Diplomarbeit

Analyse von Ruby on Rails 3 Web Content Management Systemen

Stephan Keller

19. September 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].

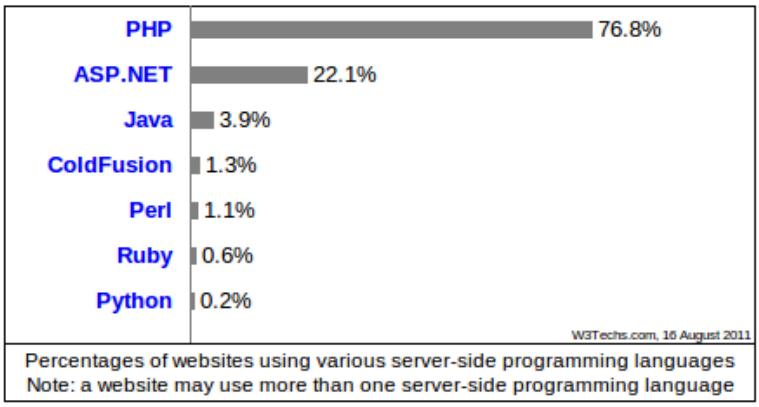

Abbildung 1.1: Nutzung verschiedener Programmiersprachen auf Servern

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 werden über ein Web Content Management System eines Drittanbieters realisiert. Die Rails-Anwendung fungiert als Zwischenstation und leitet bestimmte Anfragen an das externe WCMS weiter.

[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 Webframeworks und Technologien 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.

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 Auswertung kann abschließend eine Einsatzempfehlung für die gewählten Systeme erfolgen. 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 Entwicklungen.

2 Grundlagen

2.1 Content Management Systeme

2.1.1 Web Content Management Systeme

2.1.2 Content Management Life Cycle

2.1.3 Anforderungen an Content Management Systeme in der Zukunft

2.2 Entwicklung mit Ruby on Rails

2004 arbeitete der dänischen 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.

7 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.

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.

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

Für eine umfassende Einführung in Rails werden [Wi08] und [Rai11] empfohlen.

2.2.1 Dont Repeat yourself

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

2.2.2 Convention over Configuration

Viele Web Application Frameworks müssen vor ihrer Nutzung erst mit Hilfe zahlreicher Konfigurationsdateien und endloser Parametereinstellungen zu einem lauffähigen Gesamtsystem zusammengebaut werden. Das folgende Beispiel zeigt eine solche Konfigurationsdatei innerhalb des Java Application Frameworks Spring:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context
                                .xsd
                           http://www.springframework.org/schema/mvc
                           http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <mvc:annotation-driven />
    <mvc:view-controller path="/index.html" />
    <bean id="tilesConfigurer"
          class="org.springframework.web.servlet.view.tiles2.TilesConfigurer"
          p:definitions="/WEB-INF/tiles-defs/templates.xml" />
    <bean id="tilesViewResolver"
          class="org.springframework.web.servlet.view.UrlBasedViewResolver"
          p:viewClass="org.springbyexample.web.servlet.view.tiles2.DynamicTilesView"
          p:prefix="/WEB-INF/jsp/"
          p:suffix=".jsp" />
    <bean id="messageSource" class="org.springframework.context.support.
         ResourceBundleMessageSource"
```

```
24          p:basenames="messages" />
25    <!-- Declare the Interceptor -->
26    <mvc:interceptors>
27        <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"
28          p:paramName="locale" />
29    </mvc:interceptors>
30    <!-- Declare the Resolver -->
31    <bean id="localeResolver" class="org.springframework.web.servlet.i18n.
          SessionLocaleResolver" />
32 </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:

- 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

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

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

2.2.3 Model View Controller

Das Ruy on Rails Framework baut wie andere Frameworks auf einem Mehr-Schichten-Architektur-Modell auf. Zusätzlich kommt das bereits 1979 von dem Norweger Trygve Mikkjel Heyerdahl entwickelte Modell-View-Controller-Paradigma zum Einsatz.

2.2.4 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 unsichtbar.

REST, ein Akronym für Representational State Transfer, 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.

