Dokumentation

Flex On Rails

Webentwicklung mit Adobe’s Flex 3 und Ruby On Rails

Praktikum Softwarepraktikum 2 / WS07/08

Valentin Schwind (vs016)
Boris Wüst (bw011)

Inhalt

1 Einleitung

1.1 Abstrakt

Ziel dieses Softwaretechnik Projektes ist die Evaluierung des Adobe Flex Web Development Frameworks auf Stärken und Schwächen gegenüber anderen Frameworks, der Anwendung agiler Entwicklungsmethoden und der Integrationsmöglichkeiten in ein RubyOnRails Server-Backend.

Diese Aspekte haben wir anhand der Erstellung eines Konzepts und der prototypischen Entwicklung eines Webauftritts untersucht. Dieser Webauftritt soll die typischen Web2.0 Usability-Merkmale enthalten. Darüber hinaus besteht der Webauftritt aus „User created Content", ist dabei komplett „user-driven" und soll dadurch ein möglichst starkes „Community-Feeling" vermitteln.

Diesen Webauftritt haben wir „LinkBay" genannt. LinkBay ist eine unabhängige Link-Sharing Plattform, auf der man Links aus verschiedenen Kategorien des Internets veröffentlichen, dafür abstimmen und sich über diese austauschen kann.

1.2 Idee

Die Idee für dieses Projekt entstand nachdem wir auf das Adobe Flex Web Development Framework aufmerksam wurden und dieses von Adobe als „Flash für Web Entwickler" mit vielen Vorteilen gegenüber herkömmlichem Flash beworben wird. Wir wollen herausfinden, ob diese Behauptungen von Adobe zutreffen und Flex wirklich die neue Alternative für Web Entwickler ist oder diese nur ein „neu verpacktes" Flash für ihr Geld bekommen.

Dies können wir am besten herausfinden, indem wir Flex auf Stärken und Schwächen gegenüber anderen Web Development Frameworks, der Anwendung von agilen Entwicklungsmethoden sowie die Integrationsmöglichkeiten von Server-Backends (in unserem Fall RubyOnRails mit MySQL) hin untersuchen.

Mit einem einfachen Vergleich mehrerer Web Development Frameworks könnten wir zwar einfach, aber doch nur sehr abstrakt und ohne Hintergrundkenntnisse die Stärken und Schwächen von Flex evaluieren. Doch für die Evaluierung der Anwendbarkeit von agilen Entwicklungsmethoden müssen wir Flex in einem tatsächlichen Web Development Projekt anwenden. Hieraus ist die Idee und das Konzept für eine Web2.0 Link-Sharing Plattform entstanden, die wir später „LinkBay" getauft haben. LinkBay soll aus einem Flex (Flash) Frontend bestehen und über einen Datenservice auf Daten aus der Datenbank eines RubyOnRails Backends zugreifen.

1.3 Konzept

Durch die Link-Sharing Plattform „LinkBay" soll es Besuchern ermöglicht werden, schnell das Wichtigste und Interessanteste im Internet zu finden. Hierfür werden Links unterschiedlicher Typen von Community Mitgliedern veröffentlicht und anschließend kann die ganze Community über

die Links abstimmen, sie bewerten und Kommentare zu den Links abgeben. Diese Möglich-keiten für Mitglieder stellen eine Qualitätskontrolle der veröffentlichten Links dar. So werden für die Besucher bzw. Mitglieder uninteressante Links ausgefiltert bzw. erst am Ende einer Trefferliste angezeigt und man sieht auf einen Blick, was im Internet gerade aktuell ist und heiß diskutiert wird.

Wir wurden zu der Idee für dieses Konzept durch die Vielzahl der heute im Internet existierenden Informations-Portale inspiriert. Um heutzutage auf dem neusten Stand zu bleiben, muss man diese vielen Portale regelmäßig besuchen und die interessanten Dinge aus einer großen Menge von Informationen herausfiltern. Dies ist nicht nur mühsam sondern auch extrem zeitaufwändig. Hier eine beispielhafte Liste solcher Informationsportale:

- ShortNews, Spiegel Online, Google News (Nachrichten und aktuelle Ereignisse)
- YouTube, MyVideo, Clipfish, Metacafe, Veoh (Videos)
- Google Images, Flickr, OnlineImages (Bilder)
- Blogger, Google Blogs, Blogbar (Blogs)
- LachSchon, Lustich, IsNichWar (Lustiges im Internet)
- PodcastAlley, PodcastsDE, PodCatcher (Podcasts)

Schon diese relativ kleine Auswahl zeigt, wie viele Informations-Portale man besuchen müsste, um wirklich nichts Wichtiges im Internet zu verpassen. LinkBay soll die Informationen all dieser Quellen auf einer Seite übersichtlich und strukturiert darstellen, sodass der Besucher von LinkBay sofort sieht was für ihn interessant ist und direkt dorthin weitergeleitet werden kann. Dabei spielen neben der direkten Suche nach Links auch Kategorien und Subkategorien (spezifisch für jeden Linktyp) der jeweiligen Linktypen (All, News, Blogs, Videos, Images & Fun) sowie die Sortierung der Links eine wichtige Rolle. Dabei werden die Sortiermuster noch in zeitliche Abschnitte eingeteilt. Hier ist eine Liste der möglichen Sortiermuster für die Linklisten:

- Newest / Most Recent (die Sortierung erfolgt nach Einstelldatum & -zeit der Links)
- Most Votes (die meisten Votes in einem Zeitabschnitt)
- Best Rated (die besten Ratings in einem Zeitabschnitt)
- Most discussed (die meisten Kommentare in einem Zeitabschnitt)
- Most Rated (die meisten Ratings in einem Zeitabschnitt)

Die Zeitabschnitte werden wie folgt eingeteilt:

- Last Hour (innerhalb der letzten Stunde)
- Day (in den letzten 24 Stunden)
- Week (innerhalb letzten 7 Tagen bzw. 168 Stunden)
- Month (innerhalb der letzten 30 Tage bzw. 720 Stunden)
- Year (innerhalb der letzten 365 Tage)
- All (keine zeitliche Begrenzung, alle Links werden nach dem Suchmuster angezeigt)

Um Zugriff auf die Link-Listen zu bekommen, muss sich ein Besucher nicht extra auf unserer Seite registrieren bzw. anmelden. Doch LinkBay bietet natürlich Funktionen, die nur von registrierten und angemeldeten Mitgliedern genutzt werden können. Darunter fallen folgende Funktionen:

- Login (Anmeldung eines registrierten Mitglieds)
- Account (persönliche Daten & Übersicht der bisherigen Aktivitäten auf LinkBay)
- Submit (einen Link veröffentlichen)
- Vote (für einen bestimmten Link abstimmen)
- Rate (einem bestimmten Link eine Bewertung geben)
- Comment (ein Kommentar zu einem bestimmten Link abgeben)

Natürlich ist dieses Konzept noch nicht vollständig. Wir hoffen bei der Entwicklung der LinkBay Alpha Version neue Ideen und Eingebungen zu haben, sodass wir dieses Konzept vervollständigen und auch noch verbessern können. Zudem liegt der Erfolg bzw. Misserfolg solch einer Plattform im Detail, weshalb wir einen ausgiebigen Beta-Test mit einigen Probanden planen, um so noch kleine Unstimmigkeiten und Usability Probleme ausfindig zu machen und diese beseitigen zu können. Die Beta-Version und der dazugehörige Test wird aber bei weitem nicht mehr im Rahmen dieses Softwaretechnik Projektes stattfinden..

1.4 Entwicklungsziele im Rahmen der Alpha-Version

Da es sich bei LinkBay um ein sehr komplexes und umfangreiches Projekt handelt, das im Rahmen eines Softwareprojekts in einem Semester niemals abgeschlossen werden kann, müssen wir bestimmte Entwicklungsziele definieren, die dann den Umfang der Alpha Version von LinkBay bilden. Diese Ziele müssen hauptsächlich so ausgewählt werden, sodass eine aussagekräftige Evaluation beider Frameworks und der Integrations-Alternativen möglich ist. Zudem sollte die grundlegende Architektur von LinkBay schon in der Alpha Version feststehen.

Hier eine Übersicht der definierten Entwicklungsziele:

- Flex Frontend Design und Implementierung
- Erstellen von „Custom Components“ in Flex
- Datenbank Design für alle Primärdaten
- Festlegen aller Relationen zwischen den Models
- Implementierung aller Models durch ActiveRecords mit Relationen
- Implementierung grundlegender Funktionen durch Controller Actions
- Implementierung der Integrations-Interfaces in Flex und Rails
- Auswahl einer Integrations-Alternative nach Vergleich

2 Projektplanung & Konzeption

2.1 Arbeitsaufteilung

Im Großen und Ganzen haben wir das Projekt in vier Disziplinen aufgeteilt und einigermaßen gleichmäßig auf die Projektbearbeiter verteilt. Die vier Disziplinen sowie ihre Zuständigkeiten sehen wie folgt aus:

- Idee & Konzept Erstellung (Boris Wüst)
- Flex Frontend mit Design und Präsentationslogik (Valentin Schwind)
- Rails Backend und Datenbank mit Applications- & Businesslogik (Boris Wüst)
- Interface Erstellung und Integration von Flex Frontend und Rails Backend (Valentin Schwind & Boris Wüst)

Dadurch ist eine eindeutige Trennung zwischen Flex und RubyOnRails Entwicklung und somit auch die unabhängige Anwendung agiler Entwicklungsmethoden in beiden Teildisziplinen möglich. Die weitere Untergliederung der einzelnen Punkte wird dem jeweiligen Bearbeiter überlassen. Abgesprochen werden nur die jeweiligen „Milestones" der Gesamtentwicklung, die relativ flexibel und spontan festgelegt werden, so wie es in der agilen Softwareentwicklung üblich ist.

2.2 Zeitbedarf

Da für uns alle Teilaspekte dieses Softwareprojekts völlig neu waren, haben wir besonders viel Zeit bei der Konzepterstellung und der Einarbeitung in die jeweiligen Technologien eingeplant. Jedoch weicht gerade dort die eingeplante Zeit sehr von der tatsächlich benötigten Zeit ab, was zum großen Teil an der unzureichenden Dokumentation beider Frameworks, dem Aspekt der neuen Integrationsmethode und zu guter letzt an der zum Teil wirklich schlechten Literatur lag.

Die Entwicklung des Frontends sowie des Backends und die Datenbankerstellung haben einen geringeren Zeitaufwand gefordert, besonders im Vergleich zum Design der Schnittstellen sowie die Integration von Flex und RubyOnRails.

Hier eine Auflistung des Zeitbedarfs nach einzelnen Stufen des Projekts:

Einarbeitung RubyOnRails	- 80h
Einarbeitung Adobe Flex	- 46h
Ideen Sammlung & Konzept Erstellung	- 35h
Einrichten der Entwicklungsumgebung für Rails	- 08h
Einrichtung der Entwicklungsumgebung für Flex	- 06h
Erste Beispiel Projekte auf der jeweiligen Plattform	- 15h
Testen verschiedener Integrationsmethoden	- 15h
Erstellung des Flex Frontends	- 25h
Erstellung der Backend Datenbank mit Design	- 06h

Erstellung des RubyOnRails Backend	- 18h
Erstellung der RESTful Web Services	- 08h
Erstellung der Data-Services	- 12h
Testen der jeweiligen Komponenten	- 06h
Umstellung auf RubyAMF	- 12h
Konfiguration von RubyAMF	- 08h
Problembehandlung und Anpassung von RubyAMF	- 30h
Gesamtbedarf	**330h**

3 RubyOnRails Backend

3.1 Einrichten der Entwicklungsumgebung

Für RubyOnRails gibt es ein Pendant zu den typischen „LAMP" bzw. „WAMP" Paketen. Dieses nennt sich „InstantRails" und muss nicht einmal installiert, sondern nur im gewünschten Ordner entpackt werden und schon ist eine komplette Entwicklungsumgebung inklusive Ruby, RubyGems, RubyOnRails, MySQL und Apache für die Entwicklung bereit. Wir würden jedoch „InstantRails" nur Leuten empfehlen, die RubyOnRails einmal ausprobieren und daher keinen großen Aufwand betreiben möchten. Für die professionelle Entwicklung ist es leider nicht geeignet.

Deshalb haben wir für die LinkBay Entwicklung auch das System komplett manuell installiert und nur ausgesuchte Komponenten bzw. Versionen verwendet. Als aller erstes sollte natürlich die Ruby Laufzeitumgebung installiert werden. Sobald diese installiert ist, würden wir jedem empfehlen „RubyGems" zu installieren und für die weitere Installation zu verwenden. RubyGems ist ein Paket-Management System (ähnlich wie „apt-get" oder „Yum" für Linux) mit dem „Online Repositories" durchsucht und direkt Pakete installiert bzw. aktualisiert werden können. Mit RubyGems wird dann durch nur einen Befehl das aktuellste RubyOnRails installiert. Danach braucht man nur noch eine Datenbank seiner Wahl installieren und man ist startklar für die ersten RubyOnRails Gehversuche. Wir haben uns, wie schon erwähnt, für MySQL entschieden. Darüber hinaus haben wir mit RubyGems noch zusätzlich den Webserver „Mongrel" installiert, da dieser FastCGI unterstützt und somit eine bessere Performance als der mitgelieferte WeBrick liefert.

Am liebsten arbeiten Rails Entwickler auf der „Command line", da sie von dort aus einfach ganze Projekte oder einzelne Elemente generieren lassen und sogenannte „Rake tasks" ausführen können. Doch um professionell und effizient entwickeln zu können, benötigt man auch noch eine IDE, die unter anderem den Komfort des „Syntax Highlighting" mit sich bringt. Leider gibt es für Rails keine eigene IDE, so muss wohl oder übel eine der gängigen IDEs mit einem Rails Plugin erweitert und verwendet werden. Wir haben uns für Eclipse und das „RadRails" Plugin von Aptana entschieden. Dies hat leider sehr lange Zeit nicht funktioniert, wie es eigentlich sollte. Doch mit dem letzten Update, das erst in den letzten Tagen unserer Entwicklungszeit erschienen ist, wurden die gravierendsten Fehler behoben. Nun können wir jedem angehenden Rails Entwickler die Verwendung dieses Plugins in Verbindung mit Eclipse nur sehr empfehlen.

Hier eine Übersicht der für LinkBay verwendeten RubyOnRails Entwicklungsumgebung:

- Ruby 1.8.6, RubyGems 1.0.1
- Rails 2.0.2, Mongrel 1.1.3
- Eclipse 3.3.1.1, Aptana Studio 1.1.1
- RadRails & Ruby Developemt Tools 0.9.3
- RubyAMF 1.3.3

3.2 Allgemeines zu RubyOnRails

Bei der Umsetzung des LinkBay Backends kommen natürlich alle allgemein bekannten Vorteile von RubyOnRails zum Zuge. Um die wichtigsten zu nennen:

- Conventions over Configuration
- Model, View & Controller (MVC) Pattern
- Don't Repeat Yoursef (DRY)
- Less lines of Code

Einige Punkte hiervon können missverstanden werden. „Conventions over Configuration" sagt zum Beispiel nicht aus, dass man ganz auf Konfiguration verzichten kann, denn eher das Gegenteil ist der Fall. Zwar hat durch die RubyOnRails (nachträglich der Einfachheit halber nur Rails genannt) Konventionen alles seinen „von Rails bestimmten" Platz. Dies kommt gerade unerfahrenen Entwicklern zu Gute, die selbst noch wenig Design-Entscheidungen getroffen haben. Doch für erfahrene Entwickler und gerade im Fall von sehr speziellen Problemen bzw. Anforderungen, kann dies ein großes Hindernis sein. Also gerade diese Leute müssen sich erst einmal daran gewöhnen, dass man alles so machen muss wie das einem das Rails Framework vorschreibt, da recht wenig Freiheit für eigene Design-Entscheidungen bleibt.

Darüber hinaus muss auch in Rails einiges konfiguriert werden. Ein sehr heikles und wichtiges Beispiel sind die Rails Routen („Routes"). Diese können bei speziellen Anforderungen einen Anfänger und Profi gleichermaßen den letzten Nerv rauben, falls es mal wieder nicht so funktioniert, wie man sich das vorstellt, weil zum Beispiel die aus logischer Sicht völlig vernachlässigbare Reihenfolge der Routen wieder einmal von Rails falsch interpretiert wurde. Allgemein ist es so, dass es für alles was man in Rails konfigurieren muss bzw. kann keine schönen XML Konfigurationsdateien gibt, sondern alles in Ruby festgelegt werden muss. Zum Glück wurde nachträglich ein Ruby-Scripting Dateiformat nach dem YAML/YML Standard eingeführt, das diese Konfigurationsdateien um einiges übersichtlicher macht. Außerdem kommt man in Rails auch schon recht weit, ohne überhaupt etwas konfigurieren zu müssen. Wie schon erwähnt, liegt hier nur die Tücke im Detail bzw. an den speziellen Anforderungen und Problemen, die letztendlich auftreten, wenn man nicht nur eine kleine Standard Web Applikation entwickeln möchte.

In Sachen „lines of code" ist Rails bzw. Ruby wirklich nicht zu schlagen. Ruby macht zwar wenige Sachen wirklich anders als andere Programmiersprachen. Doch wenn es zum Beispiel um das Auflösen von Datenbank-Relationen geht, dann lässt das Rails Framework „von Haus aus" den Ruby Code wirklich kurz & knackig erscheinen, sowie es in fast keiner anderen Sprache (zumindest nicht in einer für die Webentwicklung verwendeten) bzw. keinem anderen Framework möglich ist. Der Hauptvorteil hierbei liegt aber nicht in der kürze des Codes, sondern an der wirklich guten Lesbarkeit desselbigen. Gut geschriebener RubyOnRails Code ist in den meisten Fällen gut lesbar und verständlich auch wenn man kein Ruby kann oder nicht einmal Webentwickler bzw. Informatiker ist. Dies macht Rails natürlich besonders schmackhaft für Einsteiger, die schnell.

Das „DRY" Prinzip ist nicht etwas, das wir zu den Errungenschaften von Rails zählen möchten. Es ist zwar überall zu lesen, dass es eine „Eigenschaft" von Rails bzw. Ruby wäre, doch der Prozess zum Erreichen von „DRY" wird keines Wegs von Rails bzw. Ruby unterstützt. Dieses Prinzip ist alleine dem Entwickler und dessen Programmierstiel bzw. Planung unterworfen. Wir würden sogar soweit gehen und sagen, dass „DRY" in anderen, weiter strukturierten Sprachen wie Java weitaus einfacher erreicht werden kann. Rails bzw. Ruby kann sich hierbei aber gewiss mit objektorientiertem PHP (ab PHP5) messen und hat gerade gegenüber von PHP noch den Vorteil des erzwungenen MVC Patterns, was wiederum die Anwendung von richtigem „DRY" enorm vereinfacht, aber auch den Entwickler gewissermaßen einschränkt.

Kommen wir nun zum MVC Pattern. Eine wirklich tolle Sache, die aber nichts Neues für einen ausgebildeten Webentwickler sein sollte. Diese werden eher sagen „das ist doch ein alter Hut. Verwenden wir schon seit Jahren!". Doch gerade für die „Scripter" mit mangelnden Programmierkenntnissen, oder sagen wir eher Programm-Design-Kenntnissen, die ihre überwiegend privaten Webauftritte gerne der Einfachheit halber in PHP „scripten", ist dies eine wunderbare Gelegenheit aber auch eine große Umstellung. Man wird von Rails mal wieder zu etwas gezwungen, nämlich zu gutem Programm-Design nach dem MVC Pattern Prinzip. Dadurch wird auch gewissermaßen das „DRY" Prinzip sehr gut unterstützt. Wir sagen hier gezielt nicht, dass es „erzwungen" wird, da man sich immer noch dem „DRY" Prinzip bewusst sein und es aktiv anwenden muss.

Letztendlich können „Scripter" und unerfahrene Programmierer durch den „Rails Weg" nur Gutes lernen. Auch wenn man letztendlich Rails nicht weiterhin verwendet, sei es weil man es einfach nicht mag oder es nicht geeignet ist für bestimmte Einsatzzwecke, wird man doch einige Überlegungen und Konventionen aus der „Rails Welt" mitnehmen und die neu erlangten Kenntnisse beispielsweise in PHP anwenden. So kommen wir zur Schlussfolgerung, dass Rails ein gutes Einsteiger-Framework ist, um die ersten Gehversuche in der Webentwicklung zu machen und auch vielen PHP Entwicklern weiterhelfen könnte.

3.3 RubyOnRails im Rahmen von FlexOnRails

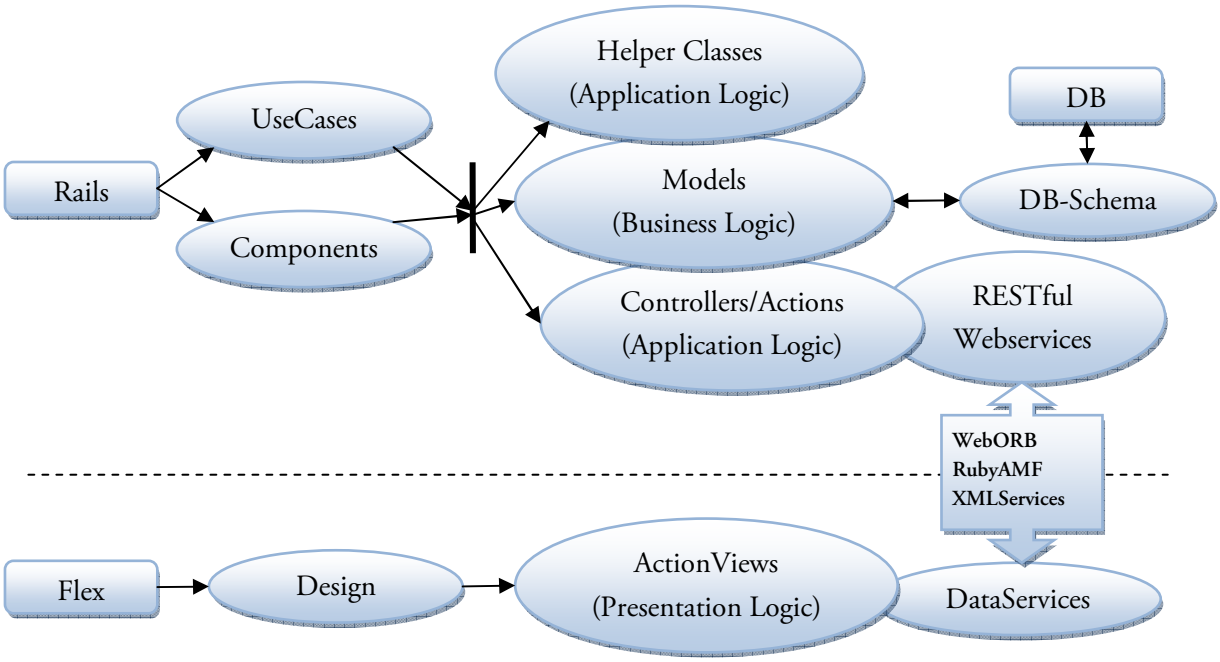
Wie im vorherigen Abschnitt schon erwähnt, kommen uns bei der Entwicklung von LinkBay eigentliche alle Prinzipien von Rails zugute. Die Verwendung von Flex als Frontend hat natürlich keinerlei Auswirkung auf die „DRY" und „less lines of code" Prinzipien von Rails. Bei „Conventions over Configuration" sieht das Ganze schon anders aus. Durch die Anwendung von RESTful Rails Web Services und dem RubyAMF Framework zur Datenkapselung und – Übertragung, muss leider sehr viel konfiguriert werden.

Das YAML Dateiformat und die Fülle von Konfigurationen machen dies sehr aufwendig, komplex, zeitraubend und nicht zuletzt sehr fehleranfällig. Neben den Routen, die alle für RESTful Web Services angepasst werden müssen, muss man auch noch das ganze RubyAMF Gateway konfigurieren und das Mapping der Online-Ressourcen zu den „ActiveRecords" in Rails aufwendig und mühsam von Hand erledigen.

Die größte Anpassung liegt jedoch im Bereich des MVC Patterns. Wir brechen zwar nicht mit dem allgemeinen MVC Prinzip, jedoch gehen wir nicht den üblichen „Rails Weg". In Rails gibt es Templates, die normalerweise für das Rendern der HTML Seiten zuständig sind. Da jedoch in „FlexOnRails" nur reine Daten vom Rails Backend an das Flex Frontend gesendet werden, die dann wiederum im Flex Frontend zu einer bestimmten View gerendert werden, gibt es nicht mehr das Mapping von Ressource bzw. Action zu einer bestimmten View. Es werden die Views also einfach komplett in das Flex Frontend ausgelagert und das Rails Backend ist nur noch für die Models und Controller zuständig.

Damit ist nun das gesamte Design und die Darstellungslogik in der Hand von Flex. Das Rails Backend ist nur noch für die Applikation- und Businesslogik zuständig und sorgt dabei, dass nur die nötigen Daten an das Frontend bzw. den Client gesendet werden. Das Frontend ist dann für die eigentliche Erstellung der View verantwortlich. Ein Beispiel neben dem von uns entworfenen Flex Frontend wäre ein XML verarbeitender RSS Reader, der auch einfach auf die zur Verfügung gestellten RESTful Web Services zugreifen kann und diese nur wissen lassen muss, dass die Daten nicht im AMF Format sondern in XML gerendert werden sollen. So haben wir noch immer die Unabhängigkeit von Model, View und Controller, die das MVC Prinzip auszeichnet, und können alle daraus resultierenden Vorteile nutzen.

Um die Aufteilung der Aufgaben und Elemente zwischen Flex und Rails noch einmal zu verdeutlichen, haben wir dies in der nachfolgenden Abbildung schematisch dargestellt:



3.4 Datenbank-Design in RubyOnRails

In Sachen Datenbank-Distributionen verwenden wir für LinkBay das aktuelle Community Release der MySQL Datenbank, da diese all unsere Anforderungen erfüllt. Zudem ist es noch kostenlos und einfach zu konfigurieren.

Doch in Rails ist es auch kein Problem eine andere Datenbank-Distribution zu verwenden, da das Framework schon Datenbank-Adapter für alle gängigen Distributionen integriert hat. Hierzu muss man lediglich in der YAML Datenbank-Konfigurationsdatei, vermutlich die erste Konfigurationsdatei die jeder Rails Neuling zu Gesicht bekommt, nur den gewünschten Adapter und gegebenenfalls ein paar Parameter anpassen.

Doch die eigentliche Neuerung in Rails ist die Verwendung von Migrations-Dateien für die Erstellung eines Datenbank-Schemas. In diesen Dateien werden schrittweise Veränderungen an der Datenbank bzw. den Tabellen vorgenommen, wie zum Beispiel das Hinzufügen von ganzen Tabellen oder auch einzelnen Feldern zu schon bestehenden Tabellen. In diesen Migrations-Dateien sollte auch immer das Vorgehen bei einem „Rollback" definiert werden, sodass die Änderungen an der Datenbank durch eine bestimmte Migration immer rückgängig gemacht werden können. Falls nichts anderes gewünscht wird, werden vor der Erstellung der Datenbank-Tabellen durch "Rake" immer alle vorhandenen Migrations-Dateien zusammengefasst und zu einem kompletten Datenbank-Schema kompiliert.

Die Anwendung eines solchen Migrations-Systems hat zum Vorteil, dass man nun auch das Entwerfen des Datenbank-Schemas völlig in den agilen Entwicklungsprozess integrieren kann und somit das Datenbank Design mit jedem Entwicklungsschritt „mit wächst". Dadurch muss man sich keine Gedanken mehr darüber machen, ob das entworfene Datenbank-Schema vielleicht durch den nächsten Entwicklungsschritt inkompatibel wird. Zudem ist die Komplexität der einzelnen Datenbank Migrationen deutlich geringer als beim Entwurf eines kompletten Datenbank Schemas zu beginn eines Projekts.

Datenbankmigrationen und Schema
Hier möchten wir nur auf den Zusammenhang und die Abhängigkeiten der Datenbank Migrationen und des Schemas eingehen. Da die Migrationen und das Schema hautsächlich von den Daten Modellen abhängen und wir diese hier noch nicht definiert haben, ist es nicht besonders tragisch falls die Migrationen bzw. das Schema nicht vollkommen nachvollzogen werden können. Die Hauptsache ist zu verstehen, wie aus den einzelnen Migrationen das komplette Datenbank Schema entsteht.

Hierzu listen wir nun die einzelnen Migrations-Dateien der Reihenfolge nach auf und zeigen abschließen das daraus entstandene Datenbank Schema:

Listing: 001_create_users.rb

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table :users do |t|

      # login information
      t.column :username,   :string,  :limit => 24, :null => false
```

```
      t.column :password,   :string,  :limit => 32, :null => false
      t.column :email,      :string,  :limit => 64, :null => false

      # personal information
      t.column :firstname,  :string,  :limit => 32, :null => false
      t.column :lastname,   :string,  :limit => 32, :null => false
      t.column :gender,     :string,  :limit => 6,  :null => false
      t.column :birthdate,  :date,                  :null => false
      t.column :city,       :string,  :limit => 32, :null => false
      t.column :country,    :string,  :limit => 32, :null => false

      # extended user information
      t.column :about,      :text
      t.column :website,    :string
      t.column :blog,       :string

      # user relations
      # t.column :city_id,    :integer, :null => false
      # t.column :country_id, :integer, :null => false

      # user timestamps
      t.column :last_login, :datetime
      t.timestamps
    end
  end

  def self.down
    drop_table :users
  end
end
```

Listing: 002_create_links.rb

```
class CreateLinks < ActiveRecord::Migration
  def self.up
    create_table :links do |t|
      # link information
      t.column :title,        :string,  :null => false
      t.column :link,         :string,  :null => false
      t.column :description,  :text,    :null => false
      t.column :permalink,    :string,  :null => false

      #link counts
      t.column :views_count,    :integer, :default => 0
      t.column :ratings_count,  :integer, :default => 0

      # link ratings
      t.column :rating_total,   :decimal,  :default => 0
      t.column :rating_average, :deciaml,  :default => 0

      # counter caches
      t.column :votes_count,    :integer, :default => 0
      t.column :comments_count, :integer, :default => 0

      # link relations
      t.column :user_id,        :integer, :null => false
      t.column :type_id,        :integer, :null => false
      t.column :category_id,    :integer, :null => false
      t.column :subcategory_id, :integer, :null => false

      # link timestamps
      t.timestamps
    end
  end
  def self.down
    drop_table :links
  end
end
```

Listing: 003_create_votes.rb

```
class CreateVotes < ActiveRecord::Migration
  def self.up
    create_table :votes do |t|

      # optional rating
      t.column :rating,   :decimal,   :null => true, :precision => 3,  :scale => 1

      # vote relations
      t.column :story_id, :integer,   :null => false
      t.column :user_id,  :integer,   :null => false

      # vote timestamps
      t.timestamps
    end
  end

  def self.down
    drop_table :votes
  end
end
```

Listing: 004_create_categories.rb

```
class CreateCategories < ActiveRecord::Migration
  def self.up
    create_table :categories do |t|

      # category title
      t.column :title,    :string,  :null => false, :limit => 24
      # category type
      t.column :type_id,  :integer, :null => false
    end
  end

  def self.down
    drop_table :categories
  end
end
```

Listing: 005_create_subcategories.rb

```
class CreateSubcategories < ActiveRecord::Migration
  def self.up
    create_table :subcategories do |t|

       # subcategory title
      t.column :title,        :string,  :null => false,   :limit => 24
       # parent category
      t.column :category_id,  :integer, :null => false
    end
  end

  def self.down
    drop_table :subcategories
  end
end
```

Listing: 006_create_types.rb

```
class CreateTypes < ActiveRecord::Migration
  def self.up
    create_table :types do |t|

      # type title
      t.column :title,  :string,  :null => false, :limit => 8
    end
```

```
  end

  def self.down
    drop_table :types
  end
end
```

Listing: 007_create_comments.rb

```
class CreateComments < ActiveRecord::Migration
  def self.up
    create_table :comments do |t|

       # comment text
      t.column :text,     :text,    :null => false
       # comment relations
      t.column :user_id,  :integer, :null => false
      t.column :story_id, :integer, :null => false

      t.timestamps
    end
  end

  def self.down
    drop_table :comments
  end
end
```

Listing: schema.rb

```
ActiveRecord::Schema.define(:version => 7) do
  create_table "categories", :force => true do |t|
    t.string  "title",   :limit => 24, :default => "", :null => false
    t.integer "type_id",                              :null => false
  end

  create_table "comments", :force => true do |t|
    t.text     "text",       :default => "", :null => false
    t.integer  "user_id",                    :null => false
    t.integer  "story_id",                   :null => false
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "links", :force => true do |t|
    t.string   "title",                     :default => "", :null => false
    t.string   "link",                      :default => "", :null => false
    t.text     "description",               :default => "", :null => false
    t.string   "type",        :limit => 8,  :default => "", :null => false
    t.string   "category",    :limit => 24, :default => "", :null => false
    t.string   "subcategory", :limit => 24, :default => "", :null => false
    t.integer  "user_id",                                   :null => false
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "subcategories", :force => true do |t|
    t.string  "title",       :limit => 24, :default => "", :null => false
    t.integer "category_id",                              :null => false
  end

  create_table "types", :force => true do |t|
    t.string "title", :limit => 8, :default => "", :null => false
  end

  create_table "users", :force => true do |t|
    t.string   "username",    :limit => 24, :default => "", :null => false
    t.string   "password",    :limit => 32, :default => "", :null => false
    t.string   "email",       :limit => 64, :default => "", :null => false
    t.string   "firstname",   :limit => 32, :default => "", :null => false
    t.string   "lastname",    :limit => 32, :default => "", :null => false
    t.string   "gender",      :limit => 6,  :default => "", :null => false
```

```
    t.date     "birthdate",                                  :null => false
    t.string   "city",       :limit => 32, :default => "", :null => false
    t.string   "country",    :limit => 32, :default => "", :null => false
    t.text     "about"
    t.string   "website"
    t.string   "blog"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "votes", :force => true do |t|
    t.decimal  "rating",     :precision => 3, :scale => 1
    t.integer  "story_id",                                 :null => false
    t.integer  "user_id",                                  :null => false
    t.datetime "created_at"
    t.datetime "updated_at"
  end

end
```

3.5 Models, Relations und ActiveRecords

In erster Linie stellen die Models aus dem MVC Pattern die primären „Entities" einer Applikation dar. Ein Model besteht dabei nicht nur aus simplen Daten, sondern beinhaltet auch noch die Beziehungen zu anderen Models und implementiert zusätzlich fundamentale Business-Logik. In unserem Fall besteht also ein Model aus den Applikationsdaten in der Datenbank, den Relationen zu den anderen Ruby Klassen und einfacher Business-Logik wie beispielweise der Validierung der eigenen Daten.

In Rails werden alle diese Anforderungen an ein Model durch die Klasse „ActiveRecord" erfüllt. Man erstellt also in Rails ein Model indem man einfach eine neue Ruby Klasse erstellt, die man von der „ActiveRecord" Klasse (ActiveRecord::Base) ableitet. Die „ActiveRecord" Klassen sind die mächtigsten im ganzen Rails Framework und bieten eine Vielzahl von Funktionen, die man am besten in der Rails API Dokumentation nachschlagen sollte. Das Prinzip „ActiveRecord" wurde jedoch nicht von den Rails Entwicklern erfunden. Es ist ein altbekanntes Software Design Pattern, das gerade in Enterprise Applikationen und verteilten Systemen sehr gerne eingesetzt wird. Eines der Hauptziele dieses Patterns ist die Entkopplung der Daten von der eigentlichen Datenquelle und die Bereitstellung dieser Daten als Klassen in der jeweiligen Programmiersprache. So kann ein Programmierer bzw. die Applikation auf Daten zugreifen, ohne sich überhaupt darum kümmern zu müssen von welcher Quelle sie eigentlich kommen und wie sie gespeichert werden. Dies wird gerade SQL scheuen Entwicklern gelegen kommen und ihnen Rails umso sympathischer machen.

LinkBay Models in RubyOnRails

In diesem Abschnitt wollen wir auf die Rails Models eingehen, die für die LinkBay Applikation benötigt werden. Zunächst aber geben wir eine kleine Übersicht über die Primärdaten der LinkBay Applikation ohne zunächst ins Detail zu gehen bzw. den Aufbau dieser Daten aufzuzeigen. Die Namensgebung sollte ziemlich selbsterklärend sein.

Models

- User

- Link
- Vote
- Comment
- Type
- Category
- Subcategory

Wir haben bei der Implementierung dieser Models einige Funktionen der „ActiveRecord::Base" Klasse verwendet, bei weitem jedoch nicht ihr volles Potential ausgeschöpft. Neben der reinen Datenhaltung haben wir natürlich auch gleich ihre Relationen in den einzelnen Models implementiert. Dies macht eine spätere Referenz in den Controllern umso einfacher. Darüber hinaus haben wir auch das Prinzip der Validierung angewandt, um zu verhindern, dass fehlerhafte Daten oder sogar unvollständige Datensätze in die Datenbank geschrieben werden. Ein weiteres, interessantes Hilfsmittel, das wir verwendet haben, sind die sogenannten „Filter". Dies sind im Grunde genommen nur weitere Instanz-Methoden des Models, die durch bestimmte Ereignisse ausgelöst werden. Man kann zum einen bestimmten, wann („before" oder „after") diese Methoden ausgeführt werden und zum anderen durch welches Ereignis sie ausgelöst werden. Diese Ereignisse sind die typischen CRUD-Aktionen („create", „read", „update" und „delete") jeder Datenbank. Mögliche Filter wären somit zum Beispiel „before_create" oder „after_delete".

So, nun aber zu den eigentlichen Models. Zuerst stellen wir eine allgemeine Definition des jeweiligen Models auf, um direkt danach die Implementierung in Ruby bzw. Rails aufzuzeigen. Fangen wir an:

User

Daten	Relationen	Helpers
username	one-to-many Links	validation
email_address	one-to-many Votes	latest_links
password (encrypted)	one-to-many Comments	latest_votes
firstname	one-to-many links_voted_on	latest_comments
lastname	one-to-many commented_links	
gender	many-to-one City	
birthdate	many-to-one Country	
city		
country		
about		
website		
blog		
last_login		
created_at		
updated_at		

Listing: user.rb

```
class User < ActiveRecord::Base

  # filters

  # validations
  validates_presence_of :username, :password, :email, :firstname, :lastname,
       :gender, :birthdate, :city, :country
  validates_uniqueness_of :username, :case_sensitive => false
  validates_uniqueness_of :email   , :case_sensitive => false

  # user relations
  has_many :links
  has_many :votes
  has_many :comments

  has_many :links_voted_on,
    :through => :votes,
    :source => :link

  has_many :commented_links,
    :through => :comments,
    :source => :link

  # instance methods
  def latest_links
    links.find(:all, :order => 'id DESC', :limit => 5)
  end

  def latest_votes
    votes.find(:all, :order => 'id DESC', :limit => 5)
  end

  def latest_comments
    comments.find(:all, :order => 'id DESC', :limit => 5)
  end
end
```

Link

Daten	Relationen	Helpers
title	many-to-one User	validation
link	many-to-one Type	generate_permalink
description	many-to-one Category	latest_votes
permalink	many-to-one Subcategory	latest_comments
views_count	one-to-many Votes	
ratings_count	one-to-many Comments	
rating_total	one-to-many	
rating_average	one-to-many	
votes_count		
comments_count		
created_at		
updated_at		

Listing: link.rb

```
class Link < ActiveRecord::Base

  # filters
  before_create :generate_permalink
```

```
  # validations
  validates_presence_of :title, :link, :description

  # link relations
  belongs_to  :user
  belongs_to  :type
  belongs_to  :category
  belongs_to  :subcategory

  has_many    :votes
  has_many    :comments

  has_many    :users_voted_on_link,
    :through => :votes,
    :source  => :user

  has_many    :users_commented_link,
    :through => :comments,
    :source  => :user

  # instance methods
  def latest_votes
    votes.find(:all, :order => 'id DESC', :limit => 5)
  end

  def latest_comments
    comments.find(:all, :order => 'id DESC', :limit => 5)
  end

  protected
    def generate_permalink
      self.permalink = self.name.downcase.gsub(/\W/, '-')
    end
end
```

Vote

Daten	Relationen	Helpers
rating	many-to-one Story	update_link_rating
created_at	many-to-one User	counter_cache
updated_at		

Listing: vote.rb

```
class Vote < ActiveRecord::Base

  # filters
  after_create :update_link_rating

  # vote relations & counter caches
  belongs_to :link, :counter_cache => true
  belongs_to :user

  # instance methods
  protected
    def update_link_rating
      if self.rating != 0
        link.rating_total += self.rating
        link.ratings_count += 1
        link.rating_average = link.rating_total / link.ratings_count
      end
    end
end
```

Comment

Daten	Relationen	Helpers
text	many-to-one Story	counter_cache
created_at	many-to-one User	

updated_at		

Listing: comment.rb

```
class Comment < ActiveRecord::Base

  # comment relations & counter caches
  belongs_to :link, :counter_cache => true
  belongs_to :user

end
```

Category

Daten	Relationen	Helpers
title	many-to-one Type	

Listing: category.rb

```
class Category < ActiveRecord::Base

  # category relations
  belongs_to :type

end
```

Subcategory

Daten	Relationen	Helpers
title	many-to-one Category	

Listing: subcategory.rb

```
class Subcategory < ActiveRecord::Base

  # subcategory relations
  belongs_to :category

end
```

Type

Daten	Relationen	Helpers
title	many-to-one Category	

Listing: type.rb

```
class Type < ActiveRecord::Base
end
```

3.6 Controllers, Actions und Helper Functions

Die Controller im MVC Pattern stellen die Applikations- und komplexere Business-Logik eines Models dar. Deshalb wird ein Controller in der Regel auch exakt nach dem Model benannt, für das er die Logik implementiert. Es können jedoch auch Controller programmiert werden, deren Appli-

kations-Logik völlig unabhängig von den Models ist. Diese werden im Rails Umfeld auch „Helpers" genannt und werden gerne in Modulen (entspricht einer kleinen Bibliothek) ausgelagert. Diese Module können problemlos wiederverwendet werden und entsprechen somit dem „DRY" Prinzip..

Jeder Controller besitzt sogenannte „Actions", welche die eigentliche Logik implementieren. Es gibt eine Reihe von „Default Actions", die normalerweise für jedes Model implementiert werden. Dieses entsprechen den im vorherigen Abschnitt erwähnten CRUD-Aktionen („create", „read", „update" und „delete") der Datenbank und sind relativ simpel. Darüber hinaus müssen dann eigene „Actions" entworfen und implementiert werden, die alle Funktionen einer Applikation abdecken. Beispielsweise werden häufig „login" und „logout" Actions im User-Contoller implementiert, da viele Applikationen nicht ohne eine User-Authentifikation auskommen. In einer Online-Banking Applikation wären die Actions „deposit" und „withdraw" im Account-Controller wohl unverzichtbar..

In Rails werden Controller durch Klassen dargestellt, die direkt vom „ApplicationController" der eigenen Applikation abgeleitet werden. Schaut man sich den „ApplicationController" an, sieht man dass dieser eigentlich von der „ActionController::Base" Klasse abgeleitet wird. Die „Actions" die im „ApplicationController" implementiert werden, stehen also jedem Controller zur Verfügung, sind somit logischerweise nicht an ein bestimmtes Model gebunden und stellen immer sehr allgemeine bzw. oft verwendete Funktionen dar. Diese Actions nennt man auch gerne „Helper Actions" bzw. „Helper Functions". Doch wie definiert man nun Actions in Rails? Diese sind nichts weiter als Instanz-Methoden der Controller-Klasse und funktionieren auch auf dieselbe Art und Weise. Jede Action hat ihre Signatur, die sich aus ihrem Namen, deren Parametern und ihrem Rückgabewert zusammensetzt. Jeder Programmierer der schon einmal objektorientiert programmiert hat, sollte sich damit innerhalb von Minuten vertraut machen und danach sofort loslegen können.

LinkBay Actions und Helper Functions
Für unsere LinkBay Alpha Applikation benötigen wir neben den typischen CRUD-Actions natürlich noch um einiges mehr. Da die CRUD-Actions meist recht simple und eintönig sind, verzichten wir hier darauf, näher ins Detail zu gehen. Um die Grundfunktionalität von LinkBay zu implementieren, sind folgende Elemente besonders wichtig:

- User Management (login, logout, logged_in?, login_required, fetch_logged_in_user)
- Link Submission and Voting (new, show, vote)
- Link Listing Function (index, list_links, fetch_links)

Auf diese Elemente werden wir in den nächsten Abschnitten genauer eingehen.

User Management
Da Teile des User-Managements bei fast allen Funktionen auf LinkBay gebraucht werden, reicht es nicht aus, diese nur im „User-Controller" zu implementieren. Deshalb haben wir die Funktionalität auf mehrere Controller aufgeteilt. Die Logik zum An- und Abmelden eines Benutzers bei LinkBay haben wir im „Account-Controller" implementiert, da diese recht selten benützt wird. Im Idealfall sollte ein Mitglied bei jedem Besuch von LinkBay sich nur einmal an- und dann wieder abmelden.

Einige Funktionen von LinkBay sind nur angemeldeten Mitgliedern vorbehalten. Falls so eine Funktion wie das Veröffentlichen eines Links von einem Besucher ausgeführt wird, müssen wir natürlich prüfen, ob dieser Besucher ein Mitglied ist und sich angemeldet hat. Da diese Überprüfung sehr häufig stattfinden muss, haben wir die sogenannten „Helper-Functions", die das für uns erledigen, zentral und somit allgemein verfügbar im „Application-Controller" implementiert. Wie das Ganze letztendlich aussieht, sieht man in den nachfolgenden Listings:

Listing: account_controller.rb

```
class AccountController < ApplicationController

  def login
    if request.post?
      @current_user = User.find_by_login_and_password(
        params[:login], params[:password])
      unless @current_user.nil?
        session[:user_id] = @current_user.id
        unless session[:return_to].blank?
          redirect_to session[:return_to]
          session[:return_to] = nil
        else
          redirect_to :controller => 'story'
        end
      end
    end
  end

  def logout
    session[:user_id] = @current_user = nil
    redirect_to :controller => 'story'
  end

  def show
    @user = User.find_by_login(params[:id])
    @links_submitted = @user.links.find(:all,
        :limit => 6, :order => 'id DESC')
    @links_voted_on  = @user.links_voted_on.find(:all,
        :limit => 6, :order => 'id DESC')
  end
end
```

Listing: application.rb

```
class ApplicationController < ActionController::Base
  before_filter :fetch_logged_in_user

  protected
  def fetch_logged_in_user
    return if session[:user_id].blank?
    @current_user = User.find_by_id(session[:user_id])
  end

  def logged_in?
    ! @current_user.blank?
  end
  helper_method :logged_in?

  def login_required
    return true if logged_in?
    session[:return_to] = request.request_uri
```

```
    redirect_to :controller => "/account", :action => "login"
      and return false
  end
end
```

Link Submission, Voting & Listing

Dies ist die Grundfunktionalität und somit das Herz unserer LinkBay Applikation. Besucher sollten natürlich in der Lage sein, sich veröffentlichte Links nach verschiedenen Sortierungsmustern anzuschauen. Angemeldete Mitglieder müssen Links veröffentlichen und für sie abstimmen können. Die ganze Logik zu diesen Funktionen haben wir im sogenannten „Link-Controller" implementiert, auf den wir jetzt genauer eingehen werden.

Um sicherzustellen dass nur angemeldete Mitglieder Links veröffentlichen und für diese abstimmen können, haben wir einen einfachen „before" Filter verwendet. Dieser prüft durch die im „Application-Controller" implementierte Helper-Function „login_required" ob der Besucher angemeldet ist oder nicht. Beim veröffentlichen des Links wird noch ein sogenannter „Permalink" aus dem Link-Titel generiert. Durch diesen stellen wir sicher dass die URL zu diesem Link gut lesbar und aussagekräftig ist, was mit einer kryptischen ID in der URL nicht der Fall wäre. Beim Abstimmen für Links ist es wichtig zu prüfen, ob das gleiche Mitglied nicht schon für diesen Link abgestimmt hat, da sonst das massive Abstimmen für eigene Links möglich wäre. Da man aber nicht automatisch beim Abstimmen eine Wertung für einen Link geben muss, kann dieses „Rating" auch noch abgegeben werden, nach dem schon für diesen Link abgestimmt wurde. Alle Link-Listing Funktionen verwenden eine Helper-Function namens „fetch_links", der man die gewünschten Eigenschaften der anzuzeigenden Links durch „Conditions" übergeben kann. Unter diese „Conditions" fallen zum Beispiel Typ, Kategorie und Veröffentlichungszeitraum der Links. Mit Hilfe dieser Helper-Function werden die verschieden Link-Listing Funktionen wie zum Beispiel „list_most_popluar" oder „list_best_rated" implementiert.

Wie die genaue Implementierung dieser Funktionen aussieht, ist im nachfolgenden Listing zu sehen:

Listing: link_controller.rb

```
class LinkController < ApplicationController

  before_filter :login_required, :only => [ :new, :vote ]

  def index
    fetch_links 'id DESC', '', '10', ''
  end

  def show
    @link = Link.find_by_permalink(params[:permalink])
    #increment the views of the link
    @link.increment!(:views_count)
  end

  def new
```

```
    @link = Link.new(params[:link])
    @link.user = @current_user
    if request.post? and @link.save
      #@link.tag_with params[:tags] if params[:tags]
      #flash[:notice] = 'Story submission succeeded'
      redirect_to :action => 'index'
    end
  end

  def list_links

    order      = params[:order]
    conditions = params[:conditions]
    limit      = params[:limit]
    offset     = params[:offset]

    fetch_links order, conditions, limit, offset
  end

  def list_most_popular
    #type        = params[:type]
    #category    = params[:category]
    #subcategory = params[:subcategory]
    # ...
    order      = params[:order]
    conditions = params[:conditions]
    limit      = params[:limit]
    offset     = params[:offset]

    fetch_links order, conditions, limit, offset


  end

  def vote
    @link = Link.find(params[:id])
    # alternativ: @link = Link.find(params[:permalink])

    # check if user has already voted on link
    unless @link.votes.exists?(:user =>@current_user)
      @link.votes.create(:user => @current_user, :rating => params[:rating])

      # calculate link rating
      if params[:rating] =! 0
        # möglich so auf attribute zuzugreifen????
        @link.rating_total += params[:rating]
        @link.increment!(:ratings_count)
        @link.rating_average = @link.rating_total / @link.ratings_count
      end
  end

  protected
    def fetch_links(order, conditions, limit, offset)
      @links = Link.find :all,
        :order      => order,
        :conditions => conditions,
        :limit      => limit,
        :offset     => offset
    end
end
```

3.7 Fazit zu RubyOnRails

Nach fast einem halben Jahr Rails-Entwicklung ist es nun Zeit ein Resümee zu ziehen. Dabei sind sowohl positive als auch negative Aspekte zu erwähnen. Wobei keine der beiden Seiten deutlich überwiegt. Fangen wir mit den positiven Aspekten an.

Was gleich zu Anfang auffällt ist, dass in Rails ein regelrechter „Kickstart" möglich ist. Wenn man ein wenig programmieren und das MVC Prinzip nachvollziehen kann, dann ist es möglich in kürzester Zeit und mit sehr wenig Quelltext eine kleine, recht ordentliche Web Applikation zu schreiben. Wir müssen schon sagen, dass dies gerade am Anfang schon sehr beflügelnd ist. In vielen Programmiersprachen und Frameworks ist der Einstieg schwer, was viele Leute schon zu beginn abschreckt. Nicht mit RubyOnRails. Deshalb hört man von fast allen Leuten soviel gutes über Rails. Doch die meisten Leute nutzen dieses Framework nicht professionell und kennen sich nicht genügend aus, um ein wirklich qualifiziertes Urteil abzugeben.

Wenn man Rails etwas genauer evaluiert, merkt man dass es wirklich gut für kleine Web Applikationen geeignet ist, die möglichst schnell und ohne viel Aufwand entwickelt werden sollen. Diese Anforderungen richten sich vor allem an private Entwickler, die schnell eine Web Applikation programmieren wollen. Diese privaten Entwickler kommen meist aus dem PHP Umfeld und sind keine professionellen Programmierer. Im schlimmsten Fall können sich nicht einmal programmieren, sondern „skripten" ihre PHP Webauftritte nur. Gerade für diese Entwickler machen die Rails Prinzipien wie „Conventions over Configuration", „DRY" und „MVC" Sinn. Da sie diese meist davor gar nicht kannten und somit von dem „Rails Weg" guten Programmierstil erlernen können und sich nur mit sehr wenig Applikations-Design beschäftigen müssen. Deshalb wird das Rails Framework in diesen Kreisen auch so gelobt (ich möchte schon fast sagen „gehypt") und wird sich dort auch weiterhin großer Beliebtheit erfreuen. Dadurch wird die Community der Rails Entwickler, von denen sich auch viele „Rails Fans" nennen, weiterhin wachsen und Rails sich immer weiter verbreiten.

Wenn man das Ganze aus professioneller Sicht betrachtet, dann sind hier leider hauptsächlich negative Punkte zu nennen. Sobald man den „Kickstart" und somit die erste Begeisterung für Rails hinter sich hat, geht es eigentlich nur noch bergab. Nicht weil Rails kein gutes Web-Entwicklungs-Framework ist, sondern weil es einfach nicht den großen Erwartungen, die durch den „Rails Hype", die „Rails Fans" und den wirklich guten Start entstehen, gerecht werden kann. Dies liegt vor allem daran, dass Rails doch einige Sachen anders angeht als man in der konventionellen Web Entwicklung gewöhnt ist. Zudem schränkt es doch gerade die professionellen Entwickler, die wissen was sie tun und die eigene Vorstellungen vom Applikations-Design haben, sehr ein. Man könnte sagen, dass es den konventionellen Entwicklungsweg gibt, der schon lange praktiziert wird und den diese Entwickler alle regelmäßig beschritten haben. Zudem gibt es nun den „Rails Weg" den man einfach neu erlernen muss und für den es keine Alternative gibt. Ergo sind die Freiheiten der Entwickler doch sehr eingeschränkt und sie müssen sich einfach an den „Rails Weg" gewöhnen und sich ihm unterordnen.

Gerade hier treten dann die meisten Probleme auf. Denn falls eine Web Applikation komplexer wird oder in eine bestehende Umgebung integriert werden soll, dann sind meistens komplexe bzw. individuelle Problemlösungen und Ansätze gefragt. Leider werden diese individuellen Problemlösungen oft nicht nur von Rails einfach nicht unterstützt, sondern ihre Umsetzung in Rails ist schlicht und einfach unmöglich. Da dieses Phänomen mittlerweile sehr oft auftritt und immer mehr Entwickler damit Erfahrung gemacht haben, gibt es für diesen Aspekt von Rails auch schon einen eigenen Namen. So folgt Rails dem „hard to tame" Prinzip, was soviel heißt, dass Rails „schwer zu bändigen" ist, wenn es um individuelle Lösungsansätze geht. Gerade in größeren und komplexeren Projekten, die aufgrund des enormen „Rails Hype" relativ früh auf Rails umgestiegen sind, machte sich dies bemerkbar. Viele der verantwortlichen Firmen haben sich sogar die RubyOnRails Framework Entwickler ins Boot geholt, um das Framework so zu verändern, dass es den Anforderungen dieser Projekte gerecht wird. Dies ist offensichtlich nicht Sinn und Zweck eines Frameworks, wenn es erst verändert werden muss, damit die Umsetzung eines Projekts möglich wird.

Gerade deshalb sind mittlerweile viele Entwickler und Firmen schon wieder von dem „Rails Zug" abgesprungen, nachdem sie durch den vorschnellen Wechsel auf Rails viel Geld bzw. Ressourcen verschwendet haben. Doch einige sehen ihren Ausflug in die Rails Welt nicht als völlige Zeit- und Geldverschwendung an, da man durch die Prinzipen und Architektur von Rails doch sehr viel für die zukünftige Web Entwicklung lernen konnte. Ein Beispiel hierfür ist www.cdbaby.com – dieser Webauftritt bestand vor dem Wechsel auf Rails aus weit mehr als 100.000 Zeilen PHP Code. Nach fast zwei Jahren Rails Entwicklung und der Anpassung des Rails Framework durch zwei Entwickler des Rails Teams, wurde die Umstellung auf Rails aufgegeben. Mit der Rails Architektur und den Prinzipien im Hinterkopf wurde dieser Webauftritt im Zeitraum von nur 2 Monaten in PHP komplett neu geschrieben und umfasst nun nur 12.000 Zeilen Code.

Ein weiterer Negativaspekt ist die heutzutage vorhandene Literatur und Dokumentation von Rails. Die API Dokumentation kann man leider nur als sehr spärlich und unvollständig beschreiben. Nachdem ich nun fast vier Rails Bücher gelesen und mir weit mehr als 15 angeschaut habe, muss ich sagen, dass eines schlechter als das andere ist. Man hat den Anschein, dass es vielleicht drei oder vier Originalbücher gibt, die schon nicht besonders gut sind. Der Rest der Bücher, zumindest die ich mir angeschaut habe, haben mehr oder minder den Inhalt der schon schlechten Originalbücher nur kopiert und ein wenig anders formuliert bzw. ihre Beispiele ein wenig abgeändert. So muss man auch hier vorsichtig sein, worauf man sich einlässt. Wenn man das Geld hat, dann sollte man am besten mehrere Bücher ausleihen bzw. kaufen und aus allen das Beste herausziehen. Sonst kommt man bei der Rails-Entwicklung nicht sonderlich weit. Wir hoffen, das wird sich in Zukunft ändern, indem sich richtige Autoren Rails widmen und gute, strukturierte und vor allem vollständige Bücher auf den Markt bringen. Zum jetzigen Zeitpunkt können wir leider nur auf die unzähligen Ruby bzw. Rails Foren verweisen. In diesen muss man zwar umständlich nach dem suchen was man wissen will, jedoch ist das Wissensangebot hier noch um einiges besser als in jedem Buch das ich gesehen habe und zudem ist es kostenlos.

Abschließend ist zu sagen, dass Rails nicht das Allheilmittel für alle Probleme ist, ganz egal was manche „Rails Fans" auch behaupten. Gerade wenn man mit wenig Aufwand kleine Web Applikationen bzw. Prototypen schreiben möchte, dann sollte man sich Rails zumindest einmal anschauen. Soll Rails in einem professionellen Umfeld eingesetzt werden, muss jeder die Stärken und Schwächen von Rails für sich selber abwägen und entscheiden, ob sich der Umstieg auf RubyOnRails wirklich lohnt.

4 Adobe's Flex 3 (Public Beta)

4.1 Allgemeines zu Flex

Flex ist ein Developer-Framework von Adobe Systems zum Erstellen von Rich Internet Applikationen. Das Flex-Framework besteht aus dem Flex-Builder, dem SDK, einem MXML und ActionScript 3.0-Compiler, den LiveCycle-Dataservices, den Flex-Charting-Komponenten, CSS-Unterstützung und der Flex-Klassen- und Komponentenbibliothek. Bis auf die Charting-Komponenten ist Flex von Version 3 an ein „Open Source"-Framework und befindet sich derzeit noch in der „Public Beta"-Phase. Zur Umsetzung des LinkBay-Frontends nutzten wir (ausgenommen die Charting-Komponenten) nahezu alle verfügbaren Features der Entwicklungsumgebung, testeten Schritt für Schritt alle implementierten Komponenten und erweiterten diese um eigene. Schließlich nutzen wir die bestehenden Schnittstellen der LiveCycle-Dataservices im Zusammenspiel mit RubyAMF (siehe nächstes Kapitel) für eine Kommunikation mit dem RubyOnRails-Backend.

Ziel jeder Flex-Entwicklung ist ein Client-seitiges Programm, das über eine der Schnittstellen der LiveCycle-Dataservices mit einem Server kommunizieren kann. In unserem Fall läuft auf dem Server eine RubyOnRails-Anwendung, während auf der Client-Seite die kompilierte Flex-Anwendung ausgeführt wird. Der User unserer Anwendung bleibt mithilfe des Flash-Players innerhalb des komfortablen und vertrauten Browsers und bekommt das Gefühl auf einer schnellen, flexiblen und effektreichen Webseite zu befinden. Die „User-Experience" ist somit ein besonderes Merkmal unserer Anwendung und brachte uns auf neue Ideen für die Usability.

Folgende Vorteile bringt eine mit Flash-/Flex entwickelte Anwendung gegenüber normalen Web-Auftritten im Internet.

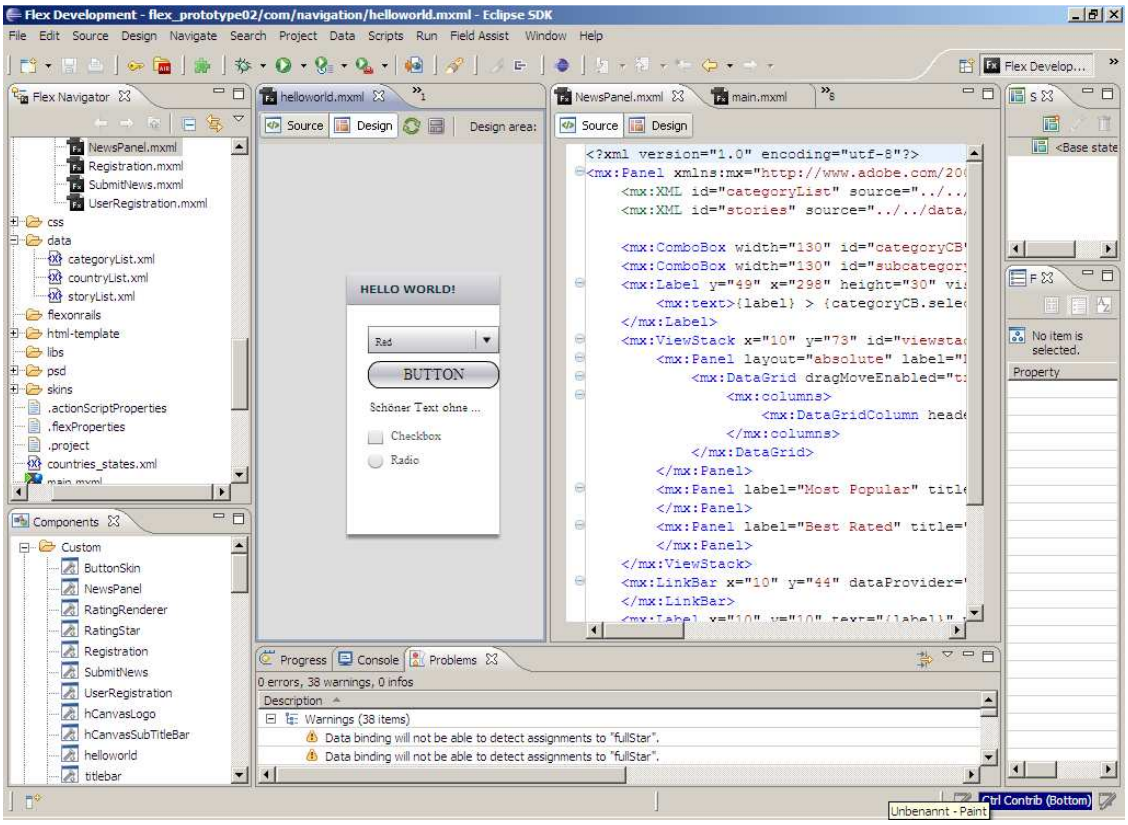
- Einmaliges Laden der Anwendung. Danach werden nur noch relevante Business-Daten (z.B. Remote-Objects per XML) übertragen. Die Netzlast sinkt.
- Darstellungsschicht (Präsentation/"Frontend") wird auf den Client ausgelagert.
- Plattformunabhängige, homogene Darstellung durch den Flash Player

Nachteile einer solchen Anwendung können sein:

- Bei großen Anwendungen muss beim erstmaligen Laden sehr viel mit übertragen werden. Programmteile (insbesondere Bilder) können zwar ausgelagert werden, doch entstehen dadurch dieselben Ladezeiten wie beim gewöhnlichen Webanwendungen. Die neue User-Experience geht verloren.
- Flash Player (9) ist Voraussetzung.
- Hohe Renderingzeiten von „einfachen" Flex-Komponenten

4.2 Flex-Builder

Der Flex-Builder bildete für uns die zentrale Entwicklungsumgebung, die in der Standalone-Version auf Eclipse 3.3.1 basiert. Auf den ersten Testumgebungen versuchten wir Flex als PlugIn in einer regulären „Classic"-Eclipse 3.3.1-Umgebung zu integrieren, scheiterten jedoch an unterschiedlichen (Versions-)Problemen im Zusammenspiel mit der RubyOnRails-Umgebung Aptana Studio 1.1.1. Also trennen wir die Entwicklungsumgebungen Flex(-Buider) und Eclipse voneinander und portieren benötigte Dateien/Workspaces je nach Bedarf. Großer Vorteil des Flex-Builders ist der Design View zur Voransicht der Anwendung ohne Kompilierung. Hier können Flex-Komponenten (eigene oder vorgegebene) per Drag'n'Drop auf die Anwendung gezogen und gegebenenfalls mit den bestehenden Datensätzen der ActionScript-Variablen direkt verknüpft werden.



Flex PlugIn für Eclipse - Abb. 4-1

4.3 MXML, ActionScript und CSS

MXML wird zur deklarativen Definition der Komponenten und Elemente auf der Benutzeroberfläche (und zum Teil sehr Entwicklerfreundlich im Design-View) eingesetzt. MXML ist eine XML-basierte, deklarative Programmiersprache und kann mit dem Flex-Builder oder mit einem Text-Editor bearbeitet werden. Wie in SQML oder HTML werden mit MXML die Eigenschaften von Komponenten beschrieben. Der Flex-Compiler übersetzt in einem Zwischenschritt die MXML-Dateien in ActionScript-Quelldateien. Ansichtszustände der MXML-beschriebenen Komponenten erleichtern für den Entwickler die Erstellung dynamischer Benutzeroberflächen und ermöglichen ein komplexes Zusammenspiel der Elemente innerhalb der Anwendung.

ActionScript ist im Gegensatz zu MXML eine imperative (befehlsorientierte) Programmiersprache und basiert zu weiten Teilen auf der Syntax von Java. ActionScript 3.0 ist eine objektorientierte Programmiersprache, die Verfahren für die rasche Erstellung von RIAs implementiert. ActionScript 3.0 bietet in Flex 3 eine hohe Performance (kürzere Kompilierungszeit, schnellere Ausführen durch den Flash Player). Die Skriptsprache beruht auf dem internationalen Standard ECMAScript und ist mit der Spezifikation für ECMAScript konform. ActionScript ist auch den Flash-Entwicklern bekannt sein und bildet einen zentralen Bestandteil jeder Flashbasierten-Webapplikation. ActionScript-Quellcode wird dazu zu einer SWF-Flashdatei kompiliert. Zum Ausführen der SWF ist der Flashplayer 9 notwendig.

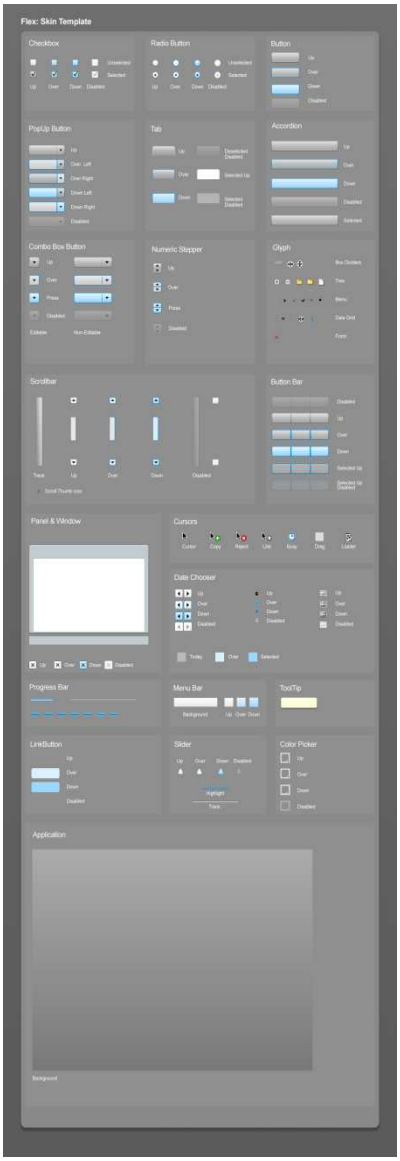
Flex unterstützt die Programmierung von CSS für die verwendeten Farben, Schriften und Beschriftungsstile. Mithilfe von CSS lassen sich sogenannte Themes entwickeln.

4.4 Graphical & Programmatic Skinning

Eine Möglichkeit Komponenten in Flex mit neuen Oberflächen (anstatt der Standard-Oberflächen) zu versehen, ist das sogenannte Graphical Skinning. Hierbei haben die Grafik-Designer die meiste Freiheiten mit ihren Grafiken (z.B. aus Photoshop) die vorgegebenen Skins der einzelnen Komponenten zu ersetzen. In Photoshop wird hier eine vordefinierte Palette grafischer Elementen durch eigene ersetzt. Der Grafiker erhält hier die Kontrolle über die Steuerelemente während sie vom Mauszeiger unberührt sind, überflogen angeklickt oder losgelassen werden. Unterschieden wird auch zwischen selektierten und unselektierten Komponenten (z.B. CheckBox) Hier zeichnet sich die große Stärke von Flex aus: Die strukturierte Implementation von selbstentworfenen Oberflächen. Hierbei wird gerne CSS oder Inline-CSS (CSS-Code innerhalb einer MXML-Datei) verwendet.

```
<mx:Style>
   Button {
      ...
      overSkin: Embed("../myButtonOverSkin.png");
      ...
   }
</mx:Style>
```

In diesem Beispiel wird für alle Buttons einer Anwendung beim MouseOver des Cursors die Bilddatei geladen. Der Nachteil dieser Methode ist, dass es trotz umfangreicher Freiheiten relativ wenige Möglichkeiten gibt, eigene Effekte zur Verbesserung der Usability zu entwerfen. Die Programmierer



Graphical Skin mit Photoshop - Abb. 4-2

und Grafiker sind gezwungen sich an die bestehenden Muster zu halten – ohne Freiheiten bei der Animierung zu bekommen.

Statt grafische Templates für die Skins der Komponenten zu benutzen, ist es möglich die API des Flash Players zum Zeichnen zu verwenden. Dafür benötigt man eine eigene Klasse, die den Skin definiert. Die Komponenten greifen auf diese Klasse zu, um die Oberfläche für sich zu zeichnen. Beim Programmatic Skinning besitzt der Programmierer die Fähigkeit auf alle Events Einfluss zu nehmen (override-Methoden). Er kann Übergänge definieren, Effekte hinzufügen, Größen verändern, usw. und dass bei wenigem Speicherplatzverbrauch und weniger Prozessorleistung (je nach Effekt unterschiedlich).

Mit Programmatic Skinning werden eigene Program-Themes definiert, die dem Programmierer alle Freiheiten bietet, die auch bei einer normalen Flash-Anwendung zu haben sind. Vorgefertigte Themes aus dem Internet wurden nicht verwendet.

4.5 Custom Components in LinkBay

Da es spezielle Elemente in der Standard-Library in Flex nicht gab, entwickelten wir für LinkBay eigene Komponenten, die wir schon zu Beginn an das Design der Applikation anpassten, während die Funktionalität nachträglich angepasst und erweitert wurde.

- RatingStar/RatingStar Renderer
  Zur Veranschaulichung wie gut ein Link bewertet wurde, bauten wir eine RatingStar-Komponente in unsere Applikation, die sich über eine Floating-Point-Variable steuern lies. Wir nahmen hierfür das bekannte 5-Sterne-System, das auf vielen Webseiten verbreitet ist als Idee, entwickelten dazu die Grafiken und schrieben einen Algorithmus, der auch Dezimalzahlen mithilfe von „halben" Sternen visualisiert. Die Rating-Star-Komponente besteht dazu aus zwei Teilen: Das eigentliches Stern-Panel, der nichts über sich weiß und ein Renderer, der fünf leere, halbe oder volle Sterne hintereinander setzt und die Komponente als Ganzes auf die Applikation bringt.

Rating Star - Abb. 4-3

- ItemRenderer des Advanced Datagrids
  Um Link-Komponenten geschickt die Datensätze aus den HTTP-Service-Schnittstellen eines Datagrids auszulesen entwickelten wir eine neue Komponente, die auf dem Datagrid basiert und erweiterten diesen um einen speziellen ItemRenderer, der die Einträge innerhalb seiner Liste befüllt und automatisch Daten in die jeweilige Zeile überträgt.

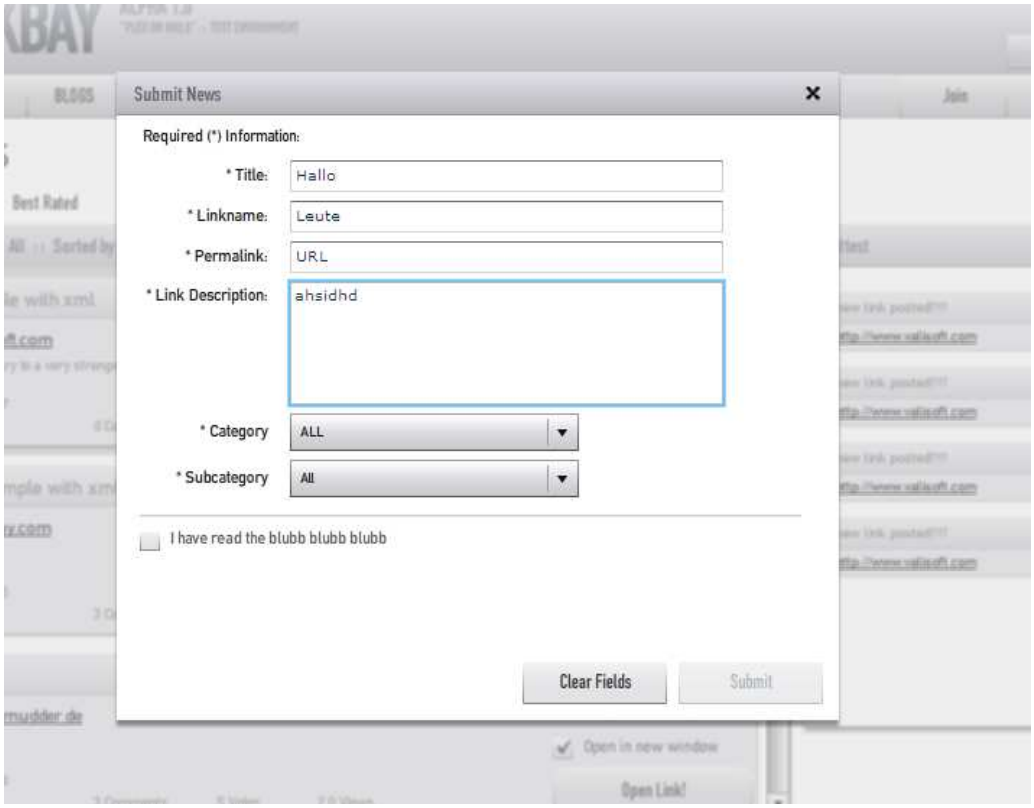
ItemRenderer für LinkBay - Abb. 4-4

Der Inhalt jeder Zeile sind Hintergrundgrafiken, Textfelder, Label, Buttons und die Rating-Star-Komponente, die in jedem „Block" unterschiedliche Daten kriegen, in ihrer Dar-

stellung aber wiederholt und gerendert werden. Der ItemRenderer verwendet für diese Datensätze ArrayCollections, die er sich dynamisch aus einer Variablen heraus lädt, sodass (in unserem Fall) die Höhe des Grids immer an die Anzahl der Items angepasst wird und die Nachrichten automatisch nach ihrer ID sortiert wurden. Zusätzliche Sortierungsverfahren (nach Datum, Rating, Klicks usw.) werden noch eingefügt.

- ToggleButton
  Zur Auswahl der Kategorien benötigten wir für LinkBay eine erweiterte Form des ToggleButtons, der nicht nur die sogenannten ViewStacks für die einzelnen Seiten generiert, sondern auch den sogenannten PopUp-Manager aufruft, damit Meldungen und Fenster erzeugt werden, falls sich der User z.B. einloggen, ausloggen oder anmelden will.

- Window Views
  Um das Gefühl des Users zu erzeugen, sich in einer performanten Anwendung und nicht auf einer einfachen Webseite zu befinden, überlegten wir uns den PopUp-Manager in Flex für ganze Fenster zu verwenden. Die bestehende Title-Windows-Komponente ersetzt die bestehende Pop-Up-Window-Definition und erzeugt ein eigenes Fenster, das ohne Programmatic Skinning (also im klassischen Flex-View) angepasst werden muss. PopUps können in Flex nur durch Programmatic Skinning realisiert werden, was für Designer unattraktiv und umständlich ist, da sie kein Design View haben, der die Komponenten im Voraus anzeigt. Mithilfe unserer Window View-Komponente können wir somit eigene Fenster erzeugen, die auch im Design View so angezeigt werden, wie sie später auch in der Applikation erscheinen.



Beispiel für ein Custom Window in LinkBay - Abb. 4-5

- Buttons
  Um zusätzlich Effekte für den „OpenLink"-Button zu verwenden, mussten wir die Standard-Implementationen des normalen Buttons erweitern. Um weiche, langsame Übergänge zwischen den „States" eines Buttons anzuzeigen, nutzen wir den Fading- und Drop-Shadow Filter-Effekt in Flex und erweiterten dadurch die Properties der bestehenden Button-Klasse.

4.6 Fonts, Preloader, Validierung

Da es in Flex die Möglichkeit gibt eigene Schriftarten in eine Anwendung einzubetten, nutzten wir dieses Feature, um die gesamte „Deutsche Mittelschrift"-Familie in die Anwendung zu integrieren. Beispiel:

```
...
[Embed(systemFont="DIN 1451 Engschrift", fontName"DIN1451Eng")]
...
```

Preloader
Der Standard-Preloader in Flex wurde durch einen eigenen Lade-Balken und Intro-Text ersetzt. Hierfür wurde eine Preloader- -Klasse und eine Preloader-Ladebalken-Klasse erzeugt, die gleich zu Beginn der Anwendung geladen werden. Ist die Anwendung fertig heruntergeladen, blendet der Ladebalken aus und die Anwendung erscheint.

Validierung
Eine wichtige Funktion der Client-Seite, die bei normalen Webanwendungen oft der Server machen muss, ist die Validierung eingegebener Formulardaten. Bei der Anmeldung, Registrierung oder Übermittlung eines neuen Links werden eingegebene Formulardaten auf ihre Richtigkeit hin überprüft. Hier mussten wir die Spezifikationen der Variablen auf der Rails-Seite (z.B. die maximale Länge eines Strings) in die Validierung auf der Flex-Seite übertragen werden. Beispiel:

```
...
<mx:StringValidator
     id="nameValidator"
     source="{nameInput}"
     property="text"
     minLength="4"
     maxLength="24"
/>
...
```

Inkorrekte oder unvollständige Angaben in den Formularfeldern mussten mit einer Fehlermeldung ausgegeben werden. Flex bietet hierbei eine Flex-seitige Notification-Funktion für Textfelder, die wir unverändert gelassen haben, da sie dem User alle fehlerhaften Eingaben korrekt und in vollständigen Sätzen mitteilt.

4.7 Fazit zu Flex

Die Arbeit mit Flex war anfangs recht einfach. Um eigene Komponenten zu entwerfen, Effekte zu nutzen und das Design anzupassen war Flex ideal. Doch umständliche, unnötige Arbeit mit States, sowie die komplizierte Flex-Klassenbibliothek, ließen uns jedoch bald zu dem Schluss kommen, das Flex für Grafik-Designer (selbst mit Programmierkenntnissen) noch gänzlich ungeeignet ist. Die Arbeitsweise von Flex mit StyleSheets war mangelhaft, da sich Klassen nicht trennen ließen. Flex erkannte nur die Standard-Klassen seiner Komponenten und überließ es dem Anwender weder vererbte Hierarchien, noch eigene Elemente ein anderes CSS-StyleSheet zu delegieren. Darüber hinaus verhielt sich der Design-View bei einigen Komponenten anders, als nach der Kompilierung.

Um komplexe und effektreiche Anwendungen zu programmieren ist Flex nicht geeignet. Wir haben uns Beispiele angesehen, die ebenfalls mit Flex realisiert wurden und erkannten, dass die Technologie nur bis einem bestimmten Grad und Umfang nützlich, erfolgreich und auch verbreitet ist (siehe Video-Online-Systeme, PhotoViewer und Office-Pendants). Sobald eine komplexe Webarchitektur verlangt wird, die auch noch mit einer anderen Technologie (in unserem Fall Rails) kommuniziert ist ungeheurer Aufwand nötig. Die mitgelieferten Libraries und Funktionalitäten versagen hier. Z.B. Gibt es in Flex keinen Sessionhandler, keine kryptographischen Tools für Passwortschutz und wenig vorgefertigten Schnittstellen zur Kommunikation (in unserem Fall nur den RPC).

Einige Kinderkrankheiten und fehlerhafte (nicht vorhandene) Steuerelemente und Komponenten, brachten uns zu dem Schluss, dass Flex nicht für jedermann geeignet ist. Bestehende Strukturen in Flex „aufzudröseln" erfordert sehr viel Einarbeitungszeit. Besonders die Arbeit mit ActionScript war zu Beginn sehr aufwendig. Libraries aus dem Internet (Caingorn, funFX), waren schwer zu implementieren und unnötig ressourcenfressend. Externe Libraries brachten weniger Mehrwert in einer Anwendung, als die Komponenten (passend für die Applikation) selbst zu programmieren. Hierfür wird jedoch sehr viel Zeit benötigt.

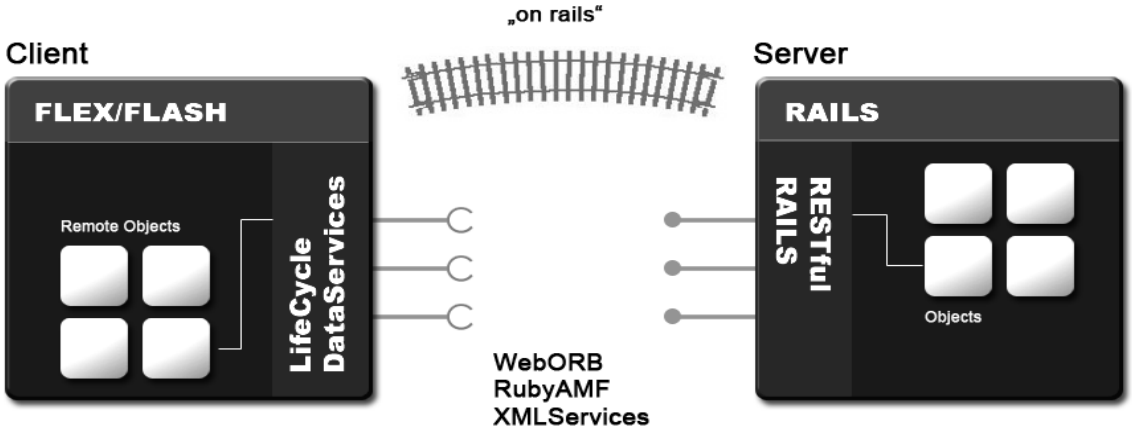
Sehr positiv zu bemerken ist die Online-Hilfe „livedocs", die im Laufe unseres Projekts von Adobe online gestellt wurde. Diese sehr umfangreiche Dokumentation von Flex 3 war quasi ein Segen und brachte uns auf viele Lösungen. Wir erkannten, dass eine solch umfangreiche Online-Hilfe bei normalen OpenSource-Projekten nicht der Fall ist (vergleiche Rails). Vermutlich gibt es hier den Ausschlag, dass keine Gemeinde, sondern ein großer Konzern hinter dem Flex-Framework steht, was auch dafür spricht, das das Nischendasein von Flex nicht von Dauer sein muss. Wir glauben, dass Flex aufgrund seiner höchst interessanten Ansätze (seit Version 3) zu einem verbreiteten Web-Framework werden könnte. Voraussetzung hierfür ist, dass das „Customizing" (z.B. die schnellere Entwicklung eigener Komponenten und Vereinfachung des Programmatic Skinning mit CSS) der Anwendung in den Mittelpunkt gerückt wird, sodass einfacher und schneller eigene Komponenten entwickelt werden können.

5 Integration mit RubyAMF

5.1 WebORB versus RubyAMF

Zur Kommunikation zwischen Client (Flex-Seite) und Server (Rails-Seite) gibt es derzeit zwei Schnittstellen, die mit einem Remoting Gateway für Rails arbeiten. Das derzeit verbreitetste, freie RPC und Flash Remoting-System nennt sich WebORB und unterliegt für nicht-kommerzielle Projekte der GPL. Es unterstützt die Adobe Message Protokolle AMF0 (Flex-Client-Protkoll) und AMF3 (Flex-Remoting-Client-Protokoll) und wird in Rails-Projekte als PlugIn hinzugefügt. Rails interpretiert so die Remoting-Methods aus Flex. Als zweites existiert eine andere Schnittstellentechnologie namens RubyAMF. Während des Projekts sind wir von WebORB auf RubyAMF umgestiegen, da wir uns bei der Konfiguration und Implementation (insbesondere bei Änderungen auf der Rails-Seite) sehr viel leichter taten. RubyAMF ist um einiges performanter als WebORB und bietet etwas mehr Freiheiten bei der Konfiguration. RubyAMF und WebORB unterscheiden sich Flex-seitig nicht in ihrer Funktionsweise. Bis auf die Definition der Schnittstellen sieht die Implementation auf der Flex-Seite bei beiden gleich aus.

Für den Prototyp und zum lokalen Testen der Flex-Anwendung verwendeten wir die Standard XML-Services, die XML-Dateien, anstatt realer Remote Objects als „Platzhalter" benutzen. Des ermöglichte uns unabhängig voneinander zu programmieren und Design-Elemente vorzeitig anzupassen. Theoretisch wäre natürlich auch eine Implementation für LinkBay nur mit XML-Services möglich gewesen. Diese unterstützen jedoch keine Remote Objects und sind nicht wirklich performant. Einige statische Elemente (Menüs, Kategorien) werden jedoch immer noch mit XML-Dateien erzeugt, da wir die Kommunikation zu Beginn auf dynamische Elemente beschränken wollten. Einige Ideen aus den Platzhaltern der XML-Definitionen flossen auch in die Programmierung auf der Rails-Seite. Wir haben eine Grafik zur Funktionsweise der Schnittstellen erstellt:



Schaubilder der Funktionsweise von Remote Objects mit FlexOnRails - Abb. 5-1

5.2 Schnittstellendefinition und –programmierung

Zur Definition der RubyAMF-Gateways benötigt Flex beim Kompilieren eine zusätzliche Konfigurationsdatei (services-config.xml), die festlegt, welche Services und Channels für eine Applikation eingerichtet wurden. Flex verwendet hierbei seine internen Remoting Services, die wiederum mit RubyAMF als Channel-Definition kommuniziert. In der Channel-Defintion wird der Host und das Gateway des RubyOnRails-Servers angegeben.

```
<services-config>
        <services>
                <service id="Rubyamf-flashremoting-service"
                         class="flex.messaging.services.RemotingService"
                         messageTypes="flex.messaging.messages.RemotingMessage">
                        <destination id="Rubyamf">
                                <channels>
                                        <channel ref="Rubyamf"/>
                                </channels>
                                <properties>
                                        <source>*</source>
                                </properties>
                        </destination>
                </service>
        </services>
        <channels>
                <channel-definition id="Rubyamf"
                                    class="mx.messaging.channels.AMFChannel">
                        <endpoint uri="http://valisoft.dyndns.org:3000/Rubyamf/gateway"
                                  class="flex.messaging.endpoints.AMFEndpoint"/>
                </channel-definition>
        </channels>
</services-config>
```

Zur Kommunikation mit Rails werden Remote Objects verwendet, für die es in Flex eine eigene Klasse gibt, die als Property die Destination der Channel-Definition hat. Im Remote Object werden Methoden definiert, die schließlich die Kommunikation mit Rails herstellt. Gibt auf der Rails-Seite eine Methode mit dem gleichen Namen, führt Rails diese aus und sendet Nachrichten zurück über das Gateway. Bei erfolgreichem Abschluss gibt es einen Rückgabewert. Es gibt also auf jede Anfrage und auf jeden Aufruf eine Antwort und eine Funktion (Event), die bei Erfolg ausgeführt wird. Bei einem Fehler der Anfrage wird die Fault-Funktion des Remote Objects aufgerufen.

```
...
    <mx:RemoteObject id="messageService" fault="onFault(event)"
                     source="MessagesController" destination="Rubyamf">
        ...
        <mx:method name="showLink" result="onBodyResult(event)" />
        <mx:method name="createLink" result="onResult(event)" />
        ...
    </mx:RemoteObject>
        ...

    <mx:Button id="createLink"
               label="Create Link"
               click="messageService.create.send({text: createLinkTxt.text});" />
...
```

Auf der Rails-Seite muss pro Controller (hier ein Platzhalter-Controller für alle Messages), das AMF-Protkoll die Anweisungen der Flex-Anwendung an Rails weiterleiten. Nach der Anweisung wird ein Rückgabewert ausgegeben, der bescheinigt ob die Operation gescheitert ist oder abgeschlossen wurde.

```
class MessagesController < ApplicationController
   ...
   def create
       if is_amf
          @message = Message.new({:text => params[0][:text]})
       else
          @message = Message.new(params[:message])
       end

    respond_to do |format|
      if @message.save
         flash[:notice] = 'Link was successfully created.'
         format.html { redirect_to message_url(@message) }
         format.xml  { head :created, :location => message_url(@message) }
         format.amf  { render :amf => "Link Saved" }
      else
         format.html { render :action => "new" }
         format.xml  { render :xml => @message.errors.to_xml }
         format.amf  { render :amf => @message.errors }
      end
   end
   ...
end
```

RubyAMF hat uns durch seine einfache Implementationsweise sehr gefallen. Es wird per Ruby-Gems-Update in eine Rails-Anwendung installiert und steht sofort zur Verfügung. Viele Tutorials im Web halfen uns mit dem Umgang und brachte uns auf neue Ideen (für die oftmals keine Zeit zur Realisierung war). Da viele überflüssige Datentransfers wegfallen, können sehr leistungsfähige Web-Applikationen erstellt werden, die klassischen HTML-Webanwendungen in Ladezeit und Benutzerfreundlichkeit (trotz AJAX!) weit überlegen sind.

Viele unserer anfänglichen Probeanwendungen basierten auf XML-Schnittstellen und verwendeten keine Remote Objects, sodass einige Probleme unerkannt blieben. Für die erste Test-Anwendung für Tests zur Integration verwendeten wir einen einfachen RubyAMF-Prototypen und das bekannte Cookbook-Beispiel, eine einfache Web-Anwendung, die auf dem CRUD-Prinzip beruht. Diese Testanwendungen zeigten uns einige der Problematiken und Lösungen auf, die Arbeit mit RubyAMF hilfreich waren.

Leider gab es zur Programmierung mit RubyAMF noch keine Bücher (Stand: Januar 2008). Daher waren wir auf Tutorials und die Hilfe von (einigen wenigen) Usern im Internet angewiesen, die sich ebenfalls schon mit dieser (noch recht jungen) Technologie zu beschäftigen hatten. Großer Dank geht an dieser Stelle an Aaron Smith, dem Chefentwickler von RubyAMF und dessen Blog/Community.

6 Design & Usability

6.1 Logo & Design

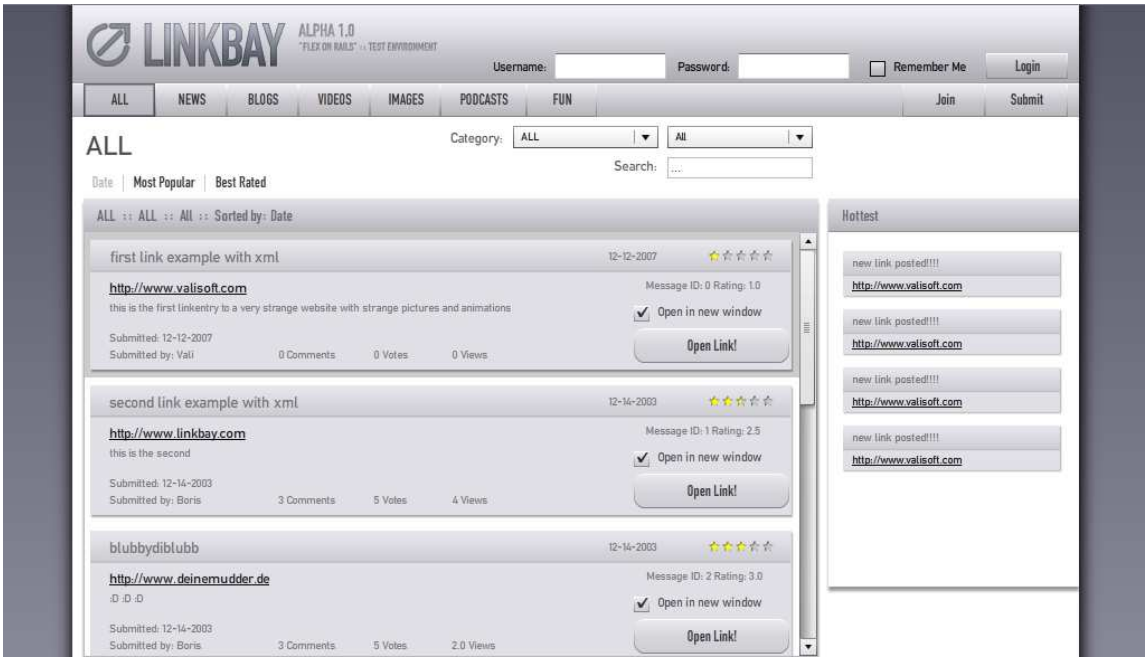
Wir haben das LinkBay Logo so einfach wie möglich gehalten und uns an den Logos bestehender News-Seiten orientiert. Die Schriftart ist die Deutsche Mittelschrift (Eng), deren Familie über ganz LinkBay verteilt ist. Das Logo entstand in PhotoShop und besteht aus mehreren Layern, die einen 3D-Effekt und die Farbverläufe erzeugen. Das LinkBay Logo setzten wir, wie bei nahezu allen derzeitigen (News-)Seiten in die obere linke Ecke.

LinkBay Logo - Abb. 6-1

Die Testumgebung erscheint in blauen Farben und soll Seriosität, Glaubhaftigkeit und Schlichtheit vermitteln. Wir hatten viele Design-Konzepte, die nach vielen Tests wieder verworfen wurden, weil sie sich schlecht implementieren ließen, unzugänglich oder einfach nicht gut genug waren. Zur Entwicklung des Seitendesigns mussten wir diverse Vorlagen ansehen, um uns einen Überblick zu verschaffen, wie andere News-Seiten aussehen. Wir schufen in PhotoShop eine neue Vorlage, die sich durch einige dezente Effekte auszeichnete und sich auch an die bestehende Seiten-Konventionen hielt (wovon wir eigentlich versuchten wegzukommen).

Aufgrund einiger Design-Konventionen, beschlossen wir die Anwendung auf eine feste Breite zu reduzieren (900 Pixel). Die Höhe der Anwendung bleibt variabel und ist von der Größe des Browsers abhängig. Wir nutzten in Flex sogenannte Constraints und Ankerpunkte zum Festsetzen bestimmter Elemente, sodass die Anwendung auch bei extremen Größenskalierungen des Browsers stabile Design-Elemente aufweisen kann. Die Anwendung ist dabei in der Mitte und am oberen Rand des Browsers „verankert", sodass der User stets einen feststehenden Fixpunkt erkennen kann, sobald sich die Größe des Browser verändern sollte.

Die Anwendung liegt zur räumlicheren Darstellung vor einem Verlauf, (der leider noch nicht in den Hintergrund der Pre-Loader-Klasse implementiert ist) und mit einem Schatten hinterlegt, der die Seite hervorheben soll. Dadurch entsteht der Eindruck, dass die Applikation auf dem Browser liegt. Alle enthaltenen Steuerelemente von LinkBay wurden schließlich nach und nach in unser Design-Konzept übertragen und wurden ausgiebig auf ihre „Look-And-Feel"-Eigenschaften hin getestet und untersucht. Der Grundgedanke war Seriosität und Eleganz auf alle sichtbaren Elemente zu übertragen und jedes Element sichtbar von anderen Komponenten abzugrenzen.



LinkBay Design - Abb. 6-2

6.2 Zur Usability

Viele Schwierigkeiten bereitete uns die Benutzerfreundlichkeit von LinkBay. Wenn sich z.B. ein User einloggt, erwartet er, dass die Felder für den „LogIn" und das „Passwort" verschwinden und stattdessen ein LogOut-Button erscheint. In früheren Versionen von LinkBay pflegten wir alle User-Aktionen auf einen eigenen Frame zu beschränken, wo man sich (ähnlich wie bei Windows) an und abmeldet. Da dies dem natürlichen Charakter einer Anwendung und keiner Webseite entspricht, erkennt man, dass die Programmlogik und Usability im Web irgendwann getrennte Wege gehen, da viele User Konventionen (im Vergleich mit anderen Webseiten) zur Bedienung erwarten.

Wir behielten jedoch unser Konzept bei, andere Formular-Daten (Registrierung, Neue Nachricht) in einem eigenen Fenster anzeigen zu lassen, da sich sonst das „Anwendungs-Gefühl" auf LinkBay nicht weiter hätte umsetzen lassen. Ob sich die User an dieses Konzept gewöhnen können, wird sich in der ersten Public-Beta-Phase entscheiden. Wir haben jedoch gute Erfahrungen damit gemacht.

Weitere Usability-Probleme waren die Positionierung der Panels und ComboBoxes für die Kategorien. Bei den vorläufigen Design-Studien wurden nämlich solche Elemente nicht berücksichtigt (was sich als Fehler erwiesen hat). Später bemerkten wir, welchen Einfluss solche Elemente auf das Design und besonders die Usability der Anwendung haben und integrierten sie mit in die Design & Usability-Planung.

6.3 Probleme mit Effekten

Die übermäßige Verwendung von Effekten führte bei der Ausführung dazu, dass auf langsamen, oder ausgelasteten Computern die Anwendung ins Stocken gerät. D.h. Frames kommen durch den

FlashPlayer ins Ruckeln, die Anwendung läuft nicht mehr flüssig und vermittelt nicht mehr den Eindruck einer performanten Anwendung (auch wenn die Zugriffe auf den Rails-Server unabhängig davon sind). Wir beschlossen daher sehr viel sparsamer mit Effekten umzugehen und Übergänge zwischen den Seiten unspektakulär aussehen zu lassen.

Bei Öffnen neuer Fenster wird in Flex der Hintergrund unscharf gemacht. Diesen Effekt behielten wir bei, da er nur kurz ist und wenige Systemressourcen verbraucht. Aber schon bei diesem Effekt bemerkten wir auf älteren PCs, dass die Anwendung sehr weit ausgelastet ist. Auch das Fading von Buttons mussten wir reduzieren, sodass die Anwendung im Hintergrund weiterhin reagiert.

Die Evaluation fand auf unterschiedlich schnellen Computern statt. Die Geschwindigkeit der Anwendung war abhängig von Prozessor, Monitorauflösung und Auslastung. Wir fanden heraus, dass beim Tabbed-Browsing (Browser mit Registrierkarten) die Leistung (Reaktionsgeschwindigkeit) der Anwendung bei vielen Effekten enorm sinkt.

7 LinkBay Ausblick

Im Moment sind wir ganz zufrieden mit dem Entwicklungsstand von LinkBay Alpha. Alle unsere gesetzten Ziele haben wir erreicht, auch wenn mehr Probleme auftraten als anfangs gedacht. Gerade die Integration, die in der Theorie so problemlos erscheint, war in der Praxis ein langes Probieren und Testen. Zudem sind wir im Laufe der Entwicklung auf weitere Probleme gestoßen, an die wir anfangs gar nicht gedacht hatten.

Probleme
Eines davon ist die Tatsache, das Google keine Flash-Webseiten (SWF Datein) indexieren kann. Bei unserem Flex Frontend handelt es sich leider um eine solche SWF-Datei. Deshalb müssen wir einen Weg finden, wie wir unseren Datenbestand im Rails-Backend einem Web Crawler zugänglich machen können. Ohne die Indexierung in verschiedenen Suchmaschinen werden die Besucherzahlen von LinkBay wohl nicht die Grenze der schon angemeldeten Mitglieder überschreiten, was sehr ärgerlich wäre.

Ein weiterer Punkt ist, dass es uns im Moment nicht möglich ist, „Deep-Linking" in unserem Flex Frontend anzuwenden. Da es sich hier nicht um generierte HTML-Seiten handelt, sind Hyperlinks zu einem Bereich oder Element auf der Seite faktisch völlig nutzlos bzw. machen überhaupt keinen Sinn. Das Flex-Framework unterstützt zwar eine Art „Deep-Linking Simulation", jedoch hat uns bis jetzt die Zeit gefehlt um dies zu implementieren, da es doch mit einem hohen Aufwand verbunden ist.

Weitere Agenda
Neben der Behebung dieser Probleme stehen auch noch andere Punkte auf unserer Agenda. Bevor wir die Beta-Version von LinkBay veröffentlichen und somit Test-Besuchern zugänglich machen können, müssen wir noch mehr Funktionen implementieren. Der Funktionsumfang von LinkBay sollte bei erscheinen der Beta-Version vollständig sein, sodass alle Funktionen von Besuchern getestet werden können. Dazu müssen wir auch noch ausführliche Tests für LinkBay im Rails Framework schreiben, die wir bei der Alpha Entwicklung ein wenig vernachlässigt haben. So können wir mögliche Probleme automatisiert reproduzieren und testen, sodass eine möglichst schnelle und einfache Fehlerbehebung möglich ist.

Sobald LinkBay auf ein „Final Release" zusteuert und somit die meisten Probleme behoben wurden, möchten wir mit Hilfe des „Adobe AIR Plugins" auch eine Desktop Applikation von LinkBay veröffentlichen. Dies wird ermöglicht, indem das Flex Frontend mit einem „Standalone Flash Player" zu einer ausführbaren Datei kompiliert wird. Hiermit lassen sich Desktop-Benachrichtigungen bei neuen News und Schnittstellen zu andern Services auf den Clienten realisieren. Mitglieder müssten nicht einmal mehr ihren Webbrowser öffnen, um auf die Inhalte von LinkBay zugreifen zu können.

Zum Abschluss noch einmal eine Übersicht der noch zu bewältigenden Aufgaben, bevor die Link-Bay Entwicklung zum Abschluss kommen kann:

- Lösung für das Google Indexing Problem
- Lösung für das Flex Deep-Linking
- Implementierung vieler Features
- Test Suite & Comprehensive-Test
- Online Beta-Version von LinkBay
- Desktop Applikation via Adobe AIR

8 Quellen

8.1 Online

http://www.themidnightcoders.com/weborb/rubyonrails/gettingstarted.htm
http://weblog.rubyonrails.org/2007/1/19/rails-1-2-rest-admiration-http-lovefest-and-utf-8-celebrations
http://railstips.org/2007/4/30/storing-secure-passwords
http://funfx.rubyforge.org/guide/
http://t3n.yeebase.com/magazin/ausgaben/artikel/von-rails-ueber-flex-zu-weborb/
http://www.rubypeople.org/
http://livedocs.adobe.com/livecycle/es/sdkHelp/programmer/lcds/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=Part1_developer_1.html
http://learn.adobe.com/wiki/display/Flex/Part+II.+Exchanging+Datas
http://www.liverail.net/articles/2006/04/16/rubyonrails-1-1-and-flex-2-0-pt-1
http://code.google.com/p/flexlib/wiki/ComponentList
http://www.liverail.net/articles/2007/6/29/tutorial-on-developing-a-facebook-platform-application-with-ruby-on-rails
http://www.lfpug.com/presentations/2006_06_26_eccles/index.htm
http://natureandtech.blogspot.com/2007/10/beginners-tutorial-to-rubyamf-with.html
http://duncandavidson.com/archives/155
http://www.benlog.org/2007/2/17/deploy-your-rails-app-in-a-subdirectory-with-apache-and-mongrel
http://bcrypt-ruby.rubyforge.org/
http://www.aidanf.net/rails_user_authentication_tutorial
http://en.wikipedia.org/wiki/Salt_(cryptography)
http://www.oreillynet.com/ruby/blog/2007/09/7_reasons_i_switched_back_to_p_1.html (wichtig!!)
http://www.sitepoint.com/article/learn-ruby-on-rails
http://forum.ruby-portal.de/
http://www.railsapi.org/
http://solutions.treypiepmeier.com/2006/12/04/rails-migration-data-types/
http://wiki.rubyonrails.com
http://www.height1percent.com/articles/2006/01/30/on-rails-migrations-and-postgresql-data-types
http://natureandtech.blogspot.com/2007/10/beginners-tutorial-to-rubyamf-with.html
http://blog.rubyamf.org/
http://flexonrails.net/
http://www.rubychan.de/
http://api.rubyonrails.org/
http://caboo.se/doc.html
http://livedocs.adobe.com

8.2 Bücher

Programming Flex 2

Paperback: 502 pages
Publisher: Adobe Dev Library (April 16, 2007)
Language: English
ISBN-10: 059652689X
ISBN-13: 978-0596526894

Build Your Own Ruby on Rails Web Applications

Paperback: 447 pages
Publisher: SitePoint (January 30, 2007)
Language: English
ISBN-10: 0975841955
ISBN-13: 978-0975841952

Practical Rails Social Networking Sites (Expert's Voice)

Paperback: 421 pages
Publisher: Apress (June 22, 2007)
Language: English
ISBN-10: 1590598415
ISBN-13: 978-1590598412

Agile Web Development with Rails, 2nd Edition

Paperback: 720 pages
Publisher: Pragmatic Bookshelf; 2 edition (December 14, 2006)
Language: English
ISBN-10: 0977616630
ISBN-13: 978-0977616633

Rails Cookbook (Cookbooks (O'Reilly))

Paperback: 534 pages
Publisher: O'Reilly Media, Inc. (January 16, 2007)
Language: English
ISBN-10: 0596527314
ISBN-13: 978-0596527310

The Rails Way

Paperback: 912 pages
Publisher: Addison-Wesley Professional; 1 edition (November 26, 2007)
Language: English
ISBN-10: 0321445619
ISBN-13: 978-0321445612