– ebenfalls im Ruby-Ordner – nicht ausführe, da ich bisher noch kein gem-Produkt benutze, das ich aktualisieren könnte. Wieder entdecke ich etwas Neues in meiner großzügigen Ruby-Installation: Sogar ein GUI-Interface, das die FOX-GUI-Library verwendet (fxruby), ist bei meiner Installation bereits vorhanden. Testweise rufe ich deshalb doch „gem update“ auf, dass mir davon sofort neuere Versionen zum Download auflistet. Da ich im Moment aber nur an Rails interessiert bin, nutze ich nur „gem install rails“ – was einwandfrei klappt, zumal ich keinen Proxy-Server zu berücksichtigen habe. Die Reaktion erfolgt denn auch rasch und bietet mir einige Abfragen an, die ich, wie in GemRails empfohlen, alle positiv beantworte. Danach darf ich gemütlich zusehen, wie RubyGems mir die einzelnen Module und die dazu gehörige Dokumentation herunter lädt, um hinterher die betreffenden Dateien in meinem Ruby-Ordner unter „...\lib\ruby\gems\1.8“ zu finden.

2.1.2.2 Grundgerüst einer neuen Anwendung

2.1.2.2.1 Das Rails-Skelett

Getreu dem GettingStartedWithRails lege ich mir nun einen Ordner für eine Rails-Applikation an (vorerst im typischen Applikationsordner „htdocs“ der Apache-Installation) und probiere den „rails“-Befehl einfach in einem Kommandofenster aus, wobei ich in meiner Windows-Umgebung ganz profan mit der DOS-Anweisung „cd“ in den gewünschten Ordner navigiere und dort eintippe:

```
rails ROR
```

Das „cmd“-Fenster ziehe ich der Kommandozeile „Ausführen...“ im Startmenü schlicht deshalb vor, weil ich es selbst schließen muss und solange eben sehen kann, was sich tut. Und es tut sich einiges, was mir am Ende in meinen neuen Ordner eine Menge Zeug einstellt. Darunter ist auch eine „Readme“-Datei, worin die grundlegende Architektur von Rails beschrieben wird: der Aufbau von Webapplikationen mit Datenbankanbindung nach dem MVC-Muster (model-view-control).

Das „Modell“ umfasst dabei die so genannte „Business Logic“, also die eigentliche Aufgabenstellung – die typischen Objekte und Interaktionen, die für diese

Arbeit maßgeblich sind, wie da sind Kunden oder Teile, Dokumente oder Journale, Aufträge oder Lieferscheine. In Rails wird dies von dem Modul „Active Record“ übernommen, das eine objekt-relationale Abbildung von ganzen Zeilen einer Datenbank zu den Programmobjekten mit ihren diversen Methoden herstellt.

Die faktische Gleichsetzung von business model und Datenbank interessiert mich dabei besonders, weil das prächtig zu meiner ML-Methode zu passen scheint.

Die Aufgabe der View-Ebene (oder Präsentation) ist dagegen die reine Darstellung der Daten. Außerdem dient sie als Schaltstelle für die Benutzer, kümmert sich also um das Layout und die Interaktion mit den Anwendern. Der Controller ist zuletzt die Schnittstelle zwischen Darstellung (View) und Modell. Was die Präsentationsschicht von den Anwendern als Befehl annimmt, muss der Controller so zerlegen und umdirigieren, dass das Modell den Befehl korrekt ausführen kann. Anschließend nimmt er das Resultat des Modells an und liefert es mundgerecht an die View weiter, die es dann entsprechend den Präsentationsvorgaben und Benutzereinstellungen aufzubereiten hat. Aufgrund der starken Verzahnung von Befehlsempfang und Befehlsweiterleitung werden beide Ebenen von Rails im Modul „Action Pack“ von den beiden Bestandteilen „Action View“ und „Action Controller“ abgearbeitet.

Das „Getting started“ liefert mir den nächsten Befehl „<tt>ruby script/server</tt>“ mit dem Hinweis, dass dies ein WEBrick-servlet sei. WEBrick ist ein HTTP-Server, geschrieben in Ruby, und wird seit Ruby 1.8.0 als Standardbibliothek jeder Installation mitgegeben. Um diesen Server aufzurufen, muss nur im Pfad der Anwendung der Befehl

```
ruby script\server
```

aufgerufen werden. Dadurch wird der WEBrick-Server hochgefahren, der sich dann im normalen Web-Browser mit

```
http://localhost:3000/
```

ansprechen lässt – und mir zwar nicht das in allen Tutorials erwähnte „Congratulation“ anbietet, aber etwas absolut Vergleichbares: „Welcome aboard, You're riding the Rails!“

2.1.2.2.2 Rails mit Apache

Da ich jedoch auch Apache verwenden will, mache ich mit dem GettingStartedWithRails weiter, das mir ebenfalls sagt, wie ich die Konfigurationsdatei „httpd.conf“ von Apache zu ändern habe. Dabei gehe ich den Weg über „Alias“, damit ich mehrere Anwendungen parallel verwalten kann. Den entsprechenden Hinweis dazu fand ich im Dokument „HowtoDeployMoreThanOneRailsAppOnOneMachine“. Die „httpd.conf“ wird somit wie folgt angepasst:

1) im Bereich „Dynamic Shared Object“ sollte die Kommentierung (#) vor folgender Zeile entfernt werden, um sie zu aktivieren:

```
LoadModule rewrite_module modules/mod_rewrite.so
```

2) und im Bereich „Alias“ wird eingefügt:

```
# rails
# aus "HowtoDeployMoreThanOneRailsAppOnOneMachine"

Alias /ROR "...Beispielpfad/ROR/public"
<Directory "...Beispielpfad/ROR/public">
        Options ExecCGI FollowSymlinks
        AddHandler cgi-script .cgi
        AllowOverride all
        Order allow,deny
        Allow from all
</Directory>
```

wobei “...Beispielpfad/RoR” der volle Pfad meines neu angelegten Rails-Ordners ist, in den ich mir vom „rails“-Befehl die diversen Unterordner – wie beispielsweise „public“ oder „log“ – anlegen ließ.

3) Wird der Alias-Weg gewählt, um Apache die Anwendung bekannt zu machen, muss die Rails-Anwendung noch „umdirigiert“ werden. In der Datei „.htaccess“ des public-Ordners ist dazu der Rewrite-Befehl mit dem Alias-Namen unterzubringen: