

## JSR 170: Das Java Content Repository und die Apache-Implementierung Jackrabbit

# Mein Name ist Hase und ich weiß Bescheid

■ VON HENDRIK BECK UND BERND RÜCKER

Die Speicherung von Inhalten jeglicher Art sowie der Zugriff darauf sind seit jeher eine der zentralen Aufgaben von IT-Systemen. Das Spektrum reicht dabei von der unternehmensinternen Datenhaltung über Unternehmenssoftware bis hin zu (häufig stark Content-lastigen) Webanwendungen. Mit dem JSR 170 (Java Content Repository) liegt ein Standard vor, der Lösungen für typische Probleme in diesem Bereich bereitstellt und den wir nun über die Apache-Implementierung Jackrabbit hier vorstellen wollen.

Der Schwerpunkt beim Zugriff auf Content lag bisher meist auf einer Abstraktion auf Datenhaltungsebene, z.B. bei einem (betriebssystemweit einheitlichen) Dateisystem oder diversen Abstraktionen beim Zugriff auf Datenbanken (SQL oder OR Mapper). Dabei ist das Handling des Contents nicht immer ganz trivial, oft verbleiben technische Details der Datenspeicherung in der Anwendung, und manche Aspekte, wie z.B. die Versionierung des Contents, bleiben unberücksichtigt. Abhilfe versucht hier das im JSR 170 spezifizierte Java Content Repository (JCR) zu schaffen. JCR definiert ein Modell, das all diese Aspekte in ein so genanntes Content Repository verlagert. Ein ebenfalls definiertes Java-API gestattet dann den Zugriff auf dieses mächtige Repository.

Das Final Release des JSR 170 wurde am 17. Juni 2005 verabschiedet, momentan liegt ein Maintenance Draft Review zur Überarbeitung der Version 1.0 vor. Außerdem wurde mit dem JSR 283 mit der Arbeit an Version 2.0 des Standards begonnen (siehe Kasten). Da bislang aber erst die Stufe eines Early Draft Review ab-

geschlossen wurde, wird in diesem Artikel die aktuelle Version 1.0 betrachtet.

### Der JCR-Aufbau

Das Ziel bei der JCR-Spezifikation war es, ein API zu entwickeln, das den in der Praxis erforderlichen Funktionsumfang für den Zugriff auf Content abdeckt. Dadurch sollte es ermöglicht werden, jegliche technischen Details, darunter auch den kompletten Persistenzaspekt, hinter diesem API zu verbergen. Durch diese vollständige Entkopplung von Anwendung und Content Repository wird es ermöglicht, die Konfiguration der Datenhaltung transparent gegenüber der Anwendung

zu halten. Dadurch kann auch die Implementierung des Repositorys gegen ein anderes Produkt ausgetauscht werden.

Anwendungen, die nun auf Content zugreifen und daher dieses API verwenden, benötigen nicht immer den kompletten denkbaren Funktionsumfang. Ein Newsticker im Web benötigt im Allgemeinen nur lesenden Zugriff auf die Newsmeldungen. Das Backend einer CMS-Anwendung dagegen benötigt sowohl lesenden als auch schreibenden Zugriff, um die Änderung von Content zu ermöglichen. Vor diesem Hintergrund wurde der Funktionsumfang des JCR API in drei Stufen unterteilt (Spezifikation und API unter [1]):

### JSR 283 – Änderungen in JCR 2.0

Mit dem JSR 283 [10] wird derzeit an einer Version 2.0 des Standards gearbeitet und laut Plan im Mai 2007 verabschiedet. Weniger als der eigentliche Kern der bisherigen Spezifikation werden wohl unterstützende Funktionalitäten überarbeitet werden. So stehen z.B. Themen zum Management von Content Repositories (Access Control, NodeType-Administration) und zur Interoperabilität verschiedener Repositories auf der Agenda, aber auch Abfragesprachen, Versionierung und Observation oder bessere Unterstützung für Remote-Zugriffe auf Repositories. Ein weiterer interessanter Punkt könnten Erweiterungen werden, die ein effektives Con-

tent-Modeling ermöglichen. Da Jackrabbit von Haus aus einiges an Funktionalität mitbringt, das über die Spezifikation hinaus geht und damit manche Unzulänglichkeiten von Version 1.0 löst (etwa die im Text angesprochene NodeType-Definition), ist man hiermit bisher sehr gut bedient. Ansonsten wird Version 2.0 einige Probleme eleganter lösen, die bisher durch die JCR-Implementierung oder projektspezifisch gelöst werden mussten. Ein Einstieg in Content Repositories ist also auch mit Version 1.0 uneingeschränkt zu empfehlen, zumal auch eine Migration von Bestandsdaten recht unkompliziert ausfallen sollte.



Quellcode auf CD

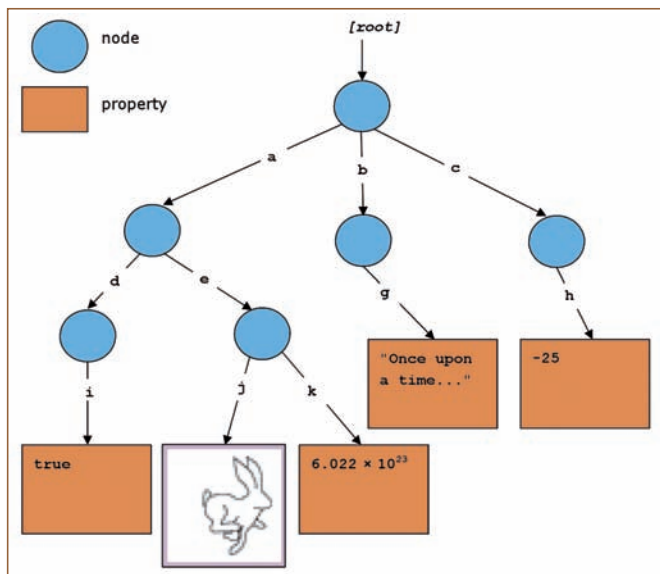


Abb. 1: JCR-Content-Baum [1]

- **Level 1 – lesender Zugriff:** Basics wie das Initialisieren einer Session, das Traversieren des Content-Baums, das Lesen von Inhalten aus dem Repository, die Möglichkeit, über XPath-Ausdrücke im Repository zu suchen, Export nach XML
- **Level 2 – schreibender Zugriff:** Hinzufügen, Verändern und Löschen von Inhalten oder XML-Import
- **optionale Funktionalitäten:** Locks, Transaktionen, Versionierung, Registrierung von Listnern z.B. auf Änderungen von Content, erweiterte Suchmöglichkeiten mittels SQL-Syntax

Erfüllt ein Repository jeweils alle Funktionalitäten von Level 1 bzw. Level 2, kann es als JCR-kompatibel (mit entsprechendem Level) bezeichnet werden. Der unterstützte JCR-Level sowie eventuell implementierte, optionale Funktionalitäten können der Beschreibung des jeweiligen Repositories (also der jeweiligen Implementierung des Standards) entnommen werden.

### Das Repository-Modell von JCR

JCR abstrahiert mit einem Repository-Modell die allgemeine Natur von strukturiertem (aber generischem) Content. Es definiert dazu einige Elemente innerhalb dieses Modells, auf die das API dann entsprechenden Zugriff ermöglicht. Es besteht im Wesentlichen aus folgenden Elementen:

- **Workspace:** Ein Repository enthält beliebig viele Workspaces. Ein Workspace definiert dabei einen in sich abgeschlossenen Bereich zur Speicherung von Content.
- **Node:** Innerhalb eines Workspace ist die Haltung von Content durch eine Menge von Items definiert. Ein Item ist dabei entweder ein Node oder eine Property. Nodes dienen hauptsächlich zur Strukturierung des Contents (ähnlich etwa einem Ordner in einem Dateisystem). Jeder Node kann beliebig viele Nodes und Properties als Kinder besitzen, jeder Workspace besitzt genau einen Root Node.
- **Properties:** In den Properties werden die eigentlichen Inhalte (also deren Werte) gespeichert. Dies können einfache Strings oder numerische Werte sein, aber auch Binärdaten (wie z.B. Bilder). Eine Property besitzt keine Kinder.

Nodes und Properties erzeugen eine baumartige Struktur, wie sie von Dateisystemen her bekannt ist. Analog zu Dateien stellen die Properties die Blätter dieses Baumes dar (Abb. 1). Das Repository-Modell macht dabei keinerlei Vorgaben bzgl. der Art oder der Struktur des zu speichernden Content. Es unterstützt dabei aber durch einige Mechanismen, hauptsächlich den sog. NodeType, durch den jeder Node definiert wird. Jeder Node besitzt genau einen sog. PrimaryNodeType, in dem seine meist eher fachlichen Eigenschaften (z.B. welche Properties und Kind-Knoten vorhanden sein müssen/dürfen) beschrieben werden. Ein NodeType-„Gästebucheintrag“ würde z.B. vorschreiben, dass ein Datum, ein Verfasser und eine Nachricht vorhanden sein müssen.

Darüber hinaus ist es möglich, MixinNodeTypes zuzuweisen, über die dem Knoten weitere, meist eher technische Eigenschaften zugewiesen werden können. Der vordefinierte NodeType *mix:versionable* sorgt z.B. dafür, dass für diesen Knoten die Versionierung aktiviert wird.

Der JCR-Standard spezifiziert dabei einige so genannte „built-in“ NodeTypes, wie z.B. *nt:file* oder *nt:folder*. Damit lassen sich bereits einfache filesystemartige Strukturen im Repository abbilden. Wesentlich mächtigere Content-Strukturen werden allerdings durch die Definition eigener NodeTypes ermöglicht, was in der Praxis auch unumgänglich ist. Listing 1 zeigt beispielhaft die Definition von NodeTypes für ein Gästebuch in der Jackrabbit-eigenen Notation CND.

### Das JCR API

Das JCR API bietet nun eine generische Schnittstelle für den Zugriff auf den Content, das heißt, für den Zugriff auf Reposi-

### Begriffsdefinitionen

- **JSR 170:** der Standard, der das im Artikel beschriebene Repository-Modell, die NodeTypes und einiges mehr spezifiziert. Außerdem definiert er mittels eines Java-APIs die Schnittstelle zum Repository.
- **JCR:** Java Content Repository ist der Name des JSR 170. Die beiden Begriffe sind also identisch.
- **Jackrabbit:** JCR stellt nur eine Spezifikation dar, die entsprechend implementiert werden muss. Jackrabbit ist eine populäre Open-Source-Implementierung und wird auch in diesem Artikel betrachtet.

### Listing 1

```
<JavaMagazin='http://www.camunda.com/
                                Java-Magazin/1.0'>
  [JavaMagazin:Eintrag]
  - JavaMagazin:datum (date)   mandatory
  - JavaMagazin:autor (string)
  [JavaMagazin:Gastebucheintrag] >
                                JavaMagazin:Eintrag
  - JavaMagazin:titel (string) mandatory
  - JavaMagazin:nachricht (string) mandatory
  - JavaMagazin:bild (binary)  multiple
```

tory, Workspaces, Nodes und Properties. Listing 2 zeigt einige grundsätzliche Funktionen des APIs, wie z.B. das Traversieren von Content, das Setzen von Properties oder das Hinzufügen neuer Nodes. Das komplette API ist unter [1] verfügbar, ausführlichere Beispiele zur Verwendung des API finden sich z.B. unter [6].

## Implementierungen

Der Standard selbst legt nun nur die Schnittstelle für den Zugriff auf den Content sowie einige Rahmenbedingungen fest. Die eigentliche Funktionalität wird dann durch geeignete Implementierungen angeboten. Von dieser jeweiligen Implementierung hängen dann z.B. die Art der Speicherung (Datenbank, File-System) oder die Art des Deployments (Standalone, Application Server) ab. Bekannte Beispiele hierfür sind:

- die Referenzimplementierung des JSR 170 der Firma Day Software
- Content Repository Extreme (CRX), kommerzielle Implementierung, ebenfalls von Day Software
- Apache Jackrabbit, populäre Open-Source-Implementierung
- Joiceira, eXo JCR, Alfresco, weitere Implementierungen

Wir werden hier auf Jackrabbit eingehen. Diese Implementierung kann momentan als die bekannteste und verbreitetste angesehen werden. Für ein konkretes

Projekt, den JSF-basierten Webshop von mad-moxx ([www.mad-moxx.de](http://www.mad-moxx.de)), fiel die Wahl nicht nur aus diesem Grund auf Jackrabbit, es war auch schlichtweg die erste am Markt erhältliche Open-Source-Implementierung.

## Jackrabbit

Jackrabbit ist im Frühjahr des letzten Jahres aus dem Incubator-Status in ein Top-Level-Projekt innerhalb von Apache aufgestiegen und liegt momentan in der Version 1.1 vor. Jackrabbit präsentiert sich als vollwertige Implementierung des Standards, deckt also neben Level 1 und Level 2 auch den optionalen Funktionsumfang ab. Darüber hinaus bringt es weitere Funktionalitäten mit, die vor allem bei der Administration des Content Repository unterstützen sollen.

Das API von Jackrabbit ist naturgemäß zu einem großen Teil von dem JCR API *javax.jcr.\** vorgegeben. Dies ermöglicht die Entwicklung von Anwendungen auf Basis von JCR ohne direkte Abhängigkeiten zu Jackrabbit und damit die vollständige Entkopplung der Applikation (z.B. eines Content-Management-Systems) vom darunter liegenden Repository.

## Einsatzszenarien

Jackrabbit kann in verschiedenen Szenarien eingesetzt werden. Einerseits kann es als Backend für ein CMS dienen. Ein bekanntes Beispiel ist hierfür das Open-Source-CMS Magnolia. Das JCR API ist in diesem Kontext besonders hilfreich, wenn eigene Anwendungen auch auf den Content zugreifen sollen, da keine proprietäre Schnittstelle des CMS verwendet werden muss. Ein völlig anderes Szenario für Jackrabbit ist der Einsatz als Content Repository in eigenen Anwendungen. Auch hier ist durch JCR die Abhängigkeit von Jackrabbit im Speziellen sehr gering.

Im Projekt war, wie in vielen Open-Source-lastigen Projekten, ein JBoss Application Server im Einsatz. Dazu soll im Folgenden einmal beispielhaft auf Deployment und Persistenz-Aspekte in dieser JBoss-Umgebung eingegangen werden. Auch wenn dies einen ziemlichen Sprung innerhalb des Artikels darstellt, möchten wir dies tun, da zu diesen The-

Anzeige

### Listing 2

```
// Aktuelle Session holen (hier mit Hilfe des tk4jcr)
Session session = RepositoryHelper.createJBossSession();

// Traversieren des Content-Baumes
Node rootNode = session.getRootNode();
Node anotherNode = rootNode.getNode(
    ("sites/mad-moxx/Impressum"));
Property title = anotherNode.getProperty("title");
System.out.println( title.getStringValue() );

// Anlegen neuer Nodes und Setzen von Properties
anotherNode.setProperty( "author", "Hendrik Beck" );
Node newNode = rootNode.getNode( "sites/mad-moxx" ).
    addNode( "Kontaktseite", "ccs:StaticContent" );
newNode.setProperty( "title", "Kontaktseite" );

// Schliessen der aktuellen Transaktion und Persistieren
// der Änderungen
session.save();
```

men wenig Dokumentation verfügbar ist und reale Projekte früher oder später vor dem Problem des Deployments stehen.

### Deployment

Wird ein CMS mit Jackrabbit als Backend eingesetzt, wird meist ein vorkonfiguriertes Komplettpaket geliefert. Möchte man aber auch mit anderen Anwendungen ebenfalls auf ein JCR Repository (z.B. Jackrabbit) zugreifen, sind mehr Ge-

#### Listing 3

```
<FileSystem class="org.jboss.portal.cms.hibernate
    .HibernateStore">
  <param name="datasource" value="java:jcrDS"/>
  <param name="isolation" value="2"/>
  <param name="auto" value="update"/>
  <param name="provider_class" value="org.hibernate
    .cache.HashtableCacheProvider"/>
  <param name="schemaObjectPrefix" value=
    "RepositoryEntry"/>
</FileSystem>
<PersistenceManager
  class="org.jboss.portal.cms.hibernate.state
    .HibernatePersistenceManager">
  <!--parameter wie beim Filesystem -->
</PersistenceManager>
```

#### Listing 4

```
public void startService() throws Exception {
  String xmlConfAsString = ...;

  // Create repository
  StringReader reader = new StringReader
    (xmlConfAsString);
  InputSource src = new InputSource(reader);
  RepositoryConfig cfg = RepositoryConfig
    .create(src, homeDir);
  repository = RepositoryImpl.create(cfg);

  log.info("Repository '"+ repositoryName + "' created");

  // RMI-Server starten
  InitialContext ctx = new InitialContext();
  RemoteAdapterFactory factory =
    new ServerAdapterFactory();
  RemoteRepository remote =
    factory.getRemoteRepository(repository);
  ctx.bind(remoteRepositoryName, remote);
  rmiRepository = remote;

  // NodeTypes registrieren
  registerCustomNodeTypes(repository);

  // Jackrabbit-Logging einschränken
  Logger.getLogger("org.apache.jackrabbit")
    .setLevel(Level.ERROR);
}
```

danken zum Deployment notwendig. Grundsätzlich gibt es drei Einsatzmöglichkeiten:

- als reine Webanwendung: Das Repository wird in ein war-Archiv integriert und ist Teil dieser Deployment-Einheit (aber auch nur durch diese Anwendung benutzbar). Der Einsatz funktioniert auch ohne Application Server und ist meist der Standard bei integrierten CMS wie Magnolia.
- als Java EE-Ressource im Application Server: Wird das Repository als Ressource betrieben, können alle Anwendungen im gleichen Server problemlos darauf zugreifen (auch ohne Netzwerk-Overhead). Dies ist für alle Umgebungen geeignet, wo aus verschiedenen Anwendungen auf den Content zugegriffen werden soll. Auch wird so das Repository an einer zentralen Stelle konfiguriert, was die eigentliche Anwendung von dieser Aufgabe befreit.
- als dezidiert Repository-Server: Sollen Clients auf das Repository zugreifen, kann ein eigener Server betrieben werden, der Zugriff per RMI, WebDAV oder Ähnliches erlaubt (dies ist nicht durch den JSR 170 spezifiziert). Clients können sowohl dezidierte CMS-Server als auch Swing- beziehungsweise RCP-Anwendungen sein.

### Das Open-Source-CMS Magnolia

Magnolia [3] ist ein Open-Source-CMS des Schweizer Unternehmens Obinary. Es soll, wie CMS anderer Mitbewerber auch, vor allem eine intuitive und einfache Oberfläche zum Bearbeiten von Webinhalten zur Verfügung stellen. Magnolia setzt als Backend komplett auf JCR, wobei im Auslieferungszustand Jackrabbit zum Einsatz kommt. Dazu werden eigene Node Types für Content-Einträge definiert und dann entsprechend gefüllt. Magnolia liegt momentan in Version 3 vor und erfreut sich wachsender Beliebtheit, auch wenn es mit der Dokumentation nicht zum Besten steht. Vor allem eine vereinfachte Integration und die Nutzung des JCR-Standards machen Magnolia aber trotzdem zu einer interessanten Wahl. Auch sorgt das Projekt für Erfahrungswerte und Feedback zu Jackrabbit.

In unserem Projekt haben wir uns für eine Java EE-Ressource, die auch gleichzeitig als RMI-Server fungiert, entschieden, damit das Content Repository zentral von allen Anwendungen verwendet werden kann. Technisch ist so auch ein Rich Client möglich, was die Integration der Content-Pflege in die Oberfläche der Warenwirtschaft ermöglicht. Allerdings hat ein erster kleiner Swing-Prototyp gezeigt, dass die RMI-Implementierung des Jackrabbit-Subprojektes jcr-rmi noch unausgereift ist und erhebliche Performanceprobleme mit sich bringt (wegen zahlreicher Remote-Zugriffe). Der Zugriff von der JSF-Anwendung aus läuft dagegen problemlos.

Die Art des Deployments ist für den Client-Zugriff auf Jackrabbit übrigens unerheblich. Unterschiedlich ist lediglich das Besorgen der benötigten *javax.jcr.Session*, über die dann der weitere Zugriff auf alle JCR-konformen Repository-Implementierungen (wie Jackrabbit) erfolgt.

### Persistenz

Die Persistenz des Contents ist ein sehr wichtiger Aspekt. Die Persistenzstrategie hat nämlich erheblichen Einfluss auf die Robustheit und Skalierbarkeit der entstehenden Anwendung. Jackrabbit unterstützt out of the box verschiedene Persistenzstrategien:

- In-Memory: Content wird lediglich im Speicher gehalten, was meist nur für Tests ausreichend ist.
- ObjectPersistenceManager: Die Daten werden in einem speziellen Binärformat direkt im konfigurierten Dateisystem abgelegt. Dabei ist gute Performance auch bei hohen Datenmengen gegeben. Allerdings gibt es Limitierungen vor allem in der Cluster-Fähigkeit der Lösung (außer man setzt auf ein im Netzwerk zur Verfügung stehendes Dateisystem).
- XMLPersistenceManager: Content wird in der Form von XML-Dateien im Dateisystem abgelegt. Dies kombiniert die Nachteile des ObjectPersistenceManager mit schlechter Performance, ist aber vor allem zu Testzwecken durchaus hilfreich.

- SimpleDbPersistenceManager: Content wird in einer relationalen Datenbank über JDBC abgelegt. Dies ist eine gängige Variante, die auch eine gute Performance erzielt. Bei geeigneter, aber dann sehr komplexer Konfiguration kann sie eigentlich allen Ansprüchen genügen.

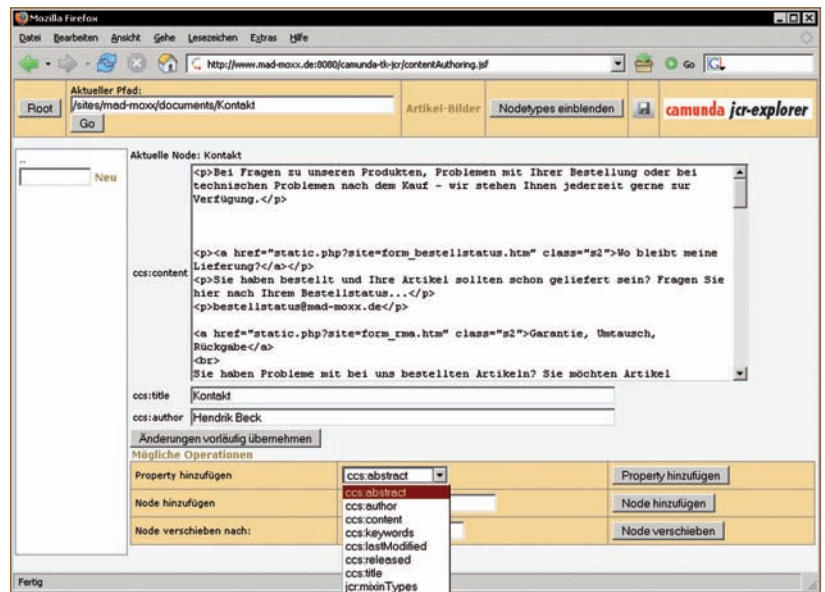
Die Unterstützung ist in kleineren Anwendungen ausreichend, wird aber schnell problematisch, wenn es um Caching, Clustering oder auch die Nutzung des Java Transaction API geht. Daher wird oft der Wunsch laut, eine Hibernate-Implementierung zu verwenden, da das Hibernate-Projekt [7] ausgereifte Lösungen für diese Probleme mitbringt. Eine solche Lösung findet sich im JBoss-Portal-Projekt [5]. Die Übernahme ins eigene Projekt ist recht einfach: Man muss den entsprechenden PersistenceManager konfigurieren (Listing 3) und die JBoss Portal Libraries verfügbar machen. Hibernate kann beim Hochfahren des Repositorys übrigens die benötigten Tabellen selbst anlegen (was über den Parameter „auto“ gesteuert wird).

Jackrabbit erstellt im Hintergrund einen Lucene-Index [8], damit der Content entsprechend durchsucht werden kann. Dieser Index kann momentan leider nur im Dateisystem abgelegt werden. Da das Repository diesen Index selbstständig aufbaut, ist es allerdings auch unproblematisch, wenn jeder Server eines Clusters seinen eigenen Index besitzt und aufbaut, wobei die Autoren diesbezüglich nicht über große Erfahrung verfügen.

### Betrieb als JBoss Service

Kommt nun, wie in unserem Fall, ein JBoss zum Einsatz, kann ein JBoss Service entwickelt werden, der das Repository hochfährt, als Ressource bereