



University of Fribourg, Switzerland

Department of Computer Science

**ENTWICKLUNG VON HTML ZU AJAX :
NATIVE PROGRAMMIERTECHNIKEN DES WEB 2.0**

Seminararbeit

Hélène Martenet

route de Choëx 93

1871 Choëx

helene.martenet@unifr.ch

01-222-785

28 Juli 2008

ABSTRACT

Diese Seminararbeit setzt sich mit der Thematik der Entwicklung der nativen Programmieretechniken des Webs 2.0 auseinander. Der Weg, der von dem statischen klassischen read-only Web zum dynamischen Benutzerbeteiligten Web 2.0 führte, wird von der technischen Seite her beschrieben. Unter anderem werden Ajax, RIAs und Webframeworks besprochen. Ausserdem wird Ruby und Ruby on Rails detaillierter vorgestellt und zu diesem Zweck wird ein Prototyp einer Webanwendung programmiert. Die Applikation wird dem Informationsaustausch zwischen ehemaligen Erasmusstudenten der Universität Freiburg an ausländischen Universitäten und Studenten die einen Erasmusaufenthalt planen, dienen.

KEYWORDS

Web 2.0, neue Webtechnologien, Webframeworks, Dynamisches HTML, Dynamische Seiten, Ruby on Rails, Rich Internet Application, Ajax, FriErasmus

INHALTSVERZEICHNIS

1. DAS WEB 2.0	1
1.1. PROBLEMSTELLUNG: EVOLUTION VS. REVOLUTION DES WEB 2.0	1
1.2. ZIELSETZUNG	3
1.3. VORGEHENSWEISE	3
2. WEBTECHNOLOGIEN VOR DEM WEB 2.0	4
2.1. DYNAMISCHES HTML	4
2.2. DYNAMISCHE SEITEN: SERVER-SIDE SCRIPTING	6
3. WEB 2.0: EINE KOMBINATION VON TECHNIKEN UND WIEDERVERWENDUNG VON FUNKTIONEN	7
3.1. AJAX UND RICH INTERNET APPLICATION (RIA)	8
3.1.1. RIA: Grundlagen und Beispiele	8
3.1.2. Grundlagen von Ajax: XMLHttpRequest	11
4. WEBFRAMEWORKS	17
4.1. WEBFRAMEWORKS: GRUNDLAGEN UND BEISPIELE	17
4.2. RUBY ON RAILS (ROR)	18
4.3. FRIERASMUS: BEISPIEL EINER RAILS-APPLIKATION	21
5. ZUSAMMENFASSUNG UND SCHLUSSFOLGERUNG	24
5.1. AUSBLICK: DAS WEB 3.0	25
LITERATURVERZEICHNIS	27
ANHANG	28

ABBLIDUNGSVERZEICHNIS

Abbildung 1: Zusammenfassendes Klassifikationsschema von Web 2.0	2
Abbildung 2: Vergleich Client-Side und Server-Side Scripting	7
Abbildung 3: Ajax und seine Bestandteile	11
Abbildung 4: Klassisches vs. Ajax-Modell einer Web-Abbildung	14
Abbildung 5: Request/Response-Verhalten im klassischen Web	14
Abbildung 6: Request/Response-Verhalten beim Einsatz von Ajax	15
Abbildung .7: Überblick von ROR	21
Abbildung 8: MVC-Struktur der Applikation FriErasmus	23
Abbildung 9: Die Hauptseite der Applikation FriErasmus	24

ABKÜRZUNGSVERZEICHNIS

AIR	Adobe Integrated Runtime
AJAX	Asynchronous JavaScript and XML
API	Application programming interface
AWT	Abstract Window Toolkit
CSS	Cascading Style Sheets
DHTML	Dynamic HyperText Markup Language
DWR	Direct Web Remoting
DOM	Document Object Model
DRY	Don't Repeat Yourself
GWT	Google Web Toolkit
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JSP	Java Server Pages
MVC	Model-View-Controller
OO	Object-oriented programming
ORM	Object/Relational Mapping
PHP	PHP: Hypertext Preprocessor
RIA	Rich Internet Application
ROR	Ruby on Rails
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
URL	Uniform Resource Locator
VB	Visual Basic
XHR	XMLHttpRequest
XML	Extensible Markup Language
XUL	XML User Interface Language

1. Das Web 2.0

1.1. Problemstellung: Evolution vs. Revolution des Web 2.0

Wie Thomas Friedman in [Friedman 2005] beschreibt, ist das Internet und seine verbreitete Benutzung eine treibende Kraft in der Globalisierungsphase des 21. Jahrhunderts. Neue Entwicklungen auf dem Bereich sollten deshalb nicht übersehen und verschlafen werden, jedoch wahrgenommen und genutzt werden.

Die Fülle an Informationen über das Web 2.0 Phänomen ist überwältigend und wächst ständig an. Ist es aber berechtigt anzunehmen, dass eine echte Revolution stattfindet oder ist es nur ein Marketingbegriff, der eine ganz normale Evolution beschreibt?

Einige Technologieexperten, wie Tim Berners-Lee, haben in Frage gestellt, ob die Benutzung des Terms „Web 2.0“ sinnvoll sei, da viele der technologischen Komponente des Webs 2.0 seit den ersten Tagen des World Wide Webs existieren. Es ist deswegen wichtig die verschiedenen Aspekte des Phänomens voneinander zu trennen: hinter dem Buzzwort verstecken sich nämlich einerseits eine *technologische*, andererseits aber auch eine *soziale* Veränderung, die sich gegenseitig beeinflussen.

In [Lassila und Hendler 2007] wird das Web 2.0 hauptsächlich als eine *soziale* Revolution in der Verwendung der Webtechnologien, als einen Paradigmenwechsel des Webs vom *Publizierungsmedium* zum *Interaktions-* und *Partizipationsmedium* beschrieben. Im Gegensatz, [Yakovlev 2007] untersucht die *Dienste* des Webs 2.0 und kommt zu der Schlussfolgerung, dass bei weitem nicht alle als revolutionär betrachtet werden dürfen.

Revolutionär oder nicht, das Web 2.0 konnte diese Neuerungen allerdings erst dank der Entwicklung seiner Programmier Techniken ermöglichen. Nur dank ihnen konnte der Webuser selber aktiv mitgestalten, anstatt statische Inhalt nur passiv zu konsumieren. Erst dann wurde es dem User ermöglicht die drei obersten Ebenen von Maslows Bedürfnispyramide (die Bedürfnisse nach Selbstverwirklichung, sozialer Anerkennung und sozialer Beziehungen) im dem Web zu decken und den Web 2.0 Hype zu ermöglichen.

Abbildung 1.1. zeigt einen Überblick verschiedener Konzepte des Web 2.0. Das hier angewandte Klassifikationsschema unterscheidet drei Achsen:

- Anwendungen und Dienste,
- Software und Systeme,
- Technologien und Programmiersprachen.

Die Technologien und Programmiersprachen werden den Mittelpunkt dieser Arbeit darstellen.

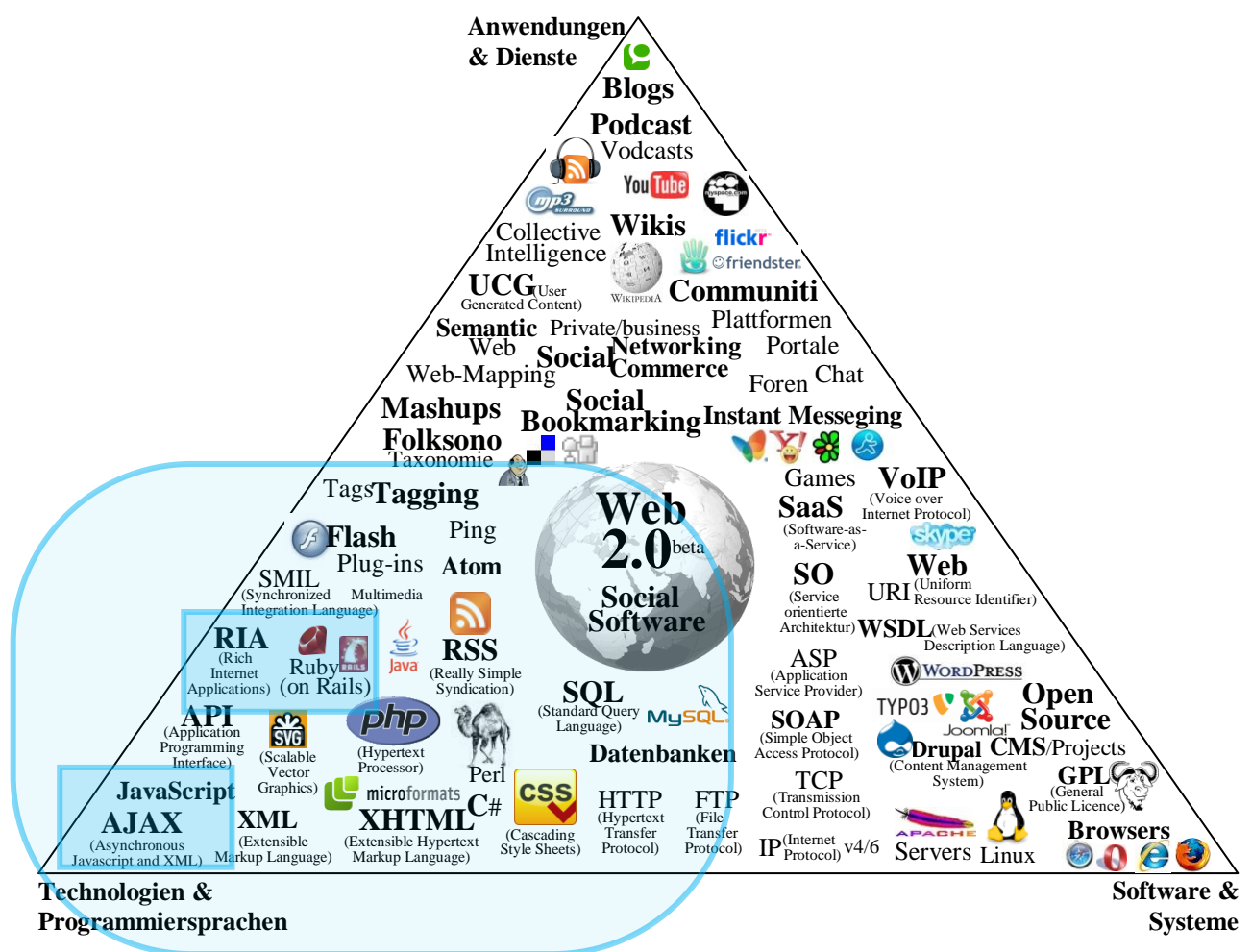


Abbildung 1: Zusammenfassendes Klassifikationsschema von Web 2.0 [Zumstein 2008]

1.2. Zielsetzung

Das Ziel dieser Seminararbeit ist das Web 2.0 von der technologischen Seite her zu betrachten und die Entwicklung der Programmier Techniken zu beschreiben, die zur veränderter Einschätzung und Benutzung des Webs beigetragen haben. Es wird versucht aufzuzeigen, dass die Technologien des Webs 2.0 durch eine Evolution und Kombination, jedoch keineswegs durch eine Revolution entstanden sind.

Welche sind wichtige, von Browsers nativ unterstützte, Programmier Techniken des Web 2.0? Welche sind ihre Grundlagen und ihre Anwendungsmöglichkeiten? In dieser Arbeit wird versucht, dem Web 2.0 ein programmiertechnisches Gesicht hinter dem Buzznamen zu geben. Zu diesem Zweck wird AJAX näher betrachtet und erläutert.

Im Web 2.0 wird versucht, nach dem Prinzip des „Agile Web Development“ Programme zu entwickeln. Dieses Vorgehen wird verwendet, um die Effizienz des Programmierprozesses zu steigern. Welche Arten von Framework können dazu eingesetzt werden? Wie kann beispielsweise Ruby on Rails helfen effizienter zu programmieren? Wie ist so ein Framework aufgebaut? In dieser Arbeit wird aufgezeigt, wie mit solchen Frameworks Webapplikationen leicht entwickelt werden können, welche als Plattformen zum Informationsaustausch zwischen Usern benutzt werden.

1.3. Vorgehensweise

Um die Evolution der Programmier Techniken zu analysieren, wird im 2. Kapitel die Technologien vor dem Web 2.0 beschreiben. Das Web folgte schon damals dem Trend immer dynamischer zu werden. Dynamisches HTML und Dynamische Seiten sind die Hauptthemen Abschnitts 2.

In Kapitel 3 wird anhand zwei wichtigen Programmier Techniken veranschaulicht, wie im Web 2.0 verschiedene Sprachen und Konzepte kombiniert werden, um dem Benutzer mehr Inhalt und Interaktivität zu verschaffen. Im Kapitel 3.1 werden AJAX und RIA's unter die Lupe genommen.

Verschiedenen Arten von Frameworks werden mit Beispielen (Serverseitig und Clientseitig) im Kapitel 4.1 erläutert. Dem Framework Ruby on Rails wird das Kapitel 4.2 gewidmet und

dazu eine kleine Ruby on Rails Webapplikation zur Veranschaulichung programmiert. Die Applikation wird dem Informationsaustausch zwischen ehemaligen Erasmusstudenten der Universität Freiburg an ausländischen Universitäten und Studenten die einen Erasmusaufenthalt planen, dienen.

In dieser Seminararbeit werden nur die wichtigsten, von den Browsern nativ unterstützten Programmieretechniken des Webs 2.0 aufgezeigt. Technologien wie zum Beispiel Flash oder Java Web Start werden nicht behandelt, da sie nicht nativ von Browsern unterstützt werden. Die Techniken des Taggings (Semantik) und die, die hauptsächlich auf XML basieren (beispielsweise: RSS, ATOM,...), werden auch nicht behandelt.

2. Webtechnologien vor dem Web 2.0

Während eines Jahrzehnts, hat sich die Art wie die Webuser das Web navigieren nicht gross verändert. Nur letzthin hat es einen Wechsel vom einfachen seitenbasierten Web zur anwendungsbasierten Web gegeben. Diese Veränderung ist durch eine Entwicklung entstanden, die immer mehr Dynamik im Inhalt und später in der Darstellung anstrebte [Vossen und Hagemann 2007].

2.1. *Dynamisches HTML*

Dynamisches HTML (DHTML), bezeichnet die Kombination von HTML, CSS, clientseitiges Scripting (meistens JavaScript) und DOM in einer Webseite. Dieser Begriff soll zum Ausdruck geben, dass es sich dabei um beträchtlich mehr als einfaches HTML handelt. Einfaches HTML ist statisch und erlaubt keine Veränderungen in der Seite. Mit DHTML ist es möglich Seiten zu entwerfen, die ihre Erscheinung nach gewissen Ereignissen verändern ohne neue Seiten oder Daten nachzuladen [Vossen und Hagemann 2007]. Mit DHTML ist es beispielsweise möglich, das sich die Farbe oder die Schrift eines Textes sich verändert, wenn der Webbenutzer den Mauszeiger darüber bewegt. Ein Bild könnte auch per Drag and Drop zwischen verschiedenen Plätzen in der Seite verschoben werden [Paulson 2005].

Der Term Dynamik sollte nicht verwirren. Beim DHTML wird das darunterliegende HTML und somit der Inhalt von der Seite nicht verändert. Nur die Erscheinung der Seite wird verarbeitet. Um einen neuen Inhalt anbieten zu können, wird ein Mechanismus zum nachträglichen Laden von Daten benötigt. Dieses ist genau was im Web 2.0 Ajax bietet.

Das „D“ in DHTML bezeichnet auch nicht Seiten, die vor der Ankunft im Webbrowser vom Server dynamisch generiert werden, sogenannte *dynamische Webseiten*. Die Dynamik in DHTML-Seiten wird nach der Ankunft im Browser ausgeführt.

- **Hypertext Markup Language (HTML):** ist eine Sprache, die aus Markups besteht und die in einem reinem Textdokument geschrieben wird. Sie enthält Informationen über die Struktur und die Darstellung der Webseite. Ein Fakt, der sicherlich zum Erfolg von HTML beigetragen hat, ist, dass HTML einen mühelosen Einstieg mit sich bringt. Die Sprache ist relativ einfach zu erlernen, da sie eine stark hierarchische Struktur in sich trägt (keine Schleifen, Verzweigungen usw.) und ein einfacher Texteditor genügt, um reine HTML-Seiten zu schreiben.

HTML birgt aber ein Problem, nämlich die Vermischung der Struktur mit der Darstellung einer Seite. Der Inhalt ist in seiner Wiederverwendung beschränkt, wenn es unnötigerweise auf einem bestimmten Medium angepasst worden ist. Unterschiedliche Medien oder Kontexte, beispielsweise mobile Geräte, Printmedien und Präsentationen können eine andere Darstellung benötigen. Deswegen ist es nicht optimal die Struktur an eine bestimmte Darstellung zu binden.

Wichtig ist auch einen Style auf einen anderen Inhalt anwenden zu können. Um beispielsweise einem Webauftritt ein konsistentes Design zu vergeben, ist es von Vorteil, wenn man ein bestimmtes Style wiederverwenden kann. [Vossen und Hagemann 2007]

- **Cascading Style Sheet (CSS):** ist seit 1996 ein Standard der W3C, welcher die Trennung von Inhalt und Darstellung ermöglicht. CSS erlaubt die Definition von Darstellungsstilen für Elemente eines strukturiertes Dokuments, wie beispielsweise ein HTML oder Extensible Markup Language (XML) Dokument [Paulson 2005].
- **JavaScript:** ist eine in 1995 bei Netscape und Sun entwickelte objektorientierte Skriptsprache. Entgegen was der Name andeutet, hat sie keinen Bezug zu der Programmiersprache Java. JavaScript erlaubt Webseiten mit Programmierlogik anzureichern: Objekte können so dynamisch verändert werden und dadurch Effekte und Interaktion mit den Users erzeugen. Formulare können somit beispielsweise schon überprüft werden, bevor sie zum Server abgeschickt werden und gegebenenfalls eine Fehlermeldung anzeigen.

- **Document Object Model (DOM):** ist eine Programmierschnittstelle (API) für den Zugriff auf HTML und XML-Dokumente und seit 1998 ein W3C Standard. DOM ist eine Baumstruktur von Objekten, die der Browser mit den HTML-Elementen der Webseite aufbaut. JavaScript benutzt dann diese Objekte, um im DHTML eine Webseite dynamisch zu verändern. Das JavaScript manipuliert nicht direkt die HTML Tags und ihren Inhalt, sondern die Objekte, die im Kontext des ausgeführten Skripts vorhanden sind.

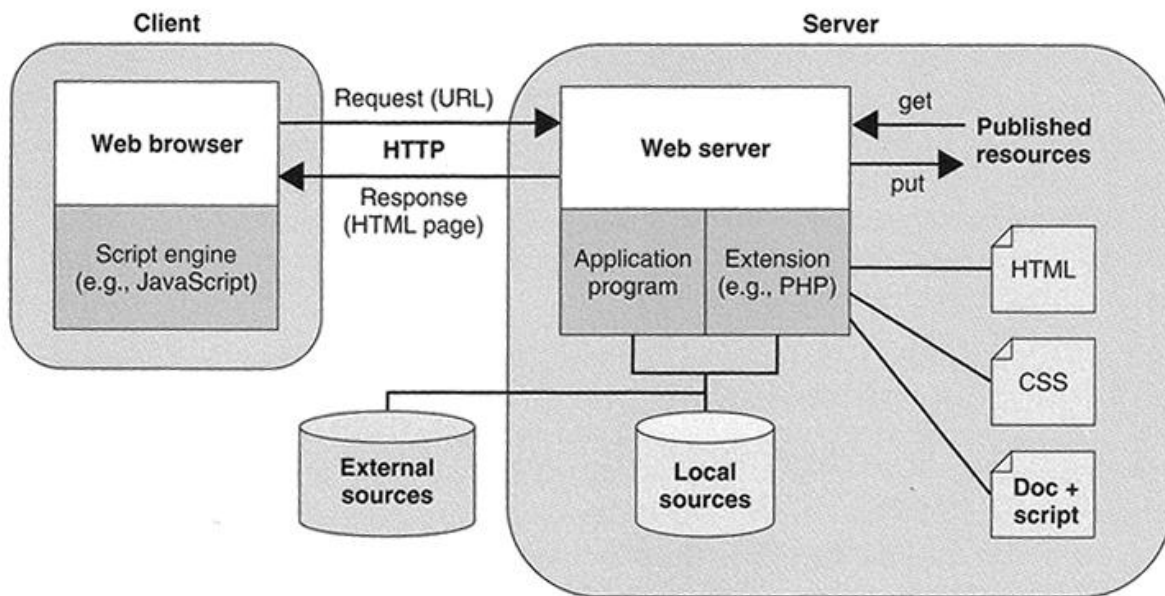
2.2. Dynamische Seiten: Server-Side Scripting

JavaScript ist eine sogenannte „Client-Side Scripting“ Sprache. Sie wird vom Webbrowser ausgeführt und erlaubt so einige Aufgaben direkt beim User zu erledigen.

Es ist aber nicht immer möglich die ganze Logik auf der Kundenseite abzuwälzen. „Server-Side scripting“ ermöglicht HTML Seiten auf der Seite des Servers dynamisch zu generieren. Dies bringt den grossen Vorteil, dass als Antwort auf eine Abfrage zurückgeschickte Webseite von verschiedenen Parametern abhängig gemacht werden kann. Mögliche Beispiele für solche Parameter sind: der Benutzeranforderungen, Zugangsrechte, Suchresultate, die von einer Datenbank zurückgegeben werden.

Die Client- und Server-Side Scripting Ansätze werden in der Abbildung 2.2. gegenübergestellt. In ihr wird ersichtlich, dass der Code beim Server-Side Scripting auf dem Server ausgeführt und so verschiedene Datenquellen gebrauchen kann, um eine Webseite zu generieren. Auf der anderen Seite ermöglicht das Client-Side Scripting schnell auf Benutzeraktion zu reagieren, ohne mit dem Server vorerst Kontakt aufnehmen zu müssen.

Es gibt viele verschiedene Server-Side Scripting Sprachen die heutzutage benutzt werden. Beispiele dafür sind Coldfusion, Perl, Python, Java Server Pages (JSP) und PHP. In der Abbildung 2.2. wird beispielhaft PHP eingesetzt.



*Abbildung 2: Vergleich Client-Side und Server-Side Scripting
[Vossen und Hagemann 2007]*

Sowohl Client als auch Server-Side Scripting basieren auf einer synchronen Kommunikation in einer Client/Server Architektur. Um die Webanwendungen noch einen Schritt dynamischer zu machen, kam letztlich die Idee auf, HTML direkt beim Client zu generieren und zwar aufgrund eines asynchronen Aufrufs von Daten vom Server. Wie im folgenden Kapitel aufgezeigt wird, führt dies zu einer weiteren Trennung der User Interface Logik vom Business Logik. Die zusammengesetzte Technik der Webentwicklung, die dahinter steckt heisst AJAX und stellt eine der wichtigsten Basen des Webs 2.0 dar.

3. Web 2.0: eine Kombination von Techniken und Wiederverwendung von Funktionen

Die Beliebtheit des Webs 2.0 basiert hauptsächlich auf zwei Grundpfeilern: Erstens das Bedürfnis von Menschen sich mithilfe des Internets auszutauschen und zweitens die Desktopähnliche Form, welche immer mehr Webanwendungen annehmen [Beck 2007]. Die grundlegenden Entwicklungsschritte, die diesen Erfolg ermöglichten, waren einerseits, die Weiterentwicklung der Datenübertragungsrates und die damit einhergehenden niedrigeren Übertragungskosten und, andererseits, neue Technologien, die immer mehr Interaktion und Kooperation ins Web ermöglichten [Kollmann 2007].

In diesem Kapitel werden zwei von den wichtigsten Technologien des Webs 2.0 vorgestellt: RIA und Mashups. Obwohl diese ganz neue Möglichkeiten und Herausforderungen mit sich bringen, bauen sie auf schon seit langem bekannten Techniken auf. In der heutigen agilen Zeit werden altbekannte Techniken erweitert und in RIA wiederverwendet; verkapselte Funktionen werden wiedergebraucht und zu Mashups kombiniert.

3.1. Ajax und Rich Internet Application (RIA)

3.1.1. RIA: Grundlagen und Beispiele

Der Begriff Rich Internet Application (RIA) bezeichnet eine Webanwendung, die die Charakteristika und Funktionalitäten einer Desktopanwendung aufweist. Der Term „Rich“ (zu Deutsch reichhaltig) bezeichnet den Fakt, dass ein Teil der Logik und der HTML-Generation auf dem Client abgewälzt wird. Die Interaktion mit dem User kann dadurch erhöht und neue Funktionalitäten ermöglicht werden, wie zum Beispiel: das Validieren von Formulareingaben in Echtzeit, kontextbezogene Hilfsmenus, Drag and Drop usw. Bekannte Beispiele von RIAs bilden unter anderem:

- Karten Applikationen: Google Maps, Yahoo Maps, Live Search Maps,
- Mail Applikationen: Outlook Web Access, Yahoo! Mail, GMail,
- Büroanwendungen: Think Free Office, Google Docs, Microsoft Office Live Workspace.

Alle verschiedenen RIA weisen den folgenden gemeinsamen Mechanismus auf: Eine Codeschicht zwischen dem Server und Client wird meistens zu Beginn der Benutzung der Applikation vom Server heruntergeladen. Sie wird oft als „Client Engine“ bezeichnet. Sie stellt eine Erweiterung des Browsers dar und ist normalerweise für die Kommunikation mit dem Server und die Darstellung des Users Interfaces verantwortlich. Dank ihr wird es dem Client möglich sein, später neuen Inhalt vom Server für den User, ohne dass dieser das merkt, herunterzuladen und darzustellen, ohne die ganze Seite neu aufzuladen. Wie im Kapitel 3.1.2. gezeigt wird, kann die Verwendung der Webseite dadurch fließender werden, da der User nicht mehr auf das klassische Aufladen der ganzen Seite warten muss [Khaled, Mugellini und Felber 2008].

Die Vorteile der RIA sind nicht zu übersehen: auf einer Seite werden mehr Reichhaltigkeit, Interaktivität und Flexibilität in den Anwendungen ermöglicht. Neue Funktionalitäten

erlauben intuitivere und benutzerfreundlichere Anwendungen. Die sonst nur auf dem Desktop verfügbaren Anwendungen werden nun webbasiert ausgeführt. Dadurch wird es möglich für die User ihre Applikationen und Daten überall nutzen zu können. [Beck 2007]

Auf der anderen Seite kann die Performance erhöht werden: Erstens braucht der Server nicht mehr das Arbeitspferd zu sein, welcher er in klassischen Anwendungen war. Einige Aufgaben werden direkt beim Kunden erledigt und so die Arbeitsverteilung zwischen Client/Server besser ausgewogen. Ein einziger Server kann dann mehr Clientanfragen gleichzeitig beantworten. Zweitens kann dank asynchroner Kommunikation die Client Engine Daten vom Server herunterladen ohne auf eine Aktion vom User (wie zum Beispiel einen Mausklick) warten zu müssen. Die Kommunikation zwischen der Client Engine und dem Server ist gegenüber die Zeit, in der der User die neuen Daten in der Seite gebrauchen wird, asynchron. Der User braucht so nicht jedes Mal zu warten. Zusätzlich dank sogenanntem „prefetching“ kann die Client Engine zukünftiges Datenbedürfnis vorhersagen und bevor der User sie anfordert, werden sie zum Client geschickt. Dieser Mechanismus macht die ganze Benutzung der Anwendung noch flüder. Google Map (siehe Kapitel 3.2.2.) benutzt diese Technik um angrenzende Mapsegmente zum Client zu schicken, bevor der User den Scroller bewegt. Drittens kann die Netzwerkbelastung auch erleichtert werden, da nicht immer die ganze Seite durch das Netz geschickt wird, wie mit klassischen Anwendungsmodellen, aber nur fehlende Daten nachgefragt werden. Dieser positive Effekt kann aber vermindert oder sogar umgekehrt werden, wenn durch Prefetching das Datenbedürfnis falsch antizipiert wird und zu viele ungenutzte Daten zum User geschickt werden.

RIA bergen aber auch ihre negativen Seiten: Da sie in einer lokalen sicheren Umgebung (sog. Sandbox) ausgeführt werden, verfügen sie nur über einen begrenzten Zugang zu den Systemressourcen. Falsche Annahmen über den Zugang zu den Ressourcen können zu fehlfunktionierenden RIAs führen.

Die Verwendung von Client-Side Script Sprache in RIAs kann auch zu Probleme führen: Erstens setzt es voraus, dass der User die Benutzung von Skriptsprachen in seinem Browser nicht deaktiviert hat. Zweitens kann es zu Performanceeinbussen kommen, wenn die Sprache interpretiert wird, wie zum Beispiel JavaScript, anstatt kompiliert, wie zum Beispiel die Programmiersprache Java.

Obwohl die Client Engine im Cache gespeichert werden kann, kann es ziemlich unangenehm lang werden, wenn die Webanwendung zum ersten Mal gestartet wird und die Client Engine vorerst heruntergeladen soll. Ein weiterer Nachteil ist, dass Suchmaschine RIAs nicht gut indexieren können. Obwohl es die beste Lösung für Webapplikationen wäre, nur gelegentlich die Verbindung zum Server zu gebrauchen und der Rest der Zeit offline arbeiten zu können, brauchen die meisten RIAs dauernd Netzwerkverbindung.

Die Webentwicklung von RIAs ist auch komplizierter, als die von klassischen Webseiten. Im Kapitel 4, werden verschiedene Webframeworks dargestellt. Sie stellen eine Lösung dar, um die Entwicklung von RIAs zu erleichtern.

In [Farell und Nezlek 2007] werden RIAs in drei Kategorien unterteilt, je nach dem wie sie entwickelt oder eingesetzt werden. Die erste Kategorie bilden die Plug-In basierte RIAs. Sie werden in einer dedizierten Umgebung programmiert und werden entweder als eine eingebettete oder als eine vom Browser gestartete autonome Anwendung eingesetzt. Beispiele für den eingebetteten Ansatz bildet Adobe's Flex und Flash Technologien (auf dem Flash Plug-In basierend), Java applet (auf dem Java Plug-In basierend) und Microsoft's Silverlight (auf dem Silverlight Plug-In basierend). Beispiele für den autonomen Ansatz bilden Java Web Start, JavaFX und Adobe's Air.

Scriptbasierte RIAs bilden die zweite Kategorie (Vgl. Kapitel 3.1.2.). Sie sind bei weitem die bekanntesten RIAs und werden vom Browser nativ unterstützt. Sie basieren auf AJAX.

Die dritte Kategorie heisst browserbasierte RIAs. Sie ist am wenigstens bekannt von den drei Kategorien. Generell umfasst sie eine User Interface Sprache, die auf XML basiert und die verschiedenen Elemente des Users Interfaces und ihre Interaktion auf einer deklarativen Weise zu beschreiben erlaubt. Die XUL (XML User Interface Language) Sprache von der Mozilla Foundation ist ein Beispiel von browserbasierte RIA. Sie kann nur im Browser ausgeführt werden die auf Mozilla basieren, wie beispielsweise Firefox.

Google Web Tool Kit (GWT) ist eine der meistbenutzten RIA Entwicklungsumgebung. Sie schlägt eine Brücke zwischen dem Plug-In Ansatz und dem Ajax Ansatz. Dieses Framework besitzt ein Java-nach-JavaScript-Compiler und so ermöglicht es Ajax Anwendungen in Java zu programmieren. Dieser Ansatz besitzt viele Vorteile unter anderem wird es möglich die vertrauten Javaklassen (Swing, AWT, etc.) und Entwicklungsumgebungen zu benutzen. Das objektorientierte (OO) Javadesign ist auch meistens einfacher zu warten als das JavaScript Design, das OO mit prozeduralem Code mischen kann [Farell und Nezlek 2007].

3.1.2. Grundlagen von Ajax: XMLHttpRequest

Scriptbasierte RIAs beruhen auf einer Technologie namens Ajax. Ajax ist ein Akronym und steht für *Asynchronous JavaScript and XML*. Es erlaubt den Inhalt einer Webseite *asynchron* zu aktualisieren ohne die *ganze* Seite erneut laden zu müssen. Dieser Vorgang wird *Partial Page Rendering* genannt, im Gegensatz zu *Full Page Reload* [Bosch 2007]. Dank Ajax ist es beispielsweise möglich, in einer Temperaturkonvertierungsanwendung, einen Anzahl Grad Celsius in einen Feld einzutippen, so dass das Resultat augenblicklich in einen anderem Feld eingeblendet wird.

Wie aus Abbildung 3.1.2. ersichtlich ist, besteht Ajax aus einem Pattern von verschiedenen Programmiersprachen. Diese waren schon zum grössten Teil vor dem Web 2.0 bekannt (Vgl. Kapitel 2.).

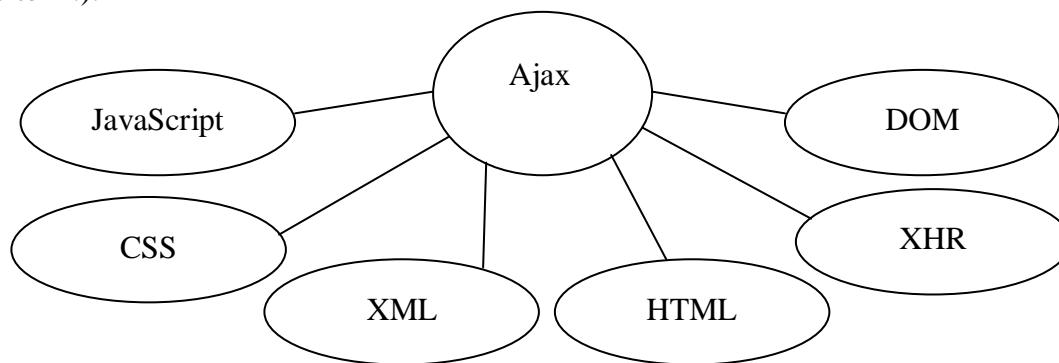


Abbildung 3: Ajax und seine Bestandteile

- **Extensible Markup Language (XML):** XML ist eine Metasprache die aus Markups besteht und seit 1998 ein W3C Standard. Sie wird zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien verwendet. Ajax benutzt XML für die Kodierung von Daten, um sie zwischen einem Server und Client übertragen zu können. Obwohl das „X“ in Ajax für *XML* steht, können andere Formate als XML für die Übertragung verwendet werden. JavaScript Object Notation (JSON), HTML und eine einfache Textdatei sind weitere Beispiele dafür.
- **XMLHttpRequest (XHR):** ist ein Application Programming Interface (API) von JavaScript, welches Verbindungen zu einem Server von der Clientseite über das Hypertext Transfer Protocol (HTTP) herzustellen erlaubt. Dieses ist zwar nur eine Erweiterung des herkömmlichen JavaScript, aber es stellt das Herz von Ajax dar. Dank ihr wird es möglich asynchron Daten (in Form von bsp. XML) abzurufen.

Wenn die Daten beim Client angekommen sind, werden JavaScript-Funktionen auf den **DOM** zugreifen, um die Darstellung (**CSS**) und auch den Inhalt (**HTML**) zu verändern. Der neue Inhalt wird in der Seite aktualisiert, ohne die ganze Seite neu laden zu müssen.

Es ist einerseits erstaunlich, dass durch die Kombination von bekannten Techniken und die Erweiterung von JavaScript (XHR) so ein riesiger Hype entstanden ist. Andererseits haben sich sehr interessante technische Möglichkeiten dadurch ergeben. Mit DHTML war es bis anhin möglich die Darstellung zu verändern, aber keine neue Inhalte konnten asynchron nachgeladen werden. In Abbildung 3.1.2.2. wird das Klassische Modell einer Webanwendung mit dem Modell einer Ajax-Webanwendung verglichen. Es findet einen Paradigmenwechsel vom Web 1.0 zum Web 2.0 statt:

Das Verhalten einer Web 1.0-Anwendung:

- *Nach einer Benutzeraktion* (Klick auf einen Link oder Button) erfolgt ein *Full Page Reload*. Das heisst, einen *HTTP-Request* wird zum Server geschickt, der Server bereitet eine neue *HTML-Seite* als Antwort und schickt sie zum Browser zurück. Es wird immer eine komplette Seite zurückgeschickt, auch wenn der Inhalt der resultierenden Seite sich nur ein bisschen verändert hat.
- Die Daten werden immer *synchron* vom Server geladen, das heisst wenn der Benutzer sie anfordert. Als Konsequenz muss der Benutzer jedes Mal warten bis die ganze neue Seite angekommen ist und angezeigt wird. In Abbildung 3.1.2.3. werden die Anfragen vom Client und Antworten vom Server zeitlich dargestellt. Das stockende Verhalten einer Webseite wird durch den unkontinuierlichen Pfeil der Benutzerinteraktion dargestellt.
- Der Browser kann meist nicht direkt auf Benutzeraktionen reagieren. Beispielsweise muss ein ganzes Formular mit verschiedenen Eingabefeldern zuerst zum Server zur Validierung geschickt werden, bevor eventuelle Fehlermeldungen in einige Felder angezeigt werden. Eine Validierung von Feldern direkt nach der Eingabe ist nicht möglich.
- Verschiedene optische Effekte und Funktionen, wie z.B. Drag&Drop, sind in einer Web-Umgebung nicht möglich [Bosch 2007].

Das Verhalten einer Web 2.0-Anwendung:

- Das Client-Engine für Ajax wird *Ajax-Engine* genannt. Sie ist dafür verantwortlich die gewünschten Daten beim Server per *HTTP-Request* anzufordern und bei der Rückkehr zu empfangen. Auf der Seite des Servers wird der Request bearbeitet und die Antwort wird in der Form von z.B. XML, HTML oder JSON File zurückgeschickt; dabei werden nur die zusätzliche nötige Daten übertragen, die der Browser noch nicht besitzt. Ein *Partial Page Rendering* erfolgt, d.h. der Browser wird *Bereiche* der Webseite neu aufbauen können ohne die ganze Seite neu laden zu müssen
- Es muss nicht mehr auf eine Aktion des Benutzers gewartet werden, bis Daten geladen werden (synchrones Vorgehen). Das asynchrone Anwendungsmodell ermöglicht ein viel fluideres Verhalten des Browsers: die Daten können einzeln und unabhängig von der Useraktivität vom Server geholt werden. Die Aktion des Benutzers wird in Abbildung 3.1.2.3. durch einen kontinuierlichen Pfeil veranschaulicht, dadurch wird es zum Ausdruck gebracht, dass der Benutzers in seiner Beschäftigung nicht aufhalten wird [Bosch 2007].
- Die Webanwendungen werden dynamischer, dadurch, dass sie im Hintergrund neue Inhalte vom Server nachladen können. Es ist beispielsweise möglich eine Validierung von Feldern direkt nach deren Eingabe durchzuführen.
- Dank verschiedener optischen Effekte und Funktionen, wie z.B. Drag&Drop, wird eine Webanwendung immer mehr die Form einer Desktop-Applikation annehmen.

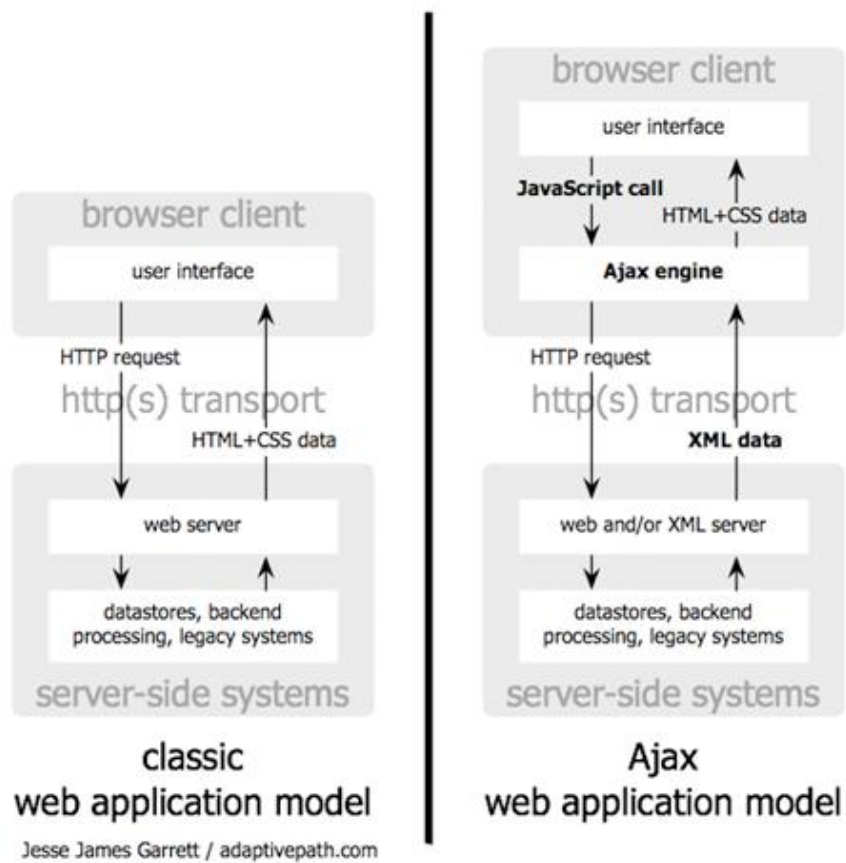


Abbildung 4: Klassisches vs. Ajax-Modell einer Web-Anwendung [Bosch 2007]

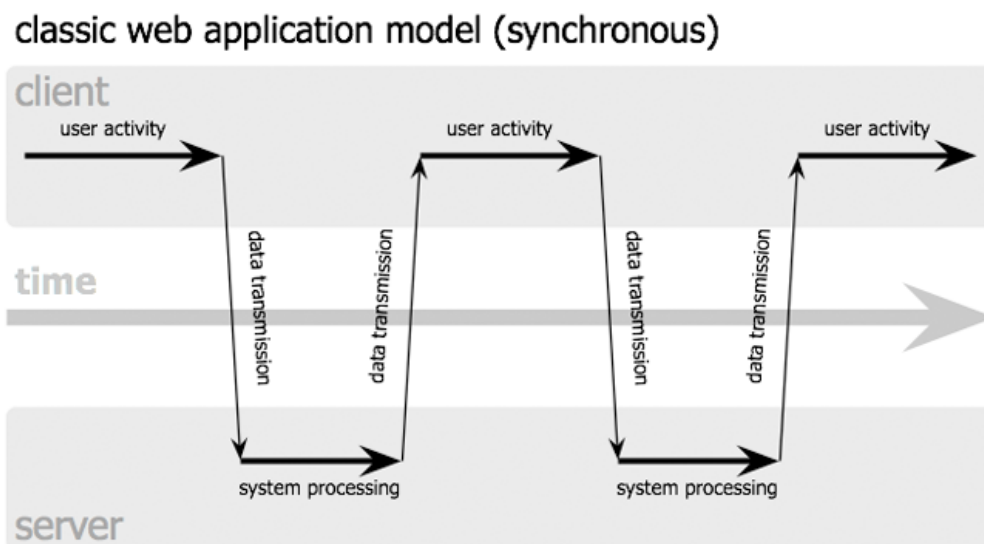


Abbildung 5: Request/Response-Verhalten im klassischen Web [Bosch 2007]

Ajax web application model (asynchronous)

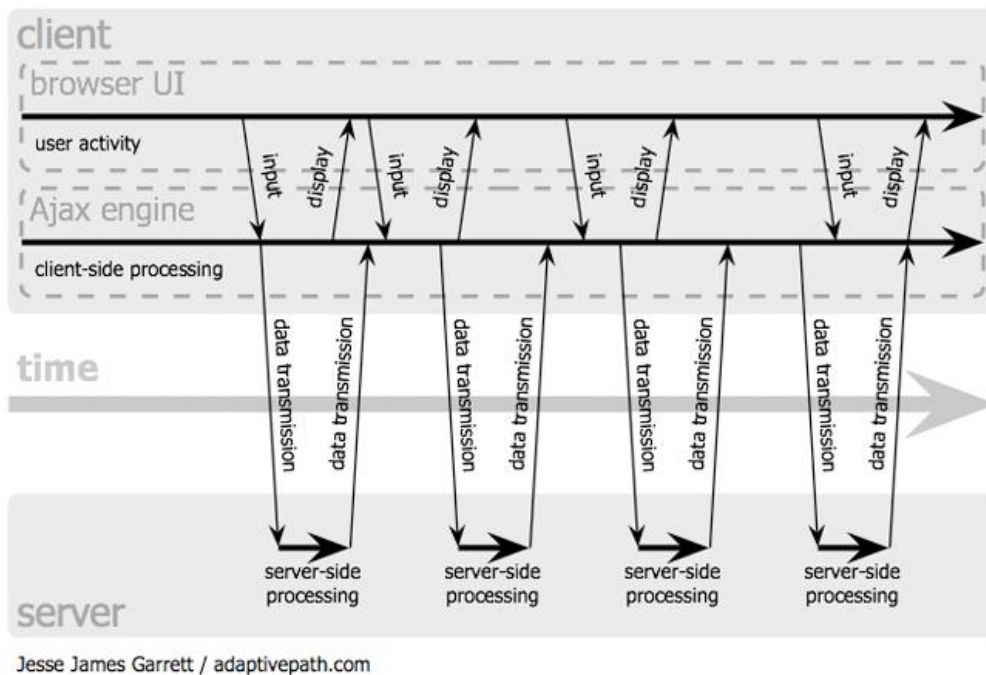


Abbildung 6: Request/Response-Verhalten beim Einsatz von Ajax [Bosch 2007]

Ajax-Befürworter weisen Ajax-Webanwendungen folgende positive Eigenschaften zu:

- Sie weisen eine bessere *Performance* als Web 1.0-Applikationen auf. Im Allgemeinen trifft dies zu, da erstens Ajax erlaubt auf die Benutzeraktionen schneller zu reagieren und zweitens muss nicht mehr gewartet werden bis die ganze Seite neu geladen wird.
- Sie sind im Allgemeinen schneller, da sie nur die minimale Menge an Daten anfordern und senden. Dadurch wird auch die Netzwerkbelastung verringert.
- Webentwickler sind im Allgemeinen schon mit den komponierenden Programmiersprachen von Ajax vertraut. Der Lernaufwand wird dadurch reduziert.
- Ajax läuft auf allen Plattformen. Der Webentwickler kann seine Applikation einmal für alle verschiedenen Systeme programmieren. Der Benutzer braucht auch keine Plug-Ins auf seinem Computer zu installieren.

Folgende Limitierungen wurden auch bekannt:

- Der intensive Einsatz von Ajax kann schnell einen hohen Grad an Komplexität aufweisen. Deswegen sind viele Toolkits und Frameworks ausgekommen, um die Entwicklung zu erleichtern. Ruby on Rails (Vgl. Kapitel 4.) ist ein Beispiel solches Frameworks. Die Ajax-Technologie ist aber noch nicht reif und benötigt weitere Toolkits und Frameworks.
- Der Testaufwand ist auch höher bei einer Ajax-Webanwendung, da alle „Ajaxifizierten“ Funktionen mit unterschiedlichen Browsern auf unterschiedlichen Betriebssystemen getestet werden müssen.
- Was der Sicherheit von Ajax-Applikationen angeht, gibt es auch gewisse Bedenken. Die Ajax-Komponente werden zwar seit Jahren benutzt, aber sie werden jetzt auf eine ganz andere unerprobte Weise eingesetzt, die mögliche Sicherheitslücken bergen können. Dem Web-Browser kommen auch immer mehr Aufgaben zu als das reine Anzeigen statischer Webseiten. Alle diese neuen Aufgaben sollten sich auch in ein neues Sicherheitsmodell des Browsers widerspiegeln.
- Kompatibilitätsprobleme in der Implementierung des JavaScripts zwischen den verschiedenen Browsern bilden ein Problem, das von Hand schwerfällig zu lösen ist. Zum Glück können Frameworks eingesetzt werden, die die gewünscht Funktionalität für die verschiedene Browser selber programmiert.
- Die Benutzer dürfen das JavaScript in ihren Browser nicht deaktivieren.
- Die Benutzer müssen sich an Ajax-Anwendungen gewöhnen und sie annehmen.
- Suchmaschinen können im Allgemeinen Ajax-Anwendungen schlecht indexieren. Das Setzen von Bookmarks und die Verwendung des Browser-Back-Buttons kann auch problematisch werden.
- Ajax ist auch nicht für Audio- und Video-Streaming geeignet, da es weder ein Audio noch ein Video-API besitzt. Sein Konkurrent Flash ist in dieser Hinsicht besser [Bosch 2007].

4. Webframeworks

4.1. Webframeworks: Grundlagen und Beispiele

Es ist durchaus möglich eine Web 2.0-Anwendung mithilfe einer Text-Anwendung und von Hand zu programmieren. Der Aufwand und der Resultat der Arbeit werden aber sogar die Mutigsten enttäuschen. Die komplexeren Programmier Techniken des Web 2.0 haben eine Explosion der Anzahl der Webframeworks zur Folge gehabt. Ein Webframework ist ein Programmiergerüst, welches Webentwickler neue Seiten und neue Funktionalitäten effizienter zu programmieren hilft.

Unter den Implementierungssprachen für RIA hat Ajax bei weitem die grösste Anzahl an Frameworks. Die sogenannten „Ajax-Frameworks“ können in Client-Side und Server-Side Frameworks unterteilt werden. Ein Client-Side Ajax-Framework ist meistens nur eine JavaScript-Datei, welche der User herunterlädt und auf welche die Webanwendung hinweist. Diese Datei kann eine objektorientierte JavaScript-Klassenbibliothek enthalten, die verschiedene Programmierbedürfnisse vereinfachen. Diese Klassen können eine Unterstützung für beispielsweise Ereignisse, Netzwerk-Kommunikation, Simple Object Access Protocol (SOAP), Cookies, visuelle Interfaces (z.B. Drag&Drop-Funktionalitäten) und ein XHR-Wrapper anbieten. Dank der XHR-Wrapper braucht der Entwickler sich nicht mehr um die Kompatibilität seiner Anwendung mit den verschiedenen Browsers (und somit verschiedene Implementierungen der XHR) Sorgen zu machen. Der Entwickler programmiert gegen den XHR-Wrapper welcher die Funktionaufrufe zur jeweiligen Browserabhängigen XHR-Implementierung weiterleitet. Bekannte Beispiele von Client-Seitige Ajax-Framework sind Dojo und Script.aculo.us; unter [Ajaxpatterns] werden weitere andere Frameworks aufgelistet. [Vossen und Hagemann 2007]

Obwohl einen grossen Teil von der Ajax-Technologie auf dem Client stattfindet, muss ein Teil der Anwendung für den Server programmiert werden, welcher den HTTP-Request des Clients entgegennimmt und eine Antwort zurücksendet. Im Gegensatz zur Client-Seite kann die Server-Seite in unterschiedlichen Technologien umgesetzt werden:

- PHP
- Visual Basic (VB)
- Java
- Ruby

sind nur einige bekannte Beispiele. Server-Seitige Frameworks bieten unter anderem vordefinierte Schnittstellen für den Empfang der HTTP-Request; so kann eine serverseitige Routine einfach angesprochen und ausgeführt werden. Beispiele von Ajax-Framework für die Programmiersprache Java sind Direct Web Remoting (DWR) und Ajax4Jsf. Diese zwei Frameworks decken sowohl die Clientseite wie auch die Serverseite ab [Bosch 2007].

4.2. *Ruby on Rails (ROR)*

Ruby on Rails ist einer der Leader in der Kategorie der Server-seitigen Frameworks des Web 2.0. und ist bis jetzt das einzige Framework für die Programmiersprache Ruby. Er unterstützt seit seinem Anfang die Programmierung mit Ajax indem er das clientseitige Framework *Prototype* und andere weitere kleine JavaScript Bibliotheken beinhaltet. Das Ziel von ROR ist den Programmierungsprozess zu vereinfachen; dafür macht er die Einfachheit, Wiederverwendbarkeit, Erweiterbarkeit, Testbarkeit, Produktivität und Wartbarkeit in der Anwendungsentwicklung zur Priorität. Sechs Prinzipien tragen zu diesen Zielen bedeutsam bei.

- **Model-View-Controller (MVC):** ist ein architektureales Pattern für Software Design, welches auf die Unterscheidung zwischen Datenmodellierung (*Data Model*), Darstellung (*View*) und Programm Kontrolle (*Controller*) basiert. Das *Modell* ist verantwortlich für die Daten, Zustand und Business Logik, während das *View* sich um das User Interface kümmert. Der *Controller* ist in der Mitte und übersetzt die Interaktion mit dem *User Interface* in eine Interaktion mit dem *Modell*. Diese Unterteilung bringt mehrere Vorteile mit sich: erstens wird der Code und die Architektur einfacher, flexibler und erweiterbarer, da die Aktivitätsbereiche in drei Komponenten unterteilt werden. Zweitens wird die Arbeitsverteilung zwischen dem Web Designer und dem Webentwickler erleichtert: der erste kann sich um dem Entwurf von HTML-Seiten kümmern während der zweite für die Programmierung der Anwendung verantwortlich ist.

- **Don't repeat yourself (DRY):** Dieses Prinzip verlangt, dass weder Daten noch Funktionalitäten mehr als einmal in der Anwendung implementiert werden. Die positive Konsequenz dieses Prinzips ist, dass die Komplexität des Codes verringert wird, da nur an einer Stelle der Code verändert werden muss, um eine Veränderung vollkommen zu implementieren. Dieses sollte auch zu einer Kostensenkung in der Wartbarkeit der Applikation führen.
- **Convention over configuration:** ROR weicht Konfigurationen im breiten Sinn des Wortes aus. Die einzige Datei, die zu konfigurieren ist, ist *database.yml*. Sie beinhaltet die Informationen über die Datenbankverbindung. ROR generiert sehr viel Code automatisch aufgrund von Vereinbarungen; Programmierer können natürlich diese Vereinbarungen überschreiben, um ihre Anwendungen auf ihre Bedürfnisse anzupassen. Ein Beispiel von über Vereinbarung automatisch generiertem Code sind Testsuites. Tests werden von ROR für jeden Controller und jedes Modell standardmässig generiert.
- **Scaffolding** (zu Deutsch: Baugerüst): ist eine Art Metaprogrammierung, welche ROR erlaubt aufgrund vom Datenbankschema die passenden Controller und Views automatisch zu generieren. Der Programmierer braucht ROR nur ein paar Instruktionen anzugeben, um eine laufende Anwendung zustande zu bringen.
- **Sofortiges Feedback:** Der Entwickler braucht nur seine Anwendung in dem Webbrowser zu aktualisieren, um sich die letzte Version seines Werk anzuschauen. Es wird keine Zeit weder für das Deployment noch für das Kompilieren benutzt.
- **Ruby:** ROR basiert auf die OO Scriptsprache Ruby. Mit vergleichsmässig wenig Code können Webentwickler viele Funktionalitäten anhand von in Blöcken kombinierten Schleifen und Tabellen implementieren. Ruby ist sehr einfach zu lesen und basiert hauptsächlich auf die Programmiersprachen Perl, Python, Smalltalk und Lisp.

ROR besteht aus drei Unter-Frameworks, welche jeder einem Teil des MVC entspricht:

- **Active Record:** stellt die Verbindung zwischen der Datenbank und den Objekten/Klassen der Applikation her. Active Record folgt dem Object/Relational Mapping (ORM) Pattern: eine Abbildung (Mapping) wird zwischen den Datenbanktabellen und den Klassen, und, zwischen den Reihen der Datenbanktabellen und der Objekte stattfinden. Eine Abstraktionslayer wird so geschaffen, und erlaubt

einerseits die Anwendung unabhängiger von der Datenbank zu halten und andererseits eine objektorientierte Handhabung der Daten. Structured Query Language (SQL) – Anfragen können durch Funktionen, die von Active Record zur Verfügung gestellt werden, ersetzt werden und so wird es dem Entwickler möglich nur Objekte handzuhaben, um seine Anwendung zu programmieren.

- **Action Controller:** ist die Steuerungszentrale. Innerhalb eines Controllers definiert der Programmierer verschiedene Aktionen und entsprechende Views werden in der Action View angelegt. Wenn der Client einen HTTP-Request zum Webserver sendet, wird es zuerst durch ein Routing entschieden zu welchem Controller die Anfrage weitergeleitet werden muss. Beim Controller wird die Aktion ausgeführt, die in der URL angegeben worden ist. Parametern können auch in der URL vorkommen. Der Controller kann noch eine Verbindung zum Active Record herstellen, um beispielsweise Daten für das View zu Verfügung zu stellen. Schliesslich wird die View zurückgeschickt, die der Aktion entspricht. Abbildung 4.2.1. stellt dar, wie die verschiedenen Komponenten zusammenarbeiten. In dem Beispiel ist aus dem angefragten Link ersichtlich, dass das Request zum Controller „Journal“ weitergeleitet wird und dort die Aktion „Show“ mit dem Parameter 1 auslösen wird.
- **Action View:** stellt das User Interface dar. Die Seiten werden meistens als RHTML oder html.erb programmiert. Diese stellen eine Mischung aus HTML und Ruby-Skript dar. Die dynamischen Teile werden durch Ruby-Skript implementiert [Chle und Kirchberg 2007].

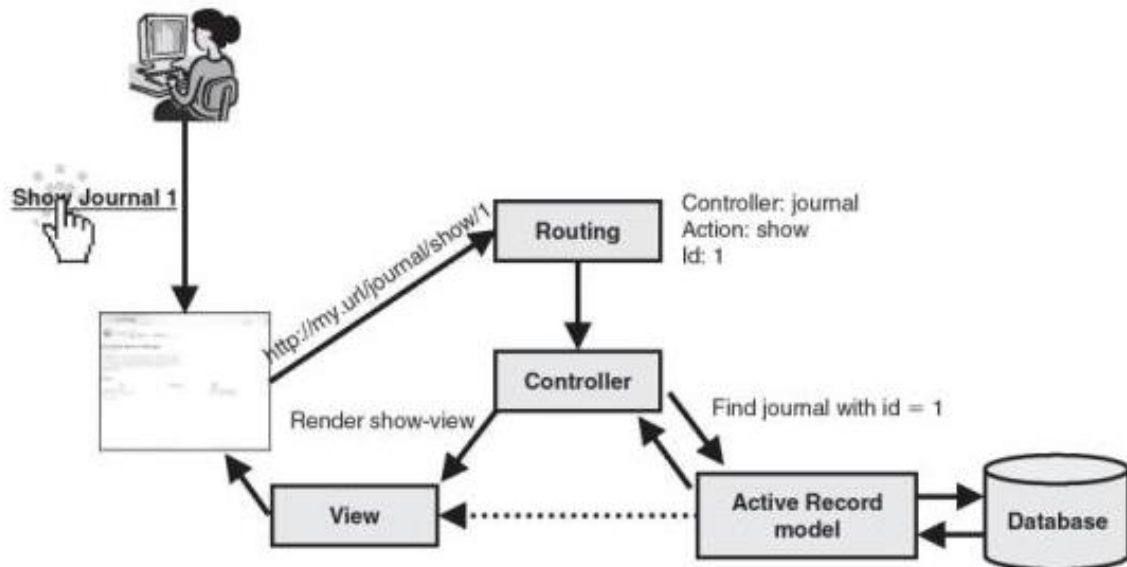


Abbildung 7: Überblick von ROR [Vossen und Hagemann 2007]

4.3. FriErasmus: Beispiel einer Rails-Applikation

FriErasmus ist eine Applikation, die für den Informationsaustausch zwischen ehemaligen Erasmusstudenten der Universität Freiburg an ausländischen Universitäten und Studenten, die einen Erasmusaufenthalt planen, dient. Diese Webapplikation ist eine typische Web 2.0 Anwendung, in welcher der Inhalt von Usern für User gedacht ist. Die beste Quelle für den Inhalt von so einer Seite ist unter den ehemaligen freiburgische Erasmusstudenten verteilt und kann so schlecht von den Universitätsangestellten zur Verfügung gestellt werden.

FriErasmus wurde als praktische Arbeit zu diesem Seminararbeit in Ruby Version 1.8.6. programmiert. Sie basiert auf dem ROR-Framework Version 2.1.0 und wurde mit einer MySQL-Datenbank Version 5.0.45 (für Mac OS X 10.4 und i686 kompiliert) entwickelt. Installationsanweisungen befinden sich auf der beiliegenden CD-Rom.

Abbildung 4.3.1. zeigt die MVC-Struktur der Applikation. Diese wurde ohne Scaffolding implementiert. Der Controller „account“ wurde in Anlehnung an das Buch [Lenz 2007] implementiert. Er holt die benötigten Daten vom Modell „user“ ab. Wenn die Aktionen „login“ und „logout“ vom User angefragt werden, führt der Controller die entsprechende Funktion aus und gibt die gleichnamige View zurück. Wenn der User sich auf der Seite „register“ befindet und sich erfolgreich ein Konto anlegt, wird die Funktion „create“ vom

Controller „account“ ausgeführt und die View „index“ vom Controller „stories“ wird angezeigt. Wenn das Konto nicht angelegt werden konnte, wird die View „register“, mit den fehlenden oder fehlerbehafteten Feldern rot umrandet wiederangezeigt.

Der Controller „stories“ arbeitet mit den Modellen „story“, „country“, „city“ und „university“. Der Controller zeigt per Default die Seite „index“ an (siehe Abbildung 4.3.2.). In ihr wird es möglich durch Klicken auf den Link „do you want to register“, „Login“ oder „Logout“ zu den Views vom Controller „account“ zu gelangen. Es ist auch möglich zu den Views „bycountry“ und „show“ vom Controller „stories“ beim Klicken auf den entsprechenden Link zu gelangen. Um eine neue „story“ schreiben zu können, muss der User eingeloggt sein. Der Controller „stories“ filtert die User und zeigt nur den eingeloggten Usern die Option „create a new story“ an. Die Webseite ist hierarchisch strukturiert, so dass es möglich ist zuerst nach Countries (Seite „index“) zu filtern, nachher nach Cities (Seite „bycountry“), nach Universities (Seite „bycity“), nach Stories (Seite „byuniversity“) und schliesslich die Stories (Seite „show“) selber anzuzeigen. Ein zusätzlicher Teil der Business Logic wurde in der Datei `stories_helper.rb` programmiert. In ihr befinden sich die Funktion für das Ausrechnen der verschiedenen Ranking der Countries, Cities, Universities und Stories. Die Resultate der Auswertungen werden den Views zur Darstellung gestellt.

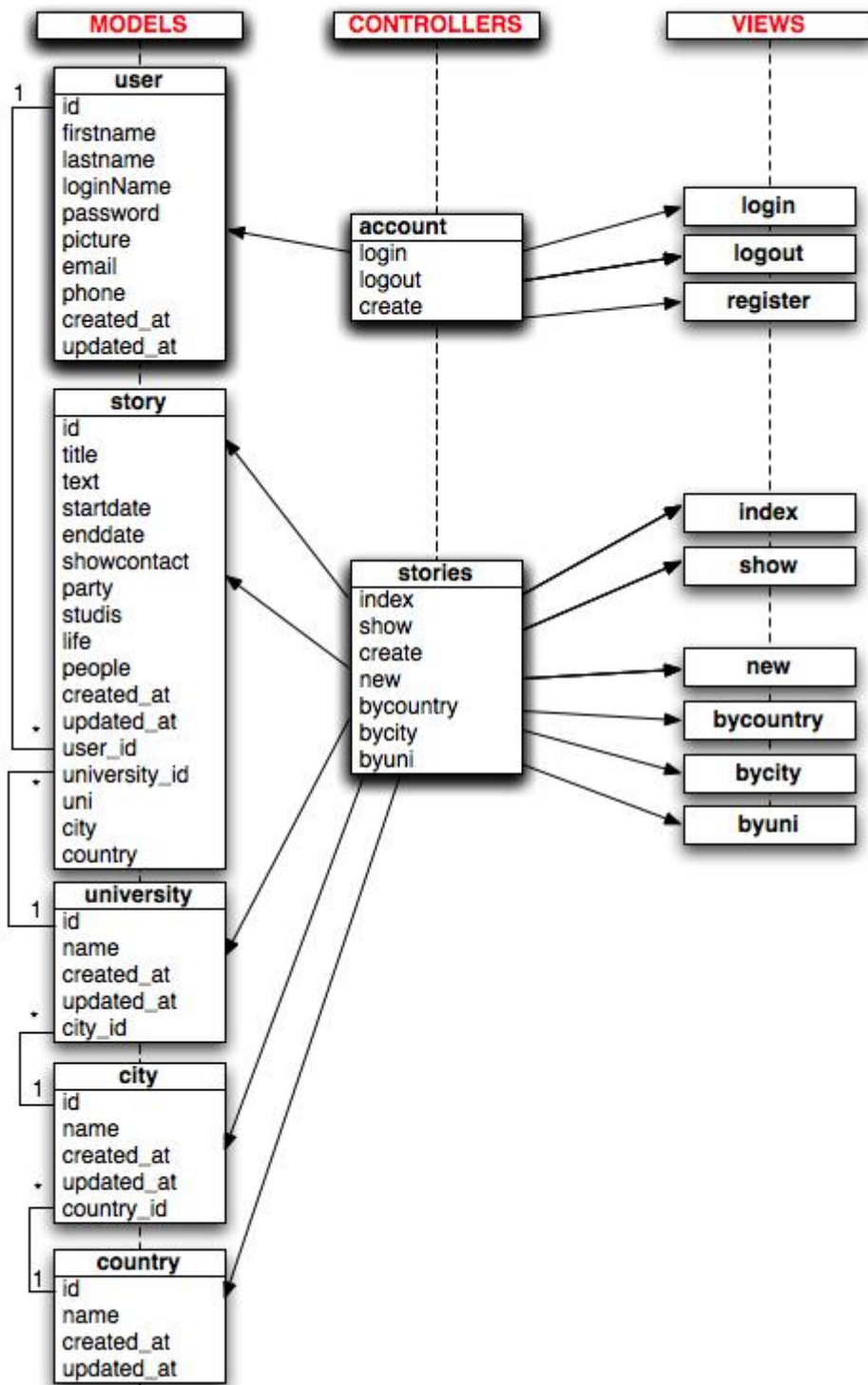



Abbildung 8: MVC-Struktur der Applikation FriErasmus

[do you want to register?](#) Not logged in. [Login](#)



Top 3 Countries	Rating
Spanien	5.0
Italy	2.67
France	2.25

[All Countries](#)

Welcome to FriErasmus, the site where you can share your experience abroad as an Erasmus student. In order to write a story yourself, please register!
Click on a country to see only the stories related to that country.

[France](#) [Italy](#) [Last](#) [RUBY](#) [Spanien](#)

Here is a list of all the stories written so far!

Title	University	Rating
fiesta	Universidad de Salamanca	5.0
FooBAR	Soeur Bonne	1.0
last	Last	1.0
Paris	La Soeur Bonne	1.0
Pasta and Pizza	Uni de Neapel	2.67

Abbildung 9: Die Hauptseite der Applikation FriErasmus

5. Zusammenfassung und Schlussfolgerung

Diese Seminararbeit hat sich mit der Thematik der Entwicklung der nativen Programmier-Techniken des Webs 2.0 auseinandergesetzt. Der Weg, der von dem statischen klassischen read-only Web-Erlebnis zum dynamischen Benutzerbeteiligten Web 2.0 führte, wurde beschrieben.

Nicht alle Techniken des Web 2.0 konnten in dieser Arbeit behandelt werden. Wichtige Techniken wie Mashups und RSS hätten noch die Webanwendung FriErasmus ergänzen können. Die Länder, Städte und Universitäten könnten beispielsweise auf einer GoogleMap Mashup angezeigt werden und die Users der FriErasmus könnten sich für ein RSS-Feed abonnieren. Die Benutzer der Seiten wären so immer auf dem Laufenden gehalten ohne immer auf die FriErasmus-Seite gehen zu müssen.

Die Programmierung mit ROR hat sich als tückischer erwiesen als angenommen. Im Hintergrund werden viele Automatismen von ROR ausgeführt, welche zwar die Arbeit erleichtern, wobei Abweichungen vom Standardmodell viel aufwendiger zu implementieren sind. Die mangelhafte Unterstützung von einer für Ruby bestimmte integrierter Entwicklungsumgebung verlangsamt ebenfalls den Programmierungsprozess. Zum Beispiel fehlen ein Autocompletion-Funktion, sowie eine hilfreicher Debugger. Hilfestellungen im Internet und in der Literatur behandeln oft unterschiedliche und nicht aktuelle Versionen von Ruby, welche inkompatibel sind. Zusätzlich muss auf dem Webserver Ruby on Rails (und zwar die richtige Version) vorhanden sein, um die Applikation zu benutzen.

Das Web 2.0 lebt von der Teilnahme der Webbenutzer, diese sind nämlich verantwortlich für die Erzeugung der Inhalte. Webseiten wie beispielsweise Facebook oder YouTube können durch die neusten Technologien attraktiver gemacht werden, aber ohne regelmässige Content-Aktualisierung von den Usern könnten sie nicht überleben. Auf einer Seite trägt der kontinuierliche Informationszufluss zu dem Erfolg des Webs bei; aber auf der anderen Seite droht einer Informationsüberfluss. Neue Lösungen werden benötigt, damit diese Unmenge von Daten effizienter gebraucht wird. Dieser neuen Aufgabe stellt sich das Web 3.0.

5.1. Ausblick: das Web 3.0

Im Hintergrund des Webs 2.0 tauchen neue Technologien auf, welche die nächste Webgeneration prägen werden. Journalist John Markoff bezeichnet in [Markoff 2006] diese nächste Entwicklung als „Web 3.0“. Diese sollte sich vor allem auf dem sogenannten *Semantic Web* stützen.

Das Ziel des *Semantic Web* ist, der riesigen Datenmenge, das im Web enthalten ist einen, für den Computer verständlichen, Sinn zu geben. Nur wenn die Daten von den Maschinen interpretiert werden können, werden diese in der Lage sein beispielsweise Homonyme auseinanderzuhalten, Synonyme als Äquivalente zu betrachten und auf einem höheren Niveau logisches Denken aufzuweisen. Das semantische Web stellt somit eine Symbiose von Webtechnologien und Wissensdarstellung (zu Englisch: Knowledge Representation) dar. Wissensdarstellung ist ein Untergebiet der Künstlichen Intelligenz, das sich damit beschäftigt Modellen von der Welt aufzubauen und aufrechtzuerhalten, die ermöglichen über sie selber und ihre assoziierten Informationen zu schlussfolgern.

Ontologien, Folksonomies und Microformate sind drei bekannte Ansätze, die heutzutage

gebraucht werden, um Daten zu strukturieren, zu klassifizieren und zu beschreiben. Mithilfe dieser Technologien werden ganz neue Möglichkeiten ermöglicht. Zum Beispiel werden Suchmaschinen fähig die Dokumente, die sich auf dem Intranet befinden zu lesen und genau den Inhalt zurückzugeben, der die Frage des Benutzers beantwortet. Die Kombination vom Semantic Web mit anderen Teilgebieten wie beispielsweise Ubiquitous Computing ist auch vielversprechend. Die zukünftigen Entwicklungen des Webs werden vielleicht zu einer weiteren sozialen Revolution führen.

Literaturverzeichnis

Ajaxpatterns: www.ajaxpatterns.org letzter Zugriff 31.07.2008.

Beck, A. (2007) Web 2.0: Konzepte, Technologie, Anwendungen. In HMD (255), S. 5-16.

Bosch, A. (2007) Ajax - Grundlagen und Funktionsweise. In HMD (255), S. 37-48.

Chle, M. und Kirchberg, P. (2007) Ruby on Rails. In Software Technology, December 2007, S. 105-108.

Farell, J. und Nezlek, G. S. (2007) Rich Internet Applications: The Next Stage of Application Development. In Int. Conf. on Information Technology Interfaces.

Friedman, T. L. (2005) The world is flat : a brief history of the twenty-first century. Farrar, Straus and Giroux, 1st, New York.

Khaled, O. A., Mugellini, E. und Felber, P. (2008) RIA: Ajax, XUL, FLEX, XAML. In Web Engineering: Modeling, Developing and Reasoning of Information Architecture, Fribourg.

Kollmann, T. (Hrsg.) (2007) Web 2.0 : Trends und Technologien im Kontext der Net Economy. Deutscher Universitäts-Verlag, Wiesbaden.

Lassila, O. und Hendler, J. (Hrsg.) (2007) Embracing "Web 3.0". IEEE, Charles Petrie.

Lenz, P. (2007) Build Your Own Ruby On Rails Web Applications. January 2007, Collingwood.

Markoff, J. (2006) Entrepreneurs See a Web Guided by Common Sense. In The New York Times 12.11.2006.

Paulson, L. D. (2005) Building Rich Web Applications with Ajax. In Computer, Vol. 38 (no. 10), Oct., 2005, S. 14-17.

Vossen, g. und Hagemann, S. (Hrsg.) (2007) Unleashing web 2.0 From Concepts to Creativity. Elsevier, Burlington.

Yakovlev, L. V. (2007) Web 2.0: Is It Evolutionary or Revolutionary? In IT Professional, Vol. 9 (6), Nov/Dec 2007, S. 43-45.

Zumstein, D. (2008) Web 2.0. Université de Fribourg, Fribourg.

Anhang

FriErasmus Installation

FriErasmus ist eine Ruby on Rails Webanwendung, welche auf einem Mac OS X Betriebssystem entwickelt wurde. Neben Ruby on Rails wurde noch eine MySQL Datenbank verwendet.

Ruby on Rails ist eigentlich plattformunabhängig, aber da unter Mac OS X 10.5 Ruby und Ruby on Rails schon vorhanden sind, erleichtert dies den Einstieg. Deshalb wird nur die Inbetriebnahme der Anwendung unter Mac OS X besprochen: diese Anleitung erklärt Schritt für Schritt, wie man FriErasmus auf einem Mac OS X 10.5 zum Laufen bringt.

MySQL 5.1

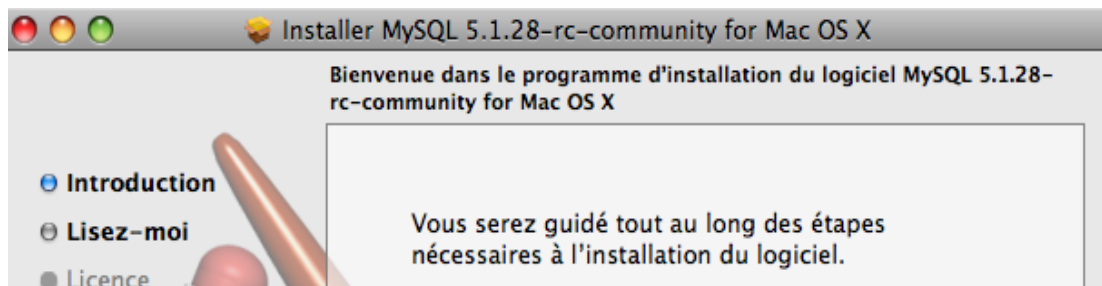
Falls auf dem Computer schon ein MySQL Server installiert ist, kann man diesen Schritt überspringen, und weiter zu Ruby gehen.

1. Den MySQL Community Server 5.1 herunterladen:

<http://dev.mysql.com/downloads/mysql/5.1.html>

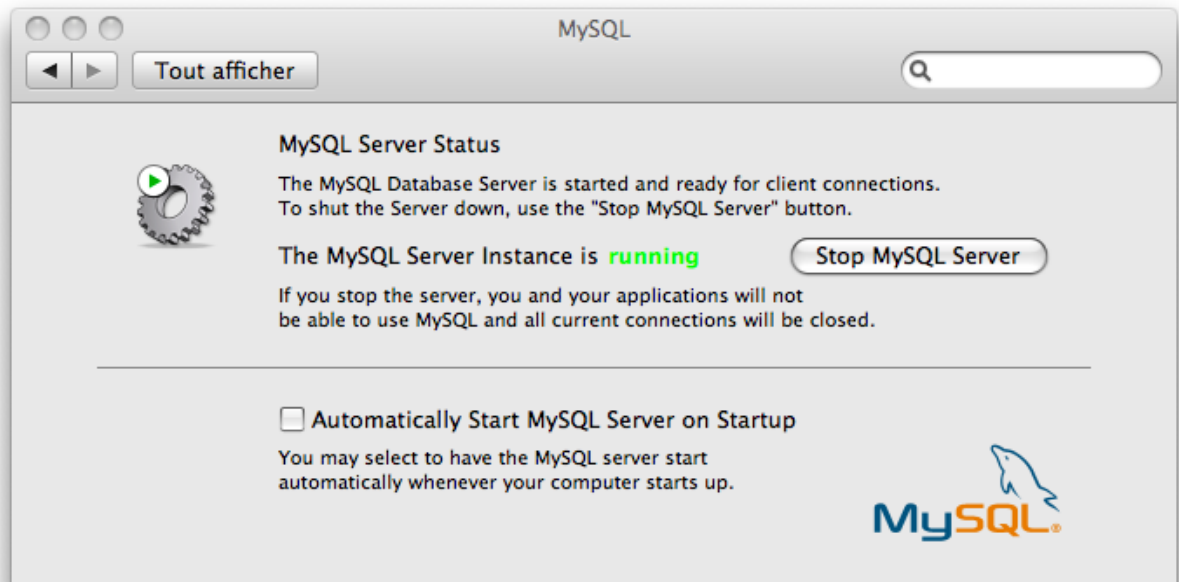
Es ist die entsprechende Architektur (Mac OSX x86) im „Package“ Format auszuwählen.

2. MySQL installieren:
 - a) Disk-Image (.dmg) öffnen
 - b) Package (.pkg) öffnen
 - c) Anweisungen des Installers folgen.



3. MySQL starten

In den Systemeinstellungen gibt es jetzt einen neuen Eintrag MySQL: in diesem Menü kann man den MySQL Server starten.



4. MySQL Administrator herunterladen:

<http://dev.mysql.com/downloads/gui-tools/5.0.html>

MySQL Administrator ist Teil eines Pakets namens MySQL GUI Tools. Hier kann man auch die Version für Mac OS X 10.4 verwenden.

5. MySQL Administrator installieren:

- a) Disk-Image (.dmg) öffnen
- b) Die Anwendung MySQL Administrator nach /Applications kopieren

Ruby und Ruby on Rails

Mac OS X 10.5 „Leopard“ wird mit Ruby 1.8.6 und Ruby on Rails 1.2.6 geliefert. Da für Ruby on Rails oft Updates verfügbar sind, gibt es ein Update System:

1. Terminal öffnen
(/Applications/Utilities/Terminal)
2. folgende Befehle eingeben (wenn das Passwort gefragt wird, muss das Passwort des eingeloggten Users eingegeben werden)

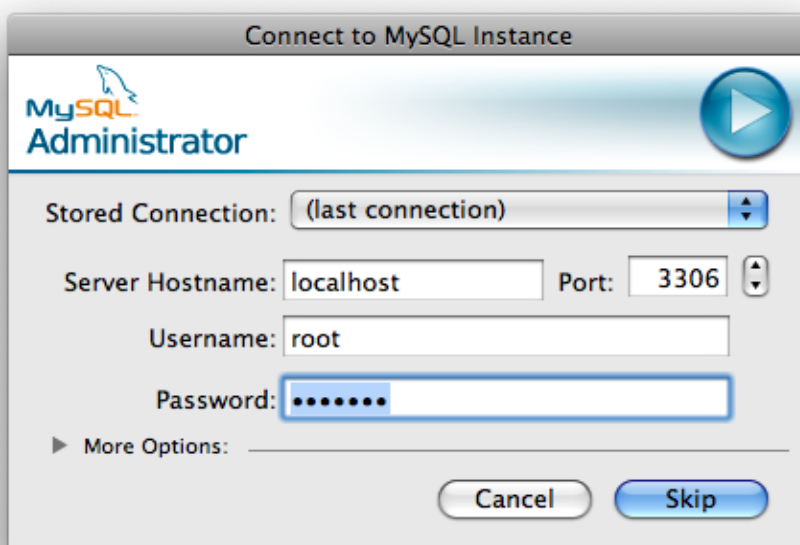
```
sudo gem update -system  
sudo gem install rails  
sudo gem update rake
```

FriErasmus Setup

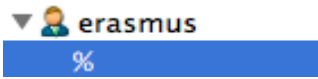
Natürlich kann man dieses Setup auch in der Kommandozeile eingeben, hier wird jedoch das Setup in den GUI Tools erläutert.

1. MySQL Administrator starten und mit dem lokalen MySQL Server verbinden. Standardmässig ist das root Passwort leer. Falls die Verbindung nicht klappt, im Terminal folgenden Befehl eingeben:

```
ln -s /var/mysql/mysql.sock /tmp/mysql.sock
```





2. Unter dem Menüpunkt „Catalogs“ folgende Datenbanken erstellen:
erasmus_development
erasmus_test
erasmus_production
3. Unter dem Menüpunkt „Accounts“ folgenden User einrichten:
Username: *erasmus*
Password: *erasmus*
4. Ist der User erstellt, kann man beim User-Eintrag auf ein Dropdown Menü klicken und den Eintrag „%“ auswählen.

5. Anschliessend kann man im Karteireiter „Schema Privileges“ die Datenbank *erasmus_development* auswählen und alle Rechte nach links verschieben.
6. Dasselbe ist für die Datenbanken *erasmus_test* und *erasmus_production* einzustellen.
7. Einstellungen speichern: „**save changes**“ klicken

Das Archiv FriErasmus.zip entpacken und im Terminal zum entpackten FriErasmus Ordner navigieren (zB. auf dem Desktop)

```
cd ~/Desktop/FriErasmus/
```

Die Tabellen werden automatisch mit folgenden Befehl erstellt:

```
rake db:migrate
```

Den Webserver mit der FriErasmus Anwendung kann man mit folgendem Befehl starten:

```
ruby script/server webrick
```

```
=> Booting WEBrick...
=> Rails 2.1.0 application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
[2008-09-16 15:35:34] INFO WEBrick 1.3.1
[2008-09-16 15:35:34] INFO ruby 1.8.6 (2008-03-03) [universal-darwin9.0]
[2008-09-16 15:35:34] INFO WEBrick::HTTPServer#start: pid=3770 port=3000
```

Der Webserver läuft nun unter der Adresse <http://0.0.0.0:3000/>

Die Webanwendung ist unter der Adresse <http://0.0.0.0:3000/stories> zu erreichen.