

Diplomarbeit

Analyse von Ruby on Rails 3 Web Content Management Systemen

Stephan Keller

17. 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	5
1.1	Ausgangslage	5
1.2	Motivation und Zielsetzung	6
1.3	Aufbau der Arbeit	7
2	Analyse der bestehenden Content Management Systeme	8
2.1	Vorbetrachtungen	8
2.2	Vorstellung Alchemy CMS	12
2.3	Vorstellung Browser CMS	14
2.3.1	Funktionsprinzipien	14
2.3.2	Erweiterungen	14
2.3.3	Verwendete Technologien	14
2.4	Vorstellung Locomotive CMS	15
2.4.1	Funktionsprinzipien	15
2.4.2	Erweiterungen	15
2.4.3	Verwendete Technologien	15
2.5	Vorstellung Refinery CMS	16
2.5.1	Funktionsprinzipien	16
2.5.2	Erweiterungen	16
2.5.3	Verwendete Technologien	16
2.6	Analyseverfahren	17
2.7	Durchführung der Analyse	17
3	Lösungsvorschläge	35
3.1	Übertragung der Konzepte des Domain Driven Design auf Ruby on Rails	35
3.2	Implementierung eines Ruby on Rails Java Content Repository	35
3.3	Übertragung des Typo3 5.0 Phoenix User-Interfaces in Rails 3.1	35

4	Anhang	36
4.1	Liste bestehender Rails 2 und 3 Web Content Management Systeme . . .	36
4.2	Ext-Direct Spezifikation für Ext Js 3.0	36

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 wurden¹.

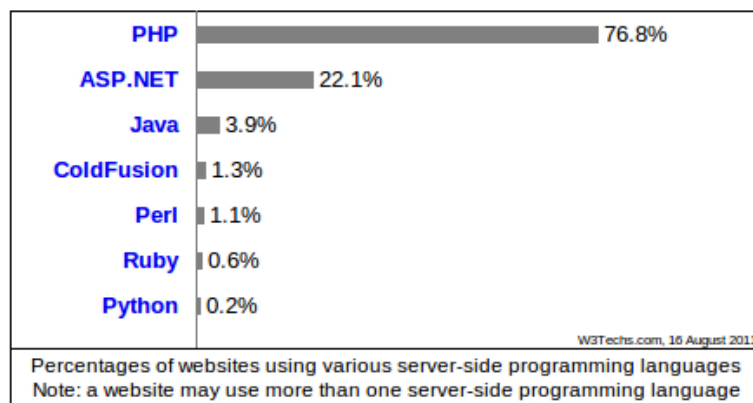


Abbildung 1.1: Nutzung verschiedener Programmiersprachen auf Servern

Auch im Bereich der Web Content Management Systeme² spiegelt sich diese Dominanz wider. Betrachtet man die Angaben des Content Management Portals cmsmatrix.org³, existieren neben den vor allem in Deutschland verwendeten Open Source-Lösungen Typo3, Drupal, Contao oder Joomla! über 500 weitere in PHP implementierte Web Content

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

²Im 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 Programmiersprachen.

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 37signals⁴, der Microblogging-Dienst Twitter⁵ und der webbasierte Hosting-Dienst Github⁶ für Software-Entwicklungsprojekte. Diese individuellen Lösungen werden dabei meist unter Zuhilfenahme eines Web Applications Frameworks realisiert, das den Entwicklungsprozess unterstützt und vereinfacht.

1.2 Motivation und Zielsetzung

Das Webframework Ruby on Rails⁷ hat sich seit der Veröffentlichung der Version 1.0 im Juli 2004 zu einem der bekanntesten Webframeworks der Ruby Fangemeinde entwickelt. Startups⁸ 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 Programmierkenntnisse⁹.
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.

⁴Projektseite von Basecamp: <http://basecamp.hq.com/>

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

⁶Github greift neben Ruby on Rails noch Webframeworks und Technologien zurück.

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

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

⁹Änderungen am Quellcode von Rails-Anwendungen erfordern immer einen Neustart der Anwendung.

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 rails-basierten Web Content Management Verwaltung zu untersuchen, um so den Einsatz eines externen WCMS überflüssig werden zu lassen.

1.3 Aufbau der Arbeit

Die vorliegende Diplomarbeit gliedert sich in sechs wesentliche Abschnitte.

Im ersten Abschnitt, in Kapitel 2, werden die für diese Arbeit notwendigen theoretische Grundlagen zu Web Content Management Systemen und dem Web Framework Ruby on Rails geschaffen. Es wird ebenfalls auf die neuen Funktionalitäten der Version 3.1 des Frameworks eingegangen. Zusätzlich werden die grundlegenden Charakteristika der Programmiersprache Ruby erläutert. Im zweiten Abschnitt, in Kapitel 3, folgt eine Analyse der bestehenden Ruby on Rails 3 Web Content Management Systeme Alchemy, RefineryCMS, BrowserCMS und Lokomotive CMS. Die ausgewählten Systeme werden dabei vorgestellt und anschließend mit Hilfe eines externen Kriterienkatalogs auf ihre Leistungsfähigkeit untersucht. Kapitel 4 überprüft die analysierten WCM-Systeme auf eventuell vorhandene konzeptionelle und programmiertechnische Schwachstellen. Darauf aufbauend werden in Kapitel 5 mögliche Lösungsansätze demonstriert und dafür notwendige theoretische Grundlagen herausgearbeitet. Kapitel 6 schließt die Arbeit mit einem Ausblick auf weitere Entwicklungen sowie einem Fazit ab.

2 Analyse der bestehenden Content Management Systeme

2.1 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 vernachlässigen¹.

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 Projektinitiatoren.

¹Im 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 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. Trotzdem 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 zu treffen. Es wurden dabei folgende Festlegungen getroffen:

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.

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 2.1: Ruby Gems mit aufgelisteten Informationen zum aktuellen Rails 3.1

2.2 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 Umbenennung 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-abs** 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 Umsetzung² ausgewählt werden.

²Der Rails 3-Entwicklungszweig von Alchemy befindet sich noch im Beta-Stadium.

Tabelle 2.1: 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	

Tabelle 2.2: 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.herokuapp.com/
	Backend	http://diplomabcms.herokuapp.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	

2.3 Vorstellung Browser CMS

2.3.1 Funktionsprinzipien

2.3.2 Erweiterungen

2.3.3 Verwendete Technologien

Tabelle 2.3: 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://diplom locomotive.herokuapp.com/
	Backend	http://diplom locomotive.herokuapp.com/admin
	Login	demo
	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	

2.4 Vorstellung Locomotive CMS

2.4.1 Funktionsprinzipien

2.4.2 Erweiterungen

2.4.3 Verwendete Technologien

Tabelle 2.4: 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	

2.5 Vorstellung Refinery CMS

2.5.1 Funktionsprinzipien

2.5.2 Erweiterungen

2.5.3 Verwendete Technologien

2.6 Analyseverfahren

2.7 Durchführung der Analyse

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	

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. Eine Angabe zusätzlicher Metadaten zu diesen Ressourcen ist nicht möglich.		Refinery CMS bietet eine einfache Medienverwaltung. Dabei wird zwischen Bildern und anderen Dateien unterschieden. Metadaten können nicht verwaltet werden.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Browser CMS verfügt über eine <i>Content Library</i> , die eine einfache Medienverwaltung von Bildern, Dateien und definierten Inhaltselementen ermöglicht. Eine Metadatenverwaltung ist nicht vorhanden.		Locomotive CMS bietet eine Asset-Verwaltung, in der selbst erstellte Inhaltselemente in Containern verwaltet werden können. Das Hinzufügen von beliebigen Metainformationen ist möglich.	

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. Er kann über installierbare Module beliebig ausgetauscht werden.		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 können so eine Vielzahl relationaler Datenbanken unterstützt werden. Zusätzlich existieren 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.	

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. Beim Zugriff zwei oder mehrerer Nutzer auf den gleichen Inhalt führt dazu, dass die Änderungen gültig werden, die zu letzt abgespeichert wurden. Optimistisches und pessimistisches Locking von Datensätzen und Inhalten wird nicht unterstützt.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
Nutzer und Administratoren können Inhalte gleichzeitig erfassen und verwalten. Beim Zugriff zwei oder mehrerer Nutzer auf den gleichen Inhalt führt dazu, dass die Änderungen gültig werden, die zu letzt abgespeichert wurden. Optimistisches und pessimistisches Locking von Datensätzen und Inhalten wird nicht unterstützt.		Nutzer und Administratoren können Inhalte gleichzeitig erfassen und verwalten. Beim Zugriff zwei oder mehrerer Nutzer auf den gleichen Inhalt führt dazu, dass die Änderungen gültig werden, die zu letzt abgespeichert wurden. Optimistisches und pessimistisches Locking von Datensätzen und Inhalten wird nicht unterstützt.	

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 Ressourcen. 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 Ressourcenverwaltung 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 <i>Content Library</i> 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 Administratoren 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 <i>browsercmsi</i> 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 Brower 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 <i>Zip</i> -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 <i>Zip</i> -Datei im Unterordner <i>data</i> abgelegt. Die Einträge liegen dabei im <i>YAML</i> -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%
<p>Der verwendete WYSIWYG-Editor <i>TinyMCE</i> 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 Ressourcen aus der Bibliothek von Alchemy Verwendung finden. Standardmäßig verfügt Alchemy bereits über die Inhaltselemente <i>Artikel</i>, <i>Text</i>, <i>Text mit Bild</i>, <i>Bilder</i>, <i>Bildergalerie</i>, <i>Überschrift</i> und <i>Intro</i>.</p>		<p>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.</p>	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 100%
<p>Wie seine Vorgänger auch können innerhalb des WYSIWYG-Editors IFrames oder anderer HTML-Code eingebettet werden. Vordefinierte Inhaltselementen, die vorhandene Ressourcen aus der <i>Content Library</i> einbinden können, müssen eigenhändig angelegt werden.</p>		<p>Wie bei Alchemy und Refinery CMS können innerhalb des WYSIWYG-Editors HTML-Fragmente angegeben werden.</p>	

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 unterstützt.		Versionierung und Wiederherstellung von Inhalten wird nicht unterstützt.	
Browser CMS 3.3.1	Erfüllt: 100%	Locomotive CMS	Erfüllt: 0%
Wird unterstützt		Versionierung und Wiederherstellung von Inhalten wird nicht unterstützt.	

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

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

Möglichkeit für <i>nicht technische</i> 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, dass 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 unterstützt	
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 unterstützt	

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

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

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 <i>erb</i> -Templates getrennt. Das Haupt-Template der Seite wird zu Beginn der Entwicklung von einem Designer festgelegt und anschließend in der Anwendung verankert (als fixe Resource im Rails Quellcode). So können Redakteure das Aussehen der Internetseite nicht beeinflussen.		Wie Alchemy verwendet Refinery CMS <i>erb</i> 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 so u.a. werden <i>erb</i> , <i>rjs</i> und <i>rxml</i> angeboten.		In Locomotive CMS kann für jede Seite ein Template angegeben werden. Die dabei verwendete Templatesprache ist <i>Liquid</i> .	

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: 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	

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	

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 Lösungsvorschläge

das ist ja geil

3.1 Übertragung der Konzepte des Domain Driven Design auf Ruby on Rails

das ist auch geil

3.2 Implementierung eines Ruby on Rails Java Content Repository

das ist auch cool

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

das auch am ende

4 Anhang

4.1 Liste bestehender Rails 2 und 3 Web Content Management Systeme

4.2 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>
</cfcomponent>
```

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 remotored 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  
});
```

Tabellenverzeichnis

2.1	Steckbrief Alchemy CMS	13
2.2	Steckbrief Browser CMS	14
2.3	Steckbrief Locomotive CMS	15
2.4	Steckbrief Refinery CMS	16

Abbildungsverzeichnis

1.1	Nutzung verschiedener Programmiersprachen auf Servern	5
2.1	Ruby Gems mit aufgelisteten Inforamtionen zum aktuellen Rails 3.1 . . .	11