exensio
intelligente informationssysteme

Grails Schnelleinstieg

2. Oktober 2009

An heutige Webapplikationen werden hohe Anforderungen gestellt. Einer hohen Nutzerfreundlichkeit, Qualität und kurzen Entwicklungszeit stehen eine Vielzahl von unterschiedlichen und komplexen Technologien, Auszeichnungssprachen und Frameworks gegenüber. Für diese Problemstellung bietet das Open Source Framework Grails durch die sinnvolle Kombination und Erweiterung von bewährten und modernen Technologien und der konsequenten Umsetzung von Grundsätzen wie „Konvention statt Konfiguration" einen umfassenden Lösungsansatz. Der folgende Artikel bietet einen Schnelleinstieg in das Grails-Framework. Dabei wird auf den technischen Hintergrund, die Anwendung und auf die Chancenbewertung durch den Einsatz des Grails-Frameworks eingegangen.

Inhaltsverzeichnis

1. Einleitung

Die Anforderungen an neue Web Applikationen steigen stetig. Häufig werden Kenntnisse in einer Vielzahl von Programmier- und Auszeichnungssprachen, Frameworks, Protokollen und dem Datenbankeneinsatz benötigt. Daneben besteht der Bedarf an schnelleren Entwicklungszeiten, konstantem Feedback bei der iterativen Entwicklung der Anwendung und Möglichkeiten für kurzfristige und flexible Änderungen.

Trotz allem sollen aber neben Flexibilität und Produktivität sichere, skalierbare und performante Technologien Anwendung finden und bereits bestehendes Kapital in Form von Quellcode und Fachkenntnissen weiterhin genutzt werden.

Im Java-Umfeld steht mit der Java EE eine leistungsstarke und vielfältig einsetzbare Softwarearchitektur zur Verfügung. Obwohl diese konsequent um bewährte Frameworks wie Spring und Hibernate erweitert wurde, bleibt die Entwicklung von Web Applikationen mit der Java EE schwerfällig. Eine niedrige Abstraktionsebene und damit einhergehende langwierige anfängliche Konfigurationen, häufige Codereproduktion und Javas statische Eigenschaften sind nur einige dafür verantwortliche Punkte.

Groovy [1] bildet als eine der vielversprechendsten dynamischen Sprachen für die *Java Virtual Machine* (JVM) die Grundlage für das Open Source Web Aplikationen Framework *Grails* [2]. Dies gewährleistet die problemlose Integration von bereits vorhandenem Java Code in neue Anwendungen und eine schnelle Erlernbarkeit für mit der *Java Programmiersprache* [3] vertraute Entwickler. Neben Groovys beeindruckenden Eigenschaften und ausdrucksstarker Syntax, nutzt Grails eine Vielzahl von bewährten Technologien, wie *Spring* [4], *Hibernate* [5], *Site Mesh* [6], *Jetty* [7], *HSQLDB* [8] und *JUnit* [9].

Paradigmen, wie *KISS* (Keep it small and simple), *DRY* (Don't repeat yourself) und *Konvention statt Konfiguration*, sollen die Arbeit mit Grails erleichtern. Letzteres ermöglicht eine bessere Übersicht innerhalb von Projekten, deren einfacheres Verständnis für Dritte und eine

erhebliche Codereduktion. Kiss steht für die Einfachheit und schnelle Erlernbarkeit von Grails und DRY für Redundanzfreiheit und einen minimalistischen Quellcode.

Grails basiert auf der *Model-View-Controller-Architektur* (MVC-Architektur) und bietet durch sein Object-Relational-Mapping-Framework *GORM* die Möglichkeit, Objekte direkt auf relationale Datenbanken abzubilden. Zudem wird die automatische Erzeugung eines Grundgerüsts der Anwendungen durch Grails Scaffolding-Funktionalität ermöglicht. Diese generieren zum Beispiel den benötigten Code für grundlegende *CRUD*-Funktionalitäten (create, read, update, delete), wie das Anlegen, Abrufen, Aktualisieren oder Löschen von Datensätzen innerhalb der Controller und der benötigten Views. Letztere werden durch so genannte *Groovy-Server-Pages* (GSP) erstellt, dem um sinnvolle Funktionen erweiterten Äquivalent zu *Java-Server Pages* (JSP).

Wer sich selbst von Grails Leistungsstärke überzeugen möchte, kann auf eine Vielzahl von Referenzen (siehe [10]) zurückgreifen. Auf Grails basierende Seiten mit täglich mehreren Millionen Hits verdeutlichen dabei, dass Grails für den Einsatz in großen Projekten und den produktiven Einsatz bereit ist.

Kurz gesagt verspricht Grails das, was bereits von vielen Web-Application-Frameworks zuvor versprochen wurde. Ein einfaches, sinnvoll strukturiertes und vorkonfiguriertes Open-Source-Framework zu sein, das bewährte und moderne Technologien einbezieht und neben Spaß bei der Entwicklung auch noch eine hohe Produktivität mit sich bringt. Begeben wir uns also gemeinsam auf die Suche nach einer Antwort auf die Fragen, ob und warum genau Grails die Möglichkeiten besitzt, diesen ehrgeizigen Ansatz zu erfüllen.

Der folgende Teil des Artikels geht tiefer auf die theoretischen Grundlagen von Grails ein, darunter seine Entstehung, technischen Grundlagen und Funktionsweisen. Kapitel 3 demonstriert direkt die praktische Arbeitsweise mit dem Framework anhand eines Beispiels. Allen, die nur einen schnellen Einblick in die Nutzung von Grails gewinnen möchten und sich mehr für das Wie als für das Warum interessieren, sei daher empfohlen, direkt mit der Lektüre von Kapitel 3 fortzufahren.

2. Theoretische Grundlagen

2.1. Technologien und Aufbau

Wie zuvor beschrieben, basiert Grails auf einer Vielzahl von unterschiedlichen, soliden Frameworks und Komponenten. Dabei versucht es die jeweiligen Stärken optimal miteinander zu kombinieren und bestehende Schwächen auszugleichen. Für ein besseres Verständnis des gesamten Frameworks, wird daher hier ein detaillierter Blick auf seinen Aufbau und verwendete Technologien geworfen.

2.1.1. Java, Groovy und die JVM

Die vielfältigen Einsatzmöglichkeiten von Java spiegeln sich insbesondere in seiner Systemunabhängigkeit aufgrund der JVM, großen Anzahl von verfügbaren Bibliotheken und umfassenden API wieder. Allerdings neigt die Javasyntax dazu, nicht sehr ausdrucksstark zu sein und häufig redundanten Code zu erzeugen. Auf der Suche nach einer flexibleren Sprache für die JVM, schaffte es Groovy, aufgrund seiner aussagekräftigen Syntax und dynamischen Eigenschaften, besonders auf sich aufmerksam zu machen. Groovycode läuft innerhalb der JVM ab und wird daher vor der Ausführung in Java-Bytecode übersetzt. Dies ermöglicht eine einfache Integration von bereits vorhandenem Javacode. Auch syntaktisch lehnt sich Groovy stark an Java an. Dadurch sind bestehende Vorkenntnisse in der Programmiersprache Java für die Arbeit mit Grails hilfreich, aber aufgrund der einfachen Erlernbarkeit von Groovy nicht zwingend nötig.

2.1.2. Hibernate und GORM

Hibernate ist ein frei verfügbares Persistenz-Framework für Java. Es erleichtert die Verwendung von relationalen Datenbanken, indem es den Zustand von Objekten in diesen speichert oder neue Objekte aus vorhandenen Datensätzen erzeugt. Es bildet die Basis für das in Grails zur Anwendung kommende Object-Relational-Mapping-Framework GORM. GORM bietet neben der Abbildung von Beziehungstypen verschiedener Kardinalitäten, wie 1:1 oder 1:n in Chen-Notation, eine Vielzahl von weiteren Möglichkeiten, wie die Verwendung von speziellen Suchkriterien oder dynamischen Suchmethoden. Dabei wird die Syntax der Abfragen in einer separaten Anwendungsschicht von der Datenquelle abstrahiert. Daher ist es nicht notwendig, aber möglich, eigene SQL-Ausdrücke für Datenzugriffe zu formulieren. Bei der automatischen Erzeugung eines Datenbankschemas, sollte jedoch darauf geachtet werden, dass von Grails nicht immer der performanteste Ansatz gewählt wird.

2.1.3. HSQLDB

Grails ermöglicht die einfache Konfiguration und Einbindung von externen Datenbanken zu verschiedenen Zwecken. Deren Verwendung ist allerdings nicht zwingend notwendig, da für jede neue Anwendung standardmäßig eine HSQL-Datenbank erzeugt wird. Dabei handelt es sich um eine komplett auf Java basierende, frei verfügbare relationale SQL-Datenbank.

2.1.4. Jetty

Jetty ist ein komplett in Java realisierter Servlet Container und Webserver, der unter der Apache 2.0 Lizenz steht und frei für den kommerziellen Gebrauch nutzbar ist. Dabei gilt Jetty als leichtgewichtig und einfach integrierbar.

2.1.5. SiteMesh

SiteMesh übernimmt innerhalb von Grails die Aufgabe des Webseiten Layouts. Es steht unter der OpenSymphony Software Lizenz, die

ähnlich der Apache 2.0 Lizenz frei für den kommerziellen Gebrauch verwendet werden kann.

2.1.6. JUnit

JUnit stellt ein Framework zum Schreiben von Testfällen dar. Innerhalb von Grails kann es für die Erstellung von Komponententests genutzt werden.

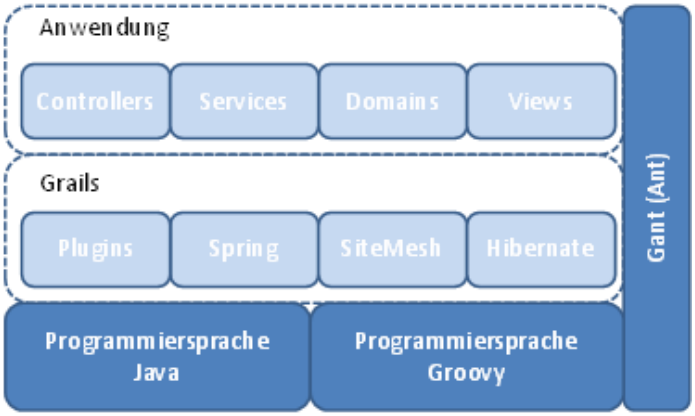

Abbildung 1: Grails Architektur

2.2. Funktionsweise

Grails basiert auf dem MVC-Architekturmuster und trennt somit das genutzte Datenmodell (Modell), die Programmlogik (Controller) und die gewählte Präsentation (View) in separate Schichten.

Die typische Bearbeitung einer Anfrage durch einen Client ist in Abbildung 2 vereinfacht dargestellt. Die Anfrage wird mit den übermittelten Parametern an einen Controller übermittelt, der diese bearbeitet und dabei auf das zugrunde liegende Datenmodell zugreifen kann. Diese Datenzugriffe können direkt über die Grails-Domainklassen mit Hilfe von GORM erfolgen.

Schließlich wird die Präsentation des Anfrageergebnisses erstellt und an den betreffenden Client verschickt. Dabei werden GSPs verwendet, bei denen es sich um gewöhnliche HTML, kombiniert mit so genannten G-Tags handelt. Die G-Tags bieten dabei eine erweiterte, auf die Verarbeitung mit Grails abgestimmte dynamische Funktionalität, wie beispielsweise Iteratoren.

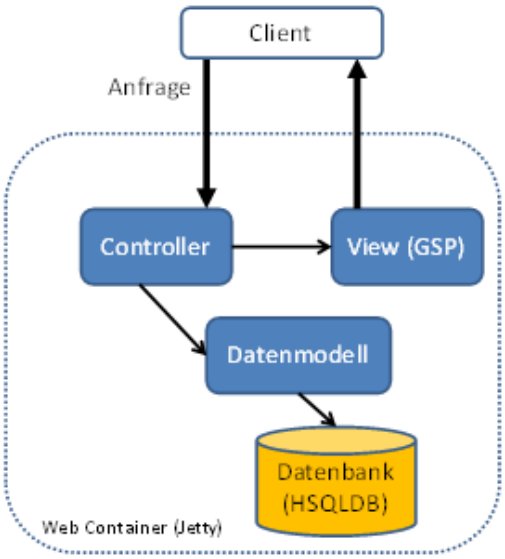

Abbildung 2: Standardisierte Anfragebearbeitet

2.3. Grundsätze

Bei der Entwicklung von Grails wurden folgende Anforderungen an das Framework in den Vordergrund gestellt.

• Bei der Erstellung einer Web-Applikation sollten stets wieder auftretende Konfigurationsaufgaben minimiert werden.

• Die Projektstrukturen sollten ein übersichtliches Arbeiten und ein schnelles Verständnis für Dritte ermöglichen.

• Es sollte möglich sein, die eigene Entwicklungsarbeit möglichst frühzeitig anhand einer lauffähigen Anwendung zu beobachten und zu validieren.

Einige der aus diesen Anforderungen resultierenden grundlegenden Prinzipien von Grails sollen im Folgenden kurz vorgestellt werden.

2.3.1. Konvention statt Konfiguration

Grails Anwendungen besitzen eine vordefinierte Projektstruktur. Innerhalb dieser Projektstruktur erfolgen häufig anfallende Konfigurationsaufgaben, wie das Einbinden einer Datenquelle, stets am gleichen Ort. Zudem versucht Grails soweit möglich, per Konvention auf Programmbestandteile zu verweisen. Wird zum Beispiel auf einen bestimmten Service oder eine View zugegriffen, versucht Grails diese automatisch an der dafür vorgesehenen Stelle zu finden.

2.3.2. Das Grundgerüst

Mit Hilfe von Grails ist es möglich, auf der Basis eines Domainmodells aufbauende Grundfunktionalitäten automatisch generieren zu lassen. Dazu gehört sowohl die Erzeugung von Datenbankschemata, als auch das zur Verfügung stellen von grundlegenden Operationen wie das Anlegen, Abrufen, Aktualisieren oder Löschen von Datensätzen und die Erstellung der dafür notwendigen Programmlogik und Views.

Diese Prinzipien ermöglichen es, mit Grails innerhalb von kurzer Zeit eine voll funktionsfähige Basisanwendung zu erstellen. Diese kann danach schrittweise an bestehende individuelle Wünsche angepasst werden. Dieser Anpassungsprozess kann innerhalb der Views oder der Controller sogar direkt zur Laufzeit, ohne einen zwingenden Neustart der Anwendung, erfolgen, was eine schnelle praktische Überprüfung von Änderungen ermöglicht.

3. Anwendung und Nutzung

3.1. Installation und integrierte Entwicklungsumgebungen

Trotz der zuvor beschriebenen reichhaltigen Funktionalität von Grails, wird für dessen Verwendung kein vorinstallierter Anwendungsserver und dank der vorhandenen HSQL-Datenbank auch nicht zwangsweise eine vorhandene externe Datenquelle vorausgesetzt. Lediglich eine Version des Java Developement Kits (mindestens Version 1.4) wird benötigt. Der Download einer Grails Distribution ist unter [11] möglich.

Für die Installation sind die folgenden Schritte durchzuführen:

1. Bei der Verwendung unter Windows und eines Installers einfach die Installationsroutine befolgen. Alternativ die vorhandene .zip oder .tar/.gz Datei in das gewünschte Verzeichnis entpacken.

2. Setzen der benötigten Umgebungsvariablen, damit die Komandozeilenfunktionen von Grails in allen Verzeichnissen zur Verfügung stehen. Dazu muss die Umgebungsvariable GRAILS_HOME um einem Verweis auf das zuvor gewählte Installationsverzeichnis ergänzt werden. Des weiteren muss die PATH Umgebungsvariable um den Eintrag „%GRAILS_HOME%\bin“ (unter Windows) ergänzt werden.

Die Verwendung einer integrierten Entwicklungsumgebung (IDE) wird für die Arbeit mit Grails nicht zwingend benötigt. Trotzdem muss nicht auf Komfortfunktionen wie Syntaxhervorhebung, Testunterstützung oder direkte Ausführung von Grails-Befehlen innerhalb der Entwicklungsumgebung verzichtet werden. Die *Netbeans IDE* [12] bietet ab der Version 6.5 von Haus aus eine integrierte Grails und Groovy Unterstützung. Allerdings kann auch beispielsweise auf die *IntelliJ IDE* [13] und *Eclipse IDE* [14] zurückgegriffen werden.

3.2. Die erste Anwendung

Die Erstellung einer einfachen Beispielanwendung namens EventOrg soll das Vorgehen bei der Arbeit mit Grails veranschaulichen. Mit ihrer Hilfe sollen die Daten von regelmäßig stattfindende Fortbildungsveranstaltungen und deren Teilnehmer verwaltet werden. Der Kunde identifiziert folgende Anforderungen an die Anwendung:

• Jede neue Veranstaltungen benötigt Angaben, wie Name, Zeit und Ort.

• Für jede Veranstaltung kann sich eine zuvor festgelegte Anzahl von Teilnehmer registrieren.

• Jeder Teilnehmer soll neben seinem Namen eine gültige Emailadresse angeben, um über eventuelle Änderungen informiert zu werden.

• Sowohl für Veranstaltungen, als auch Teilnehmer sollen Verwaltungsfunktionen, wie das Anlegen, Abrufen, Ändern und Löschen zur Verfügung stehen.

• Beim Überschreiten der maximalen Teilnehmerzahl werden die Teilnehmer nach dem "First Come, First Served"-Prinzip ausgewählt.

Diese Angaben werden durch das in Abbildung 3 mit Hilfe der Unified Modeling Language erstellte Diagramm vereinfacht repräsentiert.

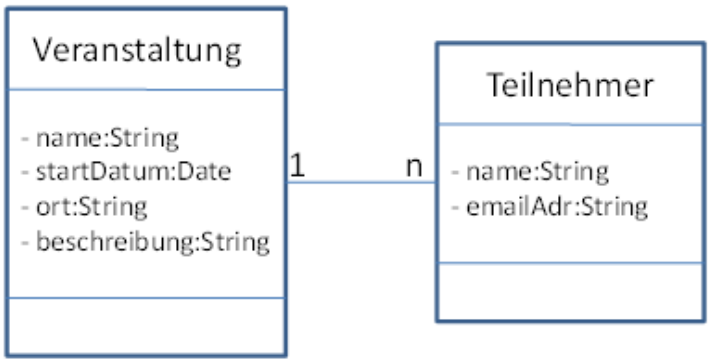

Abbildung 3: UML-Diagramm

3.3. Erstellen der Anwendung

Entwickler können bei der Arbeit mit Grails auf die automatische Erstellung von Grundgerüsten (siehe Absatz 2.3.2) für ihre Anwendungen zurückgreifen. Dies ermöglicht es, die Entwicklung des Projekts möglichst schnell anhand eines lauffähigen Prototyps zu verfolgen und zu validieren.

Für die Erzeugung dieses Grundgerüsts unserer Beispielanwendung verwenden wir den „create-app" Befehl:

> grails create-app EventOrg

Nach der Ausführung des Befehls wird ein Verzeichnis mit dem Projektname und die in Abbildung 4 dargestellte Unterordnerstruktur erzeugt.

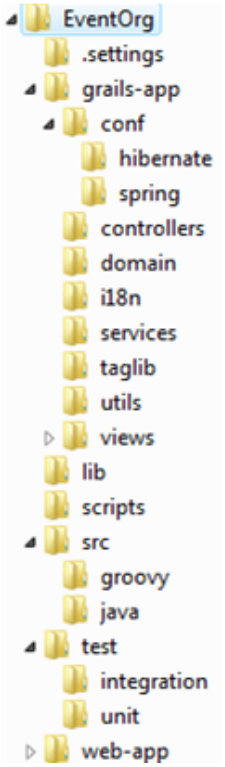

Abbildung 4: Projektstruktur

Die so erzeugte Anwendung ist bereits lauffähig, ohne dass wir jegliche weitere Konfigurationen, wie das Einrichten einer Datenquelle, durchführen müssen. Dies ist möglich, da Grails gemäß dem Prinzip Konvention statt Konfiguration (siehe Absatz 2.3.1) Projekte mit einer in vielen Fällen sinnvollen Basiseinrichtung erstellt. Im Falle der Datenquelle wird dabei auf die interne HSQL-Datenbank (siehe Absatz 2.1.3) zurückgegriffen. Eigene Datenbanken können alternativ im Verzeichnis „grails-app/config" mit Hilfe der Datei „DataSource.groovy" in die Anwendung eingebunden werden (siehe Abbildung 5). In diesem Verzeichnis befinden sich auch die weiteren von Grails erzeugten Konfigurationsdateien.

```
view plain  copy to clipboard  print  ?
01. dataSource {
02.    pooled = true
03.    driverClassName = "org.hsqldb.jdbcDriver"
04.    username = "sa"
```

```
05.     password = ""
06.  }
07.  hibernate {
08.     cache.use_second_level_cache=true
09.     cache.use_query_cache=true
10.     cache.provider_class='com.opensymphony.oscache.hibernate.OSCacheProvider'
11.  }
12.  // environment specific settings
13.  environments {
14.     development {
15.        dataSource {
16.           dbCreate = "create-drop" // one of 'create', 'create-drop','update'
17.           url = "jdbc:hsqldb:mem:devDB"
18.        }
19.     }
20.     test {
21.        dataSource {
22.           dbCreate = "update"
23.           url = "jdbc:hsqldb:mem:testDb"
24.        }
25.     }
26.     production {
27.        dataSource {
28.           dbCreate = "update"
29.           url = "jdbc:hsqldb:file:prodDb;shutdown=true"
30.        }
31.     }
32.  }
```

Abbildung 5: DataSource.groovy

3.3.1. Die Domainklassen

Auch wenn unsere Anwendung bereits lauffähig ist, fehlt es ihr bisher noch an Funktionalität. Zu diesem Zweck beginnen wir mit dem Hinzufügen von Domainklassen. Diese können mit dem Befehl „create-doamin-class" erzeugt werden. Wie im folgenden Beispiel für die Veranstaltungsklasse gezeigt.

>grails create-domain-class Veranstaltung

Klein geschriebene Klassennamen werden dabei von Grails automatisch in groß geschriebene transformiert. Nach der Ausführung werden automatisch zwei neue Dateien erstellt. Zum einen die eigentliche Domainklasse „Veranstaltung" (siehe Abbildung 6) sowie eine korrespondierende Testklasse (siehe Abbildung 7).

view plain copy to clipboard print ?

```
01.  class Veranstaltung {
02.  }
```

Abbildung 6: Erzeugte Domainklasse

view plain copy to clipboard print ?

```
01.  class VeranstaltungTests extends GroovyTestCase {
02.     void testSomething() {
03.     }
04.  }
```

Abbildung 7: Erzeugte Testklasse

Diese gibt bereits einen Hinweis auf die ausgezeichnete Testunterstützung, die Grails zur Verfügung stellt. Auch wenn diese während der kompletten Entwicklung genutzt werden sollte, wollen wir in unserem kurzen Beispiel auf die detaillierte Erstellung von einzelnen Testklassen verzichten. Als nächstes ergänzen wir die zuvor identifizierten Attribute (siehe Abbildung 3) in unserer Domainklasse „Veranstaltung" und wiederholen diese Schritte für unsere zweite Domainklasse „Teilnehmer". Assoziationen zwischen den Domainklassen werden mit Hilfe der Schlüsselwörter „hasMany" und „belongsTo" beschrieben und stets als statisch deklariert. Mit „hasMany" wird ein Attribut vom Typ java.util.Set erzeugt, in dem die Referenzen zu den zugehörigen Objekten gespeichert werden können. Die Gegenseite der Assoziation und die damit verbundene Abhängigkeit in der korrespondierenden Domainklasse wird durch „belongsTo" ausgedrückt (siehe Abbildung 8 und 9).

view plain copy to clipboard print ?

```
01.  class Teilnehmer {
02.     static belongsTo = Veranstaltung
03.     Veranstaltung veranstaltung
04.     String name
05.     String emailAdr
06.  }
```

Abbildung 8: Teilnehmer Domainklasse

```
view plain  copy to clipboard  print  ?
01.  class Veranstaltung {
02.     static hasMany = [teilnehmer:Teilnehmer]
03.     String name
04.     Date startDatum
05.     String ort
06.     String beschreibung
07.  }
```

Abbildung 9: Veranstaltung Domainklasse

3.3.2. Das Grundgerüst

Aufbauend auf einem bestehenden Datenmodell bietet Grails die Möglichkeit, die Programmlogik und die Views für grundlegende CRUD-Funktionen automatisch zu erzeugen. In Abbildung 10 sehen wir den Code, der innerhalb des Controllers ergänzt werden muss, um diese Funktionalität für unsere Veranstaltungsklasse bereitzustellen.

```
view plain  copy to clipboard  print  ?
01.  class VeranstaltungController {
02.     def scaffold = Veranstaltung
03.  }
```

Abbildung 10: Scaffold Veranstaltung

Diesen Scaffolding-Prozess (engl.: „scaffold" = Gerüst) möchten wir auch für unsere Klasse „Teilnehmer" nutzen und ergänzen darum in dem korrespondierenden Controller folgenden Code:

```
view plain  copy to clipboard  print  ?
01.  class TeilnehmerController {
02.     def scaffold = Teilnehmer
03.  }
```

Abbildung 11: Scaffold Teilnehmer

Um zu überprüfen, was diese wenigen Arbeitsschritte geleistet haben, starten wir unsere Anwendung mit dem „run-app" Befehl und rufen mit unserem Webbrowser die URL „http://localhost:8080/EventOrg" auf:

>grails run-app EventOrg

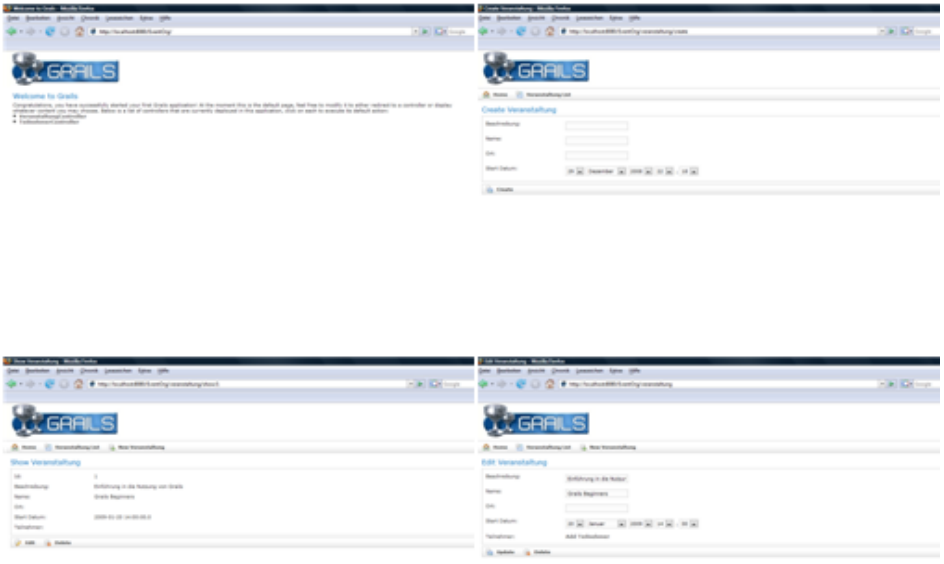

Abbildung 12: Views und CRUD-Funktionen

3.3.3. Validierung und schnelle Anpassung

Mit nur wenigen Schritten haben wir bereits eine lauffähige Webapplikation erstellt, die über grundlegende CRUD-Funktionen verfügt. In den nächsten Schritten beginnen wir mit der Anpassung unserer Anwendung und betrachten den Validierungsprozess von Domainobjekten. Bevor Domainobjekte endgültig an die Datenbank übergeben werden, überprüft Grails per Konvention, ob alle zuvor definierten Eigenschaften für das Domainobjekt vorhanden sind. Häufig ist jedoch eine frühzeitige und genauere Validierung wünschenswert. Zu diesem Zweck können innerhalb von Domainklassen *Constrains* (engl.: „constraint" = Einschränkung) definiert werden. Sie werden in Groovy-Closures (Closure) und einer eigenen „Domänenspezifischen Sprache" (DSL) verfasst. Für unsere Domainklasse „Veranstaltung" nimmt diese

Closure die folgende Form an:

```
view plain  copy to clipboard  print  ?
01. class Veranstaltung {
02.     //..
03.     static constraints = {
04.         name(maxLength:60, blank:false)
05.         startDatum(min:new Date())
06.         ort(maxSize:500, blank:false)
07.         beschreibung(maxSize:500, blank:false)
08.     }
09. }
```

Abbildung 13: Constraints Veranstaltung

Ein Großteil des in Abbildung 13 ergänzten Codes sollte bereits intuitiv verständlich sein. Innerhalb der statischen Closure werden die einzelnen Validierungsregeln aufgestellt. „Blank" drückt dabei aus, dass eine Zeichenkette nicht leer sein darf und „min" gibt einen bestimmten Mindestwert vor. Interessant sind dabei besonders nicht offensichtliche Änderungen, die Grails per Konvention vornimmt. Die Reihenfolge der einzelnen Attribute gibt dabei ebenfalls deren Nennung innerhalb der einzelnen Views vor (siehe Abbildung 14 und 15) und im Falle von „maxSize" wird das HTML Eingabefeld vom Typ „text" durch eines vom Typ „textarea" ersetzt.

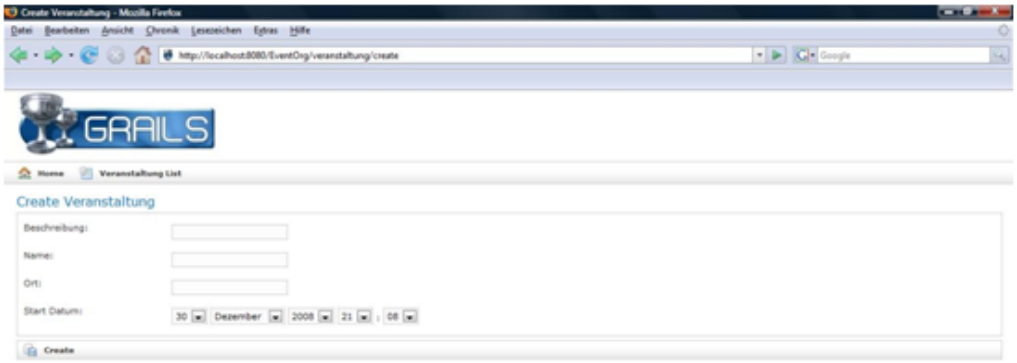

Abbildung 14: Create-View ohne Constraints

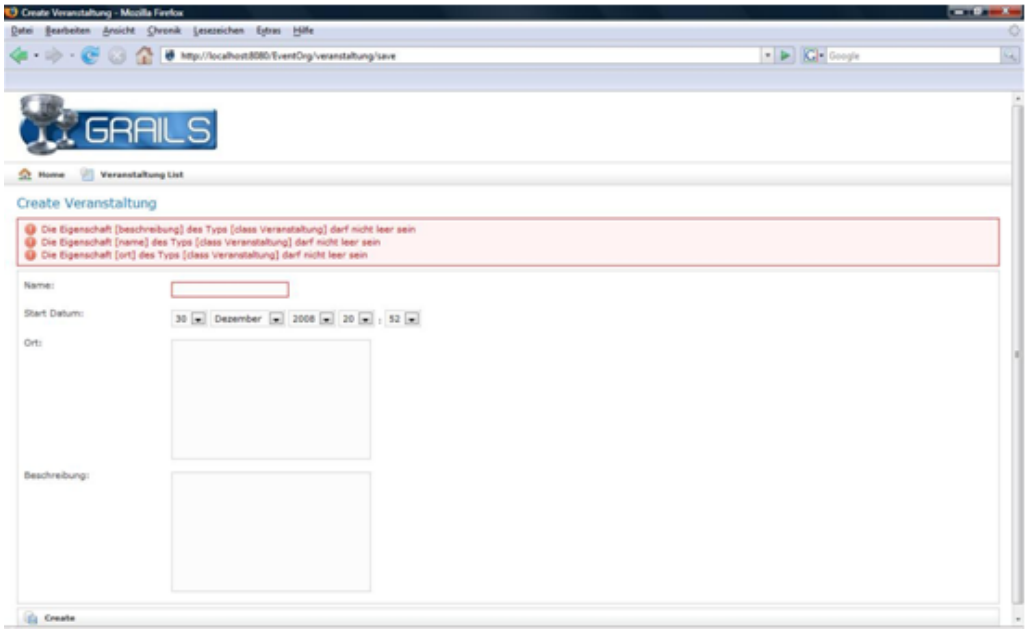

Abbildung 15: Create-View mit Constraints

Außerdem werden die automatisch generierten Fehlermeldungen für Nutzer und das Datenbankschema angepasst. Schließlich ergänzen wir die entsprechenden Constraints für unsere zweite Domainklasse (siehe Abbildung 16).

```
view plain  copy to clipboard  print  ?
01. class Teilnehmer {
02.     //..
03.     static constraints = {
04.         name(blank:false)
05.         emailAdr(email:true, blank:false)
06.     }
07. }
```

Abbildung 16: Constraints Teilnehmer

4. Fazit/Ausblick

Obwohl es sich bei Grails um ein noch recht junges Framework handelt, sind nur wenige Kritikpunkte zu finden, die gegen seinen Einsatz sprechen. Anfänglich geäußerte Kritik, wie eine eingeschränkte Toolunterstützung oder die nicht optimale IDE Integration, können aufgrund der rasanten Verbreitung, Akzeptanz und Weiterentwicklung von Grails mittlerweile nicht mehr angeführt werden. Die erfolgreiche Nutzung von Grails als Grundlage für große Projekte, wie Portalseiten mit mehreren Millionen täglichen Hits, verdeutlichen, dass Grails für den produktiven Einsatz bereit ist.
Grails zeichnet sich durch seine perfekte Integration mit Java, einfache Anwendbarkeit, Stabilität und Flexibilität aus. Dies wird zum einen dadurch erreicht, dass Grails auf bewährten und sicheren Technologien aufbaut. Dabei nutzt es Groovy, eine der vielversprechendsten dynamischen Sprachen für die JVM. Zum anderen führen Prinzipien wie Konvention statt Konfiguration oder die Möglichkeit sich das Grundgerüst einer Anwendung automatisch erstellen zu lassen zu einer übersichtlicheren, schnelleren und dynamischeren Entwicklung von verständlicheren Anwendungen, im Vergleich zum J2EE-Ansatz. Dieser Prozess wird zudem durch die große und stetig steigende Anzahl von verfügbaren Plugins unterstützt, die den Funktionsumfang von Grails konsequent erweitern. Im Gesamtbild bietet die Verwendung von Grails eine höhere Produktivität in der Anwendungsentwicklung, bei einer im Vergleich sowohl schnelleren Erstellung, als auch besseren Qualität des Ergebnisses. Die Verwendung der Apache-Lizenz 2.0 erleichtert dabei erheblich die Verwertung dieser Ergebnisse, im Vergleich zu anderen Open-Source-Lizenzen. Das Potential von Grails und seine nachhaltige Entwicklung werden zusätzlich durch die Übernahme des Grails Projekts durch Spring Source verdeutlicht.
Da sich Grails in einer rasanten Weiterentwicklung mit kurzen Entwicklungszyklen befindet, sollte auch nach zukünftigen Neuerungen Ausschau gehalten werden sollte.

Referenzen

[1] Groovy:
http://groovy.codehaus.org/

[2] Grails:
http://grails.org

[3] Java Programmiersprache:
http://java.sun.com/

[4] Spring
http://www.springframework.org/

[5] Hibernate:
http://www.hibernate.org/

[6] SiteMesh:
http://wiki.sitemesh.org/wiki/display/sitemesh/Home

[7] Jetty
http://jetty.mortbay.org/jetty/

[8] HSQLDB:
http://hsqldb.org/

[9] JUnit
http://junit.sourceforge.net/

[10] Grails - Success Stories
http://www.grails.org/Success+Stories

[11] Grails Distribution:
http://grails.org/Download

[12] Netbeans IDE:
http://www.netbeans.org/

[13] IntelliJ IDEA:
http://www.jetbrains.com/idea/

[14] Eclipse IDE:
http://www.eclipse.org/

[15] Grails Plugins:
http://grails.org/Plugins

[16] Grails Reference:
http://grails.org/doc/1.0.x/

Jörn Kuhlenkamp