chen Feldübersicht und der Leichenauflistung hin- und herwechselt. Dieses „Vergehen“ rührt schlicht daher, dass „index.rhtml“ ehemals als Eingang zur Feldverwaltung gedacht war, jedoch im Moment im direkten Aufruf nicht wirklich funktioniert.

Ändert sich vielleicht noch.

3.5.4 Fazit

Zwei Dinge haben sich während dieses vierten Schrittes herausgestellt:

1) Ruby und Rails kommen mir schon reichlich vertraut vor.

2) Mein Respekt vor dem Browser ist gewaltig gewachsen.

Angesichts des Riesenpensums, das ich noch zu lernen habe, mag Punkt 1) seltsam klingen – aber während dieses Schrittes fiel mir auf, dass ich mich ständig über die Langsamkeit des Debuggers auf meiner Maschine ärgerte und über die Art und Weise grübelte, wie ich die vielen Felder der neuen Datei sinnvoll präsentieren sollte, doch überhaupt nicht mehr daran zweifelte, es auch zu schaffen. Selbst als ich versuchte, per Javascript oder „prototype_helper.rb“ in die Eingeweide des Browsers vorzudringen, um die unangenehme Tatsache zu bereinigen, dass der Browser etwas anderes zeigte, als Ruby gemäß der damaligen Programmierung speicherte (was mir die Grenzen, zumindest meine, im Umgang mit dem Browser deutlich machte), sorgte ich mich nur noch darum, wie lange Zeit ich wohl mit der Lösungssuche verschwenden würde – hegte dagegen nicht mehr die Befürchtung, es vielleicht nie in den Griff zu bekommen.

Und was wohl am Wichtigsten ist: Ich kopiere und variiere inzwischen meine eigenen Bausteine und bewundere dabei von Mal zu Mal mehr die Modularität von Rails – mit ein Grund, warum der Mangel an interaktiven Debugger-Fähigkeiten oft gar nicht so weh tut. Wenn ein Baustein einmal funktioniert, lässt er sich mit wenig (gelegentlich sogar praktisch ohne Aufwand) an anderer Stelle wieder verwenden – wie beispielsweise die Übernahme der Drag & Drop-Funktionalität der Eigenschaften.

Ich fühle mich jetzt in Ruby und Rails schon geradezu heimisch – denn dass ich nicht alles kann und noch vieles lernen muss, blieb mir auch früher nie erspart. Es kommt in diesem Job schließlich immer wieder etwas Neues hinzu, das du noch nicht kennst und dir aneignen musst.

Punkt 2) erwies sich dagegen als wesentlich unerfreulicher.

Hatte ich bisher den Browser nur als Oberfläche gesehen, die mehr oder minder elegant zu präsentieren hatte, was ich ihr zu schlucken gab, hatte mir die Arbeit mit Ajax gezeigt, wie mächtig dieses fast unscheinbare „Oberflächentool" doch war und wie beschränkt dagegen meine Möglichkeiten sind, in seine Verarbeitung einzugreifen.

Oh ja, natürlich hatte ich zuvor auch schon daran gedacht, dass die Anwender wild drauflos blättern können und jeden, wirklich jeden längst vergangenen Arbeitsschritt wieder hervorholen können – hatte das selbst schon oft genug benutzt – doch dass die so erfreulichen Möglichkeiten von Ajax (in meiner Installation) genau auf diese Weise im Nirvana verschwinden können, fand ich dann nicht mehr wirklich lustig.

Denn Änderungen, die über „normale" HTML-Aktionen geschehen wie Texteingaben – bleiben erhalten. Es können also unangenehmerweise Zwischenstadien von Benutzeraktivitäten durch das Blättern auftauchen, die die Anwender in dieser Form gar nicht selbst erzeugt haben. Ist das Ajax-manipulierte Feld beispielsweise das erste auf dem Fenster, so werden sie wohl meist diese Stelle auch zuerst bearbeiten. Gefühlsmäßig ist ihnen dann genauso wahrscheinlich „offensichtlich", dass sie dieses Feld längst geändert haben müssen, wenn die nachfolgenden schon Korrekturen zeigen – doch einmal Blättern verändert die Sache eben: Die nachfolgenden Textfelder sind weiterhin geändert, das darüber plazierte Ajax-Feld ist dagegen wieder auf seinem ursprünglichen Zustand gelandet.

Und weil meine Erfahrung mit Benutzern zeigte, dass sie in solchen Fällen sehr störrisch und missmutig werden können, versuchte ich natürlich, diesem Problem Herr zu werden.

Schaffte ich nicht – und habe keine Ahnung, wie die Ajax-Community das lösen will bei der Masse unterschiedlicher Browser auf dem Markt. Denn der Weg, der auf den ersten Blick plausibel scheinen könnte, wäre schlicht, die Browserlogik insoweit umzukrempeln, dass die mit Javascript veränderten Instanzen nicht nur im laufenden Betrieb säuberlich unterschieden werden, sondern auch in der Historie. Das aber würde nicht nur das generelle Verhalten des Browsers ändern – und sicher nicht in allen diversen Produkten und Varianten gleichzeitig zur Verfügung stehen, – sondern auch das bisherige Blättern ad absurdum führen. Denn die Historie des Blätterns war früher vermutlich dazu gedacht, die diversen Websiten zu protokollieren, die die Anwender so nacheinander aufsuchten, und eher weniger dazu, jeden einzelnen kleinen Arbeitsschritt zu verewigen, sodass wohl ein mehrstufiges Blätterkonzept erforderlich würde.

Wie gesagt – das Ganze sieht mir nach viel Diskussionen in der Community aus. Und es zeigte mir klar auf, warum es Leute gibt, die Ajax (und den Browser) nicht als tauglich für eine Software erachten, die nicht nur zum Spaß benutzt wird.

Ich tue das nicht. Mir ist die Allgemeinheit der Oberfläche und die Flexibilität des Layouts sehr viel wert, sodass ich die Differenzen zwischen den Betriebssystemen, die sich vielleicht nie ganz beheben lassen, nicht für so bedeutsam halte. Und auch wenn ich weiß, dass Anwender sehr zickig sein können, so weiß ich auch, dass sie meistens bloß ordentlich ihren Job machen wollen. Erlaubt ihnen das eine Software, so sind sie häufig dazu bereit, über Kleinigkeiten hinweg zu sehen.

Außerdem habe ich Vertrauen in das Open-Source-Konzept. Ich denke, dass nicht nur der langsame Debugger von Ruby, sondern auch die Probleme mit Ajax, die nicht nur Oberflächengeplänkel sind, recht zügig behoben werden. Und solange muss ich eben dafür sorgen, dass wenigstens keine inkonsistenten Daten gespeichert werden.

> Nachtrag:
> script.aculo.us hat für das Blätterproblem wohl einen viel einfacheren Weg gefunden, als ich mir das so dachte – und dabei ganz nebenbei mein Vertrauen in Open Source als berechtigt erwiesen.

Das war wahrscheinlich der wichtigste Lerneffekt: Zu akzeptieren, wo die Schwächen des Browsers liegen, um auch diese Fälle beim Test der Programme zu berücksichtigen, ja eigentlich überhaupt zu wissen, dass hier ein Bedarf besteht. Im Verlauf dieses Prozesses lernte ich auch den DOM-Inspektor und die Fehlerkonsole des Firefox wieder sehr zu schätzen und habe eines praktisch aufgegeben:

Das Zwischenspeichern.

Cachen im Zusammenhang mit dem Browser ist mit Vorsicht zu genießen

Felder, die im Browser manipuiliert werden, konserviere ich nicht mehr „einfach so“ im Ruby-Programm, denn viel zuviel kann geschehen, von dem Ruby nichts mitbekommt. „Cachen“ ist zwar sicher eine arbeitsspeichernde Strategie im Zusammenhang mit den ganzen Transportprozessen auf dem Internet, es macht jedoch nur dann Sinn, wenn entweder nur wenig geschieht „am Ende der Leitung“ oder wenn gewährleistet ist, dass jede (wesentliche) Änderung auch im Cache ankommt.

Damit habe ich mein Ziel, so weitreichende Kenntnisse über Rails zu gewinnen, um damit arbeiten zu können ...

... tatsächlich erreicht.

Und deshalb werde ich diese kleine Anwendung jetzt abschließen.