FWP: AKTUELLE TECHNOLOGIEN ZUR ENTWICKLUNG VERTEILTER JAVA ANWENDUNGEN

SS 2016

Groovy/Grails

EIN ÜBERBLICK

JULIAN WILHELM

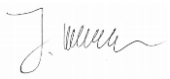
HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema

Groovy/Grails: Ein Überblick

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.



München, 26.06.2016

INHALTSVERZEICHNIS

Abbildungsverzeichnis

1 EINLEITUNG

Warum sollte man überhaupt Web-Applikationen entwickeln wollen? Dafür sprechen einige Gründe, welche je nach Art der Anwendung mehr oder weniger stark ins Gewicht fallen. Gegenüber einer klassischen Desktopanwendung hat die Web-Applikation die Vorteile plattformunabhängig zu sein und dem Nutzer Installations- und Update-Aufwand zu ersparen. Ebenso ist die Tatsache, dass Webanwendungen auch auf mobile Devices zur Verfügung stehen wichtig – immerhin haben diese seit 2014 den Desktop-PC in der Nutzer-Zahl überholt.[1]

Ist man erstmal an dem Punkt angelangt Web-Applikationen entwickeln zu wollen, so stellt sich bald die Frage nach dem besten Wege dies zu tun. Dieser sollte dabei einen möglichst standardisierten, effizienten Entwicklungsprozess darstellen mit dem selbst große, komplexe Vorhaben möglichst kostengünstig und fehlerfrei umgesetzt werden können. An diesem Punkt kommt die „Java Platform, Enterprise Edition" (Java EE) ins Spiel. Sie liefert die Spezifikation für eine Architektur für Java-basierte Webanwendungen, die dazu nötigen Webserver und einige APIs. Java EE hat damit einen Industrie-Standard gesetzt, an dem kaum ein Weg vorbeiführt – eine Alternative stellt z.B. die .NET-Plattform von Microsoft dar.

Allerdings wird die Entwicklung mit Java EE häufig als umständlich und kompliziert empfunden, weshalb Frameworks wie Spring entwickelt wurden. Ziel von Spring war es, die Entwicklung mit Java und Java EE zu vereinfachen und effizienter zu gestalten. Spring kann als Ergänzung oder auch Ersatz für Java EE zum Einsatz kommen. Auf Basis von Spring und anderen Frameworks (Hibernate, Sitemesh) wurde dann Grails als „next Generation" Java Webentwicklungs-Framework entwickelt.[2] Gründe für die Schaffung der nächsten Generation von Frameworks liegen in dem Wunsch begründet verschiedene etablierte Frameworks in einem Framework vereint vorzufinden, statt diese als Entwickler selbst verbinden zu müssen. Darüber hinaus fordern Frameworks der „alten Generation", vor allem in Kombination, oft einen hohen Aufwand, etwa in Form von Konfigurationen in XML-Dateien oder sich wiederholenden Boilerplate-Code. Inspiriert durch David Heinemeier Hansson' „Ruby on Rails" (daher auch der Name: Grails = ***G***roovy on **Rails**) wurde Grails in der Multiparadigma-Sprache Groovy (Objektorientiert und Skriptsprache), lauffähig auf der JVM, entwickelt um vollständige state-of-the-art Anwendungen in kurzer Zeit programmieren zu können.[3]

Groovy ist nach Java die zweite Standardsprache für die Java-Plattform (nach Java selbst). Das Ziel bei der Entwicklung von Groovy war es eine dynamische Sprache zu erschaffen, welche Konzepte von Sprachen wie Ruby, Perl und Python auf die Java Virtual Machine zu bringen und dabei mit Java kompatibel zu bleiben. Darüber hinaus versucht Groovy die Entwicklung von Java-Code durch syntaktischen Zucker entwicklerfreundlicher zu gestalten.[4]

Grails findet in der Industrie Anwendung etwa bei Netflix (erweitert zu „Netflix Asgard"), Sky oder dem Vodafone Music Store Anwendung.[5]

In der folgenden Arbeit soll das Framework Grails und ausgewählte Konzepte daraus im Vergleich zum Java EE Standard vorgestellt werden. Es richtet sich an Leser, die mit Java EE vertraut sind und

[1] „Mobile marketing statistics 2016".
[2] Smith und Ledbrook, *Grails in Action*, 3.
[3] Ebd., 5.
[4] König, *Groovy in Action*, 3ff.
[5] „Web Sites Using Grails".

erhebt keinen Anspruch darauf die aus Spring stammenden Grundlagen zu erläutern. Hierfür wird auf die Spring Dokumentation verwiesen.

2 GRUNDLAGEN

Im Folgenden wird die Programmiersprache Groovy vorgestellt und Grundlagen des Framework Grails dargelegt.

2.1 GROOVY

„Apache Groovy is a powerful, optionally typed and dynamic language, with static-typing and static compilation capabilities, for the Java platform aimed at improving developer productivity thanks to a concise, familiar and easy to learn syntax. It integrates smoothly with any Java program, and immediately delivers to your application powerful features, including scripting capabilities, Domain-Specific Language authoring, runtime and compile-time meta-programming and functional programming.“[6] – Der Groovy-Homepage entnommene Beschreibung

Groovy ist eine objektorientierte Programmiersprache und Skriptsprache, die dynamische und statischer Typisierung unterstützt. Sie wird auf der Java Virtual Machine ausgeführt und somit Plattformunabhängig.

Ziel der Groovy-Entwickler war es eine verbesserte – entwicklerfreundlichere – Fassung der Java-Syntax mit Konzepten aus Ruby und Python zu verbinden. Dabei zeichnet sich Groovy dadurch aus, dass es den Kompromiss zwischen der Anzahl zusätzlicher Features und der funktionierenden Integration in das Java-Umfeld sucht.

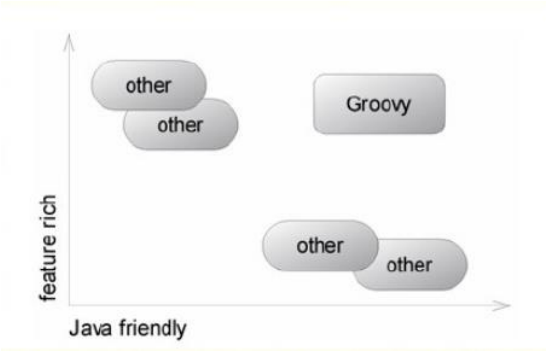


Abbildung 1: Groovys Kompromiss zwischen Feature Reichtum und Java-Integration[7]

2.1.1 Groovys Integration mit Java

Wie bereits beschrieben werden Groovy Programme auf der Java Virtual Machine ausgeführt. Dazu wird jede Groovy Source-Datei vor der Ausführung in normalen Java-Bytecode übersetzt. Zur Laufzeit ist Groovy normales Java mit einer zusätzlichen jar-Datei als Dependency. Das führt dazu, dass herkömmlicher Java-Code (mit sehr wenigen Ausnahmen) auch gültigen Groovy-Code darstellt. Somit sind alle Java-Bibliotheken auch für einen Groovy-Entwickler verfügbar und umgekehrt kann eine Groovy-Klasse in einem Java-Programm aufgerufen werden.[8] Groovy und Java sind also vollständig kompatibel zueinander.

[6] „The Groovy programming language“.
[7] König, *Groovy in Action*, 3.
[8] Ebd., 5.

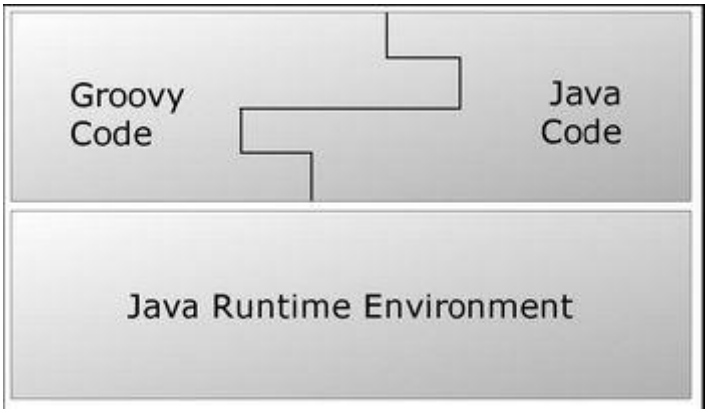


Abbildung 2: Kombinierbarkeit von Groovy und Java[9]

Groovy übernimmt viele Features aus Ruby und versucht das Schreiben von Code durch Syntaktischen Zucker und weitere aufwandssparende Maßnahmen effizienter und angenehmer zu gestalten. Dennoch sucht es dabei in Syntax-Fragen stets dem Java-Entwickler entgegen zu kommen. Deutlich werden diese Verhaltensweisen in diesem Beispiel:

```
import java.util.*;       // Java
Date today = new Date();  // Java

today = new Date()        // Groovy

require ‘date‘            # Ruby
today = Date.new          # Ruby
```

Man sieht, dass Groovy das Import-Statement überflüssig macht – Groovy importiert einige Standard-Bibliotheken automatisch. Außerdem braucht Groovy keine Semikolons, wenn es den Code auch ohne verstehen kann (etwa über Zeilenumbrüche). Gleiches gilt übrigens auch für geschweifte Klammern. Des Weiteren muss die Variable nicht als Date deklariert werden – dazu allerdings später. Darunter findet sich das Ruby-Äquivalent um zu zeigen, dass Groovy sich im Zweifel an der Java-Syntax orientiert und in diesem Fall beispielsweise das andere Package-Konzept, die Kommentar- und Objekterstellungs-Syntax vermeidet.[10]

2.1.2 Was Groovy über Java hinaus bietet

Mit der weniger aufwändigen und dennoch am herkömmlichen Java orientierten Syntax ist allerdings erst mal wenig gewonnen. Um Java-Entwickler von Groovys Nutzen zu überzeugen bedarf es mehr als das. Dabei kann man die Features die Groovy Java hinzufügt in drei Typen einteilen:

- Sprach-Features
- Spezifische Groovy Bibliotheken
- Erweiterungen von existierenden Java Standard Klassen (GDK)

Da es den Rahmen dieser Arbeit bei Weitem sprengen würde allen Groovy-Features die Beschreibung zukommen zu lassen die sie verdienen sollen an dieser Stelle stellvertretend ein paar der prominentesten Vertreter vorgestellt werden. Für weiterführende Erläuterungen sei an dieser Stelle auf die Groovy Dokumentation (groovy-lang.org/documentation) oder einschlägige Literatur wie Dierk Königs „Groovy in Action“ verwiesen.

[9] Ebd.
[10] Ebd., 5f.

2.1.2.1 Closures

Eines der unbestritten wichtigsten Features von Groovy sind die sogenannten Closures. Das Konzept der Closures stammt aus der funktionalen Programmierung und trat erstmals in LISP auf. Danach fand es Unterstützung in den meisten funktionalen Programmiersprachen (beispielsweise Haskell).[11]

Closures werden mit folgender Syntax definiert:

```
{ [closure parameters ->] closure body}
```

„`[closure parameters ->]`" repräsentiert dabei eine durch Kommas getrennte Liste von Argumenten – diese können, müssen aber nicht typisiert sein. Der Closure Body kann eine oder mehrere Anweisungen enthalten und so ganze Code-Fragmente abbilden. [12]

Closures verfügen über implizite Variablen. Wird bei der Definition eines Closure nur ein Parameter übergeben, so ist dieser immer als it erreichbar.

Beispiel:

```
def clos = {println "Hello ${it}"}
clos.call('world')
```

Weitere implizite Variablen sind this, owner und delegate. this bezieht sich auf die Instanz der Klasse in der ein Closure definiert wurde. owner stellt das umschließende Objekt des Closure dar und delegate ist per default dasselbe wie der owner, aber veränderlich – etwa in einem Builder.

Groovy ermöglicht mit seinen Closures auch das Konzept des Currying. Beim Currying wird eine Funktion mit mehreren Argumenten in eine Kette von Funktionen mit jeweils einem einzelnen Argument umgewandelt. In Groovy kann man mittels der Methode curry() „curried" Closures erschaffen.[13]

2.1.2.2 Elvis Operator

Der Elvis-Operator (?:) erweitert den ternären Auswahloperator für den Fall, dass zu prüfen ist ob eine Referenz existiert und einen Wert ungleich null besitzt. Ist dies der Fall wird der überprüfte Wert zurückgegeben, andernfalls der Alternativ-Wert hinter dem Operator.[14]

Beispiel:

```
a = b ?: c
```

2.1.2.3 Safe Navigation

Um beim Aufruf von Variablen, die möglicherweise valider Weise keinen Wert besitzen (und in diesem Fall ohne Auswirkungen bleiben sollen) NullPointerExceptions und das damit verbundene aufwändige Auffangen mit if/else-Klauseln oder mit try/catch-Blöcken[15] zu vermeiden führt Groovy das Konzept der Safe Navigation ein: `assert meinObjekt.?meinAttribut == null`

---

[11] „Closure (Funktion)".
[12] Vishal Layka, *Beginning Groovy, Grails and Griffon*, 41.
[13] Ebd., 49.
[14] König, *Groovy in Action*, 159.
[15] wider den eigentlichen Verwendungszweck

Dabei wird vor der Evaluierung des aktuellen Ausdrucks überprüft ob der Verweis vor dem Operator (.?) ein Null-Verweis ist und gibt in diesem Fall den Wert null zurück.[16]

2.1.2.4 Duck-Typing

Groovy kann statisch, aber auch dynamisch typisiert werden. Bei der dynamischen Typisierung wird auf das Konzept des Duck-Typing zurückgegriffen.

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck." - James Whitcomb Riley

Dabei wird zur Laufzeit ob ein Objekt entsprechende Merkmale unterstützt anstatt bereits beim Kompilieren auf festgeschriebene Typen zu beharren. Dies ermöglicht eine höhere Flexibilität, birgt jedoch die Gefahr von unnötigen Laufzeitfehlern. [17]

2.1.3 Java 8 als Antwort auf Groovy und Co.

Sprachen wie Groovy (Erscheinungsjahr 2003) und Scala (Erscheinungsjahr ebenfalls 2003) wurden unter anderem deshalb für die Java-Plattform entwickelt um Java um funktionale Konzepte (beispielsweise die oben behandelten Closures) zu erweitern. Dass diese Sprachen mittlerweile eine beachtliche Bekanntheit und Anhängerschaft errungen haben und in einigen kommerziellen Projekten bevorzugt eingesetzt werden zeugt davon, dass durchaus Bedarf für funktionale Programmierung auf der Java-Plattform zu bestehen scheint.

Mit dem Release von Java 8 am 18. März 2014 hat Oracle endlich auf diese Entwicklung reagiert und die unter dem Namen Projekt Lamda entwickelten Lamda-Ausdrücke in Java implementiert. Diese stellen auch die wichtigste neue Funktion in Java 8 dar und kommen mit zusätzlichen unterstützenden Funktionen auf die Java-Plattform.[18]

Zusätzlich kommt Java 8 mit einer neuen API für die Handhabung von Datums und Uhrzeit.[19] Das ist als Reaktion auf die Tatsache zu sehen, dass viele moderne Frameworks mit alternativen Bibliotheken wie etwa Joda-Time (joda.org/joda-time) arbeiten und Sprachen wie Groovy eigene Features über die alte Java time-Bibliothek gebaut haben.[20]

2.2 GRAILS

Grails ist ein Web Application Framework für die Programmiersprache Groovy.

Es bietet Konzepte wie Scaffolding, automatische Validatoren und Internationalisierung.

„Under the hood" stellt Grails eine Spezialisierung des Spring Frameworks dar und baut in vielen wichtigen Punkten auf dessen Konzepten auf.

Damit bedient es sich auch der MVC-Architektur.

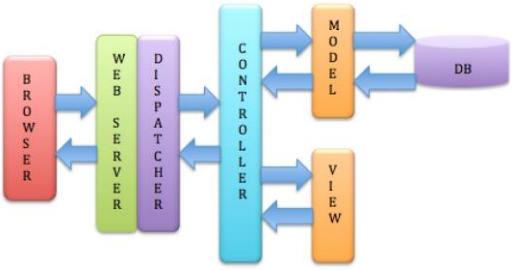


Abbildung 3: Grails MVC-Architektur

[16] König, *Groovy in Action*, 184.
[17] „Duck-Typing".
[18] „Informationen zu Java 8".
[19] Ebd.
[20] „GroovyCookbook: Dates and Times".

Außerdem spielen neben Spring verschiedene weitere etablierte Frameworks wie Hibernate und SiteMesh eine wichtige Rolle. Verbunden sind diese Frameworks und Bibliotheken mit der modernen Skriptsprache Groovy.

Nachfolgende Grafik zeigt Grails mitsamt seinem Fundament aus Java, JVM und Java SDK.

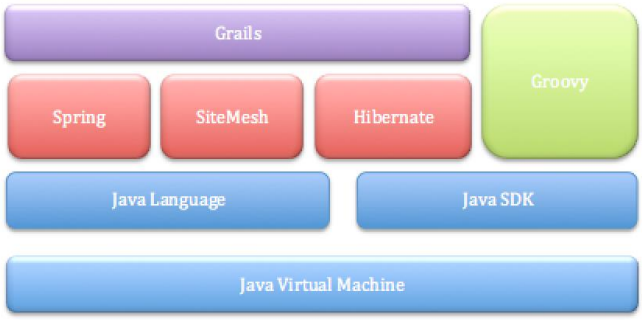


Abbildung 4: Grails und sein Fundament in der Java-Welt[21]

Wichtige Ideen neben der soliden Grundlage aus etablierter Technologien und hervorragender Java Integration sind eine agile Philosophie (unterstützt durch die agile Sprache Groovy), ein Fokus auf Produktivität und die Konzepte die im Abschnitt 3 weiter behandelt werden: Convention over Configuration, Scaffolding und Object Relational Mapping.[22]

Die treibende Kraft hinter Grails ist neben einer lebendigen Open-Source Community das Unternehmen SpringSource.

3 Ausgewählte Konzepte

Im Folgenden soll auf verschiedene wichtige Konzepte des Grails Framework eingegangen werden. Die Behandlung lehnt sich in einigen Fällen an die Spezifikation in Java EE an. Dieses Kapitel erhebt dabei keinen Anspruch auf Vollständigkeit.

3.1 Deployment

Grails kann für Entwicklungszwecke mittels des Konsolenbefehls „grails run-app“ im Entwicklungsmodus mit zusätzlichem Overhead deployed werden. Für einen produktiven Rollout sollte mittels „grails war“ ein WAR-File erstellt werden und dann händisch oder mit Hilfe von Tools wie z.B. Jenkins in Kombination mit dem in Groovy geschriebenen Gradle (vergleichbar mit Apache Ant oder Apache Maven). Das WAR-File kann dann auf jedem Java Servlet Container nach Java EE Spezifikation installiert werden. Für Grails Versionen ab Version 3.0 ist ein Apache Tomcat Webcontainer empfohlen.[23]

Mithilfe von Gant (Groovy Ant)[24] Skripten ist auch ein Deployment als EAR möglich.[25]

Grails bietet im Punkt Deployment also die gleichen Möglichgkeiten wie eine Standard JavaEE Anwendung.

[21] „Introduction to Grails | Noetic Ideas“.
[22] Smith und Ledbrook, *Grails in Action*, 6.
[23] „Wiki - Deployment“.
[24] „Gant“. https://gant.github.io/
[25] Rocher, „Gant/Grails/EAR“.

3.2 SERVLET API

Grails Controller bauen auf der Standard Java Servlet API auf, wie man sie aus anderen Frameworks, aber auch Java EE kennt. Allerdings bietet Grails in manchen Punkten „semantic sugar“ um das Arbeiten mit der Servlet API zu erleichtern.[26] Es folgt dazu beispielhaft eine kleine Übersicht:

Java Servlet	Grails Controller
request.getAttribute(“myAttr”);	request.myAttr
request.setAttribute(“myAttr”, “myValue”);	request.myAttr = “myValue”
session.getAttribute(“mAttr”);	session.myAttr
session.setAttribute(“myAttr”, “myValue””);	session.myAttr = “myValue”
servletContext.getAttribute(“mAttr”);	servletContext.myAttr
servletContext.setAttribute(“myAttr”, “myValue””);	servletContext.myAttr = “myValue”

Abbildung 5:Unterschiede zwischen Request Attributen in Java Servlets und Grails Controllern[27]

Im Wesentlichen unterscheidet sich das Arbeiten mit Grails Controllern nicht von der aus Java EE bekannten Java Servlet API.

3.3 INVERSION OF CONTROL (IOC)

„One important characteristic of a framework is that the methods defined by the user to tailor the framework will often be called from within the framework itself, rather than from the user's application code. The framework often plays the role of the main program in coordinating and sequencing application activity. This inversion of control gives frameworks the power to serve as extensible skeletons. The methods supplied by the user tailor the generic algorithms defined in the framework for a particular application.“[28] - Ralph Johnson und Brian Foote

Inversion of Control, auch das Hollywood-Prinzip genannt („Don't call us, we'll call you"), ist eine wichtige Eigenschaft von Frameworks. Eine Funktion eines Anwendungsprogramms wird bei einer Standardbibliothek registriert und von dieser zu einem späteren Zeitpunkt aufgerufen. Dadurch übernimmt das Framework die Steuerung des Programmablaufs und kann so als Skelett dienen, in das der Anwendungsentwickler seinen applikationsspezifischen Code einfügt. In diesem Punkt unterscheidet sich ein Framework auch elementar von einer einfachen Code-Bibliothek.

[26] Rocher und Brown, *The Definitive Guide to Grails 2*, 66f.
[27] Ebd., 67.
[28] Ralph E. Johnson und Brian Foote, „Designing Reusable Classes“.

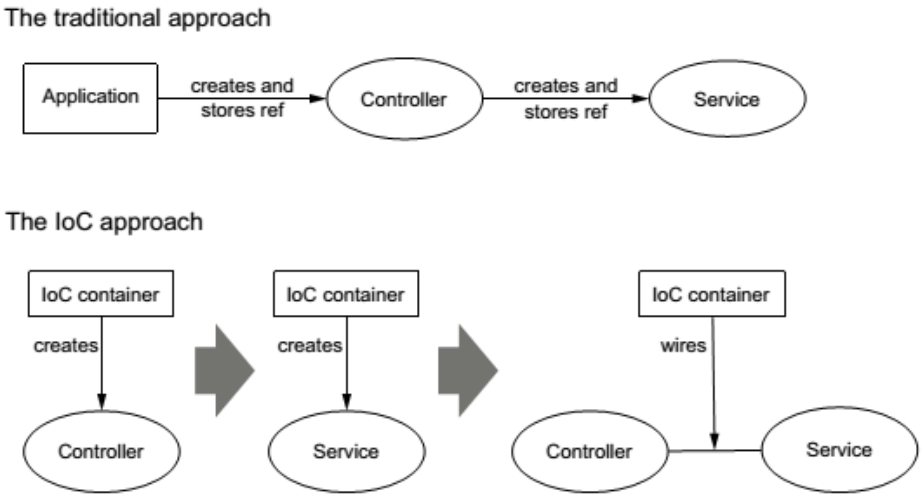


Abbildung 6: Inversion of Control im Vergleich mit traditionellem Dependency Management[29]

Java EE verfügt über verschiedene Konzepte, die auf dem IOC-Paradigma beruhen. Zwei der prominentesten davon sind die sogenannten Enterprise Java Beans (EJB) und die Contexts and Dependency Injection (CDI). Auf deren Äquivalent im Grails Framework soll im Folgenden kurz eingegangen werden.

Grails basiert sehr stark auf dem Spring MVC Framework und setzt deshalb auf Spring Beans und Spring Dependency Injection.[30]

Spring Dependency Injection verwaltet den „Lifecycle" der Objekte einer Applikation. Das bedeutet es erstellt, initialisiert und zerstört sie. Diese von Spring bzw. Grails verwalteten Objekte heißen Spring Beans und stellen tatsächlich ganz normale Java-Objekte (POJOs - *Plain Old Java Objects) dar.* Der Unterschied zwischen Beans und Java-Objekten ergibt sich erst durch die Verwaltung durch Spring. Doch damit Spring weiß, welche POJOs es verwalten soll müssen diese eine zugehörige Bean Beschreibung („bean descriptor") mit einer Bean Definition enthalten.[31] An dieser Stelle kommt Groovy/Grails ins Spiel:

Bei der Definition von Spring Beans kann auf die herkömmliche XML-Notation zurückgegriffen werden, welche in der Spring Dokumentation ausgeführt ist. Darüber hinaus besteht mittlerweile die Möglichkeit Spring Beans mittels Java Annotationen zu definieren. Neu im Vergleich zu Spring ist die sogenannte Grails' Spring Bean DSL, die mit Spring 4.0 auch den Weg in das Spring Framework gefunden hat.[32]

[29] Smith und Ledbrook, *Grails in Action*, 396.
[30] Da das Spring Framework in einer eigenen Arbeit behandelt wird soll an dieser Stelle nicht darauf vorgegriffen werden. Es wird auf die folgende Arbeit oder die Spring Dokumentation verwiesen.
[31] Smith und Ledbrook, *Grails in Action*, 396f.
[32] „Groovy Bean Configuration in Spring Framework 4".

```
beans = {
    reportGenerator(org.example.XmlReportGenerator) { bean ->
        bean.autowire = "byName"
        prettyFormat = true
    }
}
```

Abbildung 7: Bean Definition mit Grails DSL[33]

Die Definition mittels Grails DSL ist einerseits leichter auf den Augen und knapper gehalten. Andererseits – und darin liegt der eigentliche Vorteil - handelt sich dabei um Groovy Code. Das bedeutet:

- Man kann normalen Groovy Code einfügen, also Kontrollstrukturen wie Schleifen oder Bedingungen
- Man kann echte Typen für die Attributwerte verwenden[34]

Im Folgenden ein Beispiel für Groovy-Kontrollstrukturen in einer Bean-Definition:

```
import grails.util.Environment

beans = {
    if (Environment.current == Environment.PRODUCTION) {
        // Use the real web service for production
        securityService(org.example.WsClientSecurityService) {
            endpoint = "http://..."
        }
    }
    else {
        // Use a dummy service for development and testing
        securityService(org.example.DummySecurityService)  {
            userRoles = [ peter: [ "admin", "user"], tom: [ "user" ] ]
        }
    }
}
```

Abbildung 8: Groovy Kontrollstrukturen in Grails DSL[35]

Es ist auch relativ einfach möglich EJB Applikationen mit Grails zu verbinden.[36]

3.4 Convention over Configuration (COC)

Convention over configuration oder coding by convention ist ein Softwaredesign-Paradigma, welches die Komplexität von Konfigurationen zu verringern sucht, indem der Entwickler nur dann Konfigurationen selbst vornimmt, wenn er vom Standard (der Konfiguration) abweichen möchte. In traditionellen Frameworks wird eine Reihe von Konfigurationsdateien benötigt um projektspezifische Einstellungen vorzunehmen. Oft sind auch Routinetätigkeiten wie etwa die Zuordnung von Klassen zu

[33] „Spring: the foundation for Grails“.
[34] Ebd.
[35] „Spring: the foundation for Grails“.
[36] „Kickin’ down the cobblestones: Integrating Grails with EJB 2 applications“.

Datenbanktabellen oder Boiler Plate Code darunter. Die Größe und Komplexität der Konfigurationsdateien steigt dabei proportional mit dem Wachstum der entsprechenden Applikation. Dies erhöht den Entwicklungsaufwand einer Applikation und verschlechtert die Wartbarkeit. Die Verwendung von Annotationen statt Konfigurationsdateien löst das Problem nicht, sondern verschiebt es nur. Wenn sich ein Entwickler in einem Projekt mit dem Convention over Configuration Paradigma an die zugrundeliegenden Konventionen hält reduziert sich so der Wartungs- und Entwicklungsaufwand und spart so Ressourcen.[37]

In Grails sieht man die Auswirkungen der konsequenten Ausrichtung am CoC-Paradigma zuerst an der geringen Anzahl von Konfigurations-Dateien – Grails basiert die meisten Entscheidungen auf dem Source Code entnommene Defaults.

Einige Beispiele:

- Wird eine Controller-Klasse *Shop* mit einer Action *order* erstellt, so ist sie automatisch unter der URL /*meineapplikation*/*shop*/order erreichbar
- Wenn View-Dateien in ein Verzeichnis namens /views/shop/order platziert sind verlinkt Grails alles ohne eine einzige Zeile Konfiguration
- Wird in Grails eine neue Domain Klasse namens *Customer* erzeugt, so erstellt Grails automatisch die zugehörige Tabelle in der Datenbank
- Fügt man nun Felder zu *Customer* hinzu, so wird Grails automatisch on the fly die benötigten Felder in der zugehörigen Datenbanktabelle erstellen (inklusive korrekte Datentypen und Constraints wie sie in der Domain definiert sind)

Trotz allem ersetzt Grails nicht jede Form von Konfiguration - wenn man Defaults anpassen möchte ist das möglich. Grails ermöglicht das Überschreiben von Defaults ohne XML, erlaubt aber auch die klassische Hibernate-Konfiguration mittels XML-Dateien.[38]

3.5 Object Relational Mapping (GORM)

Für sein Objekt Relational Mapping (ORM) nutzt Grails GORM. Under the hood handelt es sich dabei um Hibernate3. Am dieser Stelle soll Hibernate nicht weiter behandelt werden. Es wird auf die Hibernate-Dokumentation[39] verwiesen.

Stattdessen soll kurz auf ein Konzept eingegangen werden, dass Grails aus Ruby on Rails übernommen hat: Dynamic Finders.

Dynamic Finders sind eine Möglichkeit in GORM Queries durchzuführen (neben unter anderem der Hibernate Query Language HQL). Sie sehen aus wie statische Methodenaufrufe, allerdings existieren die Methoden in keiner Weise im Code. Sie werden zur Laufzeit durch Code-Synthese, basierend auf den Attributen der entsprechenden Klasse, generiert. Sie setzen sich aus den beiden Methoden „findBy“ und „findAllBy“, einem Attribut der entsprechenden Domain-Klasse, einem von 13 Komparatoren (z.B. InList, Like, LessThan, Between, NotEqual) und einem anschließend als Parameter übergebenen Wert zusammen. Dabei können natürlich mehrere Vergleiche durch Boolesche Operatoren (And/OR) verknüpft werden. Außerdem können durch weitere Parameter eine Unterteilung in Seiten und eine Sortierung durchgeführt werden.[40]

[37] „Konvention vor Konfiguration“.
[38] Smith und Ledbrook, *Grails in Action*, 5.
[39] hibernate.org/orm/documentation/5.0/
[40] „7 Object Relational Mapping (GORM) 3.1.6“.

Ein Beispiel[41]:

```
def books = Book.findAllByTitleLikeAndReleaseDateGreaterThan(
                      "%Java%", new Date() - 30)
```

Mit Seitenunterteilung und Sortierung[42]:

```
def books = Book.findAllByTitleLike("Harry Pot%",
               [max: 3, offset: 2, sort: "title", order: "desc"])
```

GORMs Dynamic Finders sind ein Feature, welches zunächst ungewohnt und im Vergleich mit klassischen SQL-Queries wenig mächtig wirkt. Allerdings fügen sie sich gut in den Code ein, verhindert ein Auseinanderziehen von Logik, passen hervorragende in die dynamische Natur von Groovy und Grails und ersparen zumindest bei alltäglichen Lesezugriffen einiges an Arbeit.

3.6 GRAILS WEB LAYER

Grails' Präsentationsschicht besteht aus Controllern und Views. Controller – wie der Name bereits verrät – steuern eine Applikation. Sie nehmen User-Input entgegen, interagieren mit Geschäftslogik und Datenmodel und leiten den Nutzer zur richtigen Seite weiter. Ohne Controller wäre eine Web-Applikation nicht mehr als eine statische Seite.

In Grails kann ein Model – eine Map mit Daten – an die View übergeben werden. Die Schlüssel dieser Map sind die Variablennamen unter denen die Werte in der View zugänglich sind. Beispiel eines Models in einer Action:

```
def show() {
    [book: Book.get(params.id)]
}
```

Per Default wird nach dem CoC-Paradigma die View namens „*show*" gerendert.
Alternativ kann die gewünschte View auch manuell bestimmt werden:

```
def show() {
 def map = [book: Book.get(params.id)]
 render(view: "display", model: map)
}
```

Render kann auch genutzt werden um JSON oder XML auszugeben.

```
def list() {

    def results = Book.list()
    render(contentType: "application/json") {  // "text/xml" für XML
        books = array {
            for (b in results) {
                book title: b.title
            }
        }
    }
}
```

Grails unterstützt dabei auch automatisches Marshalling zu JSON oder XML:

[41] Ebd.
[42] Ebd.

```
render Book.list() as JSON  // render Book.list() as XML
```

Grails bietet natürlich auch die Möglichkeit mit *redirect* auf eine andere Seite weiterzuleiten:

```
// Call the login action within the same class
redirect(action: login)
// Redirect to a URL
 redirect(url: "http://grails.org")
```

Redirect nutzt die *HttpServletResponse* Methode *sendRedirect*.[43]

Grails Views werden mittels Groovy Server Pages (GSP) umgesetzt. Sie wurden nach der Vorlage der Java Server Pages (JSP) entwickelt und ähneln diesen. Allerdings haben sie das Ziel flexibler und intuitiver zu sein. Tatsächlich wird die Ähnlichkeit in der Dokumentation als „less attractive heritage" bezeichnet. [44]

Ein kurzer Syntax-Überblick um die Ähnlichkeit zu JSP darzustellen:

- <% %>: eingebetteter Groovy Code (Bad practice!)
- <%= %>: Werteausgabe
- <%-- --%>: serverseitige Kommentare

Alternativ zur Werteausgabe nach obigem Schema können Werte auch ähnlich der JSO Expression Language ausgegeben werden:

```
<html>
  <body>
    Hello ${params.name}
  </body>
</html>
```

Im Gegensatz zu JSP EL kann im ${..}-Block jeder Groovy-Ausdruck stehen.

GSPs verfügt über eine Reihe eingebauter GSP-Tags (Präfix: g:). Im Gegensatz zu JSP ist es allerdings nicht nötig Tag Bibliotheken zu importieren – ein Tag das mit g: beginnt wird automatisch als GSP-Tag erkannt. Natürlich können auch Custom Tag Bibliotheken erstellt werden.

GSP-Tags können einen Body und Attribute enthalten:

```
<g:example attr="${new Date()}">
   Hello world
</g:example>
```

Es folgt eine Aufstellung einiger ausgewählter GSP-Tags. Diese Aufstellung soll nur beispielhaft einige interessante Tags aufführen.

Es existieren Tags für Logik und Iteration:

- If/else:

```
<g:if test="${session.role == 'admin'}">
   <%-- show administrative functions --%>
</g:if>
```

[43] „8 The Web Layer 3.1.6".
[44] Ebd. 8.2.2 GSP Tags

```
<g:else>
   <%-- show basic functions --%>
</g:else>
```

- Each-Schleife:

```
<g:each in="${[1,2,3]}" var="num">
   <p>Number ${num}</p>
</g:each>
```

- While-Schleife:

```
<g:set var="num" value="${1}" />
<g:while test="${num < 5 }">
    <p>Number ${num++}</p>
</g:while>
```

Darüber hinaus bietet GSP Tags für Suche und Filtern von Collections:

- findAll:

```
Stephen King's Books:
<g:findAll in="${books}" expr="it.author == 'Stephen King'">
     <p>Title: ${it.title}</p>
</g:findAll>
```

- grep:

```
<g:grep in="${books}" filter="NonFictionBooks.class">
     <p>Title: ${it.title}</p>
</g:grep>
```

Zum Abschluss ein Tag, das den ärgerlichen Umgang mit mehreren Submit-Buttons erleichtert:

- actionSubmit:

```
<g:actionSubmit value="Some update label" action="update" />
```

ActionSubmit verhält sich wie ein regulärer Submit, mit dem Unterschied, dass man eine alternative Action für den Submit angeben kann.

Letztlich handelt es sich bei den Groovy Server Pages um eine Weiterentwicklung der JSP.

Nach dem kurzen Überblick über GSP und GSP-Tags folgt ein Hinweis auf die Templating-Engine SiteMesh. Sie ist standardmäßig in Grails integriert und soll an dieser Stelle nur kurz beschrieben werden. Eine detailliertere Auseinandersetzung mit der Thematik würde den Rahmen dieser Arbeit sprengen.

SiteMesh ist ein leichtgewichtiges und flexibles Framework, welches sich auf das Decorator Pattern der sogenannten „Gang of Four“ (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides) stützt um eine saubere Trennung von Inhalt und Präsentation zu ermöglichen. Dabei wird die Präsentation des Inhalts erst kurz vor der Auslieferung an einen PC oder Mobilgerät durchgeführt. [45]

[45] „Home - SiteMesh 2 - Confluence“.

3.7 GRAILS SCAFFOLDING

Scaffolding ist ein Konzept, welches es ermöglicht eine gesamte Anwendung für eine bestimmte Domain-Klasse automatisch generieren zu lassen. Dies umfasst Views und Controller Actions für CRUD-Operationen. Es kann entweder statisch oder dynamisch sein – beide Arten generieren den gleichen Code. Der Unterschied liegt darin, dass der generierte Code beim statischen Scaffolding bereits vor dem Zeitpunkt des Kompilierens für den Nutzer verfügbar und dadurch leicht anzupassen ist. Beim dynamischen Scaffolding wird der Code zur Laufzeit im Speicher generiert und ist für den Entwickler nicht einsehbar.[46]

Um Scaffolding zu nutzen muss zunächst eine Domain-Klasse angelegt sein. Diese kann wie im folgenden Beispiel Constraints enthalten. Diese stellen Regeln für die zulässigen Werte für die einzelnen Attribute der Domain-Klasse auf. Anhand dieser Constraints wird die Reihenfolge der Formular Felder in den generierten Edit- und Create-Views festgelegt und der User-Input validiert. [47]

```
static constraints = {
  name(blank:false)
  createdDate()
  priority()
  status()
  note(maxSize:1000, nullable:true)
  completedDate(nullable:true)
  dueDate(nullable:true)
}
```

Abbildung 9: Constraints einer Domain-Klasse[48]

Um vom dynamischen Scaffolding profitieren zu können benötigt man zusätzlich zu der Domain-Klasse einen zugehörigen Controller (die Zuordnung erfolgt über die bereits beschriebenen Namenskonventionen). In diesem ist die index-Methode mit dem Attribut „static scaffold = *MyDomain*“ zu ersetzen, wobei *MyDomain* die zugehörige Domainklasse darstellt. Diese kleine Anpassung sorgt dafür das zur Laufzeit List, Create, Edit und Show Views mit voll funktionsfähigen CRUD-Operationen (einschließlich Delete).[49]

Es folgen Screenshots einiger durch Scaffolding automatisch generierter Views.

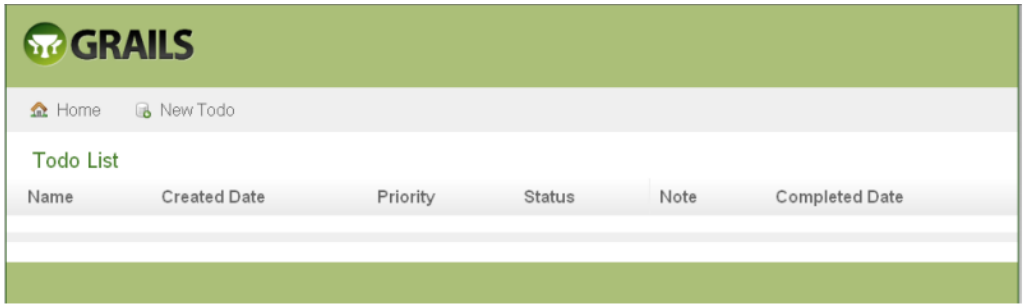


Abbildung 10: Scaffolding - List View[50]

[46] Vishal Layka, *Beginning Groovy, Grails and Griffon*, 107.
[47] Ebd., 109.
[48] Ebd.
[49] Ebd.
[50] Ebd., 110.

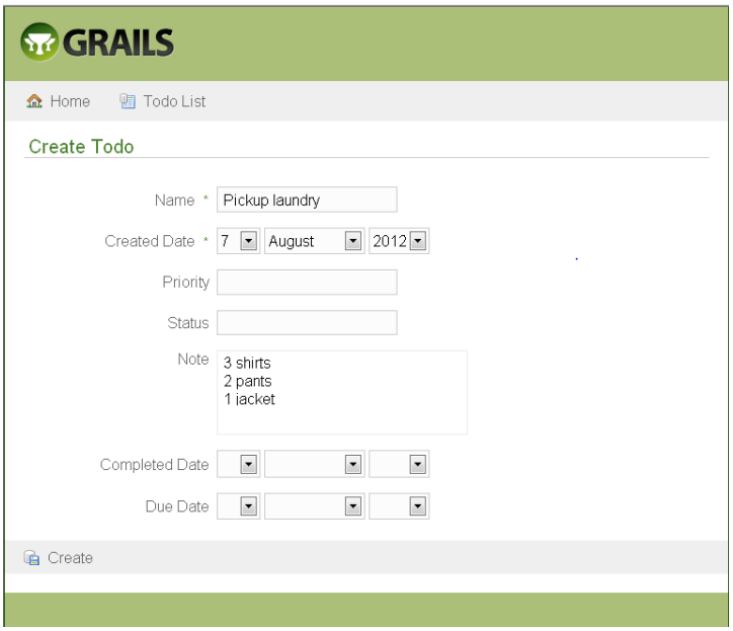


Abbildung 11: Scaffolding - Create View[51]

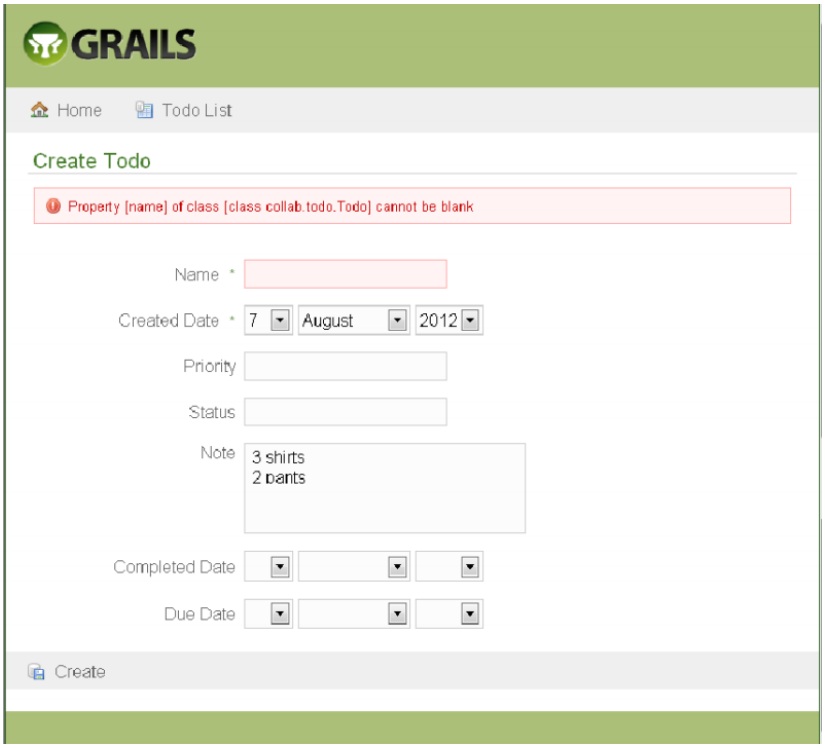


Abbildung 12: Scaffolding Validation[52]

[51] Ebd.
[52] Ebd., 111.

4 Fazit: Warum sollte man Grails (nicht?) nutzen?

Nach diesem Überblick über das Grails Framework und die kleine Einführung in manche seiner Konzepte und Technologien stellt sich abschließend die Frage ob es sich für Entwickler und Projektverantwortliche denn nun lohnt von etablierteren Lösungen umzusteigen und das nächste Projekt mit Groovy/Grails in Angriff zu nehmen.

Zunächst einmal: Was spricht gegen Grails und welche Schwierigkeiten könnten auftreten? Groovy und Grails erledigen unter voller Nutzung seiner Features viele Dinge zur Laufzeit – dies birgt natürlich die Möglichkeit vieler Fehler, deren Debugging sehr aufwändig sein kann. Zu möglichen Fehlerquellen in diesem Kontext gehört beispielsweise auch das dynamische Typing. GORM birgt Schwierigkeiten im Umgang mit Multi-Threaded Applikationen, beispielsweise kann es vorkommen, dass GORM ein neues Domain-Model zwei Mal anlegt – wobei nur eines dauerhaft gespeichert wird. In solchen Szenarien muss ein Entwickler mit großer Vorsicht vorgehen um nicht unvorhersehbarem Verhalten ausgeliefert zu sein. Dass Groovy vor Ausführung in Java-Bytecode übersetzt und interpretiert wird führt zu Performance-Einbußen – es kommt auf die Anforderungen an das Projekt und dessen Umstände an ob dieser Nachteil ins Gewicht fällt. Letztlich haben seit dem erstmaligen Release von Grails auch etablierte Frameworks weitere Entwicklungsschritte durchlaufen und in manchen Gebieten aufgeholt.

Doch was spricht dafür? Grails hat, wie aus der vorliegenden Arbeit sicherlich hervorgeht, einen großen Fokus auf effiziente Entwicklung gelegt. Dies schlägt sich in relativen kurzen Entwicklungszyklen nieder. Es eignet sich darüber hinaus bereits für sehr kleine Projekte und skaliert gut mit der Größe des Projekts. Grails ist gut dokumentiert und verfügt über eine lebendige Open Source Community – das führt auch zu einer sehr großen Anzahl nützlicher Plug-Ins (beispielsweise Spring Security). Es ist sehr leicht aufzusetzen und ermöglicht es durch seine Konventionen, GORM und das Scaffolding in sehr kurzer Zeit beachtliche Beispielanwendungen zu erstellen. Letztlich profitiert Grails auch von dem soliden Fundament auf dem es errichtet ist: etablierte Frameworks wie Spring, Hibernate, Quartz und SiteMesh liegen under the hood mit ihrer vollen Mächtigkeit vor, sind für den Nutzer in den meisten Situationen durch eine leicht bedienbare Fassade kaschiert. Grails ist – unabhängig davon, dass es bereits im dritten Release vorliegt und über 10 Jahre existiert – keine vollkommen neue Technologie. Seine Einzelteile waren bereits zum Release seit Jahren etabliert und einsatzerprobt.

Untenstehende Grafik zeigt eine Auswahl aus Unternehmen, die ihre Entscheidung bereits getroffen und mit Grails gearbeitet haben.

Abbildung 13: Unternehmen die sich für Grails entschieden haben[53]

[53] „Grails :: OCI“.

5 QUELLENVERZEICHNIS

- „7 Object Relational Mapping (GORM) 3.1.6“. Zugegriffen 29. April 2016. https://grails.github.io/grails-doc/latest/guide/GORM.html.
- „8 The Web Layer 3.1.6“. Zugegriffen 29. April 2016. http://grails.github.io/grails-doc/latest/guide/theWebLayer.html.
- „Closure (Funktion)“. *Wikipedia*, 21. Februar 2016. https://de.wikipedia.org/w/index.php?title=Closure_(Funktion)&oldid=151738378.
- „Duck-Typing“. *Wikipedia*, 26. Juni 2015. https://de.wikipedia.org/w/index.php?title=Duck-Typing&oldid=143485519.
- „Gant“. Zugegriffen 26. April 2016. https://gant.github.io/.
- „Grails :: OCI“. Zugegriffen 29. April 2016. http://www.ociweb.com/products/grails/.
- „Groovy Bean Configuration in Spring Framework 4“. Zugegriffen 26. April 2016. https://spring.io/blog/2014/03/03/groovy-bean-configuration-in-spring-framework-4.
- „GroovyCookbook: Dates and Times“. Zugegriffen 28. April 2016. http://groovycookbook.org/basic_types/dates_times/.
- „Home - SiteMesh 2 - Confluence“. Zugegriffen 29. April 2016. http://wiki.sitemesh.org/wiki/display/sitemesh/Home.
- „Informationen zu Java 8“. Zugegriffen 27. April 2016. https://www.java.com/de/download/faq/java8.xml.
- „Introduction to Grails | Noetic Ideas“. Zugegriffen 28. April 2016. http://www.bhaskarpundkar.com/introduction-to-grails/.
- „Kickin’ down the cobblestones: Integrating Grails with EJB 2 applications“. Zugegriffen 26. April 2016. http://dave-klein.blogspot.de/2008/03/integrating-grails-with-ejb-2.html.
- König, Dierk. *Groovy in Action*. Manning Publications, 2007.
- „Konvention vor Konfiguration“. *Wikipedia*, 19. Juni 2015. https://de.wikipedia.org/w/index.php?title=Konvention_vor_Konfiguration&oldid=143240216.
- „Mobile marketing statistics 2016“. Zugegriffen 25. April 2016. http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/.
- Ralph E. Johnson, und Brian Foote. „Designing Reusable Classes“. Zugegriffen 27. April 2016. http://www.laputan.org/drc/drc.html.
- Rocher, Graeme. „Gant/Grails/EAR“, 11. März 2010. http://grails.1312388.n4.nabble.com/Gant-Grails-EAR-td1588511.html.
- Rocher, Graeme, und Jeff Scott Brown. *The Definitive Guide to Grails 2*. Apress, o. J.
- Smith, Glen, und Peter Ledbrook. *Grails in Action*. Greenwich: Manning Publications, 2009.
- „Spring: the foundation for Grails“. Zugegriffen 26. April 2016. http://spring.io/blog/2010/06/08/spring-the-foundation-for-grails/.
- „The Groovy programming language“. Zugegriffen 27. April 2016. http://www.groovy-lang.org/.
- Vishal Layka. *Beginning Groovy, Grails and Griffon*. Apress, o. J.
- „Web Sites Using Grails“. Zugegriffen 26. April 2016. https://grails.org/websites?offset=0&max=12.
- „Wiki - Deployment“. Zugegriffen 26. April 2016. https://grails.org/wiki/Deployment.