Wer suchet, der findet

Das Grails-Webframework – Teil 2: Realisierung eines Portal-Frameworks

Manuel Breitfeld, Tobias Kraft, Peter Soth

In der zweiteiligen Artikelserie schreiben wir über Erfahrungen, die wir in den letzten Jahren mit Grails sammeln konnten. Im ersten Teil wurden die Grundlagen beleuchtet, im zweiten wird die Realisierung eines komplexen Portal-Frameworks mit Grails beschrieben. Der Artikel geht darauf ein, wie und warum sich ein Portal-Framework mit Grails umsetzen lässt, und soll zeigen, dass sich Grails auch für die Entwicklung von komplexeren Anwendungen, wie sie Portale darstellen, eignet.

Warum ein eigenes Portal-Framework?

Wir mussten in unzähligen Portalprojekten feststellen, dass die Systeme der bekannten Hersteller – auch im Open-Source-Bereich – über einen beträchtlichen Funktionsumfang verfügen. Mit dem Nachteil, dass wir außerordentlich viel Zeit dafür aufwenden mussten, diesen für die jeweiligen Kundenanforderungen durch Herunterstrippen anzupassen. Selbst einfache Kundenwünsche waren demzufolge meist nur äußerst aufwendig umzusetzen.

Das weckte bei uns den Wunsch nach einem schlanken und doch flexiblen System, das uns Basis-Funktionen wie Personalisierung, Content- und Dokumentenverwaltung liefert. Da wir Grails bereits in einigen Projekten erfolgreich eingesetzt haben, entschieden wir uns, selbst ein Portal-Framework zu entwickeln, in das unsere jahrelange Erfahrung einfließen sollte. Diesen „Unterbau" haben wir um Eigenschaften wie Wiki, Blog, Bewertungsmodus, Verschlagwortung und eine treffsichere Suchfunktion erweitert.

Wir benutzen dieses schlanke Framework in unseren Projekten als Basis und ergänzen es um die gewünschten Kundenanforderungen mit Hilfe von Grails. So erhält man eine maßgeschneiderte Portal-Lösung in einer angemessenen Projektlaufzeit. Darüber hinaus profitiert auch die Performance davon, da wir keinen unnötigen Ballast auf einer Produktivstellung mitlaufen lassen müssen.

Das Herzstück, der Portal-Controller

Content-Management-Systeme (CMS) – wie sie auch häufig für Portal-Lösungen eingesetzt werden – sind nicht per se für die Integration von Applikationen geeignet. Ein Problem, das bei einem Anwendungsportal gelöst werden muss, ist, dass eine Benutzer-Interaktion innerhalb eines Portlets auf einer Portal-Seite erneut an dieses weitergeleitet wird. Aus diesem Grund unterscheidet sich die URL erheblich von der eines CMS.

Bei unserem Portal-Framework hingegen legten wir viel Wert auf die Integration von Anwendungen, wobei eine Portalseite hier ein oder mehrere Portlets enthalten kann. Diese agieren wie kleine unabhängige Webanwendungen, der zentrale Portal-Controller, der ähnlich einem Dispatcher arbeitet, versorgt sie mit Informationen.

Es ist denkbar, ein Portlet auf verschiedenen Portal-Seiten einzubinden. Aus diesem Grund wird zwischen Portlet und Portlet-Instanz unterschieden, vergleichbar mit Klassen und Objekten. Ein Portlet ist die Klasse und die Portlet-Instanz das Objekt. Jede Portlet-Instanz kann über sogenannte Portlet-Preferences parametrisiert werden.

Mit diesem Hintergrundwissen ist es jetzt einfach, eine Portal-URL zu verstehen, wie sie in Abbildung 1 gezeigt wird. Zum Vergleich zeigt Abbildung 2 eine einfache Grails-URL. Das Präfix po_ (PortletInstanz) wird sämtlichen Attributen einer URL vorangestellt und damit ist es für den Portal-Controller möglich, jeden Parameter an einen dedizierten Portlet-Controller zu senden. Der Portlet-Controller selbst ist bei uns ein Standard-Grails-Controller. Demzufolge können Portlets wie eine Grails-Applikation entwickelt werden. Es ist auch denkbar, eine bestehende Grails-Anwendung als Portlet auf einer Portal-Seite anzuzeigen.

Abb. 1: Aufbau Portal-URL

Abb. 2: Aufbau Standard-Grails-URL

Tagging

Die Verschlagwortung von Inhalten, das sogenannte Tagging, ist mittlerweile im Intra- und Internet nicht mehr wegzudenken. Tags spielen in Portalen eine zentrale Rolle, um unstrukturierte Informationen, wie Content oder Dokumente rascher

wiederzufinden. Im Zeitalter von Enterprise 2.0 gehören zu Content auch Blog-Posts beziehungsweise Wiki-Seiten.

Das von uns auf Basis des Taggable-Plug-ins [TAGGABLE] entwickelte Tagging-Modul kommt ohne direkte Abhängigkeiten im Datenmodell aus, indem es in einer separaten Datenbanktabelle die Beziehung zwischen Tag und Entität verwaltet. Dies hat den Vorteil, dass neue Entitätstypen einfach dem Tagging-Mechanismus hinzugefügt werden können. Das Modul bietet folgende Funktionalitäten:

- Funktionsübergeifendes Tagging: Dieselben Tags können verschiedenen Entitäten zugewiesen werden.
- Hierarchie: Wie in Abbildung 3 zu sehen ist, lassen sich Tags hierarchisch aufbauen.

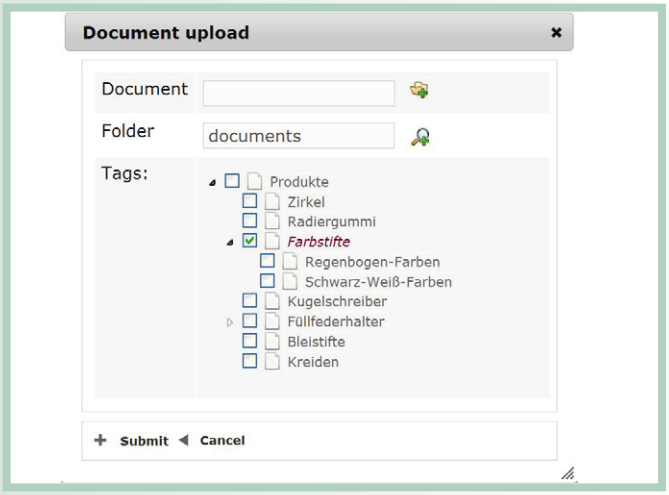

Abb. 3: Hierarchische Darstellung der Tags

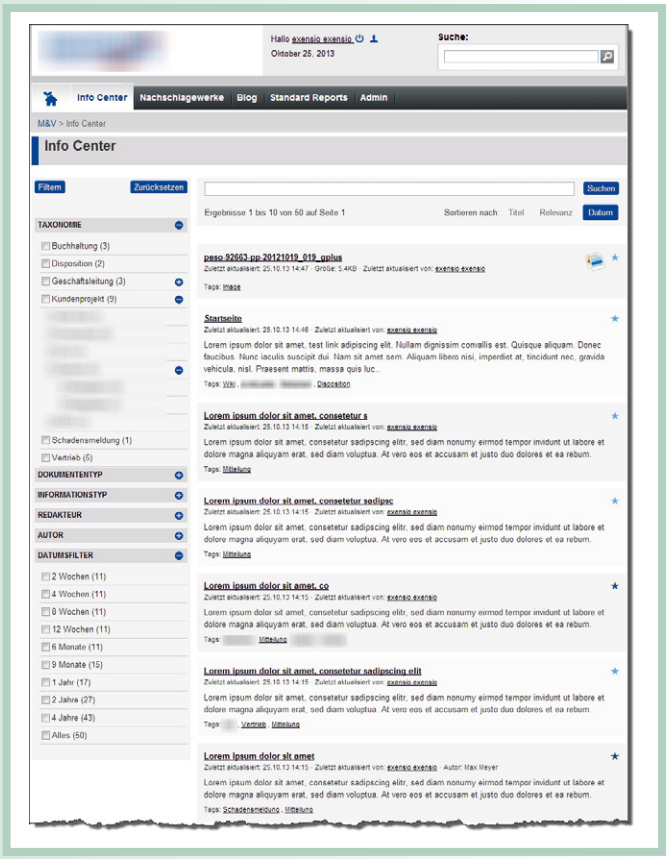

Abb. 4: Info-Center mit Taxonomie auf linker Seite

Enterprise Search mit Apache Solr und der Weg zur Search-Driven Application

Für Portale ist heute eine einwandfreie Volltext-Suche unabdingbar. Wir setzen diese Funktionalität mit Hilfe des Grails-Solr-Plug-ins um. Neben der Volltextsuche enthält das Framework auch eine facettierte Suche. Hierfür verschlagwortet man Daten bzw. Dokumente mit Tags einer Taxonomie. Die Suchergebnismenge kann damit – analog wie von Amazon bekannt – gefiltert werden (s. Abb. 4). Mit einem geeigneten Klassifikationsschema ist eine Information somit enorm rasch wiederzufinden.

Des Weiteren stellten wir im Laufe der Zeit fest, dass Solr-Abfragen teilweise um einiges einfacher als SQL-Abfragen sind. Dabei ist vor allem die „flache Struktur" der Daten im Index hervorzuheben: Wir speichern beispielsweise Nachrichten in einer SQL-Datenbank, benutzen jedoch Solr für die Abfrage, welche Rolle welche Mitteilung sehen darf. Während bei einer SQL-Abfrage hier mindestens ein JOIN (Tabelle Mitteilung zu Tabelle Rolle) notwendig wäre, steht im Solr-Index die Rolle direkt als abfragbares Feld zur Verfügung [QuerySyntax]. Genauer betrachtet folgt unser Framework aus diesem Grund dem Search-Driven Ansatz.

Um von jeder Entität die Rollen – und allgemein alle Eigenschaften – wie beschrieben abfragen und auch darstellen zu können, muss die Entität beim Speichern indiziert und beim Löschen aus dem Index entfernt werden. Das Grails-Solr-Plug-in arbeitet mit Annotationen in der Domain-Klasse, um ein Mapping von Entität zu Solr-Index abzubilden. Wir haben diese Funktionalität um flexiblere Möglichkeiten speziell im Bereich von Relationen erweitert.

Listing 1 zeigt dies anhand der BlogPost-Entität. Das Feld „subject" ist im Solr-Index als „title" verfügbar, für den Autor des BlogPosts (eine Relation) ist angegeben, dass der Vor- und Nachname sowie die E-Mail-Adresse in die entsprechenden Felder im Solr-Index gespeichert werden. Ebenfalls erwähnenswert ist die Methode **`doIndex`**, welche vor der Indizierung aufgerufen wird. Nur wenn sie **`true`** zurückliefert, wird die Entität indiziert – in diesem Beispiel soll der Solr-Index nur tatsächlich publizierte Beiträge enthalten.

Der Vorteil und der Grund, warum Solr-Abfragen einfach sind, wird hierbei deutlich: Jede Entität wird normalisiert und nur mit den Eigenschaften im Index gespeichert, die später bei einer Suche auch wirklich benötigt werden.

```
class BlogPost {
  @Solr(field='title')
  String subject
  @Solr(field='description')
  String body
  @Solr(field='status')
  PostStatus status
@SolrIndex(fields=['firstname','lastname','email'],
    indexFields=['owner_firstname','owner_lastname','owner_email'])
  User author

  boolean doIndex() {
    boolean retVal = status?.equals(PostStatus.PUBLISHED)
    return retVal
  }
}
```

Listing 1: Solr-Annotationen am Beispiel der BlogPost-Entität

Personalisierung mit dem Spring-Security-Plug-in

Eine der wesentlichsten Grundfunktionen eines Portals stellt die Personalisierung dar. Mit dieser erhält jeder Endanwender nur die Informationen, die er für seine tägliche Arbeit benötigt,

eine Informationsüberflutung wird drastisch reduziert. Für die Personalisierung benutzen wir das Spring-Security-Plug-in. Für die Authentifizierung werden mittels Spring Security die User Credentials gegen einen LDAP-Server – wie beispielsweise Active Directory – überprüft. Die Autorisierung wird mithilfe von Rollen abgebildet. Diese steuern die Sichtbarkeit von Portalseiten bzw. einzelnen Menüpunkten. Die personalisierte Anzeige von Informationen – wie zum Beispiel News – wird direkt mit Abfragen über die Apache-Solr-Suchmaschine gesteuert. Hierbei ist wesentlich, dass alle Daten auch mit den Rollen verschlagwortet (getagged) wurden, wie im obigen Abschnitt beschrieben.

Enterprise-2.0-Funktionalitäten

Web-2.0-Funktionalitäten werden auch in Unternehmen immer gefragter, sodass der Begriff Enterprise 2.0 längst etabliert ist. Dabei geht es in erster Linie um die Übertragung von Web-2.0-Konzepten auf die Kommunikationsmittel im Unternehmen. Unser Portal bietet hierfür die folgenden Module:

- Blog
- Wiki
- Kurznachrichten
- Activity Stream
- Teilen, Kommentieren und Bewerten (Voting) von Inhalten
- Integration anderer sozialer Dienste wie Twitter usw.

In diesem Artikel soll insbesondere die Flexibilität des Wiki-Moduls hervorgehoben werden. Unser Wiki kann in Form eines Portlets innerhalb des Portals benutzt werden und bietet durch die Konfiguration über Portlet Preferences die unterschiedlichsten Einsatzgebiete – auch innerhalb desselben Portals. So wird unser Wiki beispielsweise als Portlet für ein Abkürzungsverzeichnis, zur strukturierten Dokumentation von Inhalten und auch zur Pflege einer FAQ verwendet. Für Editoren ist das Wiki einfach über einen WYSIWYG-Editor zu benutzen. Über die integrierte Versionierung können Änderungen nachvollzogen und auch jederzeit frühere Revisionen wiederhergestellt werden.

Das macht das Wissensmanagement in Verbindung mit der Suche sehr effizient, denn alle Inhalte sind automatisch in der Volltextsuche auffindbar und können über verschiedene Facetten einfach gefiltert werden.

Erweiterbarkeit und Integration

Die Erweiterung und Anpassung von Funktionalitäten ist ein essenzieller Bestandteil von Softwaresystemen. Unser Portal-Framework lässt sich mit Standard-Grails-Wissen einfach erweitern: Jeder Controller ist im Prinzip ein Portlet – ohne spezielle Anpassungen! Neue Funktionalität wird demnach anhand des MVC-Konzepts entwickelt (Domain-Klasse, Controller, View und ggf. Service in Grails implementieren) und kann anschließend im Portal als Portlet zur Verfügung gestellt werden. Es ist dabei auch kein Problem, bereits in anderen Grails-Projekten entwickelte Komponenten in unser Portal zu übertragen. Fügt man anschließend noch entsprechende Solr-Annotationen der Domain-Klasse hinzu, erkennt auch die Volltextsuche die Neuentwicklung sofort.

Während die soeben beschriebene Erweiterung um neue Funktionalität eines Deployments bedarf, kann die Definition von Navigation und Portalseiten zur Laufzeit erfolgen. Ebenso lassen sich neue Portlets jederzeit hinzufügen und bereits implementierte Funktionalität durch die Vergabe oder Änderung von Portlet Preferences aktivieren.

Back-End-Systeme abstrahieren wir über Services in Grails. Hierbei ist eine Integration mittels Point-to-Point (P2P) oder Enterprise-Service-Bus (ESB) denkbar. Die P2P-Variante ist die günstigere, jedoch ist ein ESB bestimmt dann sinnvoll, wenn es viele Systeme über wiederverwendbare Services zu integrieren gilt.

Fazit

Unser Portal-Framework enthält alle wesentlichen Bestandteile eines Portals, wie man wunderbar anhand dieses Artikels sieht. Wir hatten die Absicht, mit dieser Artikelserie aufzuzeigen, dass sich mit Grails selbst komplexe Fragestellungen – zu denen auch ein Portal zählt – lösen lassen. Natürlich kann man unsere Lösung nicht mit Hilfe eines Fact-Sheets mit den bekannten Anbietern vergleichen. Denn die Idee ist hier eine andere. Unser Ziel war ein schlankes Basis-Framework, das wir mit Groovy und Grails rasch erweitern können. Das ist aus unserer Sicht gelungen, und wir setzen das Framework heute gerne und erfolgreich in Kundenprojekten ein.

Literatur und Links

[BrKrSo13] M. Breitfeld, T. Kraft, P. Soth, Das Grails-Webframework – Teil 1: Grundlagen und produktiver Einsatz, in: JavaSPEKTRUM, 6/2013

[QuerySyntax] `http://lucene.apache.org/core/2_9_4/queryparsersyntax.html`

[TAGGABLE] `http://grails.org/plugin/taggable`

Manuel Breitfeld ist nach seinem Studium der Softwaretechnik seit 2010 als Consultant für die exensio GmbH tätig. Dabei umfasst sein Tätigkeitsfeld sowohl Entwicklungsprojekte von Enterprise-Portalen und Webapplikationen als auch Beratungsprojekte im Usability- und Intranet-Bereich.
E-Mail: manuel.breitfeld@exensio.de

Tobias Kraft war nach seinem Studium des Wirtschaftsingenieurwesens in mehreren Software- und IT-Beratungshäusern als Consultant und Softwarearchitekt tätig (u. a. sd&m – heute Cap Gemini). Seit 2009 beschäftigt er sich bei der exensio GmbH mit der Architektur und Umsetzung von Enterprise-Portalen sowie Webapplikationen basierend auf Java-Technologien und dem Grails-Framework.
E-Mail: tobias.kraft@exensio.de

Peter Soth arbeitete nach seinem Studium der Elektrotechnik und Informatik mehrere Jahre als Software-Engineer und Senior Consultant bei international tätigen Unternehmen, wie Hewlett Packard, SAS Institute und BEA Systems (nun Oracle). Im Jahre 2006 gründete er die exensio GmbH mit. Seine Tätigkeitsschwerpunkte umfassen die Architekturberatung und Realisierung von Enterprise-Portal-Projekten mittels Java-Technologien.
E-Mail: peter.soth@exensio.de