Rails unterstützt REST seit der Einführung der Version 1.2 und ermöglicht es so, an Hand einer URL und den verwendeten HTTP-Methoden die richtige Aktion auf dem Server auszuführen:

GET Abfrage einer Ressource unter der angegebenen URL mit anschließender Darstellung

POST Erstellung einer neuen Ressourcen an Hand der in der Anfrage übermittelten Daten, Realisierung durch Formulare auf Clientseite

PUT Überschreiben der angeforderten Ressource mit den in der Anfrage neu übermittelten Daten

DELETE Löschen der beschriebenen Ressource

Die URL einer Anfrage repräsentiert damit nicht wie z.B. in vielen PHP-Anwendungen eine bestimmte Aktion, die beim Aufruf auf dem Server ausgeführt werden soll, sondern eine Ressource, die eindeutig zugeordnet werden kann[2]. Da aktuelle Browser nur GET- und POST-Anfragen ermöglichen, müssen entsprechende Anfragen zum Löschen und Verändern einer Ressource mit Hilfe von zusätzlichen Attributen in der Anfrage simuliert werden:

Aktion	HTTP-Methode	Wirkung
index	GET	Anzeige einer Collection von Ressourcen
show	GET	Anzeie einer einzelnen Ressource
new	GET	Erzeugt ein Formular zum Anlegen einer neuen Ressource
create	POST	Erzeugt eine neue Ressource
edit	GET	Erzeugt ein Formular zum Bearbeiten einer Ressource
update	PUT Aktualisierung einer bestehenden Ressource	
destroy	DELETE Zerstörung einer Ressource	

Eine ausführliche Beschreibung von REST und dessen Realisierung liefert [Wir08].

2.2.5 Rack und Middleware

2.2.6 Datenbankgetriebene Entwicklung

2.2.7 Generatoren

[2]Die Eindeutigkeit der Ressource muss vom Entwickler sichergestellt werden.

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. 2.1) 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.

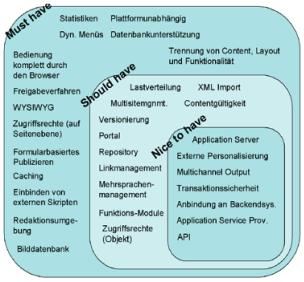

Abbildung 2.1: Feature-Matrix für Content Management Systeme

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. Diese orientieren sich dabei an den Prozessen des Content Managements und des Content Life Cycles [Vgl. Rit101].

Die von Ritter formulierten Kriterien werden in den Kapiteln 5.3.1 bis 5.3.5 nochmals aufgeführt und bilden die Grundlage für die Bewertung der Leistungsfähigkeit der ausgewählten Ruby on Rails Content Management Systeme.

2.3.1 Zielgruppe

Der externe Kriterienkatalog

2.3.2 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

2.3.3 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

2.3.4 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

2.3.5 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

2.3.6 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. 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 elegante 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 bestehende Projekte vernachlässigen[1].

[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, Refinery CMS, Browser CMS und Locomotive CMS repräsentieren daher die zum Zeitpunkt der Erstellung dieser Arbeit vielversprechendsten Rails-Implementierungen eines Web Content Management Systems. Trotz alldem ist vorweg anzumerken, dass sich diese Systeme, im Gegensatz zu anderen etablierten Open Source Content Management Systemen wie Typo3 und Drupal, noch am Anfang ihrer Entwicklungsgeschichte befinden.

Die hohe Anzahl der im Anhang der Arbeit aufgelisteten WCMS machte es erforderlich, Mindestanforderungen für die Aufnahme in die bevorstehende Analyse festzulegen. Diese sollen hier beschrieben werden:

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. Sie ist zusätzlich ein Beweis für das Engagement der beteiligten Programmierer. Eine in diesem Umfeld entstehende Software bietet daher entsprechend höheres Potenzial. Die hier ausgewählten Systeme erfüllen diese Forderung.

Unterscheidung in Front- und Backend

WCM-Systeme 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 die Inhalte der Seite zentral einpflegen und verwalten. Die ausgewählten 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:

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 RubyGems.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.

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, 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 Interkationsmöglichkeiten der Platform 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.

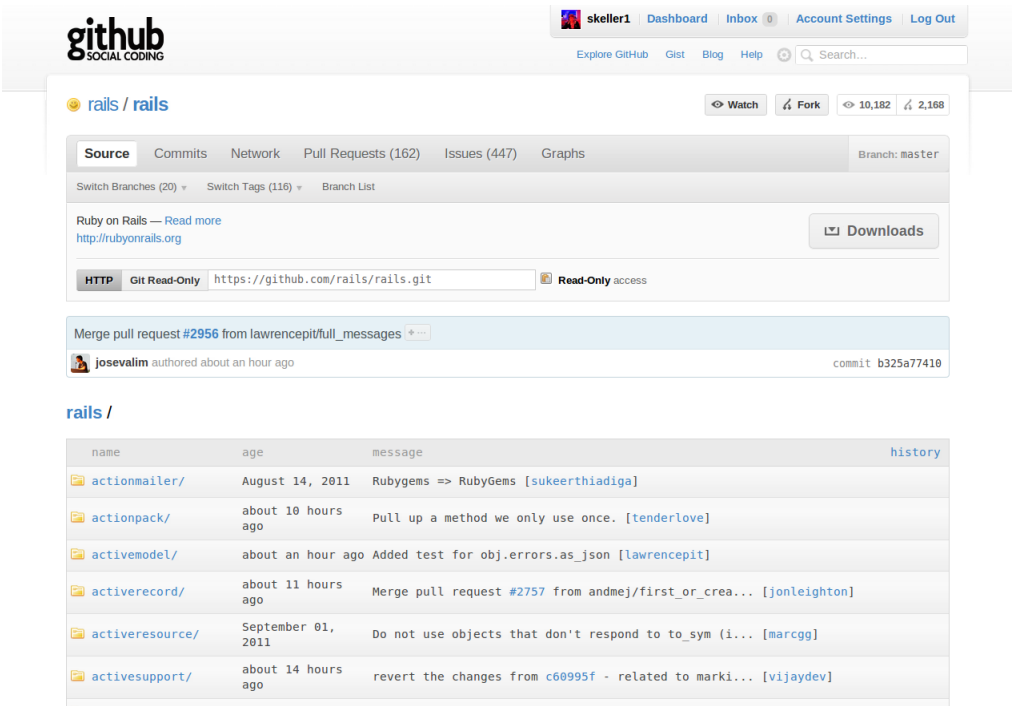

Abbildung 3.2: Beispiel für ein öffentliches Projekt auf Github.com: Das ausgewählte Projekt ist Rails 3

[2]Die Einrichtung eines kostnpflichtigen, privaten Github-Repositories ist ebenfalls möglich.

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** um die Entwickler Thomas von Deyen, Robin Böning und Carsten Fregin.
In der aktuellen Version 1.6.0 kann zwischen eine Rails 2 und 3 Umsetzung[3] ausgewählt werden.

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	

[3] Der Rails 3-Entwicklungszweig von Alchemy befindet sich noch im Beta-Stadium.

3.3.1 Funktionsprinzipien

Das Backend von Alchemy CMS verfügt u.a. über die Bereiche Seiten, Sprachen, Benutzer und Bibliothek:

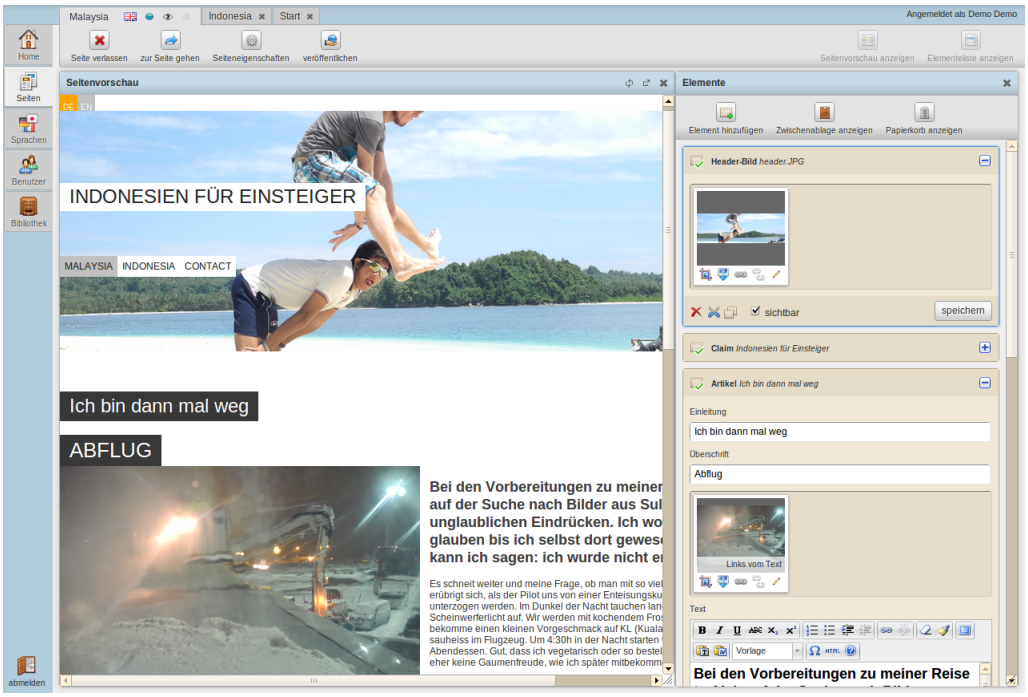

Abbildung 3.3: Backend-Ansicht von Alchemy CMS mit geöffnerter Seitenvorschau und Elementebearbeitung

Home

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

Seiten

Im Seiten-Modul des Backends findet die Verwaltung aller im System angelegter Seiten und deren Inhalte statt. Eine abgebildete Baumstruktur ermöglicht dabei einen Überblick über die verwalteten Seiten des Systems. Im Bearbeitungsmodus bietet Alchemy CMS darüber hinaus eine Aufteilung der 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 (Abb. 3.3).

Sprachen

Diese Modul ermöglicht die komfortable Verwaltung der in Alchemy CMS verfügbaren Frontend-Sprachen. Eine Alchemy Standard-Installation enthält bereits die Sprachen Deutsch und Englisch.

Benutzer

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

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 von Ressourcen 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.

Folgende offiziellen Erweiterungen sind ebenfalls verfügbar:

- alchemy-mailings[4]: Ermöglicht die Erstellung und Verwaltung von Newsletters im Backend des Systems.

[4]Komponenten-Download: https://github.com/magiclabs/alchemy-mailings

- alchemy-standard-set[5]: 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 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.

[5]Komponenten-Download: https://github.com/magiclabs/alchemy-standard-set

3.4 Vorstellung Browser CMS

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	
Philosophie	Redakteuren soll es ermöglicht werden, ohne HTML- und Rails-Kenntnisse eine Internetseite zu verwalten	

3.4.1 Funktionsprinzipien

3.4.2 Erweiterungen

3.4.3 Verwendete Technologien

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 Installation von Lokomotive CMS vornimmt und es ermöglicht, nach wenigen Minuten sofort mit der Erstellung der Seite im Browser zu beginnen[6].

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	

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

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. 3.4). 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.

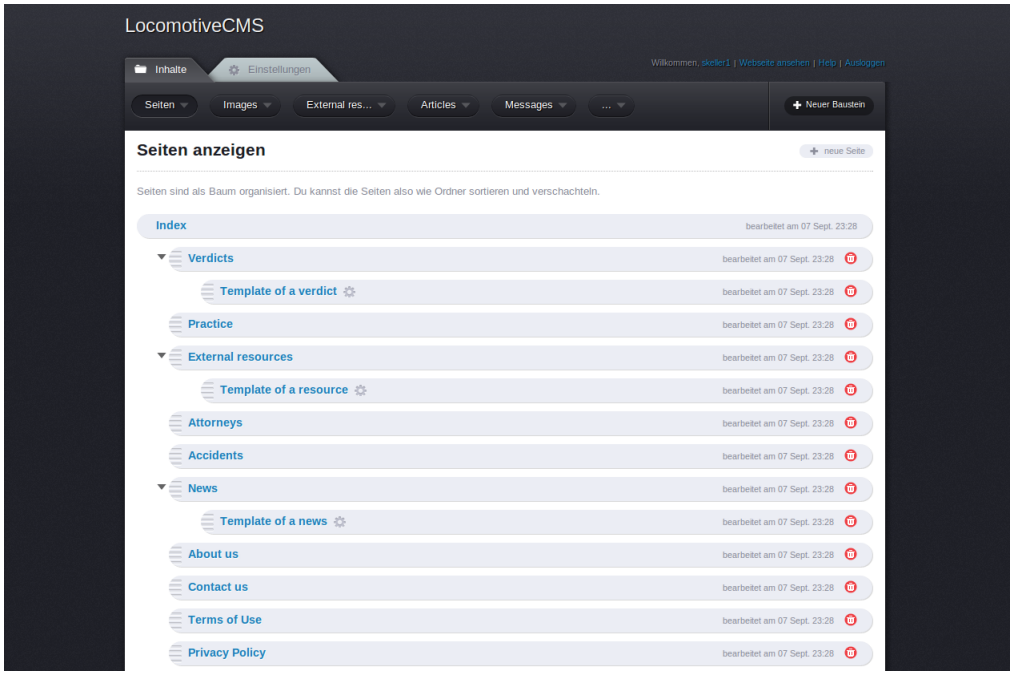

Abbildung 3.4: Backend von Locomotive CMS mit geöffnetem Seitenbaum.

Einstellungen

Im Einstellungs-Modul befinden sich die zentralen Funktionen zur Bearbeitung 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 der Seite als Gestaltungselemente dienen sollen[7].

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. 3.5). Im Gegensatz zu den hier bereits vorgestellten Systemen werden so keine Programmierkenntnisse der Anwender benötigt.

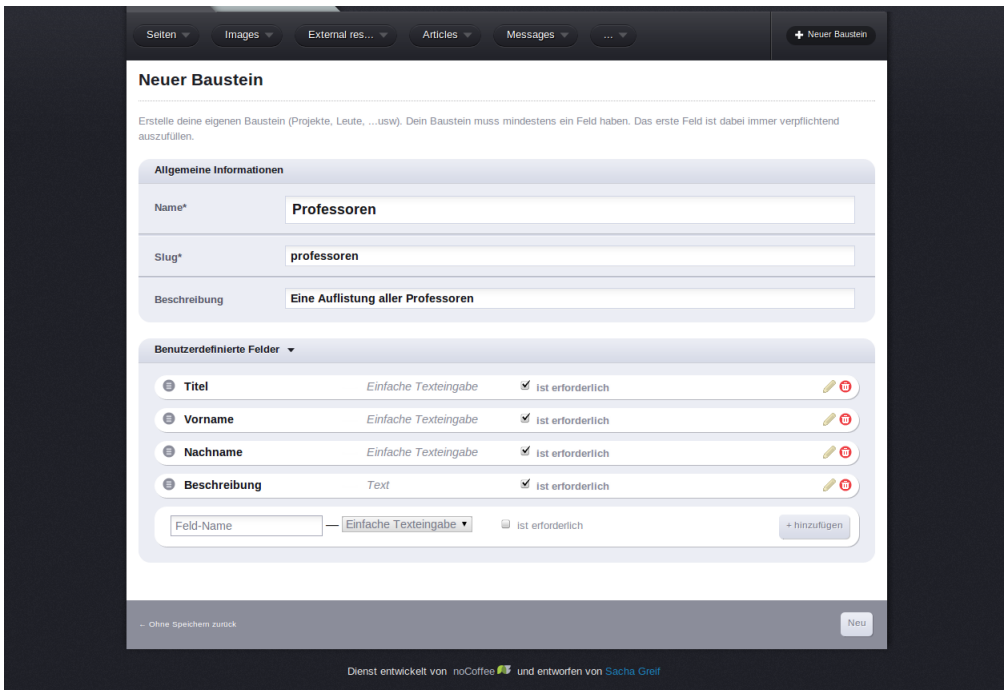

Abbildung 3.5: Ein im Backend von Locomotive CMS zusammengestelltes Inhaltselement

Die Inhaltselemente können aus folgenden grundlegenden Feldtypen zusammengebaut werden:

- Einfaches Textfeld

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

- 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[8] 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*[9] 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[10].

Im Frontend der Seite findet zusätzlich der WYSIWYG-JavaScript-HTML5-Editor Aloha[11] Verwendung. Er ermöglicht so eine Bearbeitung spezieller Inhaltselemente direkt im Frontend der Seite (Inline-Editing).

[8] Komponenten-Download: http://sizzlejs.com/
[9] Liquid ist das Ergebnis einer Extrahierung dieser Funktionalität aus Shopify
[10] 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.
[11] Projektseite: http://aloha-editor.org/

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	

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:

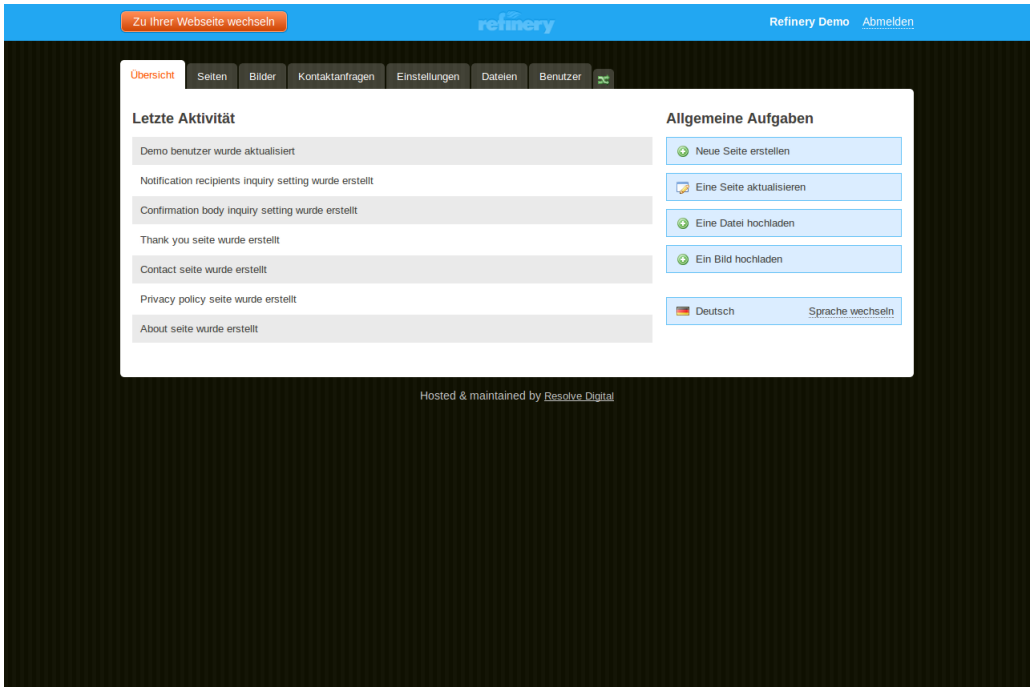

Abbildung 3.6: Backend-Ansicht von Refinery CMS

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, Metainformationen und Verhalten der Seiten einzeln bearbeitet werden. Die erstellten Seiten verfügen jeweils über einen

oder mehrere WYSIWYG-Editor-Felder[12], in die der Inhalt der Seite eingepflegt wird.

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 Anwender. 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[13] 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[14]: Darstellung von Kontaktanfragen auf der Internetseite mit zusätzlicher Verwaltungsfunktion der Anfragen im Backend von Refinery CMS
- refinerycms-news[15]: Verwaltung und Darstellung von Nachrichten im Front- und Backend-Bereich

[12] Die einzelnen Inhaltsblöcke werden als Content Sections bezeichnet. Die Anzahl der pro Seite zur Verfügung stehenden Inhaltsblöcke kann beliebig festgelegt werden. Die Positionierung der einzelnen Blöcke wird durch ein zuvor festgelegtes HTML-Template im Rails-Quellcode festgelegt.
[13] Die offizielle Bezeichnung der Parameter lautet *Refinery Settings*
[14] Komponenten-Download: https://github.com/resolve/refinerycms-inquiries
[15] Komponenten-Download: https://github.com/resolve/refinerycms-news

- refinerycms-blog[16]: 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 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 Analyse

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

[16]Komponenten-Download: https://github.com/resolve/refinerycms-blog

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.	

Inhalte (Texte, Bilder, Videos etc.) sollen zentral kategorisiert, erfasst und verwaltet werden können			
Alchemy 1.6.0	Erfüllt: 0%	Refinery CMS 1.0.8	Erfüllt: 0%
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.		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.	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
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.		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.	

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 kleiner Flaggensymbole 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 konte 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, existieren ebenfalls nicht.	

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 erlaubt das Hinzufügen von individuellen Erweiterungen.		Individuelle Inhaltselemente können durch die Verwendung der Refinery Engine Generatoren hinzugefügt werden. Für die weitere Entwicklung stehen die in Rails üblichen Entwicklungstechniken zur Verfügung.	
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.	

Integration von Inhalten anderer Webseiten, Multimedia, Applikationen, E-Commerce-Tools			
Alchemy 1.6.0	Erfüllt: 100%	Refinery CMS 1.0.8	Erfüllt: 100%
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*.		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.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
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.		Wie bei Alchemy und Refinery CMS können innerhalb des WYSIWYG-Editors HTML-Fragmente angegeben werden.	

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. Zusätzliche 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

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 unterstüzt		Wird nicht unterstüzt	
Browser CMS 3.3.1	Erfüllt: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht unterstüzt		Wird nicht unterstüzt	

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	

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: 0%	Locomotive CMS	Erfüllt: 0%
Wird nicht 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: 100%	Locomotive CMS	Erfüllt: 0%
Wird unterstützt		Wird nicht unterstützt	

3.8 Auswertung der Ergebnisse

Ergebniss der Untersuchung			
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	

4 Konzeptionelle Problemanalyse

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

1. Nutzeroberfläche
2. Datenmodellierung
3. Inhaltserstellung Die Inhaltserstellung von Refinery

4.1 Nutzeroberfläche

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

5 Lösungsvorschläge

In diesem Kapitel werden die

5.1 Implementierung eines Ruby on Rails Content Repository

Die hier betrachteten WCMS haben bei der Speicherung ihrer Daten teilweise sehr einfache Modelle inenrhalb einer Einen Möglichen Lösungsansatz bietet dabei die Umsetzung eines Content Repository. Im folgenden sollen die Konzepte dieser Lösung vorgestellt und die Vorteile für die bestehenden Ruby on Rails Content Management Systeme formuliert werden.

5.1.1 Idee und Konzept

Heutige Anwendungen (u.a. auch Web Content Management Systeme) benötigen neben der klassischen Speicherung von Daten zahlreiche zusätzliche Daten Management-Funktionalitäten. Ein Content Repository erweitert bestehende Datenbanken 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

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 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 Assocation mit ihrer Umsetzung innerhalb des neuen Web Content Management Systems Typo3 5.0.

5.1.2 Das Java Content Repository

Unter der Leitung von David Nuescheler der Firma Day Software wurde 2005 die erste Version einer Implementierung und einer API Spezifikation eines Java Content Repository veröffentlicht[3]. Die umgesetzte Referenzimplementierung wurde später unter der Leitung der Apache Foundation als Open Source Projekt Jack Rabbit[4] der Öffentlichkeit zur Verfügung gestellt.

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.
- Erstellung einer eigenständigen, komplett auf Ruby basierenden Referenzimplementierung und API nach dem Vorbild des Java Content Repository

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

5.1.4 Vorteile für die gewählten Ruby on Rails WCMS

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

- Vereinfachung des Zugriffs auf Inhalte innerhalb verschiedener Systeme
- Durch die im JCR Standard festgelegten Import- und Exportfunktionen kann ein Austausch der Inhalte zwischen einzelnen Web Content Management Systemen ermöglicht werden. Der momentane Mangel an standardisierten Exportfunktionen in den Systemen kann so ebenfalls beseitigt werden.
-

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[5] zurück. Es ermöglicht durch Kombinierung vorgefertigter Elemente eine schnelle Erstellung komplexer Oberflächen[6]. 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 herun-

[5] Informationen und Download: http://www.sencha.com/
[6] Beispielanwendungen: http://dev.sencha.com/deploy/ext-4.0.0/examples/

terladen und testen. Neben einem Download-Paket[7] auf der Projektseite von Typo3 wird zusätzlich eine aktuelle Version von Typo3 als Live-Demo[8] angeboten. Die Zugangsdaten zum Backend können im Frontend der Seite mit Hilfe eines Formulars erzeugt werden.

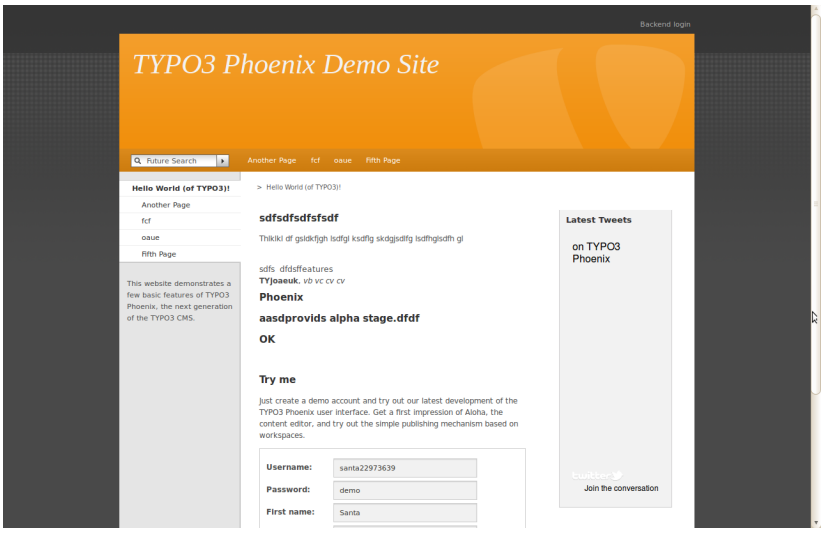

Abbildung 5.1: Frontend-Ansicht der Typo3 5.0 Sprint Release 6 Demoversion

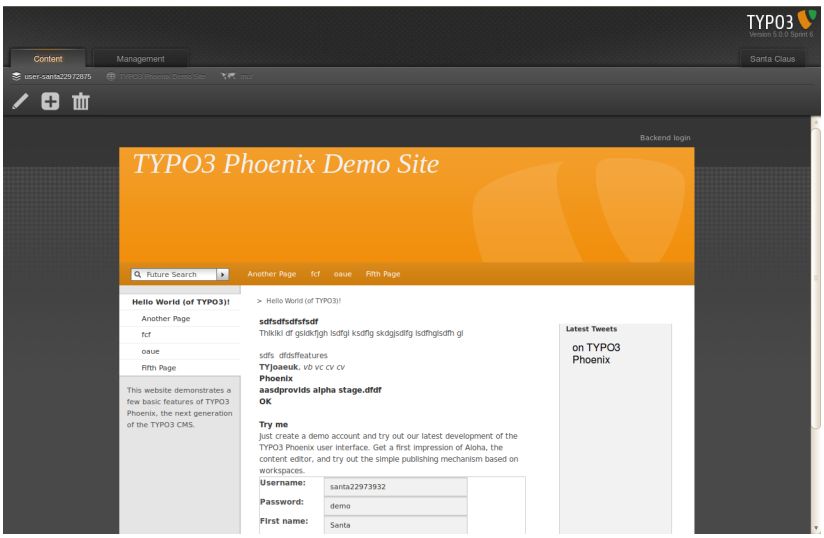

Abbildung 5.2: Backend-Ansicht der Typo3 5.0 Sprint Release 6 Demoversion

[7]Komponenten-Download: http://flow3.typo3.org/typo3-phoenix/
[8]Demoseite: http://phoenix.demo.typo3.org/

Das in der Demoversion umgesetzte Nutzer-Interface repräsentiert nur einen Teil der für Typo3 5.0 geplanten Oberfläche und Komponenten[9]. 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 mit Ext Direct

Mit Hilfe des Ext Js Frameworks können einzelne Komponenten zu einer großen Oberfläche zusammengebaut werden. Die Befüllung der Oberfläche mit Daten vom Server erfolgt dabei asynchron in Form sogenannter Ajax-Anfragen (Ajax Requests).

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

6 Zusammenfassung

6.1 Fazit

Die Analyse der ausgewählten Ruby on Rails Web Content Management Systeme hat gezeigt, dass die bestehenden Lösungen in ihrem Funktionsumfang noch sehr beschränkt sind. Ein Einsatz der Systeme kann daher nur empfohlen werden, wenn in einer Voranalyse die Anforderungen an die tatsächlich umzusetzende Internetseite definiert und abgeschätzt wurden. Ein nachträgliches Hinzufügen ist zwar möglich, erfordert jedoch einen erhöhten Mehraufwand im Vergleich zu verbreiteten Lösungen wie Typo3 oder Drupal. Der Einsatz für Privatseiten und Kleinstunternehmen ist jedoch bereits möglich, da die Systeme sehr verständlich und einfach aufgebaut sind.

6.2 Ausblick

Die entwicklelten Systeme werden ihre Defizite in den einzelnen Bereichen mit zunehmender Projektdauer minimieren und neue Funktionalitäten zur Verfügung stellen.

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
Webiva	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

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

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 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
});
```

7.3 Java Content Repository Spezifikation

Tabellenverzeichnis

Abbildungsverzeichnis