ im Template und „auto_complete_for“ im Controller – sowie die Initialisierung des Plugins. Diese läuft dabei ähnlich unkompliziert wie immer ab: Rails stellt dafür einen Ordner „vendor/plugins“ zur Verfügung, in den der Plugin-Ordner „auto_comlete“ einfach kopiert wird.

Neben der korrekten Bezeichnung der Instanzvariablen ist für ein problemloses Funktionieren dieses Plugins unbedingte Voraussetzung, dass die benötigten Javascript-Befehle aus „controls.js“ und „effects.js“ im HTML-File eingebunden werden, etwa über

```
<%= javascript_include_tag 'prototype', 'controls',
        'effects' %>
```

Anwendungsbeispiel: Der zweite Schritt von FirstRails

Wird ein bisschen mehr Flexibilität gebraucht, können die einzelnen Funktionen selbstverständlich auch gezielt eingesetzt werden. Anstatt „text_field_with_auto_complete“ kann bei Bedarf ein normales Text-Inputfeld verwendet werden, dem dann nur

```
<%= auto_complete_field 'xyz',
        :update => 'abc',
         :url => { :action => 'auto_complete_for_xyz',
                   :controller => 'application' }%>
```

hinzugefügt werden muss, um Rails zu überreden, den „Ajax.Autocompleter“ dafür einzurichten. „xyz“ ist dabei die DOM-Id des Textfeldes, dessen Eingaben unterstützt werden sollen und „abc“ diejenige des HTML-Elements, das die Ausgabe übernimmt.

Die HTML-Formatangaben aus der privaten Methode „def auto_complete_stylesheet“ der Plugin-Source „auto_complete_macros_helper.rb“ können ebenfalls dem eigenen Layout angepasst werden. Wenn beispielsweise die auszugebende Liste nicht durchscheinend sein soll, sondern darunter liegende Links oder Listenelemente zu überdecken hat, erwies sich der Überlappungsfaktor

```
z-index: 100;
```

als Definitionselement der HTML-Klasse „auto_complete“ als nützlich – jedoch nur zusammen mit einem nicht transparenten Hintergrund. Zu berücksichtigen ist hier, dass bei Verwendung von „auto_complete_field“ die im Plugin definierten Layoutbestimmungen in das HTML-Endergebnis einfließen. Das eigene Layout sollte also wenn möglich mit diesen Vorgaben abgestimmt werden.

Und zu guter Letzt wird noch das HTML-Element benötigt, das das Ergebnis der gesamten Mühe anzuzeigen hat:

```
<div class="auto_complete" id="abc"></div>
```

Hier ist wichtig, dass diese Definition unbedingt vor den auszuführenden „auto_complete“-Befehl verlagert wird, denn Javascript geht objektmäßig streng der Reihe nach vor und meldet ansonsten „element has no properties“ (schön zu sehen in der Javascript-Konsole, die im Firefox unter Extras zu finden ist). Diese Javascript-Fehlermeldung gibt als problematische Stelle in „prototype.js“ genau den Befehl an, der das gesamte anzuzeigende Listen-Objekt für den Normalfall auf „unsichtbar“ setzt, unabhängig von einer eventuell anderslautenden CSS- oder HTML-Definition. Ist es dann noch nicht definiert, läuft „prototype.js“ eben auf den besagten Fehler.

Auch die ausführende Methode, die im obigen Beispiel gemäß den „Autocomplete“-Anforderungen mit dem Namen „auto_complete_for_xyz“ versehen ist, kann nach eigenen Anforderungen erstellt werden. Sie hat dann nur selbst dafür zu sorgen, dass eine HTML-Liste (mit den nötigen <ul> und -Tags) zurückgegeben wird und dass das Ganze korrekt ausgegeben wird, beispielsweise über „render :inline“ oder „render :text“.

Anwendungsbeispiel des ursprünglichen „auto_complete“: Der dritte Schritt von FirstRails

4.1.11.7 Drag & Drop

Diese bequeme Funktionalität – einfach ein Feld mit der Maus anzuklicken und es auf ein anderes „fallen“ zu lassen, um dort irgendetwas mit dem „mitgenommenen“ Feld zu erwirken – ist ebenfalls unglaublich unkompliziert. Diesmal befindet sich der jeweilige Javascript-Code in „dragdrop.js“ im Ordner „helper/javascripts“ von ActionView, ist demnach mit

```
<%= javascript_include_tag 'dragdrop' %>
```

oder

```
<%= javascript_include_tag :defaults %>
```

zu aktivieren.

Das zu „holende“ Feld muss dabei nur mit seiner DOM-Id als „draggable“ gekennzeichnet werden.

```
<%= draggable_element("xyz", :revert => true,
        :ghosting => true)%>
```

Der Parameter „ghosting“ belässt während des „Drag“-Vorgangs alles beim Alten und gibt den Anwendern bloß eine Kopie des Feldes zum Verschieben, während der Parameter „reverse“ dafür sorgt, dass das Feld nach Beendigung der Aktion dort verbleibt, wo es hergekommen ist.

Das empfangende Feld muss dann über

```
<%= drop_receiving_element "abc",
        :url => { :action => "test" } %>
```

ermächtigt werden, die Felder vom Typ „draggable“ auch annehmen zu können. Da im obigen Fall „drop_receiving_element“ den Parameter „:update“ nicht angegeben hat, ist das Feld der DOM-Id „abc“ auch das, welches über die Controllerfunktion „test“ via Ajax wieder aktualisiert wird. Wie immer kann das zu verändernde DOM-Element jedoch über exakt diesen Parameter „:update“ beliebig verändert werden.

Hinsichtlich der interessanten Option „:ondrop“ ist freilich anzumerken, dass das Signal nicht immer so ausgelöst wird, wie es auf den ersten Blick erwartet werden könnte. So zeigte der hilfreiche „alert“-Befehl für ein Treelist-Element:

Javascript lässt sich meist recht gut über „alert“ verfolgen

```
<%= drop_receiving_element "#{line[1]}-drop",
            :update => "drop_id", :hoverclass => 'signal',
            :ondrop => "alert('xxx')",
            :url => { :action => "dropped",
                :controller => "application",
                :params => {:dropped => line[7]} },
            :with =>"'idnr='+
                new Array($F('edit_key'),$F('typ_id'))"
             %>
```

dass das „:ondrop“-Signal bereits während des Bildaufbaus des Baumes gesendet wird, nicht erst beim Drag & Drop-Vorgang selbst.

Und wie immer bei Verwendung von Ajax ist zu beachten, dass im Fehlerfall „scheinbare Reaktionslosigkeit“ vorkommen kann, die nur über das Protokoll im Rails-Anwendungsordner „log“ aufklärbar ist.

Zwei kritische Gesellen: Scrollbalken und Drag & Drop

Weiterhin interessant ist auch die Reichweite des Drag & Drop – und ihre Abhängigkeit von den jeweiligen HTML-Formatdefinitionen. So verhindert zumindest zur Zeit der Parameter „overflow: scroll“ bzw. „overflow: auto“, dass das jeweilige Feld die so bestimmten Grenzen überschreitet. Es werden zwar ein paar Workarounds (in purem Javascript und für Rails) auf dem Internet angeboten, doch im Standard war dies bis Version 1.6.2 von scriptaculous noch nicht angekommen.

Ein weiteres Problem hinsichtlich Scrollbalken ist in der FirstRails-Applikation unter den gegebenen Releaseständen festzustellen, wenn der Scrollbalken benutzt wird: Dann nämlich aktiviert das Droppen nicht mehr die DOM-Elemente direkt unter dem Cursor, sondern welche weiter oben.

Und last but not least ist bei Listen zu bedenken, dass die -Tags die gesamten enthaltenen Listenzeilen mit aktivieren, sodass in diesem Fall für

eine eindeutige Identifizierung gesorgt werden muss, wenn Drag & Drop erfolgreich für ein einzelnes Listenelement funktioneren soll.

Anwendungsbeispiel: Der dritte Schritt von FirstRails

4.1.12 Sicherheit

Bei Rails wird Sicherheit in einer unsicheren HTML-Umgebung groß geschrieben, wie das Dokument „Common security problems“ oder auch die Erweiterungen von Rails 2.0 (wie das neue Modul „RequestForgeryProtection“ des ActionController) demonstriert – mit dem Fazit, dass die (bekannten) Tücken von HTML eigentlich nur dann berücksichtigt werden müssen, wenn die Kommunikation entweder nicht vollständig über Rails-Befehle (wie etwa „save“ oder „attributes“) erfolgt oder nicht der Rails-Norm entspricht. Als Beispiel wird erwähnt:

```
gefährlich:
        Email.find_all "owner_id = 123 AND
        subject = '#{@params['subject']}'"
ok:
        subject = @params['subject']
        Email.find_all [ "owner_id = 123 AND
        subject = ?", subject ]
```

Auch ist bei HTML-Befehlen, die selber erzeugt werden, darauf zu achten, dass die HTML-Metadaten nicht im Klartext, sondern immer als Entität benutzt werden: „<“ and „>“ statt „<“ und „>“ – damit wird gewährleistet, dass der Text keine „aktiven“ Steuerzeichen enthält und der Browser nur einen harmlosen String zu fassen kriegt. Für die Gestaltung der Layouts sind in Rails schließlich die Views vorgesehen.

Um sicherzustellen, dass Textvariable „sauber“ sind, also keine Steuerzeichen enthalten, bietet Ruby eine Helfermethode „h“ an (alias der Methode „html_escape“ in Rubys Datei „erb.rb“). Sie bereitet über HTML auszugebende Variable zu ungefährlichem Text auf, wenn deren Ursprung nicht hunderprozentig kontrolliert werden kann wie beispielsweise bei Dateifeldern, die von Anwendern befüllt werden. So wird ein Text „<b>“ vom Browser

normalerweise verstanden als die Aufforderung, den folgenden Text fett auszudrucken, die Anwendung von „h“ jedoch macht aus dem Steuerzeichen eine harmlose Zeichenfolge.

```
<% test1 = '&lt;b&gt;Test1&lt;/b&gt;'%>
<%= test1%>     wird      <b>Test1</b>
<%=h test1%>    bleibt    &lt;b&gt;Test1&lt;/b&gt;

<% test2 = '<b>Test2</b>'%>
<%= test2%>     wird      Test2
<%=h test2%>    bleibt    <b>Test2</b>
```

Dürfen Anwender ihre Texte selbst layouten, sollte diese Absicherungstechnik trotzdem nicht entfallen, sondern Markup-Languages wie „Textile“, „Markdown“ oder „Rdoc“ benutzt werden, schlägt das Dokument „Common security problems“ vor.

Eine einfache Textbearbeitung unter Verwendung der Prüfroutine „ae_some_html“ von Andrian Budantsov, die aus Strings alle HTML-Tags (bis auf die zugelassenen) entfernt, findet sich in Kapitel 4 („Elemente und Contents: Vom Umgang mit Inhalten“), Sektion „Abspeicherung von Inhalten – sicheres HTML und vervollständigte Bildübernahme" des Open-Source-Tutorials „OpenRorBook: Einfache Tutorial-Verwaltung mit Rails“, gespendet vom bussole InformationsVerlag.

4.2 Konventionen und Versionen

4.2.1 Namensgebung

Singular > Modell

Plural > Datei

Wie aus dem Ergebnis des CoC-konformen Scaffold-Generators zu ersehen ist, wird mit dem Singular eines gewählten Objektnamens das Modell und mit dem Plural die Datei bezeichnet, was Sinn macht, da jeder Datensatz als Objektinstanz angesehen wird. Deshalb macht es genauso viel Sinn, dass der Controller, der die Datei kontrolliert, auch mit dem Plural versehen wird.

Ebenfalls konsequent ist Rails hinsichtlich der Instanzvariablen, die das Datenobjekt – oder die Datenobjekte – aufzunehmen hat: Sind es mehrere,

wird der Plural verwendet wie beispielsweise „@lists“, ist es nur einer, wird auch nur der Singular benutzt („@list“). Das ist überall dort bedeutsam, wo auf die Rails-Mechanismen (wie ehemals „autocomplete“) zurückgegriffen werden soll.

Weitere folgenschwere Namen werden für die Vererbung oder Zeitstempel und Lockschalter verwendet.

4.2.2 Vererbung

Vererbung persistenter Klassen – also solcher, die über ActiveRecord gehandhabt werden – kann Rails automatisch auch auf Datensatz-Ebene vermerken, sodass in den gespeicherten Datensätze die „Erbfolge“ bekannt bleibt. Dafür wird ein Dateifeld namens „type“ benötigt, das über das Attribut „Base.inheritance_column“ auch umbenannt werden kann. Liegt ein solches Feld jedoch nicht vor, so ist die Vererbung nur auf Programmebene möglich „with no special magic“, wie der Kommentar in „base.rb“ es bezeichnet.

4.2.3 Zeitstempel und Lock-Schalter

Werden in eine Datei Felder des Namens „created_at/created_on“ oder „updated_at/updated_on“ eingefügt, so versorgt Rails sie bei Neuanlage oder Aktualisierung mit den aktuellen Datumsangaben.

Soll in einer Mehrbenutzer-Umgebung „optimistic locking“ über einen Lock-Schalter in der Datenbank verwendet werden, so ist nur ein Feld des Namens „lock_version“ mit dem Typ „integer“ und dem Defaultwert „0“ in der betreffenden Datei anzulegen und darauf zu achten, dass dieses Feld als „hidden“ im Aktualisierungsfenster auftaucht.

4.2.4 Einige reservierte Namen und Sonderzeichen-Sonderfälle

Da sie Schlüsselworte der zugrunde liegenden Systeme wie MySQL, Rails oder Ruby darstellen, sollten Bezeichnungen wie:

```
file, keys, table, schema(s), module, database, Data, fields
```

für eigene Namen besser nicht gewählt werden (im Zusammenhang mit dem jeweiligen System). Für den schnellen Test „korrekter“ MySQL-Dateibezeichnungen erwies sich dabei phpMyAdmin wiederum als sehr hilfreich, da dort über SQL-Befehle wie

```
SHOW FIELDS FROM xyz
SELECT * FROM xyz WHERE abc = 'xxx'
```

sofort festgestellt werden kann, ob MySQL mit der gewünschten Namensgebung zufrieden ist.

Die entsprechende Überprüfung auf reservierte Namen durch Rails ist bei den Generatoren im Basismodul untergebracht (hier Version 2.0.2):

```
...\rails-2.0.2\lib\rails_generator\commands.rb
```

(beispielsweise C:\Ruby\lib\ruby\gems\1.8\gems\rails-2.0.2\lib\rails_generator\commands.rb)

Die darin enthaltene Methode „class_collisions“ untersucht dann, ob der gewünschte Name für das neue, vom Generator zu erstellende Teil mit bestehenden Railsklassen in Konflikt gerät.

Auf Ebene von Klassennamen existiert eine solche Vorab-Überprüfung normalerweise nicht – dort kommt es zu häufig „auf den Einzelfall“ an. So führte die ungeschickte Wahl eines Klassennamens „Data“ zu der interessanten Meldung:

```
allocator undefined for Data
```

Neben Namen können auch Sonderzeichen Probleme bereiten wie Bindestriche in Namen, die MySQL nicht gut leiden mag. Es lässt sich aber mit dem linksgerichteten Anführungszeichen („back ticks“) bekehren:

```
SHOW TABLES IN `a-bc`
```

4.2.5 Versionen und Lizenzen der verwendeten Komponenten

Die Erstanwendung „FirstRails“ wird auf der Webseite des Verlags zum Download angeboten – schrittweise, wie im Buch „Das japanische Juwel“ beschrieben.

Auf den folgenden Bausteinen hat sich die Applikation dabei als lauffähig erwiesen:

Windows® 2000, XP

Ruby 1.8.2, 1.8.4

Rails 1.1.2 – 1.1.6

Von den Rails-Modulen wurden nur „actionpack“ (1.11.2 – 1.12.5), „activerecord“ (1.13.2-1.14.4) sowie „activesupport“ (1.2.5-1.3.1) verwendet.

Es findet sich jedoch auf der Webseite des bussole InformationsVerlags auch eine auf Rails 1.2.3 angepasste Version von FirstRails zum Download bereit, sowie eine kurze Beschreibung der doch recht einschneidenden Veränderungen, die für Rails 2.0 erforderlich wären.

Weil die Entwicklung weiter voranschreitet und nicht erwartet werden kann, dass die passende Umgebung für die beschriebene Anwendung „FirstRails“ überall vorliegt, wird auch eine Ruby 1.8.2/Rails 1.1.6-Umgebung zum Download auf der Webseite hinterlegt. Um diese Umgebung zu aktivieren, muss (zumindest unter Windows®) keine vorhandene Installation entfernt werden – Umbenennung genügt. Die FirstRails-Ruby-Umgebung ist lediglich während der Testläufe mit dem Namen der aktiven Ruby-Umgebung zu versehen, sodass der Aufruf des WEBrick-Servers durch das Skript "start.bat" im FirstRails-Ordner auf diese lauffähige Umgebung treffen kann.

Mehr ist nicht erforderlich. Ist der Test abgeschlossen, kann die Echtumgebung durch Umbenennung wieder reaktiviert werden.

Für die Datenbank beschränkt sich die Anwendung auf:

MySQL 5

Unter Beta-Versionen von MySQL konnte es zu unangenehmen Ausfällen kommen, ältere MySQL-Versionen sind nach Umdefinition der langen Textfelder auf „Blob“ unter Umständen bereit mitzuspielen.

Als Browser kommen in Frage:

Firefox 2, IE 7

Ältere Firefox-Versionen können zu Abstürzen überredet werden, ältere Internet Explorer sind möglicherweise nicht einmal in der Lage, die erste Seite anzuzeigen.

Hinsichtlich des IE ist auch zu beachten, dass das Layout auf den Firefox optimiert wurde. Das heißt, dass er gelegentlich Icons nicht korrekt anzeigt oder das Layout nur suboptimal wirkt.

An HTTP-Servern wurden überprüft:

Ruby/WEBrick, Apache mit FCGI

Bei Verwendung des Apache-Servers werden die Umlaute in den Texten nicht korrekt wiedergegeben. Dies wäre zu vermeiden gewesen, wenn die MySQL-Datenbank gleich mit der Eigenschaft

Kollation „utf8_general_ci“

versehen und Rails dies über die Konfigurationsdatei „database.yml“ mit

encoding: utf8

kenntlich gemacht worden wäre – beides bedeutet, dass Texte im global einheitlichen Unicode-Format behandelt werden.

Mit einfacher CGI-Unterstützung sind darüber hinaus die Sourcen nur höchst eingeschränkt verwendbar, da die in-memory-Behandlung sich von WEBrick unterscheidet. Ob CGI verwendet wird, zeigt sich deshalb bereits an der Meldung, dass Klassenvariable nicht gefunden werden konnten.

Doch selbst mit FCGI kann es zu merkwürdigem Verhalten kommen wie beispielsweise zu der Meldung, dass Rails nicht sauber gestartet wurde, was aber mit einem zweiten Versuch "korrigiert" werden kann. Auch ist die prozessgebundene in-memory-Speicherung von FCGI gelegentlich störend, was besonders in der Baumstruktur zu unerwünschten Effekten führt. So können unterschiedliche Ruby-Prozesse etwa dann auftauchen, wenn ein WEBrick-Server aktiv ist oder bei der Aktivierung von Apache aktiv war.

InstantRails 1.7 (basierend auf Ruby 1.8.6 und Rails 1.2.3) ist ebenfalls nur eingeschränkt anwendbar. Bei der Installation ist zu beachten, dass die Datenbank-Anbindung unterlassungsmäßig ohne Passwort erfolgt, die kleine Text-Datei "currdb.txt" im Ordner "public" gelöscht oder initialisiert wird und dass der Aufruf über "application" zu einem Routing Error führt, da Rails 1.2.3 sehr viel strenger hinsichtlich der Tatsache ist, dass „application“ kein „realer“ Controller ist, sondern nur die Basis für die „echten“ Controller darstellt. Hier kann es genügen, den "dateis"-Controller anzusprechen, doch Routing-Probleme sind auch im besten Fall nicht auszuschließen. Die auf Rails 1.2.3 angepasste Version von FirstRails weist diesen Fehler dann nicht mehr auf.

Solange keine in-memory-Probleme auftauchen (wieder erkennbar an der Meldung nicht vorhandener Klassenvariablen), ist es jedoch möglich, die Grundzüge der Original-Applikation zu besichtigen.

Während der Entwicklung von FirstRails hatte sich als Arbeitsumgebung mit tatsächlich nutzbarem, wenn auch gelegentlich sehr langsamen interaktiven Debugger nur folgende Konstellation erwiesen:

```
Ruby 1.8.2
RadRails 0.6.1
```

Bei den Eclipse-RadRails-Plugins erwies sich Version 0.7.2 (zusammen mit Ruby 1.8.2) als tauglich.

Glücklicherweise ist seitdem ein schneller, integrierter Debugger aufgetaucht, der mit den neuen RadRails-Versionen von aptana schmerzfrei zusammenarbeitet.

Was nicht geklärt werden konnte, aber sich mehr als einmal als Störfaktor erwies, war der Einfluss der jeweils benutzten Firewall.

Alle Komponenten – außer selbstverständlich das Microsoft®-Betriebssystem – sind unter Open-Source-Lizenzen verfügbar. So nutzt Rails die freigiebige MIT-Lizenz, während Ruby, MySQL Apache und Firefox eigene Lizenzen vorweisen: Ruby License, MySQLs Floss-Extension, Apache License und Mozilla Public License. RadRails dagegen wurde unter der Eclipse Public License angeboten und steht seit der Übernahme durch aptana unter APL/GPL.

Die aktuellen Bedingungen der Lizenzierung oder auch der Nutzung von Markenzeichen und sonstigen Copyright-Rechten mögen wegen der raschen Entwicklung im Open-Source-Umfeld bitte auf den jeweiligen Websites nachgeprüft werden, da Open Source nicht viel verlangt für die Großzügigkeit, fremde Arbeit nutzen zu dürfen – das aber ist dann schon auf Punkt und Komma zu erfüllen. So gebietet es nicht nur der Anstand, sondern auch der Verstand: Nur wenn Abmachungen eingehalten werden, kann ein derart freies und freizügiges Teamwork-System auf Dauer überleben.

4.3 Voraussetzung: Ruby

4.3.1 Befehlsübersicht

Die Ruby-Befehlsübersicht findet sich bei den Ruby-Dokumenten unter dem Punkt „Ruby Documentation Bundle“ im Ruby Language Reference Manual (aktuell immer noch für Version 1.4.6): „Built-in functions“.

Bequemer ist sie jedoch über das interaktive Ruby-Dokumentations-Tool „ri“ zu erreichen.

Was mit den Befehlen im Einzelnen möglich ist, schildern auch die FAQs an einfachen Beispielen.

4.3.2 Die Standard-Ruby-GUI: TK

Zu den Ruby-Geschenken („Bundled Libraries“) gehört auch ein Befehlssatz für die graphische Oberfläche, die neben der Ablaufsteuerung oder Buttons so

interessante Dinge wie die Dialogbox des jeweiligen Betriebssystems zum Öffnen oder Speichern von Dateien anbietet. Leider existiert bisher in der Ruby-Dokumentation zu den Bundled Libraries noch nichts über TK, doch ein Text auf dem Internet schafft hier Abhilfe.

Zu finden ist die Schnittstelle „tk.rb“ unter

```
...\Ruby\lib\ruby\1.8
```

sowie die TK-Sourcen unter:

```
...\Ruby\lib\ruby\1.8\tk
```

Natürlich macht die Browsertechnologie von Rails eine eigenständige Fensterbearbeitung überflüssig – aber zu wissen, dass es TK gibt, kann nicht schaden.

4.3.3 Hilfreiche Dateibefehle

In Ruby (hier Version 1.8) gibt es eine Menge „vorgefertigter“ Dateimanipulationen in:

```
...\Ruby\lib\ruby\1.8
```

So lassen sich die Dateien des Rails-Ordners „public“ mit

```
dirs = Dir.entries("#{RAILS_ROOT}/public/")
```

erfassen, während der Inhalt eines File 'xxx.txt' zeilenweise in eine Textvariable „abc“ mit folgenden Instruktionen übernommen wird:

```
abc = ''
file_path = "#{RAILS_ROOT}/public/xxx.txt"
File.foreach(file_path) {|line| abc << line}    if File.exist?
(file_path)
```

Ein wenig irritierend ist die Verwendung von „new“ bei der Klasse „File“ – im einfachsten Fall öffnet dieser Befehl nämlich nur eine bestehende Datei und ist

somit ein Synonym für „File.open“. Soll die gewünschte Quelle bei Nicht-Vorhandensein erstellt werden, wird die Angabe „File::CREAT“ erforderlich:

```
f = File.new("newfile", File::CREAT|File::RDONLY)
```

Der Modus „RDONLY“ steht dabei für „Nur lesen“. Ist die betreffende Datei nicht nur bei Bedarf neu zu erstellen, sondern immer auch zu initialisieren, erweist sich die Option „TRUNC“ als nützlich, denn sie reduziert das vorhandene File auf die Länge „0“:

```
f = File.new("newfile", File::CREAT|File::TRUNC|File::RDWR,
0644)
```

Der Zusatz „0644“, der in der Ruby-Dokumentation zu „File.new“ aufgeführt ist, soll dabei – als so genannte „Optionale Berechtigung (Optional permission bits)“ – das Lesen und Schreiben („RDWR“) auf die Datei-Besitzer („owner“) einschränken.

Anwendungsbeispiel: Der letzte Schritt von FirstRails

Um Verzeichnisse anzulegen oder Dateien zu kopieren, ist weiterhin „fileutils.rb“ sehr nützlich und auch „ftools..rb“ kann helfen, etwa wenn Verzeichnisse zusammen mit allen übergeordneten Ordnern erstellt werden sollen.

4.3.4 Metaprogrammierung

Über Metaprogrammierung lässt sich wohl vieles sagen – es ist eine ganz eigene Weise, Software zu gestalten, doch auch diese „ganz eigene Weise“ ist am besten dadurch zu erlernen, dass sie einfach getan wird. Ein interessanter „Crashkurs“ bietet dazu etwa einen Einstieg über Rubys „Symbole“: die gerade für Anfänger ein wenig schwer zu verstehenden Referenzen der Form „:xyz“.

4.3.4.1 Indirektionen

Ruby führt „Texte“ aus, indem es sie in die von den Textinhalten angegebenen Objekte, Variable oder Methoden übersetzt – vorausgesetzt, dass diese

existieren. Besitzt eine Instanz „abc“ beispielsweise die Methode „xyz“, so wird diese üblicherweise über den Befehl

```
abc.xyz(*parm)
```

aufgerufen. Mithilfe einer Indirektion, die eine Textvariable „text“ benutzt, gelingt dies auch über den Befehl

```
eval(text).xyz(*parm)
```

im Falle, dass „text“ den Wert „abc“ vorweist, oder sogar

```
eval(text)
```

wenn „text“ den gesamten Befehl „abc.xyz(*parm)“ enthält.

Innerhalb von Strings (in normalen Anführungszeichen) können Indirektionen mithilfe geschweifter Klammern verwendet werden. So zeigt der Befehl

```
„blablabla #{text} blablabla“
```

nach der Ausgabe „blablabla HansOtto blablabla“ an, wenn die Variable „text“ den Wert „HansOtto“ beinhaltete.

Auch den Wert von Symbolen verschafft eval:

```
b = eval(:a.id2name)
```

Über „b“ kann danach auf den Wert des Symbols „:a“ zugegriffen werden.

