Diplomarbeit

Analyse von Ruby on Rails 3 Web Content Management Systemen

Stephan Keller

6. Oktober 2011

Hochschule für Technik, Wirtschaft und Kultur Leipzig (FH)

Dank an
Professor Dr. Ing. Robert Müller, meine Eltern Uta und Uwe Keller sowie meinem Bruder Michael Keller

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangslage

Die Skriptsprache PHP gehört weltweit zu den meist genutzten serverseitigen Programmiersprachen. Im August 2011 sind über 75 Prozent der dynamisch generierten Internetseiten mit dem PHP Hypertext Preprocessor erzeugt wurden[1].

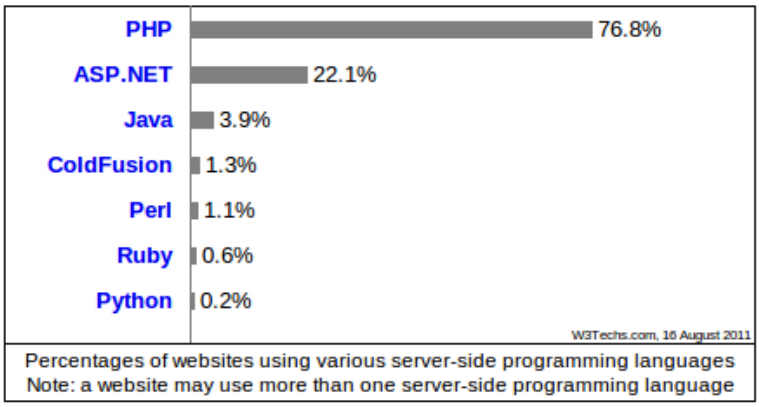

Abbildung 1.1: Nutzung verschiedener Programmiersprachen auf Servern

Auch im Bereich der Web Content Management Systeme[2] spiegelt sich diese Dominanz wider. Betrachtet man die Angaben des Content Management Portals cmsmatrix.org[3], existieren neben den vor allem in Deutschland verwendeten Open Source-Lösungen Typo3, Drupal, Contao oder Joomla! über 500 weitere in PHP implementierte Web Content

[1] W3tech erstellt täglich eine aktualisierte Auflistung über die Verwendung von serverseitigen Programmiersprachen. Es werden dabei die nach dem Alexia Ranking eine Million beliebtesten Internetseiten auf ihre Konfiguration untersucht.

[2] Im folgenden wird für den Begriff Web Content Management Systeme die Abkürzung WCMS verwendet.

[3] http://cmsmatrix.org ermöglicht eine Gegenüberstellung der Funktionalitäten von Content Management Systemen unterschiedlicher Prgrammiersprachen.

Management Systeme unterschiedlichster Ausprägung und Qualität. Ruby als Programmiersprache findet hingegen nur bei etwa 1 Prozent der erfassten Server Verwendung. Die dabei umgesetzten Projekte sind meist individuelle, browser-basierte Applikationen, die für Unternehmen und deren spezifisches Geschäftsfeld entwickelt wurden. Bekannte Vertreter sind hier u.a. die webbasierte Projektmanagement-Applikation Basecamp von 37signals[4], der Microblogging-Dienst Twitter[5] und der webbasierte Hosting-Dienst Github[6] für Software-Entwicklungsprojekte. Diese individuellen Lösungen werden dabei meist unter Zuhilfenahme eines Web Application Framework realisiert, das den Entwicklungsprozess unterstützt und vereinfacht.

1.2 Motivation und Zielsetzung

Ruby on Rails[7] hat sich seit der Veröffentlichung der Version 1.0 im Juli 2004 zu einem der bekanntesten Webframeworks der Ruby Fangemeinde entwickelt. Startups[8] sowie etablierte Unternehmen greifen zunehmend auf das Rails Framework zurück, um ihre webbasierten Geschäftsideen und -modelle zu realisieren. Wird neben der Webapplikation zusätzlich eine Internetseite zur Repräsentierung der Unternehmung benötigt, haben sich in der Praxis folgende zwei Lösungsansätze herausgebildet:

1. Bei geringem Umfang der zusätzlichen Internetseite werden die Inhalte manuell in HTML-Dateien angelegt und anschließend in die Rails-Anwendung integriert. Komfortable Möglichkeiten der Content-Verwaltung werden nicht angeboten oder später rudimentär nach implementiert. Änderungen der Inhalte sind teilweise mit erhöhtem Aufwand verbunden oder erfordern zusätzliche Programmierkenntnisse[9].

2. Komplexe Internetseiten mit vielen Inhalten und anspruchsvollem Layout werden über ein Web Content Management System eines Drittanbieters realisiert. Die

[4]Projektseite von Basecamp: http://basecamphq.com/

[5] Großteile der Programmierung von Twitter basierten bis April 2011 auf dem Ruby on Rails Framework.

[6]Github greift neben Ruby on Rails noch auf andere Webframeworks und Technologiesysteme zurück.

[7]Im weiteren Verlauf dieser Arbeit wird für das Webframework Ruby on Rails die Kurzform Rails verwendet.

[8]Der Begriff Startup bezeichnet hier junge Unternehmen, die sich mit ihrem neuartigen, meist innovativen Produkt noch nicht am Markt etabliert haben.

[9]Änderungen am Quellcode von Rails-Anwendungen im Produktivmodus erfordern immer einen Neustart des Servers.

Rails-Anwendung fungiert als Zwischenstation und leitet bestimmte Anfragen an das externe WCMS weiter.

Während der erste Lösungsansatz bei wenigen Inhalten noch vertretbar ist, erfordert die Verwendung eines externen WCMS zusätzlichen Installations- und Wartungsaufwand. Weiterhin erhöht sich der Bedarf an Programmierern, da neben Ruby nun auch andere Programmiersprachen Verwendung finden können.

Ziel der vorliegende Arbeit ist es daher, die Möglichkeiten einer komplett rails-basierten Web Content Management Verwaltung zu untersuchen, um so den Einsatz eines externen WCMS überflüssig werden zu lassen.

Dafür werden unter Verwendung der vergleichenden Methode die ausgewählten Ruby on Rails Content Management Systeme Alchemy CMS, Browser CMS, Locomotive CMS und Refinery CMS einem externen Kriterienkatalog gegenübergestellt, der allgemein gültige Anforderungen an Web Content Management Systeme formuliert. An Hand der Ergebnisse des Vergleichs kann abschließend eine Leistungsbeurteilung für die gewählten Systeme erfolgen und in wie weit sich diese für den Einsatz im Bereich des Webpublishing eignen. Zusätzlich wird die programmiertechnische Umsetzung der Systeme analysiert und auf mögliche Probleme hingewiesen.

1.3 Aufbau der Arbeit

Die vorliegende Diplomarbeit gliedert sich in sechs wesentliche Abschnitte:

Im ersten Abschnitt, in Kapitel 2, werden die für die funktionelle und programmiertechnische Analyse der Systeme notwendigen theoretischen Grundlagen zu Web Content Management Systemen und dem Ruby on Rails Webframework geschaffen. Darüber hinaus wird der für die Ermittlung der Leistungsfähigkeit der Systeme verwendete Kriterienkatalog vorgestellt.

Im zweiten Abschnitt, in Kapitel 3, folgt die funktionale Analyse der ausgewählten Ruby on Rails 3 Web Content Management Systeme Alchemy CMS, Refinery CMS, Browser CMS und Lokomotive CMS. Dazu werden die Kriterien des im Kapitel 2 vorgestellten

Katalogs mit den tatsächlich gebotenen Funktionalitäten der gewählten WCMS verglichen. Die Untersuchung schließt mit einer Zusammenfassung der Ergebnisse und einer Einschätzung für den Einsatz der WCMS ab.

Kapitel 4 überprüft die analysierten WCM-Systeme auf vorhandene konzeptionelle und programmiertechnische Schwachstellen. Darauf aufbauend werden in Kapitel 5 mögliche Lösungsansätze demonstriert und die dafür notwendigen theoretischen Grundlagen herausgearbeitet. Kapitel 6 schließt die Arbeit mit einer Zusammenfassung der herausgearbeiteten Ergebnisse ab und gibt Ausblicke auf zukünftige Entwicklungspotenziale.

2 Grundlagen

2.1 Entwicklung mit Ruby on Rails

2004 arbeitete der dänische Programmierer David Heinemeier Hansson an der Umsetzung eines webbasierten Projektmanagement-Tools mit dem Namen Basecamp[1]. Die bei der Realisierung des Projektes umgesetzten Teilkomponenten extrahierte er später und veröffentlicht sie 2005 als Framework unter dem Namen Ruby on Rails.

6 Jahre später beschreibt sich das Ruby on Rails Framework selbst mit folgenden Worten:

> Ruby on Rails is an open-source web framework that's optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration.[Rai11a]

Die Aussage bezieht sich dabei auf viele Ansätze und Entwicklungsabläufe, die innerhalb des Frameworks umgesetzt werden. Im folgenden Abschnitt sollen daher die wichtigsten Prinzipien, Paradigmen und Programmierabläufe des Frameworks zusammengefasst werden, um auf dieser Grundlage eine konzeptionelle und programmiertechnische Betrachtung in Kapitel 4 zu ermöglichen.

Für eine umfassende Einführung in Rails werden [Per10] und [Sch08] empfohlen.

2.1.1 Don't-Repeat-Yourself (DRY)

Zur Optimierung der Entwicklungvorgänge innerhalb des Frameworks propagiert Rails den Grundsatz des DRY (Don't-Repeat-Yourself). Dabei sollen Redundanzen, d.h. die wiederholte Angabe identischer Informationen jeglicher Art vermieden werden. So kann sichergestellt werden, dass sich Änderungen an einer zentralen Stelle im System (z.B.

[1] Projekt-Homepage: http://basecamphq.com/

Quellcode) in der gesamten Anwendung auswirken und Duplikate nicht mehrfach angepasst werden müssen.

2.1.2 Convention over Configuration

Viele Web Frameworks müssen vor ihrer Nutzung erst mit Hilfe zahlreicher Konfigurationsdateien und endloser Parametereinstellungen zu einem lauffähigen Gesamtsystem zusammengebaut werden[2]. Abbildung 2.1 zeigt beispielhaft solch eine Konfigurationsdatei innerhalb des Java Application Frameworks Spring[3]:

Quelltext 2.1: Konfigurationsdatei im Java Spring Framework

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xmlns:p="http://www.springframework.org/schema/p"
5         xmlns:context="http://www.springframework.org/schema/context"
6         xmlns:mvc="http://www.springframework.org/schema/mvc"
7         xsi:schemaLocation="http://www.springframework.org/schema/beans
8                             http://www.springframework.org/schema/beans/spring-beans.xsd
9                             http://www.springframework.org/schema/context
10                            http://www.springframework.org/schema/context/spring-context
                                  .xsd
11                            http://www.springframework.org/schema/mvc
12                            http://www.springframework.org/schema/mvc/spring-mvc.xsd">
13     <mvc:annotation-driven />
14     <mvc:view-controller path="/index.html" />
15     <bean id="messageSource" class="org.springframework.context.support.
           ResourceBundleMessageSource"
16           p:basenames="messages" />
17     <!-- Declare the Interceptor -->
18     <mvc:interceptors>
19         <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"
20           p:paramName="locale" />
21     </mvc:interceptors>
22 </beans>
```

Um diesen zusätzlichen und zeitraubenden Aufwand vor der eigentlichen Arbeit mit einem Framework zu vermeiden, definiert das Rails Framework zahlreiche Konventionen, die es erlauben, sofort mit der Entwicklungsarbeit zu beginnen. U.a. werden folgende Festlegungen getroffen:

[2] Rails bezeichnet diese Art der Frameworks mit ihrem Konfigurations-Overhead oft als *enterprisy*. Dies bedeutet jedoch nicht, das Rails für Anwendungsumsetzungen im Enterprise-Bereich ungeeignet ist. Matt Aimonetti stellt dies im Artikel [Aim10] heraus.

[3] Projektseite: http://www.springsource.org/

- Informationen zur Datenbankverbindung der Anwendung müssen in der Datei database.yml im Unterordner config hinterlegt werden
- Der Klassenname eines Domainenmodells wird im Singular erwartet, der dazu korrespondierende Tabellename in der Datenbank hingegen im Plural z.B. Domainenmodell Project => Datenbanktabelle projects
- Der Primärschlüssel in einer Datanbanktabelle muss vom Typ Integer sein und den Namen ID besitzen
- Rails erwartet eine definierte Ordnerstrukur für Controller, Domainmodell und Views[4]

Für den produktiven Einsatz des Frameworks müssen diese daher erlernt und akzeptiert werden, was dazu führt, dass Rails häufig als *opinionated software*[5] bezeichnet wird.

Ein Abweichen von den definierten Konventionen ist jederzeit möglich, erhöht jedoch den Aufwand des Entwicklers.

2.1.3 Model-View-Controller (MVC)

Für Software-Designer ist es eine gebräuchliche Technik, komplexe Software-Systeme durch Schichtenbildung in einzelne Bstandteile zu zerlegen [Fow03, Kapitel 1]. Das Ruby on Rails Framework baut ebenfalls auf einem Mehr-Schichten-Architektur-Modell auf. Zusätzlich kommt eine für das Rails-Framework spezifische Implementierung des bereits 1979 von dem Norweger Trygve Mikkjel Heyerdahl entwickelten Model-View-Controller-Paradigmas zum Einsatz[6]. Abbildung 2.1 charakterisiert den Ablauf einer Anfrage (Request) an eine Rails-Anwendung innerhalb des Client-Server-Modells:

1. Anfrage eines Clienten
2. An Hand der im Rails-Routing definierten Einträge wird die Anfrage eines Clienten an den registrierten Controller weitergeleitet

[4] Eine Erklärung zu Model-View-Controller folgt in Abschnitt 2.1.3

[5] Eine ausführliche Stellungnahme von David Heinemeier Hansson zu diesem Thema gibt es unter: http://www.linuxjournal.com/article/8686

[6] In der Fachliteratur wird das MVC-Paradigma häufig auf die Schichtenarchitektur einer Anwendung übertragen [BL06, S. 544 ff.]. Tatsächlich betrifft MVC in seiner ursprünglichen Form nur die Präsentationsschicht einer Webanwendung.

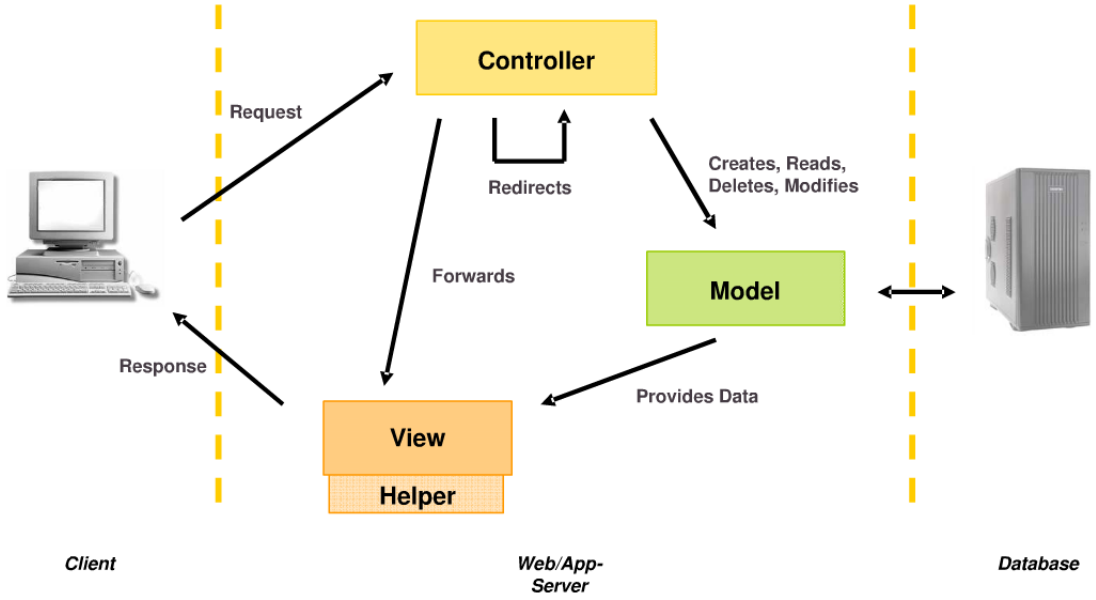

Abbildung 2.1: Abläufe innerhalb des Rails-Frameworks

3. Der Controller steuert den Ablauf innerhalb der Anwendung. Dabei greift er über ein Model auf benötigte Daten in einem Speicher (z.B. realtionale Datenbank) zu und stellt diese dem View-Layer zur Verfügung. Es ist auch möglich, dass der Controller die Anfrage an einen anderen Controller weiterleitet (Redirects).

4. Das Model bezeichnet in Rails Klassen, die einen Zugriff auf relationale Datenbanken oder andere Datenspeicher ermöglichen [HM10]

5. Der View-Layer bereitet die durch den Controller zur Verfügung gestellten Daten in der angeforderten Darstellungsform auf und gibt das Ergebnis der Anfrage aus. So wird je nach spezifiziertem Format z.B. eine HTML- oder XML-Datei erzeugt.

6. Das Ergebnis der Anfrage (die Ausgabe des View) wird vom Framework an den Server und von dort an den Clienten gesendet. Der Kommunikationsprozess mit dem Rails-Framework ist damit abgeschlossen.

2.1.4 REST

Innerhalb einer Web-Applikation erfolgt der Austausch zwischen Server und Client durch die Nutzung des HTTP-Protokolls. Dabei wird eine Anfrage (Request) an einen Server geschickt, bearbeitet und eine entsprechende Antwort (Response) mit den angeforderten Inhalten zurückliefert. Ein Großteil der Webanwendungen interpretiert dabei die im HTTP-Protokoll definierten Methoden GET und POST:

GET Anforderung an den Server, eine über die Adresszeile des Browsers angegebene Ressource zurückzuliefern. Es können zusätzlich Argumente an die angeforderte URL angehängt werden.

POST Mit Hilfe dieser Methode ist es möglich, große Datenmengen aus z.B. Formularen an einen Webserver zu verschicken. Die übergebenen Informationen werden dabei im sogenannten Body der Anfrage codiert mitverschickt und somit im Vergleich zu GET nicht in der URL sichtbar[7].

REST, ein Akronym für **RE**epresentational **S**tate **T**ransfer, erweitert die in traditionellen Webanwendungen üblichen GET und POST um die ebenfalls im HTTP-Standard enthaltenen Methoden PUT und DELETE:

PUT Die Verwendung der PUT-Methode zeigt die Neuanlage der in einer Anfrage spezifizierten Ressource an

DELETE Die Verwendung dieser Methode signalisiert dem Server, die angegebene Ressource auf dem Server zu löschen.

Da aktuelle Browser nur GET- und POST-Anfragen zuverlässig unterstützen, müssen entsprechende Anfragen zum Löschen und Verändern einer Ressource mit Hilfe von zusätzlichen Attributen in der Anfrage simuliert werden[8]. Das folgende Beispiel stellt das von Rails definierte Standard-Routing einer als *restful* angelegten Ressource Projekt dar[9].

[7] Ein Post-Request findet vor allem bei der Übermittlung von Formularen innerhalb eines Browsers Verwendung.

[8] Das Rails Framework fügt innerhalb von Formularen automatisch einen versteckten Parameter *_method* in die Anfrage, die den Namen der gewünschten HTTP-Metode (DELETE oder PUT) enthält. Im Framework wird dieser Parameter ausgelesen und ein entsprechendes Routing zur geforderten Controller-Action eingeleitet

[9] REST wird nicht über einen entsprechend formulierten Standard definiert. Vielmehr kann es als Programmierparadigma innerhalb von Web-Anwendungen verstanden werden, die eine Ansammlung von Best-Practices darstellen [Gü07].

Tabelle 2.1: Standard-Routing für eine Ressource Projekt innerhalb des Rails-Frameworks

HTTP-Methode	Anfragepfad	Ausgelöste Aktion im Controller	Wirkung
GET	/projects	index	Anzeige aller vorhandenen Projekte
GET	/projects/new	new	Anzeige eines HTML-Formulars zum erstellen eines neuen Projektes
POST	/projects	create	Erstellt ein neues Projekt mit den übermittelten Daten
GET	/projects/:id	show	Anzeige eines Projektes mit der zugeordneten ID
GET	/projects/:id/edit	edit	Anzeige eines HTML-Formulars zum Bearbeiten eines bestehenden Projektes
PUT	/projects/:id	update	Aktualisierung eines bestimmten Projektes mit den übermittelten Daten
DELETE	/projects/:id	destroy	Löschung des Projektes mit der angegebenen ID

Tabelle 2.2: Vergleich zwischen Rest-konformen und klassischen Rails-URL's

Aktion	normale URL	REST URL in Rails
show	/projects/show/12	/projects/12
delete	/projects/destroy/123	/projects/123
update	/projects/update/123	/projects/123
create	/projects/create	/projects

Tabelle 2.2 zeigt noch einmal die Trennung zwischen Ressource und Aktion innerhalb einer in Rails definierten REST-Ressource Projekt. Eine ausführliche Beschreibung von REST und dessen Realisierung liefert [Gü07].

2.1.5 Rack und Middleware

Rails ist innerhalb der Ruby-Gemeinde nicht das einzigste existierende Webframework. Mit den Projekten Sinatra[10], Merb[11], Camping[12] und Ramaze[13] stehen weitere Alternativen mit unterschiedlichen Ansätzen und Funktionsumfängen zur Verfügung. Bei der Implementierung dieser Frameworks müssen Entwickler wiederholt Adapter (Handler) zur Ansteuerung verschiedener Webserver entwickeln. Durch Rack[14], einem Ruby Webserver Interface, lässt sich die wiederholte Implementierung solcher Adapter in den einzelnen Frameworks vermeiden.

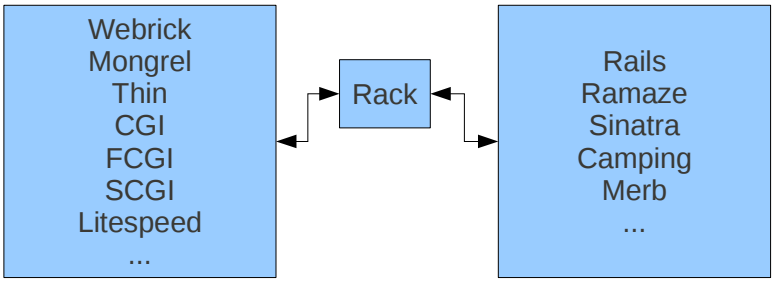

Abbildung 2.2: Rack als Vermittler (Middleware) zwischen Server und Ruby-Webframeworks

Damit Server und Frameworks untereinander kommunizieren können, muss eine Rack-Anwendung bestimmte Methoden implementieren und ein bestimmtes Rückgabeformat einhalten. An Hand des Beispiels in 2.2 sollen diese formalen Kriterien beschrieben werden:

Quelltext 2.2: Beispiel für eine einfache Rack-Anwendung

```
1 require 'rubygems'
2 require 'rack'
```

[10]Projektseite: http://www.sinatrarb.com/
[11]Projektseite: http://www.merbivore.com/
[12]Projektseite: http://camping.rubyforge.org/
[13]Projektseite: http://ramaze.net/
[14]Projektseite: http://rack.rubyforge.org/

```
3
4  class MyRackApp
5
6    def initialize(name)
7     @name = name
8    end
9
10   def call(env)
11     [200, {"Content-Type" => "text/plain"}, ["Hello #{@name}!"]]
12   end
13 end
14
15 Rack::Handler::WEBrick.run MyRackApp.new("Rack"),:Port => 3001
```

Zeile 6-8

Initialisierung der Rack-Anwendung MyRackApp unter Angabe eines zuvor übergebenen Namen (*Rack*).

Zeile 10-12

Implementierung der für eine Rack-Anwendung notwendigen Methode *call*. Diese wird vom Webserver beim Start einer Anfrage aufgerufen und muss als Antwort (Rückgabewert) einen Ruby-Hash mit den folgenden Informationen zurückliefern:

Status

Angabe des HTTP-Statuscode, *200* bedeutet in diesem Beispiel eine erfolgreiche Ausführung der Anfrage am Server[15]

Header

Angabe von im HTTP-Protokoll definierten Header-Informationen, im Beispiel wird ein einfaches Textdokument (*text/plain*) zurückgeliefert

Response

Die Antwort der Rack-Anwendung, im Beispiel wird ein String *Hello Rack!* an den Server übergeben und von diesem ausgeliefert.

Beim Aufruf der Methode *call* (Zeile 15) übergibt der Server eine Variable *env*, die alle wichtigen Informationen über die Serverumgebung und Anfrageparameter enthält.

Zeile 15

Start des Mongrel Webservers und Initialisierung der Rack-Anwendung *MyRackApp*.

[15] Die HTTP-Statuscodes sind im HTTP-Protokoll definiert und können u.a. unter folgender Quelle eingesehen werden: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Bei einem Aufruf der Adresse *localhost:3001* in einem Browser wird vom gestarteten Mongrel-Server der Rückgabewert der Methode *call* ausgegeben (Abb. 2.3).

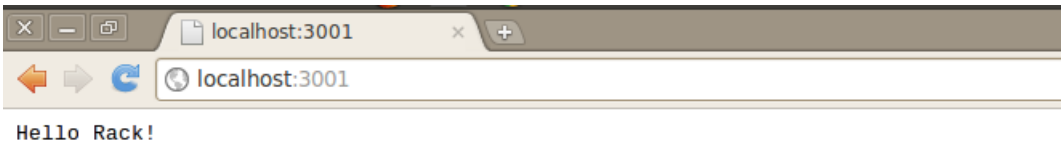

Abbildung 2.3: Ausgabe der Rack-Anwendung MyRackApp im Browser

Durch die Nutzung von Rack ergeben sich im Bereich der Ruby Webframeworks folgende Vorteile:

- Keine Codeduplizierung durch wiederholte Entwicklung von Server-Adaptern innerhalb der verschiedenen Ruby Webframeworks
- Ein Framework, das Rack unterstützt, kann sofort von allen anderen rack-unterstützenden Webservern verwendet werden
- Ein rack-unterstützender Server kann sofort mit allen rack-basierten Webframeworks eingesetzt werden.
- Vereinfachung der Webframework-Entwicklung
- Rack beinhaltet Handler für die meist verbreitesten Webserver in Ruby (CGI, FastCGI, SCGI, WEBrick, Mongrel, Litespeed u.a.)

Darüber hinaus ist es möglich, einzelne Rack-Anwendungen hintereinander zu schalten und so ein Filtersystem für ankommende Anfragen und ausgehende Antworten

vor dem eigentlichen Webframework zu realisieren. Diese Anwendungen werden Rack-Middlewares genannt und können in einer Rails-Anwendung in der Startkonfiguration des Frameworks eingebunden werden (Abb. 2.3). So kann das Ein- und Ausgabeverhalten des Railsframeworks gezielt verändert werden, da die einzelnen Rack-Anwendungen entscheiden, ob der nächste Filter (die nächste registrierte Rack-Middleware) oder eine vorzeitige Anwort an den Server geschickt werden soll.

Quelltext 2.3: Registrierung verschiedener Middlewares in einer Rails 3 Anwendung

```
1  # config/environment.rb
2
3  # Vorgefertigte Middleware aus dem Rack-Projekt
4  config.middleware.use Rack::BounceFavicon
5
6
7  config.middleware.use MyRackApp
8  config.middleware.use MyRackApp2
9  config.middleware.use MyRackApp3
10 config.middleware.use ...
```

Das Rails-Framework steht selbst am Ende der Filterliste und ist im Kontext von Rack selbst als Rack-Anwendung eingebunden (Abb. 2.4). Eine detaillierte Beschreibung von

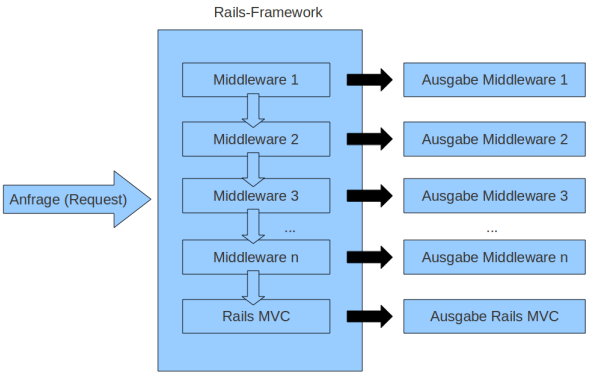

Abbildung 2.4: Möglichkeiten der unterschiedlichen Rückgabewerte durch Rack Middlewares und Rails

Rack und der Einrichtung eines komplexen Filtersystems[16] in einer Rails-Anwendung

[16]Rack stellt häufig verwendete Middlewares für verschiedene Aufgabenbereiche zur Verfügung. Die Projektseite ist unter folgender Adresse verfügbar: https://github.com/rack/rack-contrib

liefern [Neu07] und [Rai11b].

2.1.6 Generatoren

Das Ruby on Rails Framework unterstützt mit Hilfe sogenannter Generatorskripte die automatische Erzeugung von funktionsfähigem Quellcode. Der folgende Scaffold-Generator[17] erstellt z.B. nach dessen Initialisierung ein komplett funktionsbereites Codegerüst für die in Rails benötigten MVC-Komponenten einer Ressource *Projekt*.

Quelltext 2.4: Aufruf des Generators zur Erstellung einer MVC-Ressource Projekt

```
1 rails g scaffold project name:string description:text
```

Quelltext 2.5: Auflistung der erstellten Dateien des Scaffold-Generators

```
1  create db/migrate/20110926185944_create_projects.rb
2  create app/models/project.rb
3  create test/unit/project_test.rb
4  create test/fixtures/projects.yml
5  route  resources :projects
6  create app/controllers/projects_controller.rb
7  create app/views/projects
8  create app/views/projects/index.html.erb
9  create app/views/projects/edit.html.erb
10 create app/views/projects/show.html.erb
11 create app/views/projects/new.html.erb
12 create app/views/projects/_form.html.erb
13 create test/functional/projects_controller_test.rb
14 create app/helpers/projects_helper.rb
15 create test/unit/helpers/projects_helper_test.rb
16 create app/assets/javascripts/projects.js.coffee
17 create app/assets/stylesheets/projects.css.scss
18 create app/assets/stylesheets/scaffolds.css.scss
```

2.2 Web Content Management und Webpublishing

Der in der einschlägigen Literatur bereits mehrfach thematisierte Anstieg von Content[18] erfordert innerhalb der einzelnen Unternehmen ein entsprechend zielgerichtetes Management. Berechtenbreiter definiert den Begriff des Content Management mit folgenden Worten:

[17] Scaffold kann in diesem Zusammenhang mit dem Wort Grundgerüst übersetzt werden.

[18] Der Begriff des Content kann mit dem deutschen Wort Inhalt übersetzt werden und bezeichnet im Allgemeinen Informationen ...

> Content Management beschreibt die Planung, Verwaltung, Steuerung und Koordination aller Aktivitäten, die auf den Content und dessen Präsentation in Unternehmen abstellen [Ber04].

Im Bereich der internet-basierten Web-Sites und Portale hat dies zur Herausbildung des Web Content Managements geführt [Rig09, Seite 3]. Die dort verwendeten Lösungen

2.3 Externer Kriterienkatalog

Die hohe Zahl am Markt befindlicher Web Content Management Systeme führt zu einem erschwerten Auswahlverfahren. Neben vielen kostenpflichtigen, professionellen WCMS-Lösungen stehen durch die Open Source Bewegung zusätzlich zahlreiche kostenlose Softwareprodukte zur Verfügung, die sich in ihrer Leistungsfähigkeit stark unterscheiden. So kommt es häufig vor, das klein angelegte Open Source Projekte ihre Software stolz als Web Content Management System bezeichnen, obwohl nur sehr wenige Funktionalitäten implementiert sind. Um dieser Praktik entgegenzuwirken haben Vertreter der Content Management Branche eine Feature Matrix (Abb. 2.5) herausgegeben, die aktuelle Anforderungen an ein Content Management System spezifizieren soll. Die dabei entstandene Übersicht zeigt dabei eine Unterscheidung in 3 Prioritätsstufen:

Must-Have
: Diese Funktionalität muss in einem Content Management System verhanden sein.

Should-Have
: Diese Funktionalität ist nicht zwingend notwendig, kann bei entsprechender Existenz aber sehr positv wahrgenommen werden.

Nice-to-Have
: Funktionalitäten, die nur in wenigen, hochwertigen Systemen zur Verfügung stehen und über die gewöhnlichen Anforderungen hinausgehen.

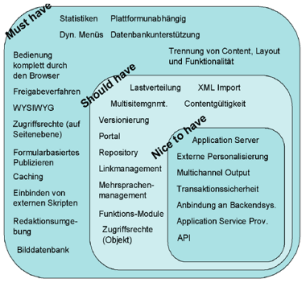

Abbildung 2.5: Feature-Matrix für Content Management Systeme

Die beschriebenen Funktionalitäten sind jedoch teilweise so allgemein formuliert, das dieser Ansatz nur als grobe Orientierungshilfe dienen kann. Der Wirtschaftsinformatiker Andreas Ritter hat daher im Rahmen seiner Bachelorarbeit *SWOT-Analyse zu*

Content-Management-Systemen einen Kriterienkatalog erarbeitet, an Hand deren die Leistungsfähigkeit von Web Content Management Systemen untersucht werden kann. Dieser orientiert sich dabei an den Prozessen des Content Managements und den bereits zuvor in Kapitel 2.2 beschriebenen Grundgedanken des Webpublishing.

Die von Ritter formulierten Kriterien werden in den Kapiteln 2.3.1 bis 2.3.5 nochmals aufgeführt und bilden die Grundlage für die Bewertung der Leistungsfähigkeit der ausgewählten Ruby on Rails Content Management Systeme.

2.3.1 Erstellung

- Mehrere Benutzer sollen gleichzeitig Inhalte verwalten und erfassen können
- Inhalte sollen – unabhängig von Zeit und Standort – durch mehrere Benutzer online verwaltet und erfasst werden können
- Für die Verwaltung und Erfassung von Inhalten sollen alle gängigen Internet-Browser (Internet Explorer, Safari und Firefox) eingesetzt werden können
- Offline Erfassung von Inhalten unter Verwendung eines lokal auf dem Rechner installierten Programms
- Inhalte sollen ohne spezielle Programmier / HTML-Kenntnisse erfasst und verwaltet werden können
- Integrierte Mediendatenbank zur Erfassung und Verwaltung von Bildern, Multimedia, Texte, Audio, Videos, usw.
- Inhalte (Texte, Bilder, Videos etc.) sollen zentral kategorisiert, erfasst und verwaltet werden können
- Inhalte sollen in einer Datenbank gespeichert werden
- Inhalte sollen mehrsprachig erfasst und verwaltet werden können
- Inhalte können während der Erfassung über eine Preview-Funktion vorab im Design der Webseite angesehen werden

- Zuordnung von standardisierten und frei definierbaren Metadaten zu Inhalten (z.B. Autor, Schlüsselwörter, benutzerdefinierte Felder) soll möglich sein
- Integration von Inhalten anderer Webseiten, Multimedia, Applikationen, E-Commerce-Tools
- Das CMS soll über eine offene API (Programmierschnittstelle) für individuelle Erweiterung verfügen
- Inhalte sollen einfach importiert / exportiert werden können - dabei kommen Formate wie z.B. XML zum Einsatz

2.3.2 Kontrolle

- Granulares Rechte- und Rollenkonzept für Anwender
- Granulares Berechtigungskonzept für einzelne Inhalte, Bereiche, Webseiten
- Schutz vor gegenseitigem Überschreiben erfasster Inhalte durch Check in/ Check out- Mechanismen
- Versionierung von Inhalten mit Möglichkeit zur Wiederherstellung vorhergehender Versionen
- Linküberprüfung: Automatische Prüfung der Gültigkeit von internen und externen Links, mit Möglichkeit zur Korrektur bzw. Benachrichtigung an eine definierte Personengruppe
- Mandantenfähigkeit: Mehrfachnutzung des Systems durch verschiedene Parteien mit kompletter Trennung der Daten und Benutzer

2.3.3 Freigabe

- Definition von Workflows inkl. mehrstufiger Freigabeprozesse für die Freischaltung von Inhalten
- Unternehmensspezifische Bearbeitungsprozesse von Inhalten, sollen über frei definierbare Workflows verwaltet werden können

- Möglichkeit für *nicht technische* User den Workflow zu kreieren, verwalten und ändern. Es soll dafür kein Scripting / Programming notwendig sein
- Möglichkeit externe Benutzer in Workflows mit einbinden zu können

2.3.4 Publikation

- Trennung von Inhalt und Design unter Verwendung von Templates
- Mehrfachverwendung von Inhalten an verschiedenen Stellen mit unterschiedlichem Layout
- Möglichkeit zur Wahl zwischen dynamischer oder statischer Generierung der Seiten / Inhalte
- Möglichkeit Inhalte für anderen Webseiten bereitzustellen (XML, Webservice)
- Navigationsstrukturen werden automatisch vom CMS generiert, publiziert und verwaltet
- Automatisches Anbieten von Druckversionen und Weiterempfehlen einer Webseite
- Inhalte sollen auf verschiedene Medien / Technologien (Cross Media Publishing, SMS / Mobile / WAP / usw.) ausgegeben werden können
- Einfache Einbindung von Fremdinhalten welche durch Drittanbieter zur Verfügung gestellt werden
- Schnittstellenunterstützung in Form von APIs sollen zur Verfügung stehen
- Barrierefreiheit bei den publizierten Seiten soll unterstützt werden

2.3.5 Terminierung und Archivierung

- Inhalte sollen archiviert werden können
- Freie Wahl des Publikationszeitraumes (zeitgesteuertes Auf- / Abschalten / Archivieren) von Inhalten

3 Funktionale Analyse der bestehenden Web Content Management Systeme

3.1 Vorbetrachtungen

Der 2006 eingeleitete Hype um das Ruby on Rails Framework hat dazu geführt, dass viele Entwickler mit der Konzeption und Umsetzung zahlreicher verschiedener Rails-Anwendungen begonnen haben. So entstanden auch im Bereich der Web Content Management Systeme zahlreiche Projekte. Ein Großteil der Vorhaben blieb jedoch in der Konzeptionsphase stecken oder die Entwicklung wurde nach wenigen Jahren eingestellt[1]. Die Ursachen sind dabei vor allem dem Entwicklungsumfeld von Rails und dem Framework selbst geschuldet:

Schnellebigkeit

Die Entwicklung des Ruby on Rails Frameworks unterliegt einem ständigen Wandel und erfordert eine ständige Anpassung des Programmierers an neue Technologien und Konzepte.

Interessenwandlung der Entwickler

Die Programmierung mit Ruby und die vielfältigen Möglichkeiten des Ruby on Rails Frameworks erleichtern die Umsetzung verschiedenster Projektideen. Es ist daher schneller möglich, dass sich Entwickler nach einiger Zeit mit anderen Projekten beschäftigen und bereits bestehende bzw. veröffentlichte Projekte vernachlässigen.

[1] Im Anhang der Arbeit findet sich eine Übersicht zu ermittelten Rails 2 und 3 Web Content Management Systemen. Diese sind zum Teil seit einigen Jahren nicht mehr weiterentwickelt wurden.

Die Realisierung eines stabilen Web Content Management Systems auf Basis von Ruby on Rails wird durch diese Faktoren erschwert und erfordert ein entsprechend tragfähiges Konzept sowie planendes Vorgehen der Projektiniziatoren.

Die hier aufgeführten Projekte Alchemy CMS, Browser CMS, Locomotive CMS und Refinery CMS repräsentieren daher die zum Zeitpunkt der Erstellung dieser Arbeit vielversprechendsten Rails-Implementierungen eines Web Content Management Systems.

Um die Zahl der zu untersuchenden WCMS einzuschränken, wurden folgende Mindestanforderungen für existierende Rails-Anwendungen festgelegt:

Open Source Software

Die hier ermittelten WCM-Systeme sind vollständige Open Source Lösungen. Ihre Veröffentlichung unterliegt dabei den in der Open Source Bewegung üblichen Lizenzen der Freien Software bzw. der Open Source Initiative (OSI).

Rails 3 Kompatibilität

Die Veröffentlichung von Rails 3 brachte vielen Verbesserungen hinsichtlich der Modularität von Rails-Anwendungen. Kern-Komponenten des Frameworks (z.B. die Persistenz-Schicht Active Record) können nun mit geringem Aufwand gegen andere Implementierungen ausgetauscht werden. Die so erreichte Flexibilität soll auch bei der Integration eines Rails WCM-Systems zur Verfügung stehen. Weiterhin sichert die Verwendung der aktuellsten Framework-Version die Unterstützung moderner Technologien und Entwicklungen innerhalb der implementierten WCMS ab.

Aktive Entwicklung

Die stetige, aktive Entwicklung an einem Open Source Projekt ist ein Merkmal für die Akzeptanz einer Software. Zusätzlich spiegelt sie das Engagement der beteiligten Programmierer wider. Ein in diesem Umfeld entstehendes Softwareprodukt bietet daher ein entsprechend höheres Potenzial. Die hier ausgewählten Systeme erfüllen diese Forderung.

Unterscheidung in Front- und Backend

WCMS unterscheiden zwischen Frontend- und Backend-Funktionalität. Das Frontend wird durch die eigentliche Internetseite repräsentiert, die mit Hilfe des Systems erzeugt wird. Im Backend können Anwender Inhalte zentral einpflegen und verwalten. Die hier ausgewählten und untersuchten Implementierungen verfügen über eine solche konzeptionelle Trennung.

3.2 Bezugsquellen

Trotz der steigenden Bekanntheit von Ruby on Rails existiert zum Zeitpunkt der Anfertigung dieser Arbeit keine Fachliteratur, die sich mit den Möglichkeiten des Web Content Managements in Rails auseinandersetzt. Vielmehr sind folgende Schwerpunktsetzungen bei den verschiedenen Autoren festzustellen:

Grundlagenbücher

Sie dienen als Einführung in Rails und verdeutlichen an Hand einfacher Anwendungen die Arbeitsweise mit dem Ruby on Rails Framework.

Fortgeschrittene Techniken mit Ruby on Rails

Rails Kern-Entwickler stellen ihre Erfahrungen mit dem Framework dar und geben Lösungsansätze für größere Unternehmensstrukturen und Projekte. Häufig vertiefte Themen sind dabei Skalierung, Performance und Refactoring.

Zur Ermittlung existierender Ruby on Rails Open Source WCMS-Software mussten daher alternative Informationsquellen herangezogen werden, die im folgenden kurz beschrieben werden sollen:

Anfragen im offiziellen IRC Channel von Ruby on Rails

Der Ruby on Rails IRC Channel ermöglicht einen konstruktiven Austausch von Rails-Entwicklern zu verschiedenen Bereichen des Rails-Frameworks. Mit der Hilfe mehrerer hundert Nutzer täglich können so Probleme und Anfragen sehr umfassend beantwortet werden. Der Ruby on Rails Channel ist erreichbar unter #rubyonrails.

RubyGems.org

Bibliotheken können die Funktionalität von Ruby enorm erhöhen. Zur Verbreitung dieser im Internet existiert u.a. der Ruby Online Community Anbieter Ruby-

Gems.org, der über 30.000 Erweiterungspakete verschiedenster Entwickler im Internet zum Download anbietet. Der Dienst verfügt über eine ausführliche Suchfunktion, mit der gezielt nach bestimmten Bibliotheken gesucht werden kann. Neben einer kurzen Projektbeschreibung und Informationen zum Entwickler wird jedem Projekt ein Datum der letzten Aktualisierung zugeordnet. Der Entwicklungsstand eines Pakets kann so besser eingeschätzt werden. RubyGems.org stellt daher einen wichtigen Startpunkt zur Ermittlung existierender Ruby on Rails WCMS dar.

Abbildung 3.1: Ruby Gems mit aufgelisteten Informationen zum aktuellen Rails 3.1

Github.com

Github.com ist ein Online-Netzwerk und Webhosting-Dienst für Programmierer. Nutzer können dort ihre individuellen Programme, umgesetzt in beliebigen Programmiersprachen, kostenlos[2] veröffentlichen. Schlüsseltechnologie des Netzwerkes ist das Versionsverwaltungssystem Git[3], welches Änderungen am Quellcode des Projektes festhält. Zusätzlich erlaubt es anderen Programmierern, bestehende Projekte zu kopieren und eigenständig weiterzuentwickeln. Anschließend können diese wieder zu einem Gesamtprojekt verschmolzen werden. Die Symbiose zwischen den Funktionalitäten von Git und den zahlreichen Interaktionsmöglichkeiten der Plattform erschaffen eine Entwicklungsumgebung, in der ein schneller und effektiver Austausch zwischen Programmierern und Projekten stattfinden kann. Innerhalb der Ruby on Rails Community hat sich Github zu einer zentralen Anlaufstelle für Rails-Programmierer entwickelt und besitzt daher zur Ermittlung bestehender Web Content Management Systeme entscheidende Relevanz.

[2]Die Einrichtung eines kostenpflichtigen, privaten Github-Repositories ist ebenfalls möglich.
[3]Projektseite: http://git-scm.com/

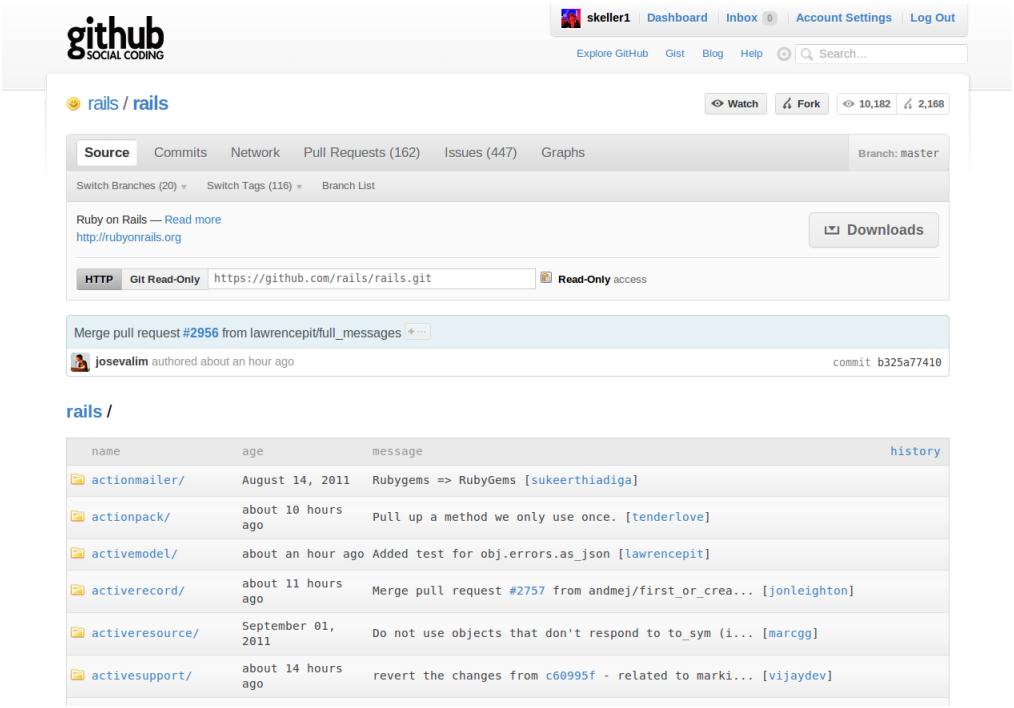

Abbildung 3.2: Beispiel für ein öffentliches Projekt auf Github.com: Das ausgewählte Projekt ist Rails 3

3.3 Vorstellung Alchemy CMS

Unter der Leitung der Hamburger Firma *macabi* wurde 2007 die proprietäre CMS Software *WashAPP* veröffentlicht. Nach der Insolvenz der Entwickler wurde das System zu nächst weiterverkauft (dabei erfolgte die Umbennung in *Webmate*), bevor es 2010 letztendlich als Open Source Software Alchemy CMS an die Öffentlichkeit übergeben wurde. Die Weiterentwicklung übernimmt seitdem die Hamburger Internetagentur *magiclabs*[4] um die Entwickler Thomas von Deyen, Robin Böning und Carsten Fregin. Die aktuelle Version 1.6.0 steht als Rails 2 und 3 Umsetzung[5] zur Verfügung. Tabelle 3.1 fasst die wichtigsten Eckdaten und Informationsquellen zu Alchemy CMS zusammen.

[4] Homepage: http://magiclabs.de/home
[5] Der Rails 3-Entwicklungszweig von Alchemy befindet sich noch im Beta-Stadium.

Tabelle 3.1: Steckbrief Alchemy CMS

Aktuelle Version	1.6.0	
Lizenz	GPLv3	
Projektseite	http://alchemy-cms.com https://github.com/magiclabs/alchemy	
Quellcode	https://github.com/magiclabs/alchemy	
IRC-Channel	nicht vorhanden	
API Dokumentation	nicht verfügbar	
Forum	http://groups.google.com/group/alchemy-cms	
Demoversion	Frontend	http://demo.alchemy-cms.com
	Backend	http://demo.alchemy-cms.com/admin
	Login	demo
	Passwort	demo
Verwendete Technologien	Ruby on Rails 3.0.x, HTML, jQuery und jQueryUI, TinyMCE - Javascript WYSIWYG Editor, SWFUpload	
Projektbeschreibung	Alchemy ist ein unglaubliches Content Managment System, welches sich gut in Rails integrieren lässt. – Absolut flexibel und kraftvoll.	
Philosophie	Der Benutzer des Systems muss nur Inhalte erstellen und ändern können Formatierung von Überschriften, Bildpositionierung und -berechnung sind Aufgaben des Entwicklers, nicht die des Redakteurs	
Zielgruppe	Privatnutzer, Kleinstunternehmen	

3.3.1 Funktionsprinzipien

Das Backend von Alchemy CMS (Abb. 3.3) verfügt u.a. über die Bereiche Seiten, Sprachen, Benutzer und Bibliothek, die im folgenden übersichtsmäßig vorgestellt werden sollen:

Home

Nach einer erfolgreichen Anmeldung im Backend bildet das Home-Modul die Startseite des WCMS, in der der Nutzer über die letzten Aktivitäten des Systems informiert wird (z.B. Auflistung der zuletzt editierten Inhalte).

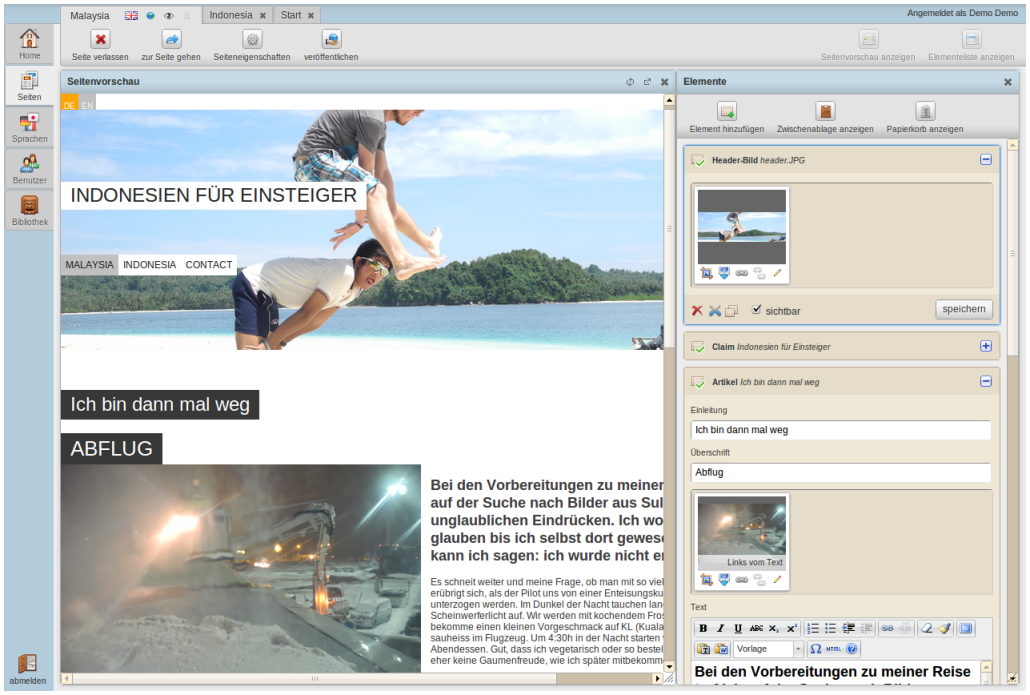

Abbildung 3.3: Backend-Ansicht von Alchemy CMS mit geöffnerter Seitenvorschau und Elementebearbeitung (Seitenbearbeitungsmodus)

Seiten

Das Seiten-Modul ermöglicht die Verwaltung und Bearbeitung aller im System vorhandener Seiten und deren Inhalte. darüber hinaus kann der von Alchemy CMS verwaltete Seitenbaum angezeigt und einzelne Seiten verschoben werden. Im Seitenbearbeitungsmodus kann die Backend-Ansicht in einen Vorschaubereich, der die aktuell ausgewählte Seite mit ihren tatsächlichen Inhalten anzeigt, und einem Elemente-Bereich, der die auf der Seite verfügbaren Inhalselemente angibt und editierbar macht, aufgeteilt werden (Abb. 3.3). Inhaltsänderungen können somit schnell nachvollzogen werden.

Sprachen

Dieses Modul ermöglicht die komfortable Verwaltung der in Alchemy CMS verfügbaren Frontend-Sprachen. Eine Alchemy Standard-Installation enthält bereits die Sprachen Deutsch und Englisch. Internetauftritte können somit jederzeit um zusätzliche Sprachversionen erweitert werden.

Benutzer

Das Benutzer-Modul ist die zentrale Anlaufstelle zur Verwaltung der am System registrierten Administratoren, Autoren und Redakteure sowie ihrer jeweiligen Befugnisse und Rechte innerhalb des WCMS.

Bibliothek

Bilder und andere Dateien werden in Alchemy CMS über das Bibliotheken-Modul verwaltet. Es ermöglicht die Auflistung aller im System zur Verfügung stehenden Ressourcen. Zusätzlich stehen Funktionen zum Hochladen, Editieren und Löschen zur Verfügung.

3.3.2 Erweiterungen

Alchemy kann durch die Erstellung von Plugins in seiner Funktionalität erweitert werden. An Hand einer definierten API wird ein Grundgerüst angeboten, mit deren Hilfe Erweiterungen bequem in alle Teile des Systems integriert werden können. Ein Plugin kann so entweder als neues Backend-Modul umgesetzt werden (es erscheint in Form eines neuen Menüeintrages im linken Bereich des Backends) oder neue Inhaltselemente zur Verfügung stellen, die dann innerhalb der Seitenbearbeitung als auswählbarer Inhaltstyp zur Verfügung stehen (Abb. 3.4).

Folgende offiziellen Erweiterungen sind ebenfalls verfügbar:

- alchemy-mailings[6]: Ermöglicht die Erstellung und Verwaltung von Newsletters im Backend des Systems.
- alchemy-standard-set[7]: Enthält CSS-Formatierungen für die in Alchemy verfügbaren Standard-Inhaltselemente.

3.3.3 Verwendete Technologien

Schwerpunkttechnologien der Nutzeroberfläche von Alchemy CMS sind die an Hand von Rails erzeugten HTML-Views und darin eingebundene JavaScript-Dateien, die mit Hilfe des JavaScript Frameworks jQuery erstellt wurden. Das Seiten-Modul greift zusätzlich

[6]Komponenten-Download: https://github.com/magiclabs/alchemy-mailings
[7]Komponenten-Download: https://github.com/magiclabs/alchemy-standard-set

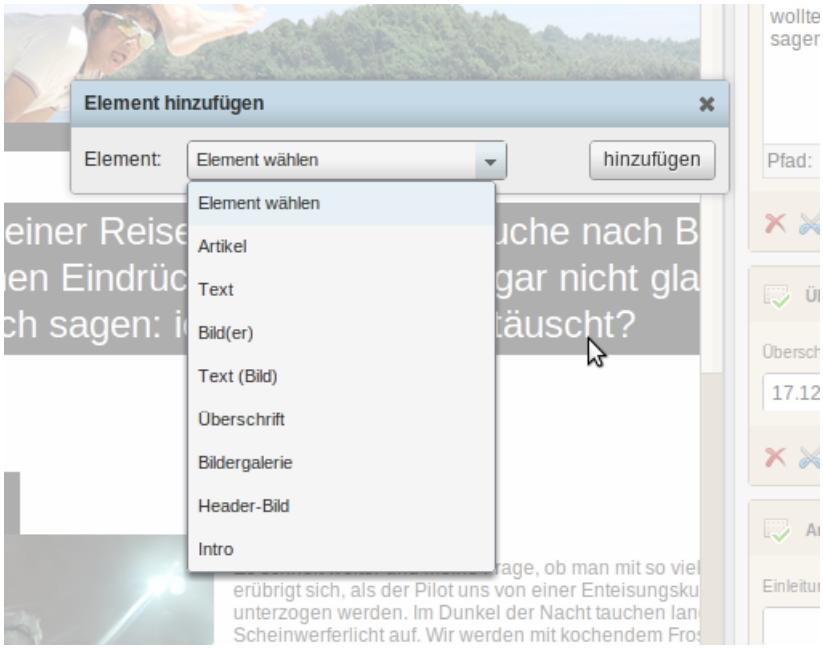

Abbildung 3.4: Standardset von Inhaltselementen in Alchemy CMS

auf die jQuery UI Bibliothek zurück, die vorgefertigte Elemente zur Erstellung von Dialogboxen und dem Tabulator-Menü liefert. Der bei einigen Inhaltselementen verfügbare WYSIWJG-Javascript Editor TinyMCE ermöglicht die Formatierung einzelner Inhalte und das Einfügen von vorgefertigtem HTML. Ein Hochladen von Ressourcen erfolgt in Alchemy über ein herkömmliches HTML Upload-Feld oder den integrierten Adobe Flash Uploader, der somit auch das Einstellen mehrerer Medien in einem Schritt ermöglicht.

3.4 Vorstellung Browser CMS

Browser CMS ist ein Open Source Web Content Management System der US-amerikanischen Agentur BrowserMedia mit Sitz in Bethesda, Maryland. Im Gegensatz zum aktuellen Versionszweig 3.x des Systems wurden die Vorgängerversionen (Versionen 1.x und 2.x) mit Hilfe der Java Programmiersprache umgesetzt. Auf Grund geänderter Marktsituation entschied sich BrowserMedia jedoch 2009 dazu, alle weiteren Versionen des CMS auf Grundlage des Rails Frameworks[8] zu entwickeln. Zusätzlich erfolgte der Umstieg auf die GPLv3-Lizenz, die eine freie Verwendung der Software ermöglicht. Eine Online-Testversion des Systems wird von BrowserMedia nicht angeboten. Um dennoch die Funktionalität des Systems zu demonstrieren, wurde eine Testinstallation von Browser CMS 3.3.1 eingerichtet[9].

Tabelle 3.2: Steckbrief Browser CMS

Aktuelle Version	3.3.1	
Lizenz	GPLv3	
Projektseite	http://browsercms.org	
Quellcode	https://github.com/browsermedia/browsercms	
IRC-Channel	nicht vorhanden	
API Dokumentation	http://rubydoc.info/gems/browsercms/	
Forum	http://groups.google.com/group/browsercms	
Demoversion	Frontend	http://diplomabcms.heroku.com/
	Backend	http://diplomabcms.heroku.com/admin
	Login	demo
	Passwort	demo
Verwendete Technologien	Ruby on Rails 3.0.x, HTML, jQuery und jQueryUI, diverse jQuery Plugins ,WYSIWYG-HTML-Editor CKEditor	
Projektbeschreibung	Menschliches Content Management mit Ruby on Rails 3 Unterstützung	
Zielgruppe	Privatnutzer, Kleine und mittelständige Unternehmen mit einer größeren Anzahl von Redakteuren	

[8] Ausführliche Informationen zum Übergang von Java auf Rails können unter folgender Quelle nachvollzogen werden: http://tinyurl.com/6eayba7

[9] Die Upload-Funktion der Testversion wurde auf Grund zu geringer Server-Ressourcen deaktiviert.

3.4.1 Funktionsprinzipien

Innerhalb des Backend (Abb. 3.5) von Browser CMS kann der angemeldete Nutzer auf alle wichtigen Funktionsbereiche des Systems zugreifen:

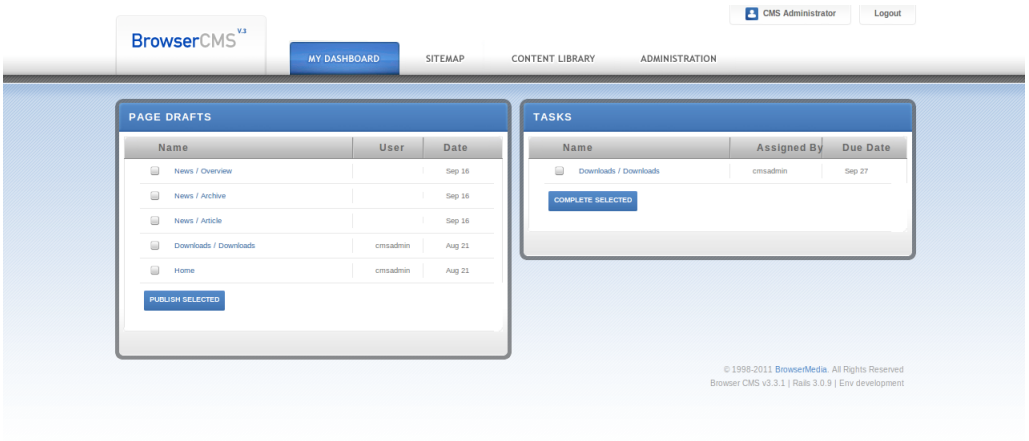

Abbildung 3.5: Backend-Ansicht von Browser CMS

Dashboard

Das Dashboard gibt eine Übersicht über die zuletzt vom Nutzer bearbeiteten Seiten sowie eine Auflistung der vom Anwender noch zu erledigenden Aufgaben.

Sitemap

Das Sitemap-Menü ermöglicht die Darstellung aller im System existierenden Seiten in einer Baumstruktur sowie die Bearbeitung aller Informationen zu einer einzelnen Seite. U.a. kann der Titel, das verwendete Template und verschiedene Meta-Informationen verändert werden.

Content Library

In der Content Library von Browser CMS können alle existierenden Inhaltselemente sortiert nach Inhaltstyp (z.B. Text, File, Image) aufgelistet und bearbeitet werden.

Administration

Im Administrator-Bereich des Systems werden alle Gruppen und Nutzer des Backend und der Internetseite verwaltet. Zusätzlich können die für die einzelnen Seiten und Inhaltselemente definierten Templates aufgelistet und bearbeitet werden.

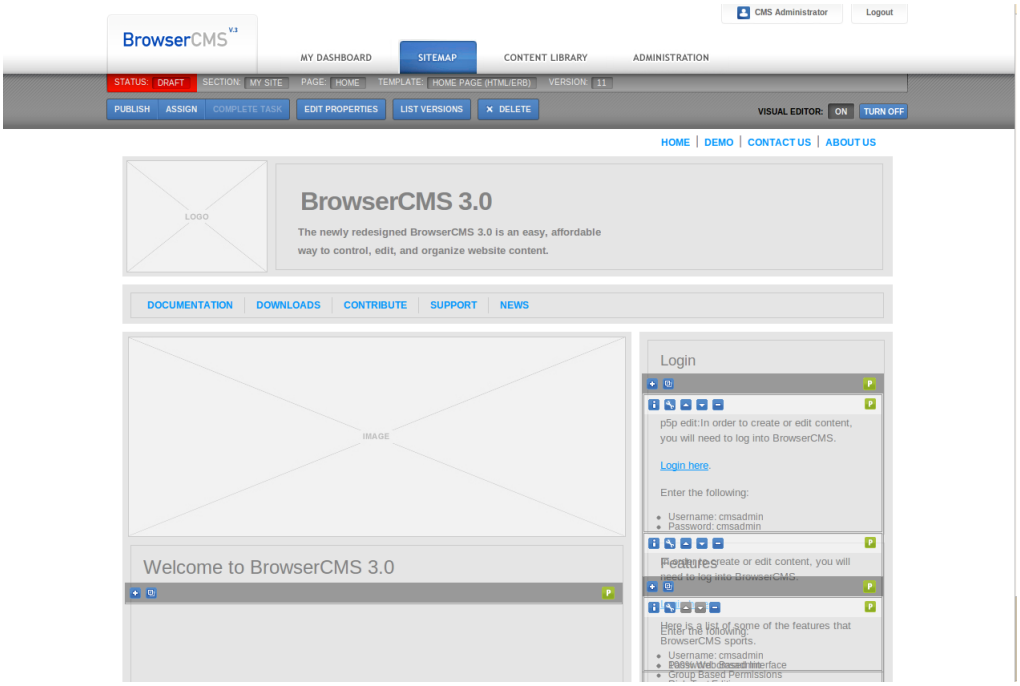

Abbildung 3.6: Ausgewählte Seite im Bearbeitungsmodus und aktiviertem Visual Editor

Die Erstellung und Anzeige von Inhalten wird in Browser CMS durch folgende 2 Prinzipien realisiert:

Content Blocks

Content Blocks sind Sammlungen von Daten mit einer definierten Anzahl an Feldern eines bestimmten Typs (z.B. Textfeld, Datumsfeld). Innerhalb der Content Library können diese verwaltet und angelegt werden.

Portlet

Portlets[10] sind spezielle Content Blocks, die festlegen, wie bereits existierende Con-

[10] Der Begriff des Portlets stammt aus der Java-Welt und bezeichnet ursprünglich beliebig kombinierbare Komponenten, die an verschiedenen Positionen einer Internet- bzw. Portal-Seite dargestellt werden können.

tent Blocks (z.B. Text, News) dargestellt werden sollen. Dazu verfügt jedes Portlet über ein im Backend editierbares Template, dass die zuvor ausgewählten Datensätze in dem gewünschten Layout präsentiert (Abb. 3.7).

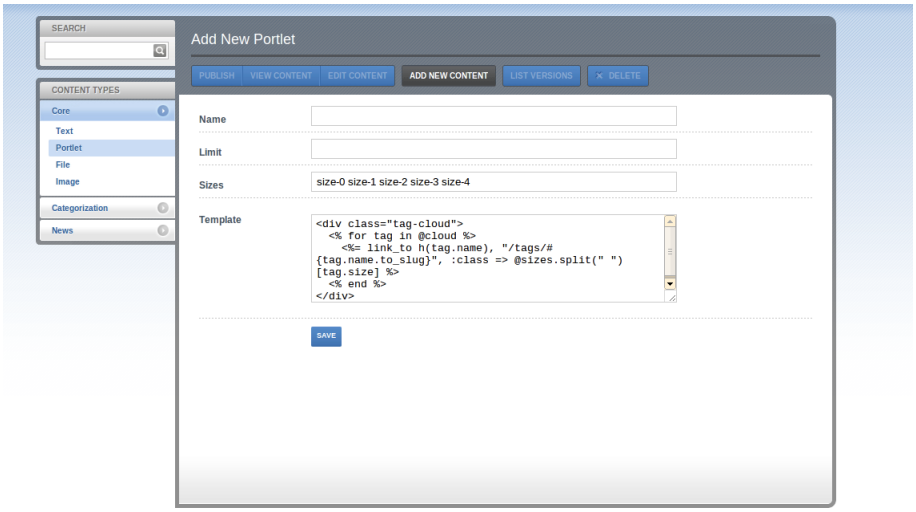

Abbildung 3.7: Tag-Portlet mit Template im Backend von Browser CMS

3.4.2 Erweiterungen

Die Funktionalität von Browser CMS kann durch das Hinzufügen von sogenannten Modulen erweitert werden. Diese beinhalten dabei neue Content-Blocks oder vordefinierte Portlets, die in der Content Library des Backends verwaltet werden können. Im Internet stehen zusätzlich folgende vorgefertigten Erweiterungen bzw. Module zur Verfügung[11]:

- browsercms-news[12]: Darstellung von kurzen Nachrichten im Frontend der Seite.
- browsercms-blog[13]: Erstellung und Verwaltung von Blog-Einträgen inklusive Kommentar- und Tagging-Funktionalität.
- browsercms-events[14]: Erstellung und Verwaltung von Event-Einträgen. Diese kön-

[11] Eine ausführliche Liste mit allen Erweiterungen findet sich unter folgender Adresse: http://modules.browsercms.org/modules
[12] Komponenten-Download: https://github.com/browsermedia/bcms_news
[13] Komponenten-Download: https://github.com/browsermedia/bcms_blog
[14] Komponenten-Download: https://github.com/browsermedia/bcms_event

nen wahlweise alle zusammen in einer Listenansicht oder in Einzeldarstellung individuell präsentiert werden.

- browsercms-rankings[15]: Erlaubt dem Internetseitenbenutzer eine Bewertung der besuchten Seite abzugeben.

3.4.3 Verwendete Technologien

Das komplette Backend von Browser CMS wird mit Hilfe von in Rails gerenderten HTML-Views erzeugt. JavaScript kommt lediglich bei der Darstellung verschiedener Feldtypen (z.B. Datumsdialog und Auswahlbox) innerhalb der Content-Blöcke und beim Anlegen von Textinhalt durch die Verwendung des JavaScript-WYSIWYG-Editors FCKEditor (Abb. 3.8) zur Anwendung. Dieser ermöglicht die komfortable Formatierung eingestellter Texte sowie die Einbindung vorformatierter HTML-Inhalte.

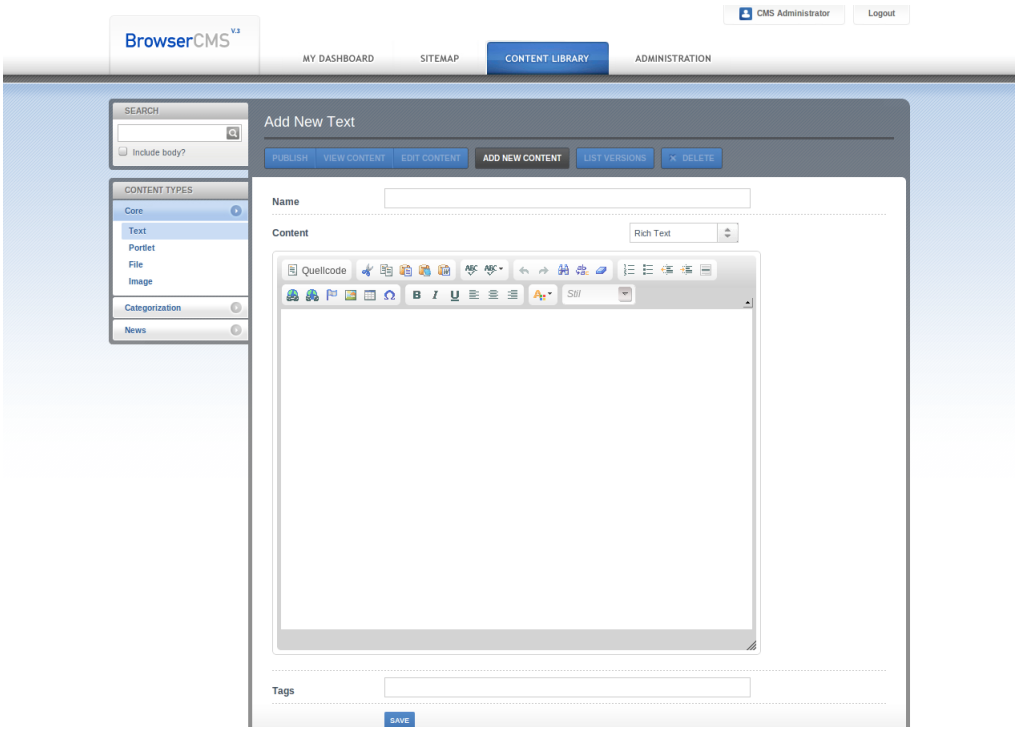

Abbildung 3.8: Erstellung eines neuen Datensatzes vom Inhaltstyp (Content Block) Text mit Hilfe des FCKEditor

[15]Komponenten-Download: https://github.com/browsermedia/bcms_rankings

Die Erstellung der Content-Blöcke (Content Blocks) und Portlets wird durch einen in Rails umgesetzten Generatorskript ermöglicht und vereinfacht somit die Entwicklungsarbeit für den Programmierer (Abb. 3.1).

Quelltext 3.1: Aufruf des Generator-Skripts zur Erstellung eines neuen Inhaltselements Produkt

```
1 rails g cms:content_block Product name:string price:integer description:html
```

3.5 Vorstellung Locomotive CMS

Lokomotive CMS ist ein Open Source Content Management System der Ruby on Rails Entwickler Didier Lafforgue und Jacques Crocker sowie dem Designer Sacha Greif. Es wird unter der MIT-Lizenz der Open Source Initiative vertrieben und steht in den Rails-Versionen 2 und 3 als Downloadpaket zur Verfügung. Neben den Möglichkeiten der eigenständigen Installation bieten die Initiatoren des Systems auch den kostenpflichtigen Dienst *Bushi.do* an, der eine automatische Komplettinstallation inklusive Hosting und Serverumgebung zur Verfügung stellt. Nach nur einem Klick kann sofort mit der Erstellung der Seite im Browser begonnen werden[16].

3.5.1 Funktionsprinzipien

Locomotive CMS unterscheidet im Backend der Anwendung in folgende 2 Funktionsbereiche:

Inhalte

Im Inhalte-Bereich des Backends werden alle im System angelegten Seiten und Inhaltselemente verwaltet und bearbeitet (Abb. 3.9). Die Darstellung jeder einzelnen Seite und der Inhaltselemente kann darüber hinaus durch Angabe eines Templates online verändert werden (mehr dazu in Abschnitt 3.5.3), was eine Umsetzung komplexer Seitenlayouts sicherstellt.

Einstellungen

Im Einstellungs-Modul befinden sich die zentralen Funktionen zur Bearbeitung

[16] Auf der Projektseite von Lokomotive CMS wird die automatische Installation mit Hilfe von Bushi.do beschrieben: http://www.locomotivecms.com/

Tabelle 3.3: Steckbrief Locomotive CMS

Aktuelle Version	keine Angabe von Entwicklungsversionen	
Lizenz	MIT License	
Projektseite	http://www.locomotivecms.com/	
Quellcode	https://github.com/locomotivecms/engine	
IRC-Channel	#locomotivecms	
API Dokumentation	http://rubydoc.info/github/resolve/refinerycms	
Forum	http://groups.google.com/group/refinery-cms/	
Demoversion	Frontend	http://diplomalocomotive.heroku.com/
	Backend	http://diplomalocomotive.heroku.com/admin
	Login	demo@demo.de
	Passwort	demo123
Verwendete Technologien	Ruby on Rails 3.0.x, HTML, jQuery und jQueryUI, diverse jQuery Plugins ,WYSIWYG-HTML-Editor Aloha und TinyMCE, MongoDB, Template Sprache Liquid	
Projektbeschreibung	Locomotive ist ein Open Source CMS für Rails. Es ist sehr flexibel und unterstützt Heroku und Amazon S3.	
Philosophie	Verwaltung kleiner Internetseiten Komplexe Inhaltselemente dank MongoDB selbst erstellen	
Zielgruppe	Privatnutzer, Kleinstunternehmen	

der vorhandenen Backend-Nutzer sowie globaler Locomotive CMS-Einstellungen (registrierte Domains, Import-/Export von Seiten, Bearbeitung von allgemeinen Metainformationen des Systems). Zusätzlich können in einem Template-Bereich einzelne, zentrale Dateien verwaltet werden, die innerhalb des Seitenlayouts als Gestaltungselemente dienen sollen[17].

3.5.2 Erweiterungen

Erweiterungen werden in Locomotive CMS als Bausteine bezeichnet und repräsentieren individuell zusammengestellte Inhaltselemente. Sie können im Backend von Locomotive CMS über einen komfortablen Bearbeitungsdialog erstellt werden (Abb. 3.11). Im

[17] Die für layoutspezifische Verwendung hochgeladenen Dateien werden in Locomotive CMS als Snippets bezeichnet

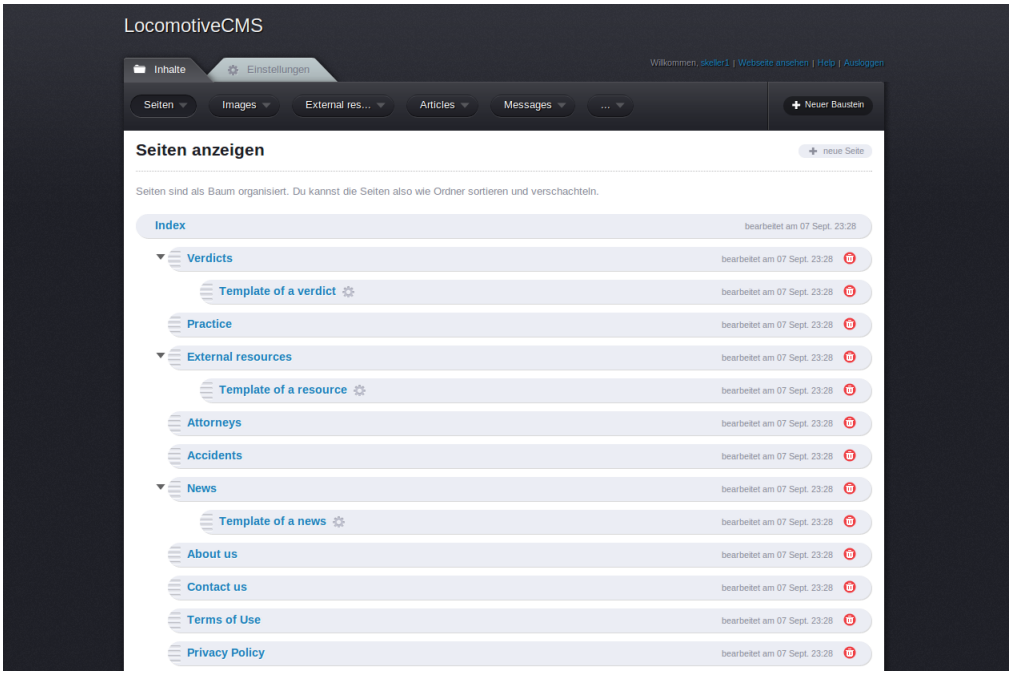

Abbildung 3.9: Backend von Locomotive CMS mit geöffnetem Seitenbaum.

Gegensatz zu den hier bereits vorgestellten Systemen werden so keine Programmierkenntnisse der Anwender benötigt. Ebenfalls entfällt der Aufwand zur Installation einer Erweiterung.

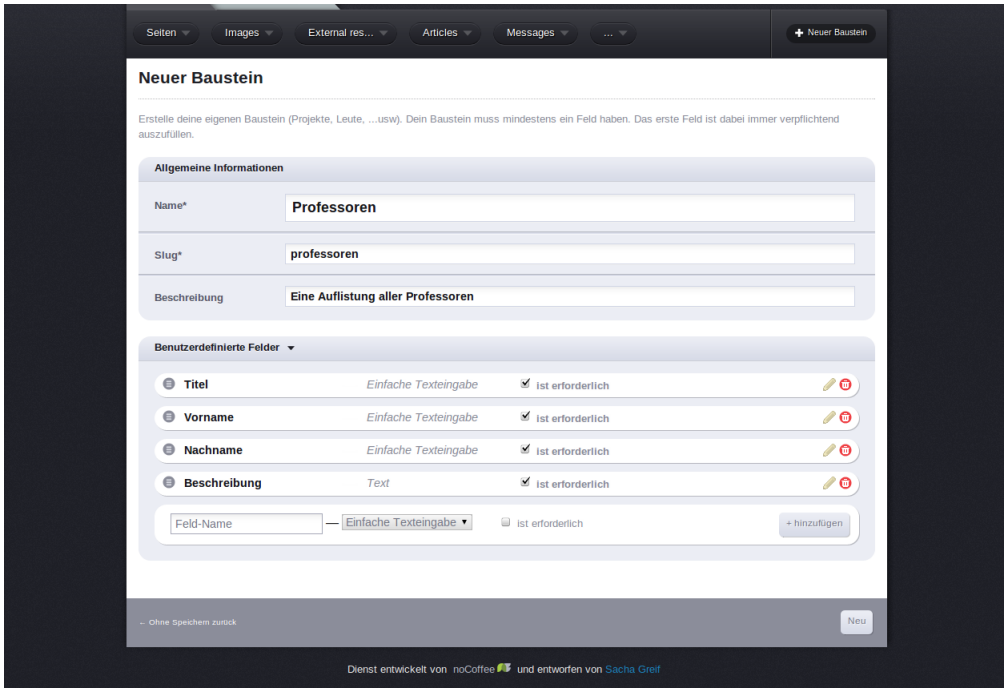

Abbildung 3.10: Ein im Backend von Locomotive CMS zusammengestelltes Inhaltselement

Die Inhaltselemente können aus folgenden grundlegenden Feldtypen zusammengebaut werden:

- Einfaches Textfeld
- Text
- Auswahlbox
- Checkbox
- Datum
- Datei
- has one (ermöglicht die Zuordnung genau eines anderen Inhaltselements des angegebenen Bausteintyps)
- has many (ermöglicht die Auswahl mehrerer anderer Inhaltselemente des angegebenen Bausteintyps)

3.5.3 Verwendete Technologien

Das Backend von Locomotive CMS besteht aus in Rails gerenderten HTML-Views mit eingebundenen JavaScript-Dateien, die einzelne Funktionalitäten bereitstellen. Das HTML wird dabei zusätzlich durch die Verwendung der JavaScript-Bibliothek Sizzle[18] manipuliert, die es erlaubt durch Angabe eines CSS-Selektors bestimmte Elemente des HTML zu erfassen. Zur Erstellung von Templates wird die Ruby Template-Sprache Liquid eingesetzt. Sie hat ihren Ursprung in der kommerziellen Online E-Commerce-Plattform *shopify*[19] und beitet Möglichkeiten der Vererbung und Überschreibung vorheriger definierter Templates. Der im Backend eingesetzte WYSIWYG-Editor TinyMCE ermöglicht die komfortable Eingabe von Text und HTML-Elementen[20].

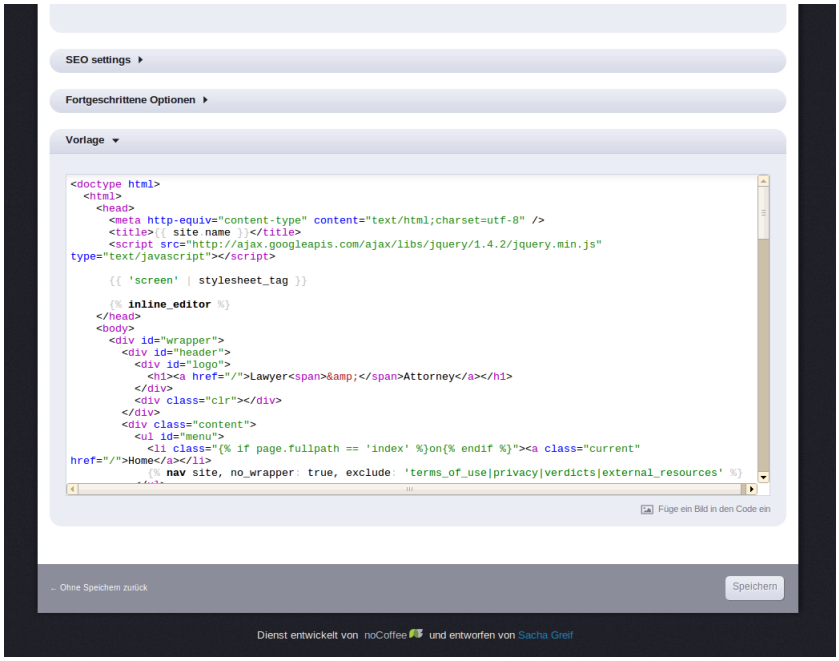

Abbildung 3.11: Seitenbearbeitung in Locomotive CMS mit editierbarem Liquid-Template

Im Frontend der Seite findet zusätzlich der WYSIWYG-JavaScript-HTML5-Editor

[18] Komponenten-Download: http://sizzlejs.com/

[19] Liquid ist das Ergebnis einer Extrahierung dieser Funktionalität aus Shopify

[20] Um den Editor innerhalb der einzelnen Seiten zu aktivieren, müssen im Liqiud-Template der Seite entsprechende Befehle aufgerufen werden. Nach einem erneuten speichern der Seite steht der Editor im Backend zur Verfügung und kann mit Inhalten gefüllt werden.

Aloha[21] Verwendung. Er ermöglicht so eine Bearbeitung spezieller Inhaltselemente direkt im Frontend der Seite (Inline-Editing[22]).

[21] Projektseite: http://aloha-editor.org/

[22] Die Entwicklung dieses Features befindet sich noch im Betastatdium und kann noch nicht zuverlässig verwendet werden.

3.6 Vorstellung Refinery CMS

Refinery CMS - in der Kurzform oft als Refinery bezeichnet - ist ein freies Open Source Web Content Management System des neuseeländischen Entwicklerteams Resolve Digital, dessen Entwicklung 2004 durch David Jones eingeleitet wurde. Nach einer fünfjährigen, eingeschränkten Entwicklungsphase, in der nur wenige Bereiche des Systems verbessert wurden, erfolgte am 28. Mai 2009 die Veröffentlichung der ersten Open Source Software Version. In der Folgezeit wurde das CMS durch die Kernentwickler David Jones, Philip Arndt, Steven Heidel und Uģis Ozols auf das aktuelle Rails 3 umgestellt und eine erste stabile Version 1.0.0 veröffentlicht (28. Mai 2011).

Tabelle 3.4: Steckbrief Refinery CMS

Aktuelle Version	1.0.8	
Lizenz	MIT License	
Projektseite	http://refinerycms.com	
Quellcode	https://github.com/resolve/refinerycms	
IRC-Channel	#refinerycms	
API Dokumentation	http://rubydoc.info/github/resolve/refinerycms	
Forum	http://groups.google.com/group/refinery-cms/	
Demoversion	Frontend	http://demo.refinerycms.com
	Backend	http://demo.refinerycms.com/refinery
	Login	demo
	Passwort	demo
Verwendete Technologien	Ruby on Rails 3.0.x, HTML, jQuery und jQueryUI, WYSIWYG-HTML-Editor Wymeditor, HTML 5 Multi-Upload	
Projektbeschreibung	Erweiterbares Ruby on Rails „CMS Framework“ mit Ruby on Rails 3 Unterstützung	
Philosophie	Realisierung einer benutzerfreundlichen, einfachen Oberfläche Einfaches Hinzufügen von Funktionalität an Hand der in Rails bekannten Entwicklungsabläufe Aktive Community durch Google Group und IRC, die eine schnelle Hilfe ermöglichen	
Zielgruppe	Privatnutzer, Kleinstunternehmen	

3.6.1 Funktionsprinzipien

Das Backend bildet in Refinery CMS die zentrale Anlaufstelle zur Erstellung und Verwaltung aller Inhalte, Einstellungen und Nutzer. Über ein zentrales Menü kann auf die Funktionsmodule des Systems zugegriffen werden, welche im folgenden vorgestellt werden:

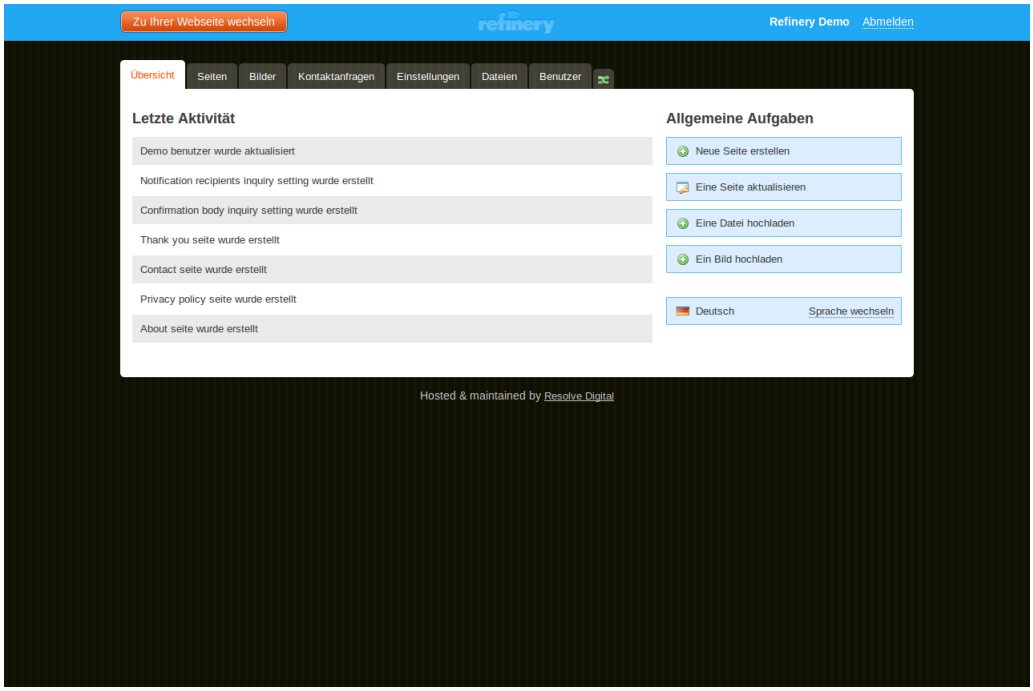

Abbildung 3.12: Backend-Ansicht von Refinery CMS

Übersichts-Modul

Nach der Anmeldung am System bildet das Übersichts-Modul (o.a. Dashboard) die Startseite des Systems. Dort können in einer einfachen Form die letzten Aktivitäten innerhalb des CMS eingesehen werden. Zusätzlich werden Schnellaufrufe zu systemspezifischen Funktionen angeboten.

Seiten-Modul

Das Seiten-Modul listet alle angelegten Seiten und Unterseiten in Form einer Baumstruktur auf. Zusätzlich können Titel und Metainformationen bearbeitet

werden. Darüber hinaus verfügt jede Seite über einen oder mehrere WYSIWYG-Editor-Textfelder[23], in die der Inhalt der Seite eingepflegt werden kann (Abb. 3.13).

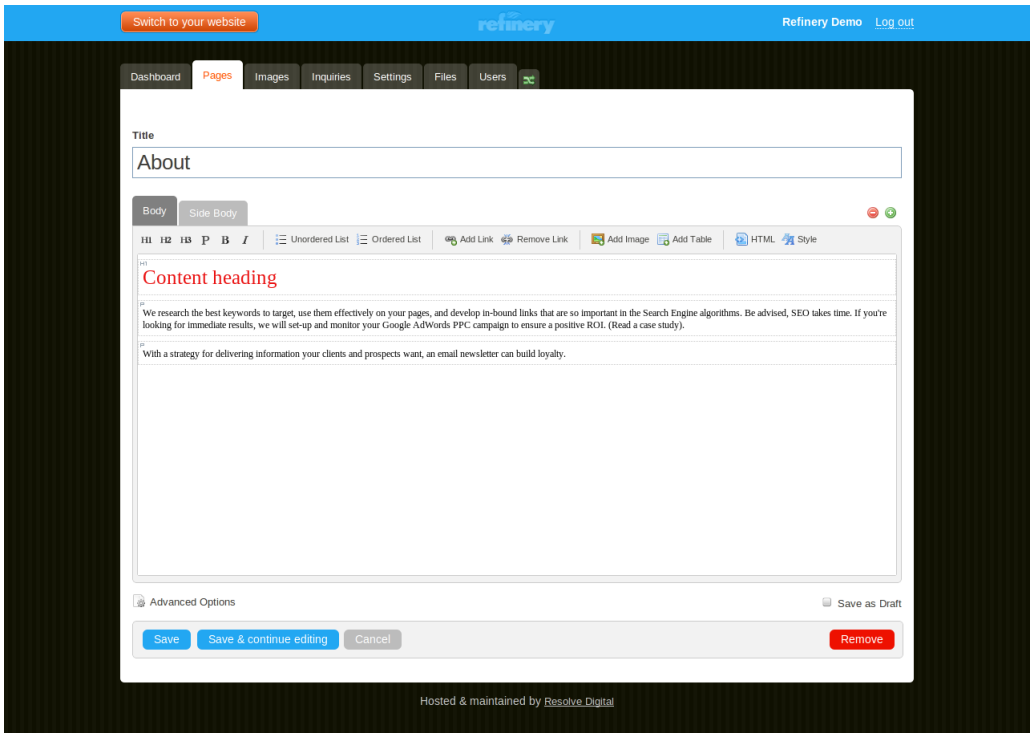

Abbildung 3.13: Seitenbearbeitung in Refinery CMS mit geöffneten Content Sections Body und Site Body

Bilder- und Dateien-Modul

In Refinery wird durch die Wahl der Menüpunkte Bilder und Dateien die Ressourcenverwaltung des Systems geöffnet. Dort können anschließend Mediendateien verschiedenster Formate hochgeladen, bearbeitet und durchsucht werden.

Benutzer-Modul

Das Benutzer-Modul erlaubt die Verwaltung der am System registrierten Anwen-

[23] Die einzelnen Inhaltsblöcke werden als Content Sections bezeichnet. Die Anzahl der pro Seite zur Verfügung stehenden Inhaltsblöcke kann beliebig festgelegt werden. Die Darstellung und Positionierung der einzelnen Blöcke wird durch ein zuvor festgelegtes HTML-Template im Rails-Quellcode festgelegt festgelegt.

der. U.a. können dort Nutzername, Passwort und Berechtigungen für die Verwendung anderer Module gesetzt werden.

Einstellungs-Modul

Die einzelnen Module und Erweiterungen von Refinery können durch zuvor definierte Parameter[24] in ihrem Verhalten oder Ausehen beeinflusst werden. Das Einstellungs-Modul listet die in einer CMS-Installation vorhandenen Konfigurationsoptionen auf und ermöglicht eine nachträgliche Editierung und Löschung dieser.

3.6.2 Erweiterungen

Erweiterungen werden in Refinery als Engines bezeichnet. Sie werden durch Aktivierung innerhalb der Rails-Anwendung (im Quellcode) installiert und anschließend im Benutzer-Modul von Refinery den einzelnen Anwendern zugeordnet. Zum Zeitpunkt der Erstellung dieser Arbeit existieren u.a. folgende Erweiterungen:

- refinerycms-inquiries[25]: Darstellung von Kontaktanfragen auf der Internetseite mit zusätzlicher Verwaltungsfunktion der Anfragen im Backend von Refinery CMS
- refinerycms-news[26]: Verwaltung und Darstellung von Nachrichten im Front- und Backend-Bereich
- refinerycms-blog[27]: Engine zur Erstellung kurzer Beiträge inklusive Kategorisierungs- und Kommentarfunktion

3.6.3 Verwendete Technologien

Refinery CMS ist ein technologisch sehr einfaches System. Die Realisierung der Backend-Funktionalitäten wird durch die konsequente Verwendung von HTML 5 erreicht. Zur Generierung der Dialoge, die u.a. innerhalb des Bild- und Dateien-Moduls eingesetzt werden, greift Refinery auf die jQuery UI Bibliothek zurück. Der im Seiten-Modul integrierte WYSIWYG-Editor ist ein für die Anforderungen des CMS angepasster, XHTML konformer JavaScript WYMeditor. Durch die Unterstützung von HTML 5 innerhalb des

[24] Die offizielle Bezeichnung der Parameter lautet *Refinery Settings*
[25] Komponenten-Download: https://github.com/resolve/refinerycms-inquiries
[26] Komponenten-Download: https://github.com/resolve/refinerycms-news
[27] Komponenten-Download: https://github.com/resolve/refinerycms-blog

Backend können Datenübertragungen von Medien an den Server in Form von Multi-Uploads realisiert werden. Die sonst übliche Nutzung von Flash entfällt und vereinfacht damit das System zusätzlich. Die bereits angesprochenen Erweiterungen (Engines) sind eigenständige Rails-Anwendungen, deren Grundgerüst mit Hilfe eines Refinery-Engine-Generators erzeugt wird. Die geringen technologischen Abhängigkeiten und Anforderungen des Systems erlauben es, Teile des Backends bei Bedarf anzupassen, zu verändern oder komplett auszutauschen.

3.7 Durchführung der funktionalen Analyse

Um die Leistungsfähigkeit der ausgewählten Systeme im Bereich des Webpublishing einschätzen zu können, werden die vorgestellten Kriterien des vorhergehenden Abschnitts mit den ausgewählten WCMS in Tabellenform gegenübergestellt. Es erfolgt dabei eine Festlegung der Erfüllung dieser Kriterien in folgende 2 Stufen:

Tabelle 3.5: Mögliche Auswertungsstufen für die umgesetzten Funktionalitäten der WCMS

Erfüllt 100%	Das hier vorgestellte WCMS erfüllt die aus dem Kriterium definierten Funktionalitäten. Dies kann auch durch Installation einer Zusatzkomponente erreicht werden.
Erfüllt 0%	Das hier vorgestellte WCMS erfüllt die aus dem Kriterium definierten Funktionalitäten nicht. Es existieren darüber hinaus keine Erweiterungsmöglichkeiten für das System oder die Erfüllung ist nur durch eigenständige Implementierung (Programmierung) erreichbar.

Zusätzlich werden bei bestehenden Einschränkungen und Problemen zu jedem WCMS noch Erläuterungen aufgeführt, die eine genauere Einschätzung des tatsächlichen Funktionsumfangs liefern können.

3.7.1 Erstellung

Mehrere Benutzer sollen gleichzeitig Inhalte verwalten und erfassen können			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Nutzer und Administratoren können Inhalte gleichzeitig erfassen und verwalten.		Nutzer und Administratoren können Inhalte gleichzeitig erfassen und verwalten.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Nutzer und Administratoren können Inhalte gleichzeitig erfassen und verwalten.		Nutzer und Administratoren können Inhalte gleichzeitig erfassen und verwalten.	

Inhalte sollen – unabhängig von Zeit- und Standort – durch mehrere Benutzer online verwaltet und erfasst werden können			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Vollständig unterstützt		Vollständig unterstützt	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Vollständig unterstützt		Vollständig unterstützt	

Offline Erfassung von Inhalten unter Verwendung eines lokal auf dem Rechner installierten Programms			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	

Integrierte Mediendatenbank zur Erfassung und Verwaltung von Bildern, Multimedia,Texten, Audio, Videos, usw.			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Alchemy bietet eine Bibliothek, in der Bilder und Dateien verwaltet werden können.		Refinery CMS bietet eine einfache Medienverwaltung.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Browser CMS verfügt über eine *Content Library*, die eine einfache Medienverwaltung von Bildern, Dateien und definierten Inhaltselementen ermöglicht.		Locomotive CMS bietet eine Asset-Verwaltung, in der selbst erstellte Inhaltselemente in Containern verwaltet werden können.	

Inhalte sollen ohne spezielle Programmier / HTML-Kenntnisse erfasst und verwaltet werden können			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Alle Inhalte können über den TinyMCE-Javascript WYSIWYG Editor erfasst und formatiert werden.		Alle Inhalte können über den integrierten WYSIWYG-Editor Wymeditor erfasst und formatiert werden. Der Editor ist fest in das System integriert und kann nicht ausgetauscht werden. Ein Plugin, dass die Verwendung eines anderen Editors ermöglicht ist bereits in Planung.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
In Browser CMS findet der WYSIWIG-FCKEditor Verwendung. Zusätzlich stehen verschiedene Module zur Verfügung, die einen Austausch des Editors gegen andere Lösungen ermöglichen.		Alle Inhalte können über zwei integrierte WYSIWYG-Editoren erfasst und formatiert werden. Im Backend steht der Javascript Editor TinyMCE zur Verfügung. Im Frontend findet der HTML5-WYSIWYG-Editor Aloha zur Manipulierung der Seiteninhalte Verwendung (befindet sich noch in der Entwicklung).	

Inhalte sollen in einer Datenbank gespeichert werden			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Alchemy verwendet Active Record als Datenbankpersistenzschicht. Durch die Verwendung von Migrationen könen so eine Vielzahl relationaler Datenbanken unterstützt werden. Zusätzlich exisitieren einige Adapter, um auch dokumentenbasierte Datenbanken anzusteuern.		Refinery greift ebenfalls auf Rails' Active Record zurück und unterstützt damit mehrere relationale und dokumentenorientierte Datenbanken.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Wie bei Alchemy und Refinery CMS wird hier auch auf Active Record zurückgegriffen. Die Entwickler garantieren auf Grund fehlender Tests jedoch nur die Unterstützung von SQLite und MySQL-Datenbanken. Tendenziell können aber alle von Active Record unterstützten Datenbanken eingesetzt werden.		Locomotive CMS greift im Gegensatz zu seinen Konkurrenten auf die dokumentenorientierten Datenbank MongoDB zurück. Relationale Datenbanken werden somit nicht unterstützt. Eine Umsetzung von Locomotive mit Active Record ist jedoch geplant.	

<table>
<tr><td colspan="4">Inhalte (Texte, Bilder, Videos etc.) sollen zentral kategorisiert, erfasst und verwaltet werden können</td></tr>
<tr><td>Alchemy 1.6.0</td><td>Erfüllt: 0%</td><td>Refinery CMS 1.0.8</td><td>Erfüllt: 0%</td></tr>
<tr><td colspan="2">Die Bibliothek von Alchemy unterstützt lediglich eine Auflistung von Resourcen. Bilder und Dateien können damit nur in Form einer Listenansicht inspiziert werden. Eine Zuordnung zu Kategorien oder eine Anlage von Ordnerstrukturen zur Erleichterung der Orientierung ist nicht möglich. Die Verwaltung großer Datenmengen scheint daher nur schwer möglich.</td><td colspan="2">Ähnlich wie bei Alchemy gleicht die Resourcenverwaltung nur einer einfachen Auflistung von Bildern und anderen Ressourcen. Eine Kategorisierung der Inhalte ist nicht möglich. Ebenfalls können keine Ordner zur sinnvollen Strukturierung der Ressourcen erstellt werden. Die Verwaltung großer Datenmengen wird dadurch schnell zu einem Geduldsakt.</td></tr>
<tr><td>Browser CMS 3.3.1</td><td>Erfüllt: 0%</td><td>Locomotive CMS</td><td>Erfüllt: 0%</td></tr>
<tr><td colspan="2">Die Content Library von Browser CMS listet wie ihre Vorgänger lediglich die angelegten Bilder oder Dateien auf. Möglichkeiten zur sinnvollen Organisation (Kategorien, Ordner) großer Datenmengen sind nicht vorhanden.</td><td colspan="2">Die Inhaltsverwaltung kann nicht kategorisiert werden. Wie bei seinen Vorgängern sind die Datensätze lediglich in Listenform aufgeführt. Eine logische Strukturierung mit Hilfe von Ordnern ist nicht möglich.</td></tr>
</table>

Inhalte können während der Erfassung über eine Preview-Funktion vorab im Design der Webseite angesehen werden			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 0%
Redakteure können ihre erstellten und editierten Inhalte im Backend durch ein Preview-Fenster sichtbar machen. Änderungen an Inhaltselementen können somit sofort nachvollzogen werden.		Refinery CMS verfügt über keine Preview-Funktion der Inhalte. Ist ein Inhaltselement im Backend neu angelegt oder bearbeitet wurden, wird dies auf der Internetseite sofort sichtbar.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 0%
Wie bei Alchemy werden Inhalte erst nach ihrer Veröffentlichung sichtbar. Bis dahin kann jedoch im Frontend durch Inline-Editing der Seite jedes Inhaltselement bearbeitet werden.		Locomotive CMS bietet wie RefineryCMS keine Preview-Funktion. Änderungen und neu angelegte Inhalte werden direkt veröffentlicht.	

Zuordnung von standardisierten und frei definierbaren Metadaten zu Inhalten (z.B. Autor, Schlüsselwörter, Benutzerdefinierte Felder) soll möglich sein			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Metadaten zu Inhalten können nicht vergeben werden.		Inhalte werden als einfache Datensätze betrachtet und besitzen daher keine definierten Metadaten.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Metadaten können zu einzelnen Inhaltselementen in Form einer Tag-Liste hinzugefügt werden. Diese wird in der Datenbank als Text abgespeichert und bei ihrer Nutzung in einzelne Teil-Strings zerlegt. Das Hinzufügen zuvor definierter Metadaten ist nicht möglich.		Zu den verschiedenen Inhaltselementen können beliebig viele Metadaten hinzugefügt werden. Auch die Darstellung von 1:1 und 1:n-Beziehungen ist möglich. Diese Funktionalität wird dabei vor allem durch die Verwendung der dokumentenbasierten Datenbank MongoDB möglich.	

Inhalte sollen mehrsprachig erfasst und verwaltet werden können			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
In Alchemy können Inhalte mehrsprachig angelegt werden. Durch die Auswahl einer bestimmten Sprache wird ein entsprechender Seitenbaum mit allen existierenden Inhalten zu der ausgewählten Sprache erzeugt.		Refinery CMS kann Inhalte mehrsprachig verwalten und ausgeben. Zur Aktivierung der Funktionalität müssen nur die zu unterstützenden Sprachen in einer Konfigurationsdatei angegeben werden (dies kann von Administtratoren im Backend vorgenommen werden). Alle Sprachen werden dabei in einem einzigen Seitenbaum verwaltet. Vorhandene Übersetzungen zu einer bestimmten Seite werden durch Einblendung von Flaggensymbolen kenntlich gemacht.	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
In Browser CMS kann durch die Installation der Erweiterung *browsercmsi* die Unterstützung von mehrsprachigen Inhalten erreicht werden. Der Plugin-Anbieter konnte die 100% Rails 3-Kompatibilität der Erweiterung jedoch nicht garantieren. Von einem Einsatz dieser Lösung in einer Produktiv-Umgebung wird daher abgeraten. Innerhalb der Bewertung von Browser CMS werden daher 0% beim Erfüllungsgrad angegeben.		Inhalte können nur einsprachig verwaltet werden. Erweiterungen, die diese Funktionalität herstellen können, konnten nicht ermittelt werden.	

Das CMS soll über eine offene API (Programmierschnittstelle) für individuelle Erweiterung verfügen			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Eine flexible Plugin-DSL (Domain Specific Language) erlaubt das Hinzufügen von individuellen Erweiterungen.		Individuelle Inhaltselemente können durch die Verwendung der Refinery Engine Generatoren hinzugefügt werden. Die weitere Entwicklung Anpassung des Codegerüst folgt dann mit den innerhalb des Rails Framework üblichen Entwicklungstechniken und -abläufen.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Ähnlich wie bei Refinery CMS können neue Module und Inhaltstypen mit Hilfe von speziellen Rails-Generatoren erzeugt werden.		Neue Inhaltstypen lassen sich im Backend durch ein einfaches User-Interface zusammenstellen. Mit wenigen Klicks sind so schnell neue Elemente erstellt. Programmierkenntnisse sind nicht notwendig.	

Für die Verwaltung und Erfassung von Inhalten sollen alle gängigen Internet-Browser (Internet Explorer, Safari und Firefox) eingesetzt werden können			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Vollständig unterstützt		Vollständig unterstützt	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Vollständig unterstützt		Vollständig unterstützt	

Inhalte sollen einfach importiert / exportiert werden können - dabei kommen Formate wie z.B. XML zum Einsatz			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Alchemy verfügt über keine integrierten Import und Export-Funktionalitäten.		Es existieren auf Nutzerebene keine Möglichkeiten des Im- und Exports. Durch sogenannte Seed-Dateien ist jedoch ein nachträgliches Befüllen der Datenbank möglich. Der Aufruf erfordert jedoch Kenntnisse in Ruby on Rails und ist daher für Normalanwender/Redakteure nicht sinnvoll.	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 100%
In Browser CMS können Inhalte nicht importiert und exportiert werden. Entsprechende Features müssten erst eigenständig implementiert werden.		In Locomotive CMS kann ein kompletter Internetauftritt mit seinen Inhalten und Ressourcen importiert und exportiert werden. Zum Austausch der Inhalte findet eine *Zip*-Datei Verwendung, die alle benötigten Ressourcen (Bilder, Dateien, Templates usw.) sowie Inhalte der Datenbank einschließt. Ressourcen werden dabei in vordefinierten Ordnerstrukturen abgelegt. Die Datenbankeinträge aus MongoDB werden innerhalb der *Zip*-Datei im Unterordner *data* abgelegt. Die Einträge liegen dabei im *YAML*-Format vor.	

<table>
<tr><td colspan="4">Integration von Inhalten anderer Webseiten, Multimedia, Applikationen, E-Commerce-Tools</td></tr>
<tr><td>Alchemy 1.6.0</td><td>Erfüllt: 100%</td><td>Refinery CMS 1.0.8</td><td>Erfüllt: 100%</td></tr>
<tr><td colspan="2">Der verwendete WYSIWYG-Editor TinyMCE erlaubt in seiner HTML-Ansicht das Einbinden von Fremdinhalten anderer Seiten (z.B. IFrame). Zusätzlich ist die Erstellung von eigenen Inhaltselementen mit Hilfe der Alchemy Plugin DSL-API denkbar. So können auch die verschiedenen Resourcen aus der Bibliothek von Alchemy Verwendung finden. Standardmäßig verfügt Alchemy bereits über die Inhaltselemente Artikel, Text, Text mit Bild, Bilder, Bildergalerie, Überschrift und Intro.</td><td colspan="2">Refinery CMS verwaltet jede Internetseite innerhalb eines flexiblen WYSIWYG-Editors. Die Integration von vordefiniertem HTML-Code kann dabei durch Nutzung der HTML-Ansicht des Editors erreicht werden.</td></tr>
<tr><td>Browser CMS 3.3.1</td><td>Erfüllt: 100%</td><td>Locomotive CMS</td><td>Erfüllt: 100%</td></tr>
<tr><td colspan="2">Wie seine Vorgänger auch können innerhalb des WYSIWYG-Editors IFrames oder anderer HTML-Code eingebetet werden. Vordefinierte Inhaltselemente, die vorhandene Ressourcen aus der Content Library einbinden können, müssen eigenhändig angelegt werden.</td><td colspan="2">Wie bei Alchemy und Refinery CMS können innerhalb des WYSIWYG-Editors HTML-Fragmente angegeben werden.</td></tr>
</table>

3.7.2 Kontrolle

Granulares Rechte- und Rollenkonzept für Anwender			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
In Alchemy existieren vordefinierte Rollen (Registriert, Author, Redakteur,Administrator). Das Anlegen weiterer Rollen zur besseren Differenzierung ist jedoch nicht möglich.		Refinery CMS besitzt kein Rechte- und Rollenkonzept. Dem Anwender kann lediglich der Zugang zu bestimmten Plugins erlaubt oder entzogen werden, um so den Funktionsumfang einzuschränken.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 0%
In Browser CMS wird in einer Standardinstallation zwischen den Rollen Gast, CMS Administrator und Content Editor unterschieden. Zusätzlich können weitere Backend-Gruppen angelegt werden.		Locomotive CMS besitzt ein einfaches Rechte- und Rollenkonzept. Es wird zwischen Administratoren, Designern und Autoren unterschieden. Das Anlegen weiterer Gruppen ist nicht möglich.	

Granulares Berechtigungskonzept für einzelne Inhalte, Bereiche, Webseiten			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Die in Alchemy vordefinierte Rollen Registriert, Author, Redakteur und Administrator bestimmen den Funktionsumfang eines Anwenders im Backend. Angelegte Inhalte können jedoch nicht einzelnen Nutzern zugeordnet werden.		Nutzer können alle Inhalte und Bereiche einer Webseite editieren, solange sie zur Nutzung des bestimmten Plugins berechtigt wurden.	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Der Zugriff auf bestimmte Seiten (Seitenbaumzweige) kann eingeschränkt werden. Zusätzlich bietet Browser CMS ein Erstellen von Frontend-Nutzergruppen an, um so bestimmte Seiten des CMS nur exklusiv ausgewählten Nutzern zur Verfügung zu stellen. Die Zugriffsberechtigung auf installierte Plugins kann ebenfalls für jeden Nutzer individuell festgelegt werden. Leider ist es nicht möglich, einzelne Inhaltselemente für bestimmte Nutzer unzugänglich zu machen.		Der Zugriff auf Seiten und Inhalte kann nicht individuell gesteuert und beeinflusst werden. Besitzt ein Anwender das Recht zum Editieren und Anlegen von Inhalten (Nutzergruppe Redakteur), können alle Inhalte im gesamten CMS bearbeitet werden.	

Schutz vor gegenseitigem Überschreiben erfasster Inhalte durch Check in/ Check out-Mechanismen			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt. Die automatische Versionierung von Inhaltselementen erlaubt jedoch ein nachträgliches, manuelles Sichten und Zusammenfügen verschiedener Versionen.		Wird nicht unterstützt	

Versionierung von Inhalten mit Möglichkeit zur Wiederherstellung vorhergehender Versionen			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Versionierung und Wiederherstellung von Inhalten wird nicht unterstüzt.		Versionierung und Wiederherstellung von Inhalten wird nicht unterstüzt.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 0%
Wird unterstützt		Versionierung und Wiederherstellung von Inhalten wird nicht unterstüzt.	

Mandantenfähigkeit: Mehrfachnutzung des Systems durch verschiedene Parteien mit kompletter Trennung der Daten und Benutzer			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstüzt		Wird nicht unterstüzt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstüzt		In Locomotive CMS können mehrere Internetauftritte gleichzeitig verwaltet werden. Eine Trennung der verschiedenen Nutzer und Daten wird jedoch nicht angeboten.	

Linküberprüfung: Automatische Prüfung der Gültigkeit von internen und externen Links mit Möglichkeit zur Korrektur bzw. Benachrichtigung definierter Personengruppen			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstüzt		Wird nicht unterstüzt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstüzt		Wird nicht unterstüzt	

3.7.3 Freigabe

Möglichkeit für *nicht technische* User den Workflow zu kreieren, verwalten und zu ändern. Es soll dafür kein Scripting / Programming notwendig sein			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Nutzer können in Alchemy keinen Workflowprozess kreieren. Es ist jedoch möglich, das Redakteure die von Autoren durchgeführten Änderungen kontrollieren und anschließend veröffentlichen. Ein Austausch zwischen beiden Nutzergruppen ist nicht möglich (z.B. kurze Mitteilung an den Redakteur). Redakteure müssen so Änderungen der Seiteninhalte selbst erkennen.		Wird nicht unterstüzt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Ein Workflowprozess kann in Browser CMS nicht erzeugt werden. Autoren ist es nur möglich, ihre durchgeführten Änderungen an andere Backend-Nutzer mit Veröffentlichungsrechten weiterzuleiten (Simulierung eines einfachsten Workflows).		Wird nicht unterstüzt	

Definition von Workflows inkl. mehrstufiger Freigabeprozesse für die Freischaltung von Inhalten			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstüzt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstüzt	

Möglichkeit externe Benutzer in Workflows mit einbinden zu können			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstüzt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstüzt	

Unternehmensspezifische Bearbeitungsprozesse von Inhalten sollen über frei definierbare Workflows verwaltet werden können			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstüzt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstüzt	

3.7.4 Publikation

Trennung von Inhalt und Design unter Verwendung von Templates			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Inhalt und Design werden in Alchemy durch die Verwendung von *erb*-Templates getrennt. Das Haupt-Template der Seite wird zu Beginn der Entwicklung von einem Designer festgelegt und anschließend in der Anwendung verankert (als fixe Ressource im Rails Quellcode). So können Redakteure das Aussehen der Internetseite nicht beeinflussen.		Wie Alchemy verwendet Refinery CMS *erb* als Template-Sprache. Trotz der so erreichten Trennung zwischen Inhalten und Design erschwert die fehlende Möglichkeit der Anpassung im Backend den Umgang mit dem gesamten WCMS. Dies gilt für das Haupttemplate der Seite sowie für alle Erweiterungen.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Browser CMS unterstützt die Verwendung verschiedener Template-Sprachen. In einer Standard-Installation werden u.a. *erb*, *rjs* und *rxml* angeboten.		In Locomotive CMS kann für jede Seite ein Template angegeben werden. Die dabei verwendete Templatesprache ist *Liquid*.	

Mehrfachverwendung von Inhalten an verschiedenen Stellen mit unterschiedlichem Layout			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 0%
Inhaltselemente und Seiten können in Alchemy kopiert und wiederverwendet werden. Die Zuordnung eines neuen Templates muss durch den Administrator erfolgen (Änderung am Rails-Quellcode).		Inhalte und Seiten können nicht kopiert und mehrfach verwendet werden.	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Es können nur Inhalte mehrfach verwendet werden.		In Locomotive CMS kann nur für jede Seite ein neues Template angegeben werden. Inhalte sind den einzelnen Seiten zugeordnet und nur dort verwendbar.	

Navigationsstrukturen werden automatisch vom CMS generiert, publiziert und verwaltet			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Wird unterstützt		Wird unterstützt	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Wird unterstützt		Wird unterstützt	

Barrierefreiheit bei den publizierten Seiten soll unterstützt werden			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Barrierefreiheit kann bei entsprechender Umsetzung der Verwendeten Templates und CSS-Dateien erreicht werden.		Barrierefreiheit kann bei entsprechender Umsetzung der Verwendeten Templates und CSS-Dateien erreicht werden.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Barrierefreiheit kann bei entsprechender Umsetzung der Verwendeten Templates und CSS-Dateien erreicht werden.		Barrierefreiheit kann bei entsprechender Umsetzung der Verwendeten Templates und CSS-Dateien erreicht werden.	

Inhalte sollen auf verschiedene Medien / Technologien (Cross Media Publishing, SMS /Mobile / WAP / usw.) ausgegeben werden können			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	

Möglichkeit Inhalte für anderen Webseiten bereitzustellen (XML, Webservice)			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 100%
Wird nicht unterstützt		Es gibt Plugins mit optionaler Ausgabe als RSS-Feed oder XML. Eine Bereitstellung ausgewählter Inhalte ist damit möglich.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 0%
Die Darstellung von Inhalten als RSS-Feeds ist möglich. Es muss jedoch selbst eingerichtet werden.		Wird nicht unterstützt	

Möglichkeit zur Wahl zwischen dynamischer oder statischer Generierung der Seiten / Inhalte			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 0%
Wird unterstützt		Wird nicht unterstützt	

Einfache Einbindung von Fremdinhalten welche durch Drittanbieter zur Verfügung gestellt werden			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
Über den integrierten WYSIWYG-Editor können HTML-Fragmente eingebunden werden.		Über den integrierten WYSIWYG-Editor können HTML-Fragmente eingebunden werden.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Über den integrierten WYSIWYG-Editor können HTML-Fragmente eingebunden werden.		Über den integrierten WYSIWYG-Editor können HTML-Fragmente eingebunden werden.	

Automatisches Anbieten von Druckversion und Weiterempfehlen einer Webseite			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	

3.7.5 Terminierung/Archivierung

Freie Wahl des Publikationszeitraumes (zeitgesteuertes Auf- / Abschalten / Archivieren) von Inhalten			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	

Inhalte sollen archiviert werden können.			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
Wird nicht unterstützt		Wird nicht unterstützt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Refinery CMS erlaubt das Archivieren von kompletten Seiten. Es ist jedoch nicht möglich, bestimmte Inhalte zu Archivieren.		Wird nicht unterstützt	

3.8 Auswertung der Ergebnisse

Der Erfüllungsgrad der einzelnen Kriterien hat für die untersuchten WCMS folgendes Gesamtergebnis herausgestellt:

		Web Content Management System			
Bereich	**Maximum**	Alchemy CMS	Browser CMS	Locomotive CMS	Refinery CMS
Erstellung	1400	1000	1000	1000	900
Kontrolle	600	0	200	0	0
Freigabe	400	0	0	0	0
Publikation	1000	500	600	400	500
Terminierung/Archivierung	200	0	0	0	0
Gesamtpunkte	3600	1500	1800	1400	1400

Die untersuchten Systeme zeigen bei der Erfüllung der für Webpublishing relevanten Kriterien sehr unterschiedliche Resultate. Im Bereich der Inhaltserstellung vefügen die Systeme über solide Funktionalitäten. Die Webpublishingbereiche Freigabe, Kontrolle und Terminierung/Archivierung decken dagegen bei allen untersuchten Systemen funktionale Deffizite auf. In der Gesamtbetrachtung liefert Browser CMS den größten Funktionsumfang. Vor allem durch die umfassendere Nutzer- und Rechteverwaltung kann sich das System von Alchemy CMS und Locomotive CMS absetzen, die funktional gesehen im Mittelfeld angesiedelt sind. Refinery CMS bietet auf Grund der nur minimal implementierten Rechteverwaltung die geringste Gesamtfunktionalität aller untersuchten Systeme. Die Ergebnisse der durchgeführten Analyse (Kaiptel 3.7) werden im Folgenden für jedes System noch einmal separat in einer Übersicht zusammengefasst.

Alchemy CMS

Erstellung

- Umfassende Möglichkeiten zur Erstellung und Verwaltung von Seiten und verschiedenen Inhaltselementen in verschiedenen Sprachen
- Template-Erstellung und -verwaltung findet im Quellcode der Rails-Anwendung statt
- Umsetzung komplexer Seitenlayouts möglich

Kontrolle

- eingeschränkte Rechteverwaltung mit festgelegten Gruppen für Front- und Backend (Registriert, Author, Redakteur, Administrator)
- kein umfassendes Linkmanagement

Freigabe

- Einfacher Freigabeprozess für Internetseiten und Inhalte
- keine umfassende Workflowfunktionalität

Publikation

- Trennung der Inhalte vom Design durch Verwendung von Templates
- umfassende Seiten- und Inhaltsverwaltung mit Möglichkeiten zum Kopieren und Verschieben der Inhalte (Zwischenablagefunktion)

Terminierung/Archivierung

- Seiten und Inhalte können nicht archiviert oder zeitraumgebunden veröffentlicht werden

Browser CMS

Erstellung

- Umfassende Möglichkeiten zur Erstellung und Verwaltung von einsprachigen Seiten und verschiedenen Inhaltselementen
- Template-Erstellung und -verwaltung im Backend der Anwendung
- Umsetzung komplexer Seitenlayouts möglich

Kontrolle

- Rechteverwaltung mit vordefinierten Gruppen für Front- und Backend sowie Möglichkeiten zum Erstellen eigener Gruppen
- kein umfassendes Linkmanagement

Freigabe

- Einfacher Freigabeprozess für Internetseiten und Inhalte
- keine umfassende Workflowfunktionalität

Publikation

- Trennung der Inhalte vom Design durch Verwendung verschiedener Template-Sprachen
- Wiederverwendung von Inhaltenselementen

Terminierung/Archivierung

- Archivierung von erstellten Seiten
- keine zeitraumgebundene Veröffentlichung von Seiten und Inhalten möglich

Locomotive CMS

Erstellung

- Umfassende Möglichkeiten zur Erstellung und Verwaltung von einsprachigen Seiten und verschiedenen Inhaltselementen
- Inhaltselemente können im Backend der Seite individuell erstellt werden
- Umsetzung komplexe Layouts mit verschiedenen Seiten- und Inhaltselementen möglich

Kontrolle

- eingeschränkte Rechteverwaltung mit vordefinierten Gruppen für das Backend
- kein umfassendes Linkmanagement

Freigabe

- keine Abbildung von Freigabeprozessen für Internetseiten und Inhalte möglich
- keine Workflowfunktionalität

Publikation

- Trennung der Inhalte vom Design durch Verwendung der Template-Sprache Liquid
- Wiederverwendung von Inhaltenselementen
- eingeschränkte Möglichkeiten zum Verschieben von Seiten und Inhalten

Terminierung/Archivierung

- Archivierung und Terminierung von Inhalten und Seiten ist nicht möglich

Refinery CMS

Erstellung

- Erstellung und Verwaltung von mehrsprachigen Seiten und Inhaltsböcken
- Inhaltselemente können im Backend der Seite individuell erstellt werden
- Umsetzung komplexe Layouts mit Hilfe von Rails Views möglich

Kontrolle

- keine Rechteverwaltung für Seiten, Inhalte und Nutzer
- kein umfassendes Linkmanagement

Freigabe

- keine Unterstützung von Freigabeprozessen
- Erstellte Seiten können in den Bearbeitungsmodus gesetzt werden und sind dadurch nicht mehr erreichbar
- keine Workflowfunktionalität

Publikation

- Trennung der Inhalte vom Design durch Verwendung der in Rails üblichen ERB-Views
- keine Wiederverwendung von Inhaltenselementen möglich
- keine Möglichkeiten zum Verschieben und Kopieren von Inhalten

Terminierung/Archivierung

- Archivierung und Terminierung von Inhalten und Seiten ist nicht möglich

4 Konzeptionelle Problemanalyse

In diesem Abschnitt der Arbeit werden die ausgewählten Systeme auf konzeptionelle und programmiertechnische Probleme untersucht. Die Betrachtung wird dabei in folgende Teilbereiche aufgeteilt:

1. Erweiterungen
2. Nutzeroberfläche

4.1 Erweiterungen

Innerhalb des Rails Frameworks ist die Verwendung von Generatoren ein häufiges Mittel zur schnellen Entwicklung von funktionsfähigem Code (siehe Kapitel 2.1.6). Die Mehrheit der hier vorgestellten Systeme[1] bedient sich zur Realisierung neuer Inhaltslemente ebenfalls eines Generatorskripts, der ein für das jeweilige CMS funktionsbereites Grundgerüst erzeugt. Exemplarisch soll hier das Ergebnis eines Generatoraufrufes in Refinery CMS aufgezeigt werden (Abb. 4.2). Das erstellte Inhaltselement Projekt verfügt dabei über die Felder Titel und Beschreibung und wurde durch den in Abb. 4.1 dargestellten Befehlsaufruf realisiert.

Quelltext 4.1: Aufruf des Refinery Engine Generators

```
rails g refinery_engine project name:string description:text
```

Quelltext 4.2: Erzeugte Dateien des Refinery Engine Generators

```
1 create  vendor/engines/projects/app/controllers/admin/projects_controller.rb
2 create  vendor/engines/projects/app/controllers/projects_controller.rb
3 create  vendor/engines/projects/app/models/project.rb
4 create  vendor/engines/projects/app/views/admin/projects/_actions.html.erb
```

[1] Locomotive CMS benötigt durch die Möglichkeit der Erstellung individueller Inhaltselemente im Backend keine installierbaren Erweiterungsmodule

```
5  create  vendor/engines/projects/app/views/admin/projects/_form.html.erb
6  create  vendor/engines/projects/app/views/admin/projects/_projects.html.erb
7  create  vendor/engines/projects/app/views/admin/projects/_records.html.erb
8  create  vendor/engines/projects/app/views/admin/projects/_project.html.erb
9  create  vendor/engines/projects/app/views/admin/projects/_sortable_list.html.erb
10 create  vendor/engines/projects/app/views/admin/projects/edit.html.erb
11 create  vendor/engines/projects/app/views/admin/projects/index.html.erb
12 create  vendor/engines/projects/app/views/admin/projects/new.html.erb
13 create  vendor/engines/projects/app/views/projects/index.html.erb
14 create  vendor/engines/projects/app/views/projects/show.html.erb
15 create  vendor/engines/projects/config/locales/en.yml
16 create  vendor/engines/projects/config/locales/fr.yml
17 create  vendor/engines/projects/config/locales/lolcat.yml
18 create  vendor/engines/projects/config/locales/nb.yml
19 create  vendor/engines/projects/config/locales/nl.yml
20 create  vendor/engines/projects/config/routes.rb
21 create  vendor/engines/projects/db/migrate/create_projects.rb
22 create  vendor/engines/projects/db/seeds/projects.rb
23 create  vendor/engines/projects/features/manage_projects.feature
24 create  vendor/engines/projects/features/step_definitions/project_steps.rb
25 create  vendor/engines/projects/features/support/paths.rb
26 create  vendor/engines/projects/lib/generators/refinerycms_projects_generator.rb
27 create  vendor/engines/projects/lib/refinerycms-projects.rb
28 create  vendor/engines/projects/lib/tasks/projects.rake
29 create  vendor/engines/projects/readme.md
30 create  vendor/engines/projects/refinerycms-projects.gemspec
31 create  vendor/engines/projects/spec/models/project_spec.rb
```

Die erzeugten Dateien lassen sich in folgende Funktionsbereiche zusammenfassen:

Zeile 1-2 Der Generator erzeugt zwei Rails-Controller zur Steuerung der Logik im Frontend und Backend des Content Managment Systems.

Zeile 4-14 Automatische Generierung von HTML-Views zur Darstellung aller im Frontend und Backend benötigten Komponenten. U.a. werden ein HTML-Formular zum Anlegen neuer Projekte im Backend (Zeile 5 *_form.html.erb*) und eine Auflistung von zur Verfügung stehenden Aktionen im Backend erstellt (Zeile 4 *_actions.html.erb*)

Zeile 3 und 21 Erstellung des in Rails benötigten Models Project (Zeile 3 *project.rb*) und der zur Speicherung in der Datenbank benötigten Migration (Zeile 21 *create_projects.rb*)

Zeile 20 Erstellung einer Routing-Datei, welche die für das Frontend und Backend benötigten URL's bzw. Routen in der Rails-Anwendung registriert.

Zeile 15-19 Erstellung der für die Unterstützung von Mehrsprachigkeit notwendigen Konfigurationsdateien im YAML-

Zeile 23-31 Erzeugung der benötigten Dateien zur Unterstützung der Entwicklung von Tests sowie der Registrierung der Engine innerhalb der Rails-Anwendung (Zeile 27)

Bei Bedarf an weiteren Inhaltselementen ergibt sich schnell die Erkenntnis, das die Realisierung der Inhaltselemente mit Hilfe eines Generators sehr viel redundanten Rails-Code erzeugt. Damit verstößt diese Art der Erweiterungsentwicklung gegen den in Rails propagierten Ansatz des DRY (Don't repeat yourself).

Die auf Grund dieser Konzeption einhergehenden Probleme sollen im folgenden kurz zusammengefasst werden:

1. Jedes Inhaltselement besitzt seine eigene Darstellungsrepräsentierung in Form von HTML-Views. Änderungen am Backend-Design des WCMS erfordern so eine Anpassung sämlicher verwendeter Erweiterungen. Die Erweiterung und das eigentliche WCMS werden so sehr stark von einander abhängig.

2. Jedes Inhaltselement wird in einer eigenständigen Datenbanktabelle gespeichert. Ein Inhaltselement Projekt benötigt so z.B. die Datenbanktabelle Projekt mit den Tabellenfeldern Name und Beschreibung. Bei häufiger Verwendung zusätzlicher Inhaltselemente entsteht somit schnell eine beachtliche Anzahl an zusätzlichen Tabellen[2].

3. Die für die Inhaltselemente benötigten Datenbankschemas müssen in Form von in Rails üblichen Migrationen verwaltet werden. Dies erfordert einen entsprechenden Mehraufwand bei der Pflege der Erweiterungen.

4. Ein Import und Export von Inhaltselementen in das WCMS ist nur möglich, wenn die notwendigen Datenbanktabellen zuvor erzeugt wurden.

5. Inhaltselemente müssen für ihre Erreichbarkeit im Frontend und Backend des WCMS im Routing der Rails-Anwendung registriert werden. Bei der Nutzung vieler Erweiterungen entstehen so zahlreiche zusätzliche Routingeinträge.

[2]Die Zahl der Datenbanktabellen kann in Rails mit Hilfe von Single Table Inheritance (STI) minimiert werden.

6. Durch die Installation/Verwendung der umfassenden Erweiterungen wird die Gesamtgröße der Rails-Anwendung unnötig vergrößert.

7. Durch das Laden zusätzlicher Rails-Controller und Models benötigt die Rails-Anwendung zusätzlichen Arbeitsspeicher und weist einen verlängerten Boot-Prozess auf.

Der Verstoss gegen das Dry-Prinzip wird vor allem bei der Umsetzung der einzelnen Erweiterungs-Controller innerhalb von Refinery CMS ersichtlich. Dort wird mit Hilfe der im Controller verfügbaren Klassenmethode *crudify* das gesamte Grundgerüst des Controllers dynamisch erzeugt[3]. Die Methode kann dabei verschiedene Parameter aufnehmen, an Hand deren die Ausgabe gesteuert werden kann. Alle neu erzeugten Inhaltselemente sind in ihrer Grundstruktur als Rest-basierte Ressourcen anzusehen, die mit Hilfe eines angepassten Generators in das jeweilige Backend des WCMS eingebunden werden.

Quelltext 4.3: Projects-Controller mit verwendeter crudify-Methode und optionalen Parametern

```
1 module Admin
2   class ProjectsController < Admin::BaseController
3
4     crudify :project,
5             :title_attribute => 'name', :xhr_paging => true
6
7   end
8 end
```

4.2 Nutzeroberfläche

Die in den vorgestellten Content Management Systemen umgesetzten Nutzeroberflächen sind zum Großteil durch die Kombination individueller HTML, CSS und JavaScript-Dateien zusammengestellt. Diese werden dabei

[3]Die Methode *crudify* erstellt mit Hilfe von Ruby-Metaprogrammierung die gesamte Logik des Controllers. Alle Aktionen, die der Controller verwaltet, werden so erst zur Laufzeit der Rails-Anwendung generiert.

5 Lösungsvorschläge

In diesem Kapitel werden Lösungsvorschläge zur Beseitigung der in Kapitel 4 beschriebenen Probleme formuliert.

5.1 Implementierung eines Ruby on Rails Content Repository

Die in den analysierten WCMS umgesetzten Implementierungen der Erweiterungsentwicklung verstoßen zum Teil gegen das in Rails gebotene Prinzip des DRY (vgl. Kapitel 4.1). Eine konzeptionelle Änderung innerhalb dieses Systembereiches hätte somit Auswirkungen auf die gesamte WCMS-Infrastruktur der Inhaltsspeicherung. Dennoch soll hier der mögliche Lösungsansatz eines Content Repository konzeptionell vorgestellt werden.

5.1.1 Idee und Konzept

Heutige Webanwendungen (u.a. auch Web Content Management Systeme) benötigen neben der klassischen Speicherung von Daten zahlreiche zusätzliche Daten-Management-Funktionalitäten. Ein Content Repository soll dieser Entwicklung Rechnung tragen und erweitert daher das Leistungsspektrum bestehender Datenbankprodukte u.a um folgende zusätzlichen Funktionalitäten:

- Speicherung strukturierter und unstrukturierter Daten (z.B. Binär- und Textformate oder Metadaten)
- Zugangskontrolle
- Sperrung (Locking)

- Transaktionen
- Versionierung
- Überwachung der Daten
- Volltextsuche

Der Zugriff auf das Content Repository wird dabei durch eine API genau definiert und garantiert somit die Nutzung der zusäzlichen Funktionalitäten. Andere Entwickler können auf das Content Repository und seine Zusatzfunktionen kontrolliert zugreifen, ohne etwas über die Infrastruktur des Content Repository und seine Implementierungsdetails zu wissen.

Im Bereich der Java Enterprise Content Management Systeme werden Content Repositories bereits erfolgreich eingesetzt. Namhafte Beispiele sind dabei das kommerzielle ECMS CQ5 und CRX[1] von Adobe und das u.a. als Open Source Software verfügbare Alfresco ECMS[2]. Innerhalb der PHP-Entwicklergemeinde wird ebenfalls gerade an der Umsetzung eines auf PHP basierten Content Repositories gearbeitet. Initiatoren sind dabei vor allem die Typo3 Association mit einer Implementierung innerhalb des neuen Web Content Management Systems Typo3 5.0[3].

5.1.2 JCR-Das Java Content Repository

Unter der Leitung von David Nüscheler der Firma Day Software wurde 2005 die erste Version einer Implementierung und API Spezifikation eines Java Content Repository veröffentlicht[4]. Die umgesetzte Referenzimplementierung wurde später unter der Leitung der Apache Foundation als Open Source Projekt Jack Rabbit[5] der Öffentlichkeit zur Verfügung gestellt.

Zur Speicherung der Inhalte definiert das Java Content Repository ein einfaches hierarchisches Datemmodell, das folgende Objektstruktur aufweist:

[1] Nach der Übernahme von Day Inc. durch Adobe im Sommer 2010 wird das angebotene Content Management System CRX als Adobe Projekt weitervertrieben.

[2] Informationen zu Alfresco und dem Content Repository: http://www.alfresco.com/products/platform/

[3] Das TYPO3CR ist als eigenständiges Paket im PHP-FLOW3-Framework verfügbar und ein zentraler Bestandteil von Typo3 5.0

[4] Die Entwicklung ging im Java Specification Request (JSR) 170 als offizieller Standard in die Programmiersprache Java ein. Momentan wird an der Veröffentlichung der Version 2.0 des Standards gearbeitet(JSR-283).

[5] Projektseite: http://jackrabbit.apache.org/

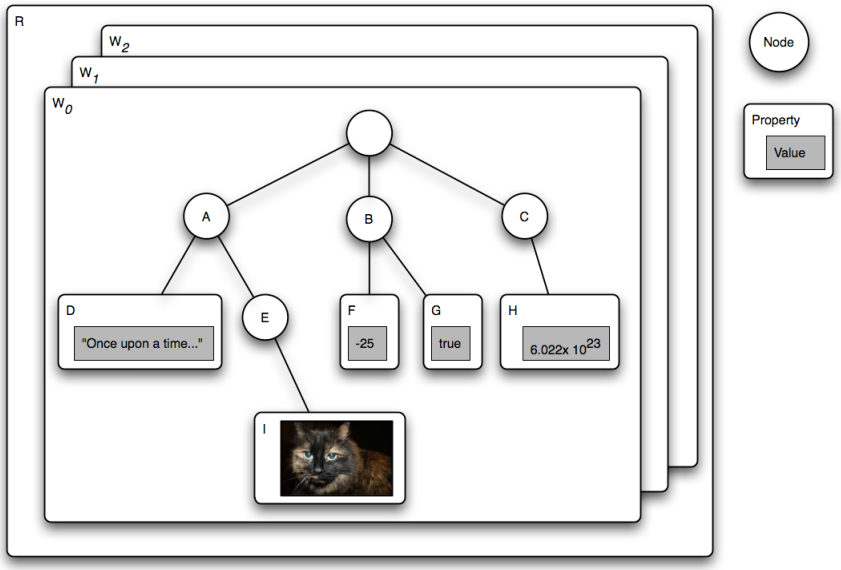

Abbildung 5.1: JCR-Datenmodell

Workspace

Ein JCR-Repository besteht aus einem oder mehreren Workspaces (Arbeitsbereichen), die jeweils mehrere Items in einer Baumstruktur beliebiger Tiefe verwalten können. Jeder Workspace lässt sich durch einen Namen eindeutig identifizieren (W_0, W_1, W_2) und enthält mindestens ein Item (root node).

Item

Ein Item kann entweder ein Node (Knoten) oder ein Property (Eigenschaft) sein.

Node

Knoten (Nodes) eines Workspaces bilden die Struktur der zu speichernden Daten ab. Ein Knoten kann daher keine oder weitere andere Kinderknoten (child notes) enthalten.

Property

Eine Property kann keine anderen Items beinhalten, aber dafür den Inhalt in Form von sogenannten Values abspeichern. Dabei kann ein Property-Node kein oder mehrere Values beinhalten.

5.1.3 Umsetzungsvarianten innerhalb von Ruby on Rails

Durch die Verfügbarkeit eines Ruby-Interpreter in Java (jruby) ergeben sich für die Umsetzung in Ruby on Rails folgende 2 Möglichkeiten:

- Verwendung der Open Source Java Implementierung Jack Rabbit und Spezifkation einer API zur Nutzung dieser innerhalb von Rails. Für die weitere Arbeit mit Rails ergibt sich daher die Abhänngigkeit zu jruby und den unterstützten Erweiterungen und Funktionalitäten. Erste Implementierungsversuche und Demonstrationsanwendungen wurden bereits erstellt[6].
- Erstellung einer eigenständigen, komplett auf Ruby und dem Rails Framework basierenden Referenzimplementierung und API nach dem Vorbild des Java Content Repository

5.1.4 Vorteile für die gewählten Ruby on Rails WCMS

Die Umsetzung eines Content Repository in Ruby kann für die bestehenden Web Content Management Systeme folgende Vorzüge bringen:

- Beseitigung des DRY-Verstosses bei den untersuchten Rails-WCMS, da die Inhalte in dem durch das Content Repository zur Verfügung gestellten hierarchischen Datenmodell gespeichert werden können.
- Wegfall der bisher notwendigen Datenbankmigrationen und zusätzlichen Datenbanktabellen, da das Content Repository die gesamte Infrastruktur bereitstellt und die Speicherung der Daten übernimmt.
- Vereinfachung des Zugriffs auf Inhalte innerhalb verschiedener Systeme durch Verwendung einer definierten API
- Fehlende Webpublishing-Funktionalitäten der existierenden Rails WCMS können durch eine Implementierung eines Content Repository als Infrastruktur allgemein zur Verfügung gestellt werden (z.B. Versionierung, Suchfunktionen).

[6] Auf Github.com wurde eine Rails 2 - Anwendung mit Verwendung der . Die Rails-Anwendung ist unter folgender Internetseite erreichbar: https://github.com/wpc/jcr-rails-demo

- Durch die im JCR-Standard festgelegten Import- und Exportfunktionen kann ein Austausch der Inhalte zwischen verschiedenen Web Content Management Systemen ermöglicht werden.
- Ein Content Repository kann als Erweiterung auch von anderen Rails-Anwendungen verwendet werden, die mit verschiedenartig strukturierten Inhalten umgehen müssen.

5.2 Übertragung des Typo3 5.0 Phoenix User-Interfaces in Rails 3.1

Content Management Systeme erfordern bei steigender Funktionalität ein entsprechend komplexeres Nutzer-Interface. Die sinnvolle Realisierung entsprechender Oberflächen ist mit der u.a. in Refinery CMS gewählten Generierung von HTML-Views und einzelnen JavaScript-Dateien nur noch schwer möglich. Zur Umsetzung solcher Projekte empfiehlt sich daher der Einsatz alternativer Technologien. Die neue Version 5.0 des PHP basierten Web Content Management Systems Typo3 greift bei der Generierung der gesamten Backend-Oberfläche auf das Java Script Frameworks Ext JS 4[7] zurück. Es ermöglicht durch Kombinierung vorgefertigter Elemente eine schnelle Erstellung komplexer Oberflächen[8]. Die Veröffentlichung des Typo3-Projektes unter der GPLv3-Lizenz macht eine Weiternutzung der in der Version 5.0 geplanten Oberfläche generell möglich. Im folgenden sollen daher die notwendigen Schritte zur Integrierung des Typo3 5.0 User-Interfaces in eine Rails 3.1 Anwendung beschrieben werden.

5.2.1 Typo3 5.0

Die Entwicklung von Typo3 5.0 befindet sich noch in einer frühen Phase. Interessenten können jedoch bereits Entwicklerversionen in Form sogenannter Sprint Releases herunterladen und testen. Neben einem Download-Paket[9] auf der Projektseite von Typo3 wird zusätzlich eine aktuelle Version von Typo3 als Live-Demo[10] angeboten. Die Zugangs-

[7] Informationen und Download: http://www.sencha.com/
[8] Beispielanwendungen: http://dev.sencha.com/deploy/ext-4.0.0/examples/
[9] Komponenten-Download: http://flow3.typo3.org/typo3-phoenix/
[10] Demoseite: http://phoenix.demo.typo3.org/

daten zum Backend können im Frontend der Seite mit Hilfe eines Formulars erzeugt werden.

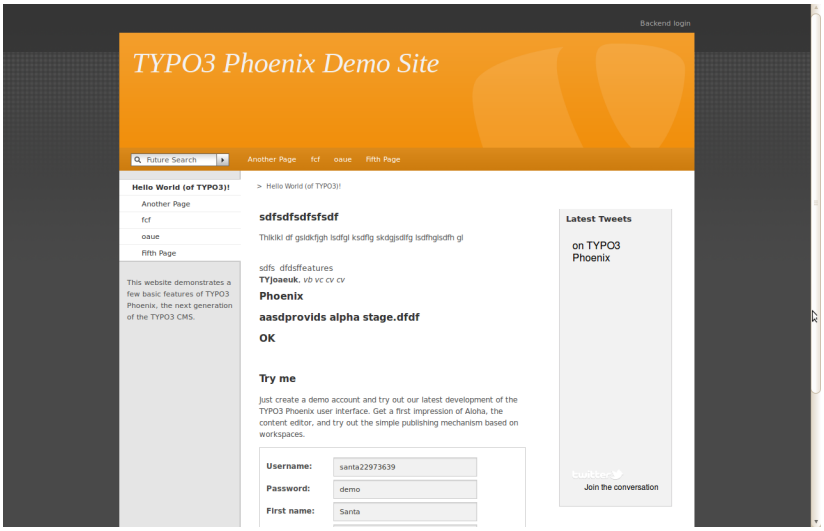

Abbildung 5.2: Frontend-Ansicht der Typo3 5.0 Sprint Release 6 Demoversion

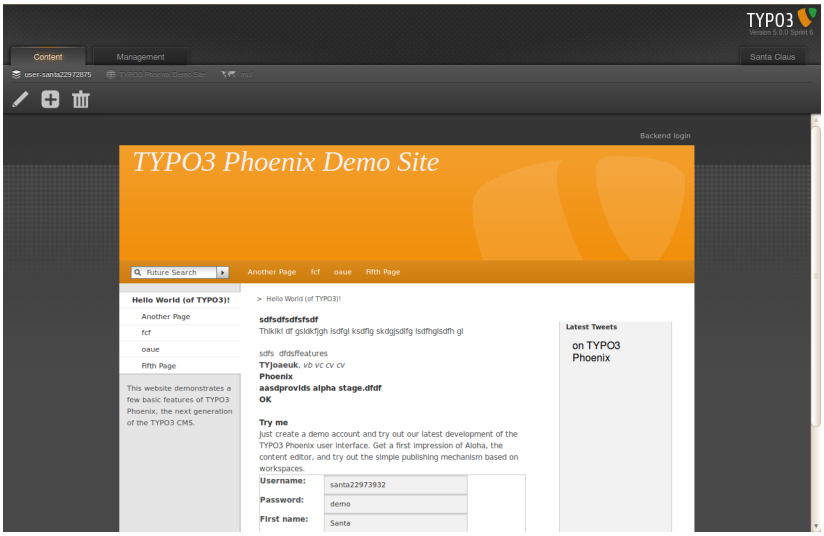

Abbildung 5.3: Backend-Ansicht der Typo3 5.0 Sprint Release 6 Demoversion

Das in der Demoversion umgesetzte Nutzer-Interface repräsentiert nur einen Teil der für Typo3 5.0 geplanten Oberfläche und Komponenten[11]. U.a. sind folgende Bestandteile bereits umgesetzt wurden:

- Login-Seite zur Anmeldung im Backend von Typo3 5.0
- Content-Modul im Backend mit integrierter Vorschau der aktuell ausgewählten Seite und beschränkten Möglichkeiten der Inhaltsbearbeitung mit Hilfe des für Typo3 5.0 geplanten Aloha-Editors
- Management-Modul zur Verwaltung der im System angelegten Seiten in Form einer Baumstruktur
- Dashboard des angemeldeten Nutzers mit Auflistung der editierten Inhalte

5.2.2 Ext JS und Ext Direct

Das Typo3-User-Interface setzt bei der Kommunikation zwischen den einzelnen Interface-Komponenten und dem Server auf die Nutzung von Ext Direct. Ext Direct beschreibt dabei einen in Ext JS definierten Sprachstandard, mit dessen Hilfe serverseitige Funktion clientseitig per Javascript aufgerufen (gesteuert) werden können. Für ein besseres Verständnis wird in Abbildung 5.1 eine im jQuery Framework formulierte Ajax-Anfrage und der entsprechende Ext Direct Ausdruck gegenübergestellt.

Quelltext 5.1: Ajax-Anfrage an einen Server im jQuery-Framework

```
1
2  // jQuery Ajax Request
3  $.ajax({
4    url: '/ajax?ajaxID=MyController_myMethod&amp;parameter1=someValue',
5     success: function( data ) {
6      alert(data);
7      }
8    }
9  });
10
11 // Ext Direct Aufruf
12 Namespace.MyController.myMethod("some_Value", function(data, e) {
13    alert(data);
14 });
```

[11] Bilder und ausführliche Erläuterungen zur neuen Typo3 5.0 Oberfläche sind unter folgender Adresse zu finden: http://typo3.org/teams/usability/t35ui/

Beide Javascript-Beispiele resultieren in einer asynchronen Ajax-Anfrage, die die entsprechende Methode serverseitig aufruft. Für Ext Direct ergeben sich jedoch folgende zusätzliche Vorteile:

1. Keine wiederholte Angabe einer URL zum Aufruf der serverseitigen Funktionen.
2. Vereinfachung des Javascript-Quellcodes, da im Vergleich zu herkömmlichen asynchronen Anfragen weniger Quellcode (Javascript) geschrieben werden muss
3. Vereinfachung der cleintseitigen Javascript-Programmierung, da der Aufruf von client- und serverseitigen Methoden namentlich übereinstimmt.

Eine umfassende Beschreibung von Ext Direct ist dem Anhang der Arbeit beigefügt (Anhang 7.3). Dort werden alle Konzepte und notwendigen Komponenten für eine serverseitige Unterstützung von Ext Direct erläutert.

Um Ext Direct auch innerhalb einer Rails-Anwendung nutzbar zu machen, wurde im Rahmen dieser Diplomarbeit die Rails 3.1 kompatible Erweiterung *extr* entwickelt[12]. Der für Ext Direct notwendige Router (Anhang 7.3) wurde mit Hilfe einer Rack Middleware realisiert, die alle Ext JS-Anfragen an die entsprechenden Servermethoden (Controller-Methoden) weiterleitet. Schematisch ergibt sich somit folgender Anfrageablauf innerhalb der Rails-Anwendung:

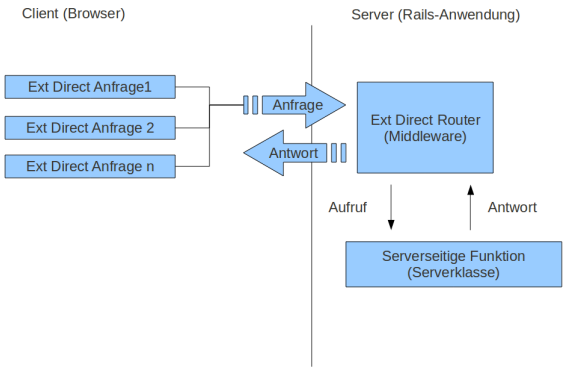

Abbildung 5.4: Schema des Ext Direct Routing mit *extr*

[12]Für Ruby und das Rails-Framework existieren bereits Implementierungen, die jedoch nicht vollständig zu Rails 3 und 3.1 kompatibel sind. Aus diesem Grund wurde die Umsetzung einer eigenen Erweiterung ins Auge gefasst.

Die Installation und Nutzung der Erweiterung *extr* wird im Anhang dieser Arbeit demonstriert. Auf Grund des frühen Entwicklungsstatus unterstützt die Erweiterung serverseitig noch nicht alle der geforderten Ext-Direct-Funktionalitäten. Im folgenden sollen bestehende Beschränkungen und Probleme erläutert werden:

- Ext Direct-Anfragen mit Parametern aus Formularen und Datei-Uploads werden von der Rails Middleware noch nicht vollständig an die entsprechende Zielmethode (controller action) weitergeleitet.
- kein Schutz vor CSRF/XSRF-Angriffen[13]

[13] http://guides.rubyonrails.org/security.html#cross-site-request-forgery-csrf

6 Zusammenfassung

6.1 Fazit

Im Rahmen dieser Arbeit konnten die ausgewählten Ruby on Rails Web Content Management Systeme Alchemy CMS, Browser CMS, Locomotive CMS und Refinery CMS auf ihre Webpublishing-Fähigkeiten untersucht werden. Dabei zeigte sich, dass der allgemein in Content Management Systemen etablierte Redaktionsprozess (Erstellung, Verwaltung, Publikation und Archivierung) sehr unterschiedlich unterstützt wird. Ein Einsatz der Systeme kann daher nur empfohlen werden, wenn in einer Voranalyse die Anforderungen an die tatsächlich umzusetzende Internetseite genau definiert und mit dem gewünschten Rails WCMS abgeglichen werden. Zwar bietet Rails komfortable Möglichkeiten zur Anpassung der Systeme, dies erfordert jedoch einen erhöhten Mehraufwand im Vergleich zu verbreiteten Lösungen wie Typo3 oder Drupal, die im Bereich des Webpublishing bereits einen Großteil der hier untersuchten Funktionalitäten erfüllen. Die in Kapitel 4 aufgezeigten Implementierungsdetails zeigen darüber hinaus, das die Systeme hinsichtlich ihres Datenbankdesigns und der User-Interface-Umsetzung Optimierungspotenzial besitzen. Vor allem der in Rails übliche Einsatz von Generatorskripten und HTML-Views bei der Erweiterungsentwicklung lässt die Arbeit mit diesen Systemen schnell unübersichtlich und aufwendig erscheinen.

6.2 Ausblick

Die vorgestellten WCMS sind zum Teil erst innerhalb der letzten 2 Jahre (2009) an die Öffentlichkeit übergeben wurden. Durch die Leistungen der Open Source-Bewegung ist daher mit weiteren funktionalen und technischen Verbesserungen zu rechnen. Die Umsetzung eines rails-basierten Content Repository kann dabei einen wichtigen Entwicklungsimpuls innerhalb der Ruby on Rails Web Content Management Systeme bedeuten.

7 Anhang

7.1 Liste bestehender Rails 2 und 3 Web Content Management Systeme bzw. Blogging-Software

OpenSource Web Content Management Systeme mit Rails 2.x Unterstützung		
Projektname	**Projektseite im Internet**	**Aktive Weiterentwicklung**
Alchemy CMS	http://magiclabs.github.com/alchemy/	Ja
Skyline CMS	http://www.skylinecms.nl/	Ja
Webiva	http://webiva.org/	Ja
Railfrog	http://railfrog.com/	Nein
Radiant	http://radiantcms.org/	Ja
zenacms	http://zenadmin.org/	Ja
Compages	http://compages.wordpress.com/	eingestellt
Comatose	http://comatose.rubyforge.org/	eingestellt
Mephisto	https://github.com/halorgium/mephisto	eingestellt
Rubricks	http://rubricks.org/	eingestellt
Typhus	http://typus.heroku.com/	eingestellt
Station	https://github.com/atd/station	Ja
Vrame	https://github.com/9elements/vrame	eingestellt
Ansuz CMS	https://github.com/knewter/ansuz	eingestellt
Geego CMS	http://gitorious.org/geego-cms#more	eingestellt

Open Source Web Content Management Systeme mit Rails 3.x Unterstützung		
Projektname	**Projektseite im Internet**	**Aktive Weiterentwicklung**
Alchemy CMS	http://magiclabs.github.com/alchemy/	Ja
Browser CMS	http://browsercms.org	Ja
Locomotive CMS	http://www.locomotivecms.com/	Ja
Refinery CMS	http://refinerycms.com/	Ja
adva-cms	http://adva-cms.org/	Ja
typo (Blogging)	http://fdv.github.com/typo/	Ja
Surtout CMS	http://surtoutcms.com/	noch nicht veröffentlicht
Nesta CMS	https://github.com/gma/nesta	ja

Closed Source Web Content Management Systeme mit Rails 3.x Unterstützung		
Projektname	**Projektseite im Internet**	**Aktive Weiterentwicklung**
Blocks	http://www.blocksglobal.com/	Ja

7.2 Crudify-Methode von Refinery CMS 1.0.8

Implementierung der Crudify-Methode in Refinery CMS 1.0.8

```
1  # Base methods for CRUD actions
2  # Simply override any methods in your action controller you want to be customised
3  # Don't forget to add:
4  #   resources :plural_model_name_here
5  # or for scoped:
6  #   scope(:as => 'module_module', :module => 'module_name') do
7  #     resources :plural_model_name_here
8  #   end
9  # to your routes.rb file.
10 # Full documentation about CRUD and resources go here:
11 # -> http://api.rubyonrails.org/classes/ActionDispatch/Routing/Mapper/Resources.html#
       method-i-resources
12 # Example (add to your controller):
13 # crudify :foo, :title_attribute => 'name' for CRUD on Foo model
14 # crudify :'foo/bar', :title_attribute => 'name' for CRUD on Foo::Bar model
15 # Note: @singular_name will result in @foo for :foo and @bar for :'foo/bar'
16
17 module Refinery
18   module Crud
19
20     def self.default_options(model_name)
21       singular_name = model_name.to_s.split('/').last
22       class_name = "::#{model_name.to_s.camelize.gsub('/', '::')}".gsub('::::', '::')
23       plural_name = singular_name.to_s.gsub('/', '_').pluralize
24       this_class = class_name.constantize.base_class
25
26       {
27         :conditions => '',
28         :include => [],
29         :order => ('position ASC' if this_class.table_exists? and this_class.
               column_names.include?('position')),
30         :paging => true,
31         :per_page => false,
32         :redirect_to_url => "main_app.refinery_admin_#{plural_name}_path",
33         :searchable => true,
34         :search_conditions => '',
35         :sortable => true,
36         :title_attribute => "title",
37         :xhr_paging => false,
38         :class_name => class_name,
39         :singular_name => singular_name,
40         :plural_name => plural_name
41       }
42     end
43
44     def self.append_features(base)
```

```
45        super
46        base.extend(ClassMethods)
47      end
48
49      module ClassMethods
50
51        def crudify(model_name, options = {})
52          options = ::Refinery::Crud.default_options(model_name).merge(options)
53          class_name = options[:class_name]
54          singular_name = options[:singular_name]
55          plural_name = options[:plural_name]
56
57          module_eval %(
58            def self.crudify_options
59              #{options.inspect}
60            end
61
62            prepend_before_filter :find_#{singular_name},
63                                  :only => [:update, :destroy, :edit, :show]
64
65            def new
66              @#{singular_name} = #{class_name}.new
67            end
68
69            def create
70              # if the position field exists, set this object as last object, given the
                    conditions of this class.
71              if #{class_name}.column_names.include?("position")
72                params[:#{singular_name}].merge!({
73                  :position => ((#{class_name}.maximum(:position, :conditions => #{
                        options[:conditions].inspect})||-1) + 1)
74                })
75              end
76
77              if (@#{singular_name} = #{class_name}.create(params[:#{singular_name}])).
                    valid?
78                (request.xhr? ? flash.now : flash).notice = t(
79                  'refinery.crudify.created',
80                  :what => "'\#{@#{singular_name}.#{options[:title_attribute]}}'"
81                )
82
83                unless from_dialog?
84                  unless params[:continue_editing] =~ /true|on|1/
85                    redirect_back_or_default(#{options[:redirect_to_url]})
86                  else
87                    unless request.xhr?
88                      redirect_to :back
89                    else
90                      render :partial => "/refinery/message"
91                    end
92                  end
```

```
93                else
94                  render :text => "<script>parent.window.location = '\#{#{options[:
                        redirect_to_url]}}';</script>"
95                end
96              else
97                unless request.xhr?
98                  render :action => 'new'
99                else
100                 render :partial => "/refinery/admin/error_messages",
101                        :locals => {
102                          :object => @#{singular_name},
103                          :include_object_name => true
104                        }
105               end
106             end
107           end
108
109           def edit
110             # object gets found by find_#{singular_name} function
111           end
112
113           def update
114             if @#{singular_name}.update_attributes(params[:#{singular_name}])
115               (request.xhr? ? flash.now : flash).notice = t(
116                 'refinery.crudify.updated',
117                 :what => "'\#{@#{singular_name}.#{options[:title_attribute]}}'"
118               )
119
120               unless from_dialog?
121                 unless params[:continue_editing] =~ /true|on|1/
122                   redirect_back_or_default(#{options[:redirect_to_url]})
123                 else
124                   unless request.xhr?
125                     redirect_to :back
126                   else
127                     render :partial => "/refinery/message"
128                   end
129                 end
130               else
131                 render :text => "<script>parent.window.location = '\#{#{options[:
                        redirect_to_url]}}';</script>"
132               end
133             else
134               unless request.xhr?
135                 render :action => 'edit'
136               else
137                 render :partial => "/refinery/admin/error_messages",
138                        :locals => {
139                          :object => @#{singular_name},
140                          :include_object_name => true
141                        }
```

```
142               end
143             end
144           end
145
146           def destroy
147             # object gets found by find_#{singular_name} function
148             title = @#{singular_name}.#{options[:title_attribute]}
149             if @#{singular_name}.destroy
150               flash.notice = t('destroyed', :scope => 'refinery.crudify', :what => "
                      '\#{title}'")
151             end
152
153             redirect_to #{options[:redirect_to_url]}
154           end
155
156           # Finds one single result based on the id params.
157           def find_#{singular_name}
158             @#{singular_name} = #{class_name}.find(params[:id],
159                                                    :include => #{options[:include].map
                                                         (&:to_sym).inspect})
160           end
161
162           # Find the collection of @#{plural_name} based on the conditions specified
                  into crudify
163           # It will be ordered based on the conditions specified into crudify
164           # And eager loading is applied as specified into crudify.
165           def find_all_#{plural_name}(conditions = #{options[:conditions].inspect})
166             @#{plural_name} = #{class_name}.where(conditions).includes(
167                                 #{options[:include].map(&:to_sym).inspect}
168                               ).order("#{options[:order]}")
169           end
170
171           # Paginate a set of @#{plural_name} that may/may not already exist.
172           def paginate_all_#{plural_name}
173             # If we have already found a set then we don't need to again
174             find_all_#{plural_name} if @#{plural_name}.nil?
175
176             per_page = if #{options[:per_page].present?.inspect}
177               #{options[:per_page].inspect}
178             elsif #{class_name}.methods.map(&:to_sym).include?(:per_page)
179               #{class_name}.per_page
180             end
181
182             @#{plural_name} = @#{plural_name}.paginate(:page => params[:page], :
                    per_page => per_page)
183           end
184
185           # Returns a weighted set of results based on the query specified by the user.
186           def search_all_#{plural_name}
187             # First find normal results.
188             find_all_#{plural_name}(#{options[:search_conditions].inspect})
```

```
189
190             # Now get weighted results by running the query against the results already
                    found.
191             @#{plural_name} = @#{plural_name}.with_query(params[:search])
192           end
193
194           # Ensure all methods are protected so that they should only be called
195           # from within the current controller.
196           protected :find_#{singular_name},
197                     :find_all_#{plural_name},
198                     :paginate_all_#{plural_name},
199                     :search_all_#{plural_name}
200         )
201
202         # Methods that are only included when this controller is searchable.
203         if options[:searchable]
204           if options[:paging]
205             module_eval %(
206               def index
207                 search_all_#{plural_name} if searching?
208                 paginate_all_#{plural_name}
209
210                 render :partial => '#{plural_name}' if #{options[:xhr_paging].inspect}
                        && request.xhr?
211               end
212             )
213           else
214             module_eval %(
215               def index
216                 unless searching?
217                   find_all_#{plural_name}
218                 else
219                   search_all_#{plural_name}
220                 end
221               end
222             )
223           end
224
225         else
226           if options[:paging]
227             module_eval %(
228               def index
229                 paginate_all_#{plural_name}
230
231                 render :partial => '#{plural_name}' if #{options[:xhr_paging].inspect}
                        && request.xhr?
232               end
233             )
234           else
235             module_eval %(
236               def index
```

```
237                 find_all_#{plural_name}
238               end
239             )
240           end
241
242         end
243
244         if options[:sortable]
245           module_eval %(
246             def reorder
247               find_all_#{plural_name}
248             end
249
250             # Based upon http://github.com/matenia/jQuery-Awesome-Nested-Set-Drag-and-
                    Drop
251             def update_positions
252               previous = nil
253               params[:ul].each do |_, list|
254                 list.each do |index, hash|
255                   moved_item_id = hash['id'].split(/#{singular_name}\\_?/)
256                   @current_#{singular_name} = #{class_name}.find_by_id(moved_item_id)
257
258                   if @current_#{singular_name}.respond_to?(:move_to_root)
259                     if previous.present?
260                       @current_#{singular_name}.move_to_right_of(#{class_name}.
                              find_by_id(previous))
261                     else
262                       @current_#{singular_name}.move_to_root
263                     end
264                   else
265                     @current_#{singular_name}.update_attribute(:position, index)
266                   end
267
268                   if hash['children'].present?
269                     update_child_positions(hash, @current_#{singular_name})
270                   end
271
272                   previous = moved_item_id
273                 end
274               end
275
276               #{class_name}.rebuild! if #{class_name}.respond_to?(:rebuild!)
277               render :nothing => true
278             end
279
280             def update_child_positions(node, #{singular_name})
281               node['children']['0'].each do |_, child|
282                 child_id = child['id'].split(/#{singular_name}\_?/)
283                 child_#{singular_name} = #{class_name}.where(:id => child_id).first
284                 child_#{singular_name}.move_to_child_of(#{singular_name})
285
```

```
286                 if child['children'].present?
287                   update_child_positions(child, child_#{singular_name})
288                 end
289               end
290             end
291           )
292         end
293
294         module_eval %(
295           def self.sortable?
296             #{options[:sortable].to_s}
297           end
298
299           def self.searchable?
300             #{options[:searchable].to_s}
301           end
302         )
303
304
305       end
306
307     end
308
309   end
310 end
```

7.3 Ext-Direct Spezifikation für Ext Js 3.0

Ext.Direct
Specification Draft

Ext.Direct is a platform and language agnostic technology to remote server-side methods to the client-side. Furthermore, Ext.Direct allows for seamless communication between the client-side of an Ext application and any server-side platform. Ext 3.0 will ship with 5 server-side implementations for Ext.Direct. Each of these are called Ext.Direct stacks (need better name).

Server-side Stacks available at Ext 3.0 release:

- PHP
- Java
- .NET
- ColdFusion
- Ruby – (Merb)

There are many optional pieces of functionality in the Ext.Direct specification. The implementor of each server-side implementation can include or exclude these features at their own discretion. Some features are not required for particular languages or may lack the features required to implement the optional functionality.

Some examples of these features are:

- Remoting methods by metadata – Custom annotations and attributes
- Programmatic, JSON or XML configuration
- Type conversion
- Asynchronous method calls
- Method batching
- File uploading

This specification is being released into the open so that community members can make suggestions and improve the technology. If you do implement a router in a different language and/or server-side framework and are interested in contributing it back to Ext, please let us know.

An Ext.Direct server-side stack needs at least 3 key components in order to function.

The name and job of each of these components:

- Configuration – To specify which components should be exposed to the client-side
- API – To take the configuration and generate a client-side stub

- Router – To route requests to appropriate classes, components or functions

Configuration

There are 4 typical ways that a server-side will denote what classes need to be exposed to the client-side. These are Programmatic, JSON, XML configurations and metadata.

Languages which have the ability to do dynamic introspection at runtime may require less information about the methods to be exposed because they can dynamically determine this information at runtime.

Some considerations that are language specific:

- Ability to specify named arguments to pass an argument collection. When using an argument collection order of the arguments does not matter.
- Ability to make an argument optional
- Ability to dynamically introspect methods at runtime
- Ability to associate metadata with classes or methods
- Requirement to convert an argument to a specific class type before invoking the method on the server-side
- Requirement to specify how many arguments each method will get

As you can see different languages require different approaches to describe their server-side methods. What may work for one language will may not be enough information the other. The important outcome of the configuration is to describe the methods that need to be remotable, their arguments and how to execute them.

Programmatic Configurations:

Programmatic configuration builds the configuration by simply creating key/value pairs in the native language. (Key-Value Data Structures are known by many names: (HashMap, Dictionary, Associative Array, Structure, Object). Please use the term which feels most natural to the server-side of your choice.

PHP Example:

```
$API = array(
  'AlbumList'=>array(
      'methods'=>array(
          'getAll'=>array(
              'len'=>0
          ),
          'add'=>array(
              'len'=>1
          )
      )
  )
);
```

Here we are instructing the server-side that we will exposing 2 methods of the AlbumList class getAll and add. The getAll method does not accept any arguments and the add method accepts a single argument. The PHP implementation does not need to know any additional information about the arguments, their name, their type or their order. Other server-side implementations may need to know this information and will have to store it in the configuration.

JSON Configuration:

JSON Configuration stores the exact same information in a JSON file on the file-system and then reads that in.

```
{
    AlbumList: {
        methods: {
            getAll: {
                len: 0
            },
            add: {
                len: 1
            }
        }
    }
}
```

XML Configuration:

XML Configuration stores the exact same information in an XML file on the file-system and then reads that in.

```
<AlbumList>
    <methods>
        <method name="getAll" len="0" />
        <method name="add" len="1" />
    </methods>
</AlbumList>
```

Configuration by Metadata:

PHP does not support adding custom metadata to methods.

Consider this example of a ColdFusion component which has added a custom attribute to each method (cffunction) called remotable.

ColdFusion Example:

```
<cfcomponent name="AlbumList">
    <cffunction name="getAll" remotable="true">
        <cfreturn true />
    </cffunction>
    <cffunction name="add" remotable="true">
        <cfargument name="album" />
        <cfreturn true />
    </cffunction>
</cffunction>
```

The ColdFusion Ext.Direct stack is able to introspect this component and determine all of the information it needs to remote the component via this custom attribute.

API

The API component of the Ext.Direct stack has an important function. It's job is to transform the configuration about the methods to be remoted into client-side stubs. By generating these proxy methods we can seamlessly call server-side methods as if they were client-side methods without worrying about the interactions between the client and server-side.

The API component will be included via a script tag in the head of the application. The server-side will dynamically generate an actual JavaScript document to be executed on the client-side.

The JS will describe the methods which have been read in via the configuration file. This example uses the variable name Ext.app.REMOTING_API. It will describe the url, the type and available actions. (Class and method).

By including the API.php file via a script tag. The server side will return the following:

```
Ext.app.REMOTING_API = {
    "url":"remote\/router.php",
    "type":"remoting",
    "actions":{
        "AlbumList":[{
            "name":"getAll",
            "len":0
        },{
            "name":"add",
            "len":1
        }]
    }
};
```

Router

The router accepts requests from the client-side and *routes* them to the appropriate Class, method and passes the proper arguments.

Requests can be sent in two ways, via JSON-Encoded raw HTTP Post or a form post. When uploading files the form post method is required.

JSON-Encoded raw HTTP posts look like:

```
{"action":"AlbumList","method":"getAll","data":[],"type":"rpc","tid":2}
```

The router needs to decode the raw HTTP post. If the http POST is an array, the router is to dispatch multiple requests. A single request has the following attributes:

- action – The class to use
- method – The method to execute
- data – The arguments to be passed to the method – array or hash
- type - “rpc” for all remoting requests
- tid – Transaction ID to associate with this request. If there are multiple requests in a single POST these will be different.

Form posts will be sent the following form fields.

- extAction – The class to use
- extMethod – The method to execute
- extTID – Transaction ID to associate with this request
- extUpload – (optional) field is sent if the post is a file upload
- Any additional form fields will be assumed to be arguments to be passed to the method.

Once a request has been accepted it must be dispatched by the router. The router must construct the relevant class and call the method with the appropriate arguments which it reads from *data.* For some languages this will be an array, for others it will be a hash. If it is an array then the order of the arguments will matter. If it is hash then the named arguments will be passed as an argument collection.

NOTE: At this point there is no mechanism to construct a class with arguments. It will ONLY use the default constructor.

The response of each request should have the following attributes in a key-value pair data structure.

- type – 'rpc'
- tid – The transaction id that has just been processed
- action – The class/action that has just been processed
- method – The method that has just been processed
- result – The result of the method call

This response will be JSON encoded. The router can send back multiple responses with a single request enclosed in an array.

If the request was a form post and it was an upload the response will be sent back as a valid html document with the following content:

```
<html><body><textarea></textarea></body></html>
```

The response will be encoded as JSON and be contained within the textarea.

Form Posts can only execute one method and do not support batching.

If an exception occurs on the server-side the router should also return the following error information when the router is in **debugging** mode.

- type – 'exception'
- message – Message which helps the developer identify what error occurred on the server-side
- where – Message which tells the developer where the error occurred on the server-side.

This exception handling within the router should have the ability to be turned on or off. This exception information should never be sent back to the client in a production environment because of security concerns.

Exceptions are meant for server-side exceptions. Not application level errors.

Optional Router Features

- Ability to specify before and after actions to execute allowing for aspect oriented programming. This is a powerful concept which can be used for many uses such as security.

Client Side portions of Ext.Direct

To use Ext.Direct on the client side you will need to add each provider by the variable name you used in the API. For example:

```
Ext.Direct.addProvider(Ext.app.REMOTING_API);
```

After adding the provider all of your actions will exist in the global namespace.
The client-side will now be able to call the exposed methods as if they were on the server-side.

- AlbumList.getAll
- AlbumList.add

An additional argument will be added to the end of the arguments allowing the developer to specify a callback method. Because these methods will be executed asynchronously it is important to note that we must use the callback to process the response. We will not get the response back immediately.

For example we do NOT want to do this:

```
var albums = AlbumList.getAll();
```

This code should be written like this:

```
AlbumList.getAll(function(url, response) {
  // process response
});
```

7.4 Nutzung von Extr in einer Rails 3.1-Anwendung

Im folgenden Abschnitt wird die Installation und Nutzung des erstellten Plugins Extr in einer neu erstellten Rails 3-Anwendung demonstriert. Das erstellte Paket ist noch in der Entwicklungsphase und konnte nicht vollständig auf seine Sicherheit überprüft werden[2]. Der Quellcode der Anwendung kann unter folgender Adresse heruntergeladen werden:

Extr: https://github.com/skeller1/extr

Für die folgenden Beschreibungen wird eine funktionsbereite Rails 3.1-Installation vorausgesetzt.

1. Erstellung einer neuen Rails 3-Anwendung *directtest*

```
rails new directtest
```

2. Aktivierung des Plugins/Gems in der Gemfile der erstellten Rails-Anwendung

```
# ...
gem "extr", :git => "git://github.com/skeller1/extr.git"
# ...
```

3. Aktivierung des Plugins/Gems in der Gemfile der Rails-Anwendung

Gemfile

```
# ...
gem "extr", :git => "git://github.com/skeller1/extr.git"
# ...
```

4. Erstellung einer Rails 3 Standard-Ressource Projekt und Erzeugung der notwendigen Datenbanktabelle.

```
rails g scaffold project name:string description:text
rake db:migrate
```

5. Einbinden der notwendigen Ext-Direct-Bibliotheken mit Hilfe der vom Plugin zur Verfügung gestellten Helfermethoden (View Helpers). Der Namensraum des Ext-Direct-Provider ist *Rails*.

[2]Sicherheitsbedenken in den Bereichen CSRF und XSS wurden bei der Vorstellung des Paktes aufgezeigt.

app/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
<head>
  <title>Extr</title>
  <%= stylesheet_link_tag "extr/application" %>
  <%= javascript_include_tag "extr/application" %>
  <%= csrf_meta_tags %>
  <%= ext_base_tag %>
  <%= ext %>
  <%= ext_direct_provider "Rails" %>
</head>
<body>
    <%= yield %>
</body>
</html>
```

6. Registrierung der Ext-Direct-fähigen Controller-Methoden (Actions) im Projekte-Controller (projects controller). Die erstellten Methoden sind nur als Beispiel zu verstehen und können entsprechend angepasst werden.

app/controllers/projects_controller.rb

```
class ProjectsController < ApplicationController

  # activate extr for this controller
  include Extr::DirectController


  #disable Rails 3 authenticity_token for demonstration, not recom. in production
  skip_before_filter :verify_authenticity_token


  #register 2 ext direct controller action with different, optional controller
       name (MyDirectController)
  direct "MyDirectController",
    :getChildProject => 1,
    :someOtherMethod => 2


  def getChildProject
    # render a random project name as json response
    render :json => {:name => "Project#{Random.rand(11)}"}.to_json
  end
```

7. Im Index-HTML-View der Projekt-Ressource (http://localhost:3000/projects) kann nun durch einen Javascript-Aufruf eine Anfrage an die registrierten Ext-Direct-Controller-Methoden gesendet werden. Abbildung 7.1 zeigt das Ergebnis eines mit

Hilfe des Mozilla Firefox Plugins Firebug temporär ausgeführten JavaScript-Codes. Als Javascript-Callback wird der Name des Projektes zurückgegeben (*Project9*).

Javascript zum Aufruf des Projekt-Controllers in der Rails-Anwendung

```
1 Rails.MyDirectController.getChildProject("param",function(r,e){
2  alert(r.name);
3 });
```

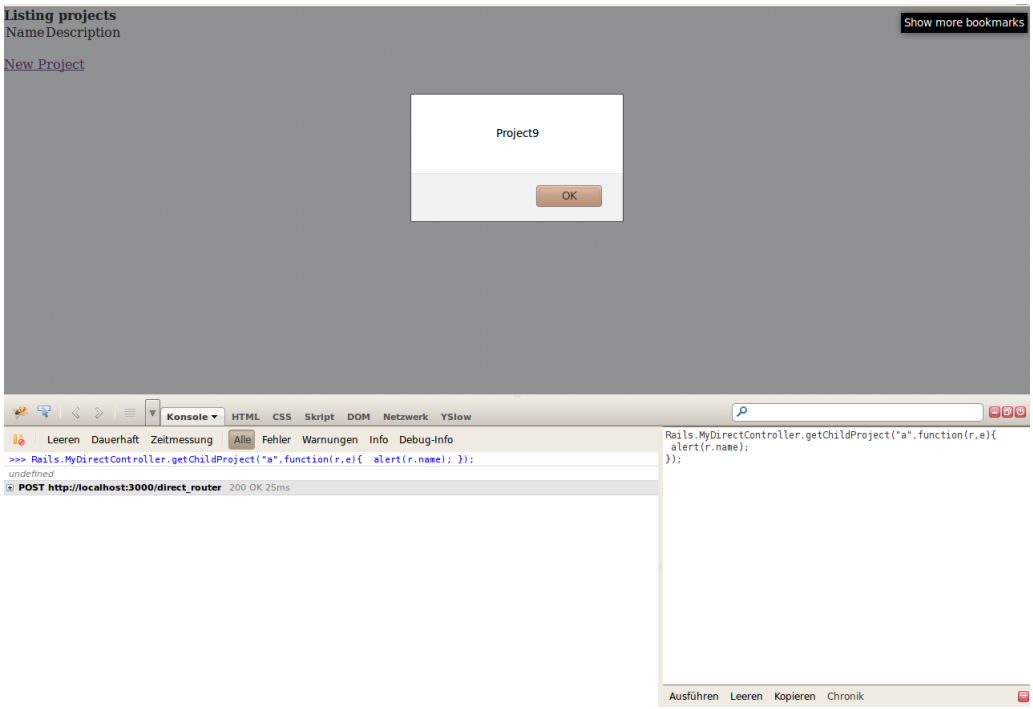

Abbildung 7.1: Mit Hilfe von Firebug formulierte Javascript-Ext-Direct-Anfrage mit ausgegebener Antwort der Rails-Anwendung

7.5 Java Content Repository Spezifikation

8 Literaturverzeichnis

[Aim10] Aimonetti, Matt: *Rails and the Enterprise*. http://weblog.rubyonrails.org/2010/3/24/rails-and-the-enterprise. Version: 2010

[Ber04] Berchtenbreiter, Ralph: *Grundlagen von Content-Management-Systemen und Ansätze ihrer Bedeutung für das CRM*. Galileo Computing, 2004

[BL06] Bernhard Lahres, Gregor R.: *Praxisbuch Objektorientierung – Professionelle Entwurfsverfahren*. Galileo Computing, 2006. – ISBN 978–3–89842–624–6

[Fow03] Fowler, Martin: *Patterns für Enterprise Application-Architekturen*. mitp, 2003. – ISBN 978–3–8266–1378–4

[Gü07] Günther, Tobias: *Theorie einer neuen Web-Architektur in Ruby On Rails: REST in Rails*. http://t3n.de/magazin/rest-rails-teil-1-theorie-neuen-web-architektur-ruby-219794/. Version: 2007

[HM10] Hussein Morsy, Tanja O.: *Ruby on Rails 2*. Galileo Computing, 2010. – ISBN 978–3–89842–779–1

[Neu07] Neukirchen, Christian: *Introducing Rack*. http://chneukirchen.org/talks/euruko-2007/neukirchen07introducingrack.pdf. Version: 2007

[Per10] Perrotta, Paolo: *Metaprogramming Ruby*. Pragmatic Programmers, 2010. – ISBN 1–934356–47–6

[Rai11a] Rails: *Rails 3.1*. 2011. – http://rubygems.org/gems/rails

[Rai11b] Rails: *Rails on Rack*. http://guides.rubyonrails.org/rails_on_rack.html. Version: 2011

[Rig09] Riggert, Wolfgang: *ECM - Enterprise Content Management*. Vieweg+Teubner, 2009. – ISBN 978–3–8348–0841–7

[Sch08] Schmidt, Maik: *Enterprise Recipes with Ruby and Rails*. Pragmatic Programmers, 2008. – ISBN 1–934356–23–9

Tabellenverzeichnis

Abbildungsverzeichnis

Quelltext

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter ausschließlicher Verwendung der angegebenen Literatur, Quellen und Hilfsmittel angefertigt habe.

Ort, Datum	Stephan Keller