Geschickt für ein solches „indirektes“ Vorgehen ist auch das Ruby-Modul „ObjectSpace“, das Zugriff auf alle aktuellen Objekte ermöglicht, die Ruby gerade im Griff hat. Beispielsweise erlaubt die Sequenz:

```
yyy = []
ObjectSpace.each_object(Xyz) do |db|
        yyy << db
        end
```

alle Instanzen der Klasse „Xyz“ abzuarbeiten und sie in einem Array „einzulagern“. Zu beachten hier: Der Klassenname darf nicht als String, also nicht in Hochkomma, geschrieben werden.

Auch diese Funktion ist interessant:

```
ObjectSpace._id2ref(object_id)
```

Sie wandelt eine Objekt-Identität in eine Referenz um, die sich dann über Parameter weiterreichen lässt. Wird an „_id2ref“ beispielsweise die Variable „__id__“ übergegeben, kann jedes Objekt eine Referenz auf sich selbst verteilen.

Anwendungsbeispiel: Der zweite Schritt von FirstRails

Zu guter Letzt sei noch ein interessanter Methodenaufruf erwähnt: Mithilfe von „send“ können Objekte beauftragt werden, eine ihrer öffentlichen Funktionen auszuführen, natürlich mit Parametern.

```
xyz.send('funktionsname', parm1, parm2, parm3)
xyz.send(:funktionsname, parm1, parm2, parm3)
```

Und natürlich geht auch

```
string = 'funktionsname'
xyz.send(string, parm1, parm2, parm3)
```

Anwendungsbeispiel: Der vierte Schritt von FirstRails

4.3.4.2 Mehr über eval

```
eval << -EOS
```

erlaubt es unter anderem, dass Variable zur aktuellen Instanz/Klasse hinzugefügt werden können – und zwar mit einem textabhängigen Namen.

Während nämlich

```
@#{xyz} = ...
```

im Normalfall zu einer Fehlermeldung führt, schlicht, weil alles, was hinter dem Zeichen „#“ auftaucht, als Kommentar angesehen wird, führt

```
eval <<-EOS
 @#{xyz}  = „xyz“
EOS
```

zu einer String-Instanzvariablen des aktuellen Objekts mit dem Inhalt „xyz“, die nun überall, wo dieses Objekt auftaucht, ebenfalls existiert, obwohl sie nie in der Klassendefinition aufgeführt war – und, Zugriffsmöglichkeit vorausgesetzt, mit

```
abc = eval(„@#{xyz}“)
```

abgefragt werden kann.

Die Abfrage auf Existenz fällt zwar ein wenig ungeschickter aus als bei den üblichen Variablen, die per „defined?“ überprüft werden können, ist jedoch auch nicht wirklich problematisch:

```
Begin
  abc = eval(„@#{xyz}“)
rescue
  eval <<-EOS
    @#{xyz} = ...
  EOS
  retry
end
```

Anwendungsbeispiel für diese Art „flexibler Variablengestaltung“ ist Rails selbst, das dem Ruby-Objekten „Class“ unter anderem die Funktion „cattr_reader“ beifügt.

4.3.4.2.1 module_eval und instance_eval

Ganze Methoden können Klassen und Instanzen mit module_eval (synonym für class_eval, Methode der Klasse „Module“) und instance_eval (Methode der Klasse „Object“) hinzugefügt werden.

Der einführende „Crashkurs“ gibt folgendes Beispiel:

```
class BigMeanGiant
end

%w(fee fi fo fum).each do |name|
 BigMeanGiant.class_eval <<-EOS
  def #{name}()
   puts 'Giant says:  #{name.upcase}!'
  end
 EOS
end
```

Die so erzeugten Methoden lass sich dann ganz normal mit:

```
BigMeanGiant.new.fee
```

aufrufen. Der mit

```
xyz.class_eval << EOS
```

erzeugte Code für die Klasse „xyz“ kann jedoch vom Debugger nicht interpretiert werden und das heißt, dass bei interaktivem Debuggen nicht in den Code „hinein gesteppt“ werden kann.

Besser ist es deshalb wohl, obige Anweisung zu erweitern:

```
xyz..class_eval <<-EOS, __FILE__, __LINE__+1
```

wie es Rails (in etwa) bei „cattr_reader“ tut. Warum „in etwa“? Weil Rails den Befehl noch klammert und auf die Addition mit „1“ verzichtet. Was trotzdem nicht geht, ist, den Wert der in diesen Methoden aufgeführten, indirekt „benannten“ Variablen auch zu erfahren.

Eine weniger exotisch wirkende Lösung für das oben angeführte Beispiel ist die Verwendung des Ruby-Befehls „define_method“ der Klasse „Module“, um Instanzenmethoden zu erzeugen:

```
class BigMeanGiant
  %w(fee fi fo fum).each do |name|
    define_method(name) {puts name}
  end
end
```

Als Nachteil wird in diesem Fall angesehen, dass die mit define_method erstellte Methode wohl nicht im AST des jeweiligen Programmes auftaucht.

4.3.4.3 Proc, Continuation und Marshal

Prozedur-Objekte („proc“) speichern Code inklusive der aktuellen lokalen Variablen. Als Objekte lassen sie sich dabei ganz normal referenzieren, sodass diese Referenzen als „function pointer“ dienen können. Ruby stellt darüber hinaus Objektmethoden als eigenständige „Methodenobjekte“ zur Verfügung, wobei nicht nur lokale, sondern auch Instanzvariable erhalten bleiben und auch diese Methodenobjekte sind referenzierbar, also über eine den „function pointern“-vergleichbare Option ansprechbar.

Continuations sind ebenfalls interessant – Kernel-Objekte, die einen bestimmten Ort in einem Ruby-Programm repräsentieren, der damit sogar dann noch angesprochen werden kann, wenn sich dieser Code außerhalb des aktuellen Programm-Ablaufs („out of scope“) befindet.

Auch das Ruby-Modul „Marshal“ ist sehr nützlich – es speichert („serializes“) Objekte in einem String (oder falls angegeben, in eine andere Ausgabeform):

```
string = Marshal.dump(obj)
```

und rekonstruiert sie auch wieder:

```
obj = Marshal.load(string)
```

Marshal bearbeitet freilich nicht jedes beliebige Objekt – so werden komplexe Objekte, die „binding“ benötigen, oder auch „procedure“-Objekte, Instanzen der Klasse „IO“ oder singleton-Objekte nicht erfolgreich behandelt und ergeben

einen „TypeError“, wie das interaktive Ruby-Dokumentations-Tool „ri“ vermeldet.

Komplexe Objekte sind beispielsweise verschachtelte Arrays oder Hashs, die somit zu dieser Fehlermeldung führen können. Leider geschieht dies nicht bereits beim Abspeichern – derartige Objekte werden von Marshal noch widerspruchslos in alle möglichen Formate einer Datenbank (binary, Blob, Text) hinterlegt, beim Zurückholen ergibt sich dann aber der angekündigte Typ-Fehler, wenn Ruby im Modus „verbose“ läuft (was beim Debuggen in der Regel der Fall ist):

```
incompatible marshal file format (can't be read)
        version %d.%d required; %d.%d given
```

während ansonsten derselbe Vorgang die Meldung:

```
marshal data too short
```

erzeugt.

Verschachtelte Arrays oder Hashs sind dabei solche, deren Elemente aus weiteren Arrays oder Hashs bestehen, wobei einfache Formen solcher Verschachtelungen – wie Arrays, die aus Arrays einzelner String-Elemente aufgebaut sind – noch problemfrei verarbeitet werden. Ab welcher Verarbeitungsstufe Marshal sich weigert, korrekt zurückzuspeichern, lässt sich über die Marshal-Versionen im Debug erkennen, die beim Abspeichern und beim Zurückladen überprüft werden können: Wie die Dokumentation mitteilt, sind dies genau die ersten beiden Bytes der Marshal-Daten.

Anwendungsbeispiel: Der zweite Schritt von FirstRails

4.3.4.4 Alias

Über ein Alias lassen sich diverse Ruby-Objekte wie (bereits bestehende) Methoden, Operatoren, globale Variable und Verweise eines regulären Ausdrucks ($&, $', $', $+) „um-referenzieren“, wobei die Parameter für „alias“ sowohl Namen als auch Symbole sein dürfen.

Rails benutzt dies sehr häufig bei Methoden, um die Programmierung zu „verschlanken“. So findet sich die wichtige HTML-Funktion „tag“, die Tags erstellt, im helpers-Ordner des ActionView-Moduls in „tag_helper.rb“. Sie wird jedoch in „active_record_helper.rb“ mit einer Fehlerbehandlung „umgeben“ für den Fall, dass Fehlerbehandlung überhaupt möglich ist. Anstatt umständlich mit diversen Abfragen und/oder „super“ zu operieren, verwendet Rails das Ruby-Aliasing, da die umbenannte Funktion „tag_without_error_wrapping“ dadurch auf die bereits bestehende Originalversion von „tag“ in „tag_helper.rb“ zugreift:

```
alias_method :tag_without_error_wrapping, :tag
def tag(name, options)
        if object.respond_to?("errors") &&
object.errors.respond_to?("on")
            error_wrapping
                (tag_without_error_wrapping(name, options),
                object.errors.on(@method_name))
        else
            tag_without_error_wrapping(name, options)
        end
end
```

4.3.4.5 Yield

Yield ist ebenfalls ein interessantes Angebot von Ruby. Es „führt aus“, was gerade zur Ausführung ansteht und ist eine der eher ungewohnten Erscheinungen, die von Rubys Gleichbehandlung von Variablen und Methoden lebt – ungewohnt zumindest für Programmierer „der alten Schule“, die es gewöhnt sind, dass Parameter deutlich erkennbar übergeben werden.

Nicht so bei „yield“, wie das Beispiel „What does “&” prepended to an argument mean?“ in den FAQ zeigt, das den Parameter-Bezeichner „&“ erläutert: Dieser wandelt einen ausführbaren Ruby-Befehlsblock in ein Prozedur-Objekt um und übergibt dieses daraufhin der betreffenden Methode – die diese Befehle sodann per „yield“ verwirklicht, was auch für den Aufruf und die Verfügbarkeit der notwendigen Parameter sorgt.

Das „Ungewöhnliche“ für Programmierer, die diese Strategie nicht kennen, ist schlicht, dass die Methode, die per „yield“ ihre parametrisierten Befehlsketten ausführen muss, diese nicht deutlich erkennbar als Parameter erhält, sodass aus der reinen Source nicht einmal ein Anhaltspunkt zur Verfügung steht, von woher eigentlich der zu bearbeitende Inhalt kommt.

Das einfache Beispiel aus den FAQ führt dies vor. Mit den Definitionen

```
square = proc { |i| i*i }
```

und

```
def meth2
        print yield(8), "\n"
end
```

führt der Aufruf

```
meth2 &square
```

dann zum Ergebnis „64“.

Als praktisches Anschauungsbeispiel kann die Fehlerbehandlung von Rails in „validations.rb“, Funktion „each_full“ dienen.

4.3.4.6 Send

Nicht weniger interessant ist Rubys Befehl „send“ – mit ihm nämlich lässt sich eine Methode einfach über ihren Namen aufrufen.

Wozu das gut sein soll?

Nun, Ruby erlaubt es doch sehr flexibel, Methoden in Klassen und Instanzen einzufügen und diese flexibel eingefügten Funktionen wollen dann auch verwendet werden – genau dafür ist ein schönes Beispiel in Rails zu finden, denn „record“-Objekte weisen die einzelnen Dateifelder als Methoden auf.

Übrigens – diese Scaffold-Templates

```
...\rails-2.0.2\lib\rails_generator\generators\components\scaffold\templates
```

sind generell ein schönes Anschauungsbeispiel für „anschmiegsame“ Programmierung mit Hilfe von Indirektionen.

4.3.5 Objekt-Gleichheit

Ruby nutzt die übliche Syntax von „=“ als Zuordnungselement und „==“ als Vergleichsoperator. Für Case-Statements tut „===“ im Prinzip dasselbe wie „==“, wird jedoch typischerweise von abgeleiteten Klassen überschrieben, wie es im interaktiven Ruby-Dokumentations-Tool „ri“ heißt.

Bei der Zuordnung über „=“ ist zu beachten, dass diese Zuordnung keine rein wertmäßige ist, sondern dass tatsächlich dasselbe Objekt über eine Referenz zur Verfügung gestellt wird. Das heißt, dass eine Veränderung dieses Objektes automatisch die über „=“ zugeordneten Variablen mit ändert. Das kann gelegentlich bei den praktischen Arrays oder Hashs irritieren, wie im folgenden Fall:

```
top = @html[i+1][1]
while level > top
  ...
  top.succ!
end
```

In diesem Fall („succ!“) wird das Original-Element „top“ verändert – und das ist identisch mit „@html[i+1][1]“, sodass auch das Array „@html“ nicht mehr die alte Form aufweist.

Soll das Array in der alten Form erhalten bleiben, tut deshalb

```
top = @html[i+1][1]
while level > top
  ...
  top = top.succ
end
```

bereits das Gewünschte. „freeze“, also das (endgültige) „Einfrieren“, kommt dafür kaum in Frage und auch „clone“ und „dup“ scheinen nicht erforderlich zu sein – zwei Kopiervarianten, die das Objekt selbst zusammen mit seinem aktuellen Zustand duplizieren (ohne jedoch referenzierte Objekte mitzuverdoppeln) und sich möglicherweise nur in den abgeleiteten Klassen unterscheiden, wie die Dokumentation „ri“ sagt.

4.4 Einzelne Rails-Befehle

4.4.1 cattr_reader, cattr_writer, cattr_accessor, mattr_reader...

Für Klassen führt Rails eine Erweiterung der Ruby-Klassen „Class“ und „Module“ durch und ergänzt die vorhandene Instanzfunktionalität um entsprechende Klassenmethoden in „attribute_accessors.rb“, untergebracht auf dem Pfad von „Active Support“ (hier Versions 2.0.2):

```
...\activesupport-2.0.2\lib\active_support\core_ext\class
...\activesupport-2.0.2\lib\active_support\core_ext\module
```

4.4.2 Find

In den Grundlagen „base.rb“ des „Active Record“-Moduls findet sich nicht nur die allgemeine, mächtige Funktion „find()“, sondern auch ihre diversen Abarten wie beispielsweise „find_by_sql“, das ein beliebiges SQL-Statement auszuführen erlaubt oder die interessanten „find_by_xyz“, wobei „xyz“ tatsächlich nur ein Feldname ist – oder mehrere – und trotz dieser vereinfachenden Schreibweise noch Optionen erlaubt.

Bemerkenswert bei „find_by_xyz“ ist, dass die verwendeten Feldnamen nicht indiziert sein müssen, dass aber im Falle, dass sie indiziert sind, diese als Sortierung dient – was übrigens auch für die anderen find-Befehle gilt.

Nicht mehr zum Umfang des nativen Rails 2.0 gehört „pagination“, das (nunmehr als Plugin „classic_pagination“) Controllern auf vielfältige Weise ermöglicht, eine Auswahl von Datensätzen zu besorgen und sie der Oberfläche zur Verfügung zu stellen – und das nicht nur „einfach so“, sondern sortiert und vor allem mit einer „Page“-Funktion, die auf diese Datensätze in einer

vorgebbaren Schrittfolge zugreift. Gesteuert wird diese seitenweise Listenbehandlung über eine Instanzvariable „xyz_pages“, die automatisch von Rails oder nach Bedarf selbst erstellt werden kann, was wie immer ausführlich in der Dokumentation von „pagination.rb“ nachzulesen ist.

Zu beachten ist nur, dass jede dieser stufenweisen Listenbearbeitungen zwar immer von derselben Controller-Funktion durchgeführt wird, dass aber Parameter bei den folgenden Aufrufen nicht länger berücksichtigt werden. Soll deshalb das Einlesen der Datensätze nicht nur beim ersten Mal bedingt ablaufen, so ist zu beachten, dass die geforderte Bedingung tatsächlich bei jedem Aufruf noch bekannt ist. Die Schwierigkeit dabei ist, dass es sich in diesem Fall wieder um die Kommunikation mit dem Browser handelt, bei der die ständigen Initialisierungen ein Aufbewahren von Variablen problematisch macht. In einfachen Fällen sollte die Bedingung somit bereits vor dem Aufruf dieser Listenbearbeitung in der View erfolgen – im Klartext: Die Abhängigkeit wird am bequemsten durch mehrere Funktionen abgedeckt, die intern dann natürlich parametrisiert auf die gleiche Methode zugreifen.

Anwendungsbeispiel des ursprünglichen „paginate“: Der vierte Schritt von FirstRails

4.4.3 Render und Redirect

Auch „render“ bietet gleich eine ganze Sammlung von Funktionen an, da es sich mit dem Komplex von Aufbereitung und Transport der Daten an die Oberfläche beschäftigt. In „base.rb“ von Action Controller wird es im Kommentarteil unter

```
# == Renders
```

überblicksweise erläutert. So erfolgt die Datenübergabe über Instanzvariablen, die automatisch vom Controller an das Template übergeben werden.

Im „protected“-Bereich dieser Source finden sich dann die Ausführungen zu den tatsächlichen Funktionen wie der allgemeinen und wieder sehr mächtigen render()-Methode und ihren diversen Varianten wie „render :inline =>“, das

einen (sogar ausführbaren) Text ausgibt oder „render :action => "xyz"“ („render_action "xyz"“), das die Ausgabe der Daten auf diejenige der Aktion „xyz“ umdirigiert.

Im Gegensatz zu „render :action“ als dem reinen „Ausgeben“, das Templates nur aufbereitet, ist die „Umleitungsfunktion“ über „redirect_to“ dagegen ein vollständiger Methodenaufruf.

Render-Elemente finden sich jedoch noch an anderer Stelle: in der Source „base.rb“ von ActionView, um die Ausgabe direkt auf der Ebene der Templates zu ermöglichen. Ein Beispiel in der Dokumentation ist:

```
<%= render "shared/header",
        { "headline" => "Welcome", "person" => person } %>
```

Nützlich ist auch

```
render :file
```

das eine Datei über die Angabe des absoluten Pfads des Templates ausgibt oder das bereits erwähnte:

```
render :inline => text
```

das den Inhalt der Variable „text“ an der angegebenen Stelle im Template ausgibt – und so nebenbei bemerkt in dieser Variable sogar Javascript-Befehle erlaubt.

Anwendungsbeispiel: Das Welcome-Fenster von Rails, der zweite Schritt, der letzte Schritt von FirstRails

Weiterhin interessant ist:

```
render :nothing => true
```

das an der angeforderten Stelle einfach nichts ausgibt. Im Zusammenhang mit Ajax bedeutet dies, dass kein Ergebnis sichtbar wird, was beispielsweise für „remote_function“ nützlich sein kann.

Anwendungsbeispiel: Der vierte Schritt von FirstRails

4.4.3.1 Render :template

Von besonderem Interesse ist auch „render :template“, das den relativen Pfad des Templates akzeptiert, den es über den mitgelieferten String erhält:

```
render :template => "application/index"
```

Durch den Aufruf der View-Source mit Pfad eignet sich diese Template-Variante des Render-Befehls für eine „Querverbindung“ zwischen unterschiedlichen Controllern, da sie es einem Controller erlaubt, auf Templates aus anderen als dem eigenen View-Ordner zuzugreifen – dies ist dann interessant, wenn ein Controller die Oberflächen eines anderen nutzen soll. Hier ist jedoch zu beachten, dass die gesamte Programmlogik dann gewährleisten muss, dass alle erforderlichen Instanzvariablen, die diese „fremde Sicht“ von ihrem eigenen Controller erwartet, natürlich mitgeliefert werden müssen. Und weil ein anderer Controller den gesamten Kontext des Programmablaufs verändert, sollte jede Aktion in solch einem Prozess, ob in einem Template oder im Programmcode, ihren maßgeblichen Controller immer über den Parameter „:controller“ sicherstellen.

4.4.3.2 Partials

Auch die Variante „render :partial“ ist hervorzuheben, beschrieben unter

```
# === Rendering partials
```

Diese „partials“ sind Teilformate, die von anderen Oberflächenelementen, seien es „vollständige“ Templates wie „html.erb“-Dateien oder wiederum Partials – oder auch von Controllern, die diese Oberflächen steuern – verwendet werden können und vor allem hinsichtlich Ajax gebraucht werden. Denn Ajax erlaubt es, nur diejenigen Elemente zu aktualisieren, die auch tatsächlich geändert wurden. Wie üblich regelt das CoC-Prinzip den Umgang mit den „partials“: So sind sie mit führendem Unterstrich zu benennen, werden aber ohne diesen Unterstrich aufgerufen. Sie können dann ihrerseits

Funktionen anstoßen wie Helfermethoden, auszuführende Aktionen in den verschiedenen Rails-Befehlen wie „link_to...“ oder „observe ...“ oder auch pures Javascript.

Die Partial-Elemente machen dabei überhaupt keine Schwierigkeiten bei einer Mehrfachverwendung. Sie lassen sich beliebig nebeneinander im Layout plazieren und aufrufen,wobei der Parameter „:locals“ erlaubt, im jeweiligen Template-Element eine lokale Variable zu definieren, die einer Aktion oder Helfermethode mitgegeben kann. Erfolgt beispielsweise im Layout der Aufruf des Partial-Elementes mit

```
<div id="abc">
   <%= render :partial => "xyz", :locals => { :name => "david" }
%>
</div>
```

so kann im Partial selbst eine Helfermethode mit dieser lokalen Variablen „name“

```
<% Helferlein(name) %>
```

aufgerufen und im Controller der Helfermethode mit

```
def Helferlein(name)
```

benutzt werden, wobei der Parameter „name“ in beiden Fällen gehorsam den Wert „david“ übermittelt, der der lokalen Variable bereits beim Aufruf des „render :partial“ mitgegeben worden war.

4.4.3.3 Querverbindungen zwischen diversen Controllern und Templates

Partial-Elemente lassen sich ebenfalls von verschiedenen Controllern benutzen, hier muss jedoch die „Template“-Schreibweise verwendet werden, die den genauen (relativen) Standort des Partials angibt:

```
render :partial  => "abc/xyz", :locals => { :name => „david"}
```

Bei der Interaktion zwischen diversen Controllern und Partial-Elementen darf die Art der Interaktion freilich nicht vergessen werden – wird eine Verarbeitung ohne Ajax-Kommandos aufgerufen, so macht es wenig Sinn, in ihrem Verlauf ein Partial-Element direkt anzusprechen. Da ohne Ajax immer die gesamte HTML-Seite ausgegeben wird, erscheint in diesem Fall das Partial nämlich nur als ein einsames einziges Element auf einer ansonsten leeren Seite, selbst wenn der Controller das bunteste Rails-Layout verwenden würde. Um das gesamte Fensterbild zu erhalten, muss deshalb unter diesen Umständen ein „normales" Template verwendet werden, das über „yield" (das das alte „@content_for_layout" ersetzte) von Rails in eine vollständige HTML-Datei umgewandelt wird – inklusive des gewünschten Partial-Elementes. Manchmal hilft bei der Kombination verschiedener Oberflächenelemente und der auf ihnen auszuführenden Aktionen jedoch nichts anderes, als ein wenig Javascript zu integrieren, um das gewünschte Ergebnis zu erhalten.

Anwendungsbeispiel: Der vierte Schritt von FirstRails

Nicht nur die Ausgabe, sondern eine ganze Aktion wird über

```
redirect_to :action => "index", :controller => 'application'
```

durchgeführt. So kann also nicht nur ein Template von mehreren Stellen aus aufgerufen werden, sondern auch die gesamten Funktionen anderer Controller.

Anwendungsbeispiel: Der dritte Schritt von FirstRails

Parameter werden einfach durch Symbole hinzugefügt, sollten jedoch nicht unbedingt „:params" lauten:

```
redirect_to :action => "error", :controller =>
'application', :confirm => info
```

Anwendungsbeispiel: Der vierte Schritt von FirstRails

Zu beachten ist unbedingt bei einer etwaigen direkten Controller-Controller-Interaktion, dass Rails bei Controller-Methoden immer ein automatisches Rendering durchführt, bevor die Kontrolle wieder an die Views übergeht –

werden deshalb weitere, hintereinander verkettete Controller-Methoden aufgerufen, muss dort, wo nicht direkt mit dem Browser kommuniziert wird, ein Render-Befehl tunlichst vermieden werden.

4.4.4 Link_to...

„Link_to“ und die Ajax-Version „Link_to_remote“ sind Helferfunktionen im helpers-Ordner des ActionView-Moduls in den Source-Dateien „url_helper.rb“ und „prototype.helper“, die die entsprechenden Anker-Tags aus den Vorgaben erstellen.

In den anzuzeigenden Texten dürfen dabei weitere HTML-Inline-Elemente vorkommen. Ist beispielsweise das Element „span“ mit roter Textfarbe definiert, so zeigt

```
<%= link_to „abc <span>def</span>“, :action => „xyz“,
        :controller => „ijk“,
        :params => { „p1“=> „p1“, „p2“  => „p2“}%>
```

das Resultat tatsächlich als Link an, dessen Textteil „def“ in Rot gehalten ist.

Anwendungsbeispiel: Der zweite Schritt von FirstRails

4.4.5 Listen-Tags <select>, <option>

Rails bietet einige Möglichkeiten an, HTML-Auswahllisten zu erstellen wie „select“ in „form_options_helper.rb“ im Ordner „helpers“ von ActionView, das im Template über

```
<%= select("abc", "def", xyz)
```

ausgeführt wird, wobei „abc“ der Name eines Modells ist, „def“ das zu identifizierende Feld des jeweiligen Datensatzes (im Standardfall also „id“) und „xyz“ ein Array, das über die HTML-<option>-Tags aufbereitet werden soll.

Ein anderer Befehl, angeboten in derselben Source „form_options_helper.rb“ ist:

```
options_from_collection_for_select
```

Er erlaubt eine noch bequemere Verarbeitung der so häufig vorkommenden Datei-Listen (wie „@lists“) aus den Rails-Modellen.

Welches der Listenelemente von den Anwendern am Ende ausgewählt wurde, wird dem Controller sodann über „@params“ mitgeteilt (beispielsweise für die Funktionen „create“ oder „update“), und zwar über den Parameter, der im HTML-<select>-Tag unter „name“ spezifiziert wurde. Und wie immer ist der Typ des Parameters aufgrund der Kommunikation via URL ausschließlich „String“.

Anwendungsbeispiel: Der erste Schritt, der dritte Schritt von FirstRails

4.4.6 Checkbutton

In „form_helper.rb“ im Ordner „helpers“ findet sich auch eine bequeme Möglichkeit, Checkbuttons ins HTML-Layout einzufügen. Das einfachste Beispiel im Kommentar lautet dazu:

```
<%= check_box "person", "single" %>
```

„person“ stellt darin das Objekt dar, das als Instanzvariable „@person“ vorzuliegen hat, „single“ ist die erforderliche Methode (in Rails meist ein Objekt-Attribut), das auf diese Weise bei jedem Aktualisierungsvorgang automatisch über die Checkbutton-Funktion mit berücksichtigt wird.

Ist ein solcher direkter Zusammenhang zu einem Datenfeld jedoch nicht gegeben, steht in „form_tag_helper.rb“ mit „check_box_tag“ noch eine zweite Variante zur Verfügung:

```
<%= check_box_tag "abc", value, checked,
        :onclick => "if ($('abc').checked) {...}" %>
```

was von Rails übersetzt wird in:

```
<input id="abc" name="abc" onclick=
        "if ($('abc').checked) {...}"  type="checkbox"
        value="Inhalt von value" />
```

Im Falle, dass der Wert der lokalen Variable „checked“ „true“ ist, wird im Input-Tag noch die Option „checked="checked"“ hinzugefügt.

4.4.7 Verify und Validate

Verify prüft den Aufruf von Methoden auf notwendige Bedingungen hin und gehört zum Action Controller-Komplex „verification.rb“. Die Prüfung selbst erfolgt vor der gewünschten Operation und leitet den Aufruf im negativen Fall auf eine andere Funktion um. Erforderliche Voraussetzungen werden dem Befehl dabei als Parameter mitgegeben, wobei wie gewöhnlich ausführliche Beschreibungen und Beispielfälle in den Kommentaren zu finden sind.

Validate ist dagegen ein Angebot von Rails, Datensätze auf Korrektheit überprüfen zu lassen und findet sich im ActiveRecord-Komplex unter „validations.rb“. Wie die dortigen Kommentare sowie das Kapitel „UnderstandingValidation“ im Rails-Wiki ausführen, erfolgt diese Prüfung, bevor ein Datensatz gespeichert wird, wobei Fehlermeldungen in einem Fehlerobjekt „@errors“ hinterlegt werden. Ist „@xyz“ die Instanzvariable eines Rails-Datensatzes mit einer Validierung, so kann diese freilich über

```
@xyz.save(false)
```

auch ignoriert oder über

```
@xyz.valid?
@xyz.errors.empty?, @xyz.errors.count,
@xyz.errors.on('feldname'), @xyz.errors.each...
```

zu einem sonstigen Zeitpunkt in der Verarbeitung genutzt werden. Besonders interessant ist die direkte Verwendung dieser Methoden unter dem

Gesichtspunkt, dass Rails über Ajax die sofortige Prüfung von Benutzereingaben erlaubt.

Anwendungsbeispiel: Der zweite Schritt von FirstRails

4.4.8 Flash

Auch „flash“ gehört zum Action Controller-Modul und erlaubt es, temporäre Objekte zwischen Aktionen auszutauschen. Laut Dokumentation wird es bevorzugt für Meldungen aller Art verwendet: „flash.rb“.

Wird im Controller etwa

```
flash[:notice] =
      "Dateibeschreibung \"#{params[:key]}\" fehlt"
```

definiert, so kann in der aufgerufenen View dann mit

```
<%= flash[:notice] %>
```

der betreffende Text ausgegeben werden.

Anwendungsbeispiel: Der zweite Schritt von FirstRails

4.4.9 Listen und Baumstrukturen (Tree, nested set) und Rails 2.0

Auch diese Funktionalität wurde aus dem nativen Rails seit 2.0 entfernt, was aber kein großer Verlust ist aufgrund der großen Bequemlichkeit von SQL-Abfragen. So kann die erforderliche Funktionalität problemlos durch eigene Methoden ersetzen werden, die die entsprechenden SQL-Befehle dann durchführen – ein Beispiel findet sich in Kapitel 2 („Das Inhaltsverzeichnis als Menü: Baumstrukturen“), Sektion „Find_all_by... – SQL ohne Probleme“ des Open-Source-Tutorials „OpenRorBook: Einfache Tutorial-Verwaltung mit Rails“, gespendet vom bussole InformationsVerlag.

Doch auch „acts_as_tree“, das Baumstrukturen realisiert, existiert noch weiterhin als Plugin.

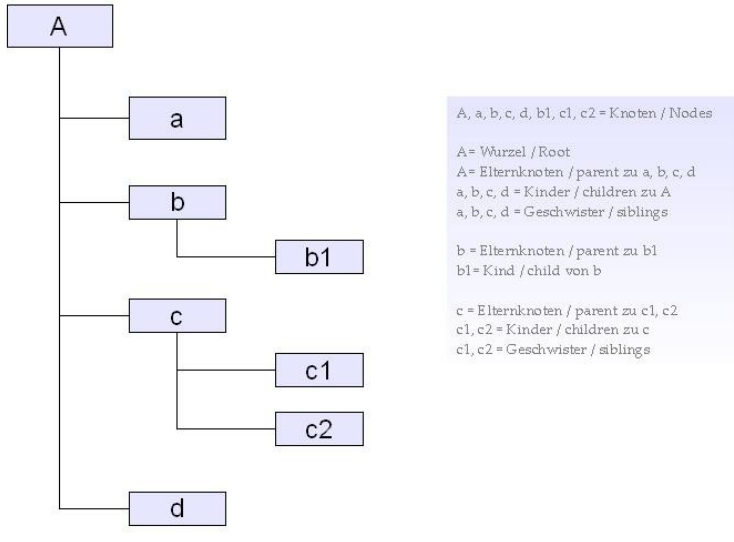
Mit Hilfe dieses Plugins kann ein Modell dann für Baumstrukturen mit dem Befehl

```
acts_as_tree
```

erweitert werden. Dafür benötigt es einen „:foreign_key“ innerhalb der Datenstruktur, in dem der Eltern-Datensatz gespeichert werden kann – und zwar über seine Id, weshalb Rails unterlassungsmäßig davon ausgeht, dass dieses Feld „parent_id“ heißt (es wird bei Bedarf auch jeder andere Name akzeptiert). Der Befehl „create“ von ActiveRecord legt auch in diesem Fall weiterhin einen ganz normalen Datensatz „xyz“ an, wogegen mit

```
xyz.children.create
```

ein Datensatz erzeugt wird, der im besagten Feld „parent_id“ die eindeutige Satznummer des Elternteils abspeichert – was beliebig tief verschachtelt werden kann.



Die Dokumentation von „tree.rb“ erläutert noch weitere Funktionen, mit denen die so hinterlegten Baumstrukturen bearbeitet oder ausgewertet werden können. So liefert

```
xyz.children
```

das Array zurück, in dem die nächste Ebene der Baumstruktur, die direkten eigenen „Kinder“ also, als ActiveRecord-Objekte untergebracht sind. Analog dazu gibt

```
xyz.parent
```

Auskunft über den Elternsatz, während

```
xyz.siblings
```

alle „weiteren“ Geschwister derselben Ebene aufführt.

Anwendungsbeispiel des ursprünglichen „acts_as_tree“: Der dritte Schritt von FirstRails

4.4.10 Oberflächengestaltung mit Templates

4.4.10.1 Verwendung von CSS-Stylesheets

Die Freizügigkeit der Oberflächengestaltung durch CSS-Stylesheets wird durch Rails sogar noch freizügiger, wie in HowtoCorrectlyUse-StylesheetsInYourTemplates beschrieben wird. Dafür existiert der Befehl für Templates:

```
<%= stylesheet_link_tag "xyz" %>
```

zu finden in der Source „asset_tag_helper.rb“ im „helpers“-Ordner von ActionView. Wie das HowTo beschreibt, können auch hier Rubys Indirektionen verwendet werden. So ist es etwa möglich, den Befehl in eine Variable einzubetten, was besonders zusammen mit den „Layout“-Templates von Rails interessant ist. Dadurch wird eine einheitliche Steuerung der Oberflächengestaltung erreicht – interessant beispielsweise, wenn eine

mehrmandentenfähige Software geplant ist, die für unterschiedliche Mandanten auch unterschiedliche Erscheinungsbilder zeigen soll.

Anwendungsbeispiel: Der zweite Schritt von FirstRails

Zwei kritische Gesellen: Scrollbalken und Drag & Drop

Gelegentlich kann CSS (bzw. HTML) jedoch auch ein wenig zickig sein im Zusammenhang mit all den diversen Frameworks, die hier am Werk sind. So tritt eine merkwürdige „Verschiebung“ beim Drag & Drop auf die Baumstruktur (unter den gegebenen Releaseständen) auf, wenn diese per Scrollbalken im Bild bewegt wurde. Dies zeigte sich daran, dass nicht mehr diejenigen Baum-Elemente, die sich direkt unter dem Cursor befanden, für den Drag & Drop-Vorgang aktiviert wurden, sondern einige weiter oben.

Dabei schien es tatsächlich so, als handle es sich bei dieser örtlichen „Aktivitätsverlagerung“ um den Bereich, den der Scrollbalken vom oberen Bildrand entfernt war.

Obwohl im obigen Beispiel der Cursor auf das Element „adapter“ verweist, ist „verwendung“ einige Plätze weiter oben aktiviert.

Wird der Scrollbalken testweise entfernt, verhält sich der Baum wieder völlig normal.

4.4.11 Für das Protokoll

Gelegentlich geschehen Dinge, die sich nur aus der Interaktion diverser Einzelelemente erklären lassen und die somit oft schwer wiederholbar sind – und dadurch kaum Information enthalten. Auch wenn die nächste Aktualisierung von Betriebssystem, Firewall oder sonst einem beteiligten Element die Sache wieder bereinigen könnte, sollen einige Problemchen nicht unerwähnt bleiben.

Denn dann ist es nicht so überraschend, wenn sie aus heiterem Himmel wieder zuschlagen.

4.4.11.1 MySQL

Während die stabile Version des MySQL-Servers zuverlässig ihren Dienst tut, zeigte sich die Beta-Version gelegentlich sehr zickig im Umgang mit den ersten Rails-Gehversuchen, ließ sich jedoch brav sofort wieder hochfahren. Das änderte sich erst mit dem Auftauchen des Störfaktors „Firewall“.

4.4.11.2 Firewall

Im Laufe der ersten Spielereien mit Ajax fing der Rechner an, ein höchst seltsames Verhalten an den Tag zu legen und zwar im Zusammenhang mit „observe_fields“ und dem Versuch, dieses per Instanzvariable „@observe“

```
@observe = 'observe_field(\"AAA\",
        :update = \"target_id\",
        :url => { :action => \"tree_response\" })'
```

in einer Helpermethode zu definieren und dann über

```
<%= eval(@observe) %>
```

in einem „rhtml“-Template aufzurufen. An jenem seltsamen Tag (an dem Deutschland die Weltmeisterschaft verlor) führte dies unter den Bedingungen der vorliegenden Konstellation mit zuverlässiger Regelmäßigkeit zu einer nicht endenwollenden Höchstbelastung des Systems durch einen Windows-Dienst „svchost.exe“ – der sich sogar durch Standby oder Benutzerwechsel nicht beruhigen ließ. Doch was gar mit dem Ruhezustand angestellt wurde, soll kein zweites Mal ausprobiert werden. Zwar ließ sich der Prozess ohne Widerrede beenden, doch wenigstens eine der DLLs, die dieser Dienst wohl bearbeitet hatte, war für die Rails-Umgebung von zentraler Bedeutung gewesen: Während die hübsche Railsmaske „Welcome aboard“ (ein einfaches html-File) immer noch aufgerufen werden konnte, schien die Kommunikation mit Rails höchst empfindlich gestört zu sein – war schlicht nicht mehr vorhanden, verursacht durch das Fehlen des Services „SharedAccess“. Dieser ist durch seine Bedeutung für Internetverbindungen jedoch so direkt mit der Windows®-Firewall gekoppelt, dass es bei der Benutzung einer anderen Schutz-Software (außer der von Windows) unter Umständen nicht mehr möglich ist, den Dienst ohne Neustart wieder hochzufahren.

Das Problem trat häufig im Zusammenhang mit Template-Befehlen auf, bei denen Fehler vorgekommen waren – was kein Wunder ist: Der loop-verursachende Service war „ERSvc“, der normalerweise über den Service „Fehlerberichterstattungsdienst“ verwaltbar ist. Besonders übel schien es im Zusammenhang mit den Rails-Steuerzeichen „<%%>“ gewesen zu sein. Da ein Code-Schnippsel im Rails-Wiki, das ebenfalls einen Template-Befehl über Strings erzeugt, diese Steuerzeichen im Template-Code beließ, wurde das merkwürdige Verhalten schlicht als Hinweis dahingehend interpretiert, nicht zu gewagte Spielchen damit zu treiben.

Anwendungsbeispiel: Der zweite Schritt von FirstRails

Erst als die benutzte Firewall deinstalliert und durch ein anderes Produkt ersetzt worden war, löste sich das Ärgernis in Wohlgefallen auf.

Die Moral von der Geschicht'? Die Kommunikation von Applikation, Web-Server und Browser ist eine vielschichtige Sache und für Nicht-Experten nicht

immer ganz leicht zu durchschauen oder: Viele Köche verderben manchmal den Brei, sogar bei Ruby on Rails.

Wenn also Ruby plötzlich nicht mehr so will oder der Debugger langsamer geworden ist – dann liegt das unter Umständen nicht nur an irgendwelchen vergessenen oder falschen Einstellungen. Manchmal liegt es tatsächlich nur daran, dass die Firewall wieder einmal aktualisiert wurde.

5 Ausgewählte Links

5.1 Die Definition der Information

Information und Länge auf der Information	http://www.bussole.de/html/DefInf.htm

5.2 FirstRails – Download von Sourcen und Umgebung

Sourcen der im Buch beschriebene Anwendung in fünf Schritten mit Datenbankbeschreibungen und lauffähiger Umgebung	http://www.bussole.de/Rails/FR.htm

5.3 Der Verlag bussole IV und seine Bücher

bussole IV	http://www.bussole.de/
Aufbruch zu neuen Ufern	http://www.bussole.de/html/buch04.htm
Die Physik der Information	http://www.bussole.de/html/buch03.htm
Die Fliege - oder - Das Handwerk der Datenbank-Programmierung	http://www.bussole.de/html/buch02.htm
Nicht für Jedermann	http://www.bussole.de/html/buch01.htm
Die Individualität liegt in den Daten	http://www.bussole.de/html/buch00.htm

5.4 Lizenzen

Generelle Anforderungen an Open-Source-Lizenzen	http://www.opensource.org/docs/definition.php
Rails (MIT)	http://www.opensource.org/licenses/mit-license.php
Ruby	http://www.ruby-lang.org/en/LICENSE.txt

Ruby-Logo, Creative Commons Attribution ShareAlike 2.5	http://www.rubyidentity.org/ http://commons.wikimedia.org/wiki/Image:Ruby_logo.svg http://en.wikipedia.org/wiki/Creative_Commons
MySQLs Floss-Extension	http://www.mysql.de/company/legal/licensing/foss-exception.html
Apache License	http://www.apache.org/licenses/LICENSE-2.0.txt
Firefox (MPL)	http://www.mozilla.org/MPL/
RadRails (EPL)	http://www.eclipse.org/legal/epl-v10.html

5.5 Bibliotheken und Verzeichnisse

Deutsche Bibliothek	http://dnb.ddb.de/
Sourceforge	http://sourceforge.net/
Rubyforge	http://rubyforge.org
Wikipedia (deutsch)	http://de.wikipedia.org/wiki/Hauptseite
Wikipedia (englisch)	http://en.wikipedia.org/wiki/Main_Page

5.6 Ruby – Links und Projekte

Ruby	http://www.ruby-lang.org/de/
Ruby-Download	http://www2.ruby-lang.org/en/20020102.html http://ftp.ruby-lang.org/pub/ruby/
Ruby-Installer	http://rubyinstaller.rubyforge.org/wiki/wiki.pl
Einstieg in Ruby	http://www.ruby-doc.org/gettingstarted/ http://www.loudthinking.com/arc/000199.html http://www.math.ias.edu/doc/ruby-docs-1.8.2/faq-en/rubyfaq.html
Ruby auf Wikipedia	http://de.wikipedia.org/wiki/Ruby_%28Programmiersprache%29 http://en.wikipedia.org/wiki/Ruby_%28programming_language%29
Tutorial	http://www.ruby-doc.org/docs/Einfuhrung_in_Ruby/ http://ruby.brian-schroeder.de/course/slides.pdf

	http://www.oreillynet.com/ruby/blog/2005/12/digging_into_ruby_symbols_1.html
Ruby-Manual	http://www.ruby-doc.org/docs/ruby-doc-bundle/Manual/man-1.4/index.html
Ruby-Syntax	http://www.ruby-doc.org/docs/ruby-doc-bundle/Manual/man-1.4/syntax.html
Reservierte Worte	http://www.ruby-doc.org/docs/ruby-doc-bundle/Manual/man-1.4/syntax.html#resword
Std Library Dokumentation	http://stdlib.rubyonrails.org/
RubyGems – Sourcen zum Sammeln	http://gems.rubyforge.org/ http://docs.rubygems.org/
WEBrick – HTTP-Server	http://www.webrick.org/
Mongrel – HTTP-Server	http://mongrel.rubyforge.org/
Interaktive Shell mit ri-Dokumentation	http://rubyforge.org/projects/fxri/
IDE in Ruby	http://rubyforge.org/projects/freeride/

5.7 Rails – Links und Projekte

Rails	http://www.rubyonrails.org/
Rails-Wiki	http://wiki.rubyonrails.org/rails http://wiki.rubyonrails.org/rails/pages/Howtos
Tutorial	http://rubyforge.org/projects/openrorbook/ (bussole IV) http://ruby-rails.info/tutorial (deutsch) http://wiki.rubyonrails.com/rails/pages/Tutorial http://wiki.rubyonrails.com/rails/pages/GettingStartedWithRails http://www.digitalmediaminute.com/article/1816/top-ruby-on-rails-tutorials http://www.onlamp.com/pub/a/onlamp/2005/06/09/rails_ajax.html http://www.erikveen.dds.nl/distributingrubyapplications/rails.html

Rails-Klassen	http://api.rubyonrails.org/
	http://ap.rubyonrails.com/
	http://ar.rubyonrails.com/
RadRails – IDE für Rails	http://sourceforge.net/projects/radrails
	http://aptana.org/download_radrails.php
Rails-Installation	http://wiki.rubyonrails.com/rails/pages/GemRails
	http://instantrails.rubyforge.org/wiki/wiki.pl
Rails-Einführung (PP)	http://obiefernandez.com/presentations/WorkinOnTheRailsRoad.ppt
Firma von David Heinemeier Hansson	http://37signals.com/
	http://www.loudthinking.com/
Projektverwaltung in Rails	http://www.basecamphq.com/
Social Software in Rails	http://www.43things.com/
Todo-Lists in Rails	http://www.tadalist.com/

5.8 Weitere Open Source – Links und Projekte

Open-Source-Definition	http://www.opensource.org/docs/definition.php
Eclipse-Plattform	http://www.eclipse.org/platform/
	http://www.eclipse.org/europa/
Apache	http://www.apache.org/
MySQL - Datenbank	http://www.mysql.de/
phpMyAdmin	http://sourceforge.net/projects/phpmyadmin/
PostgreSQL - Datenbank	http://www.postgresql.org/
Scintilla-Editor	http://www.scintilla.org/
Linux	http://www.linux.de/
Ubuntu	http://www.ubuntu.com/
	https://help.ubuntu.com/
Xubuntu	http://www.xubuntu.org/
Ubuntu-Foren	http://forum.ubuntuusers.de/
	http://ubuntuforums.org/
Dualboot	https://help.ubuntu.com/community/WindowsDualBoot
ISQL-Viewer	http://isqlviewer.com/

6 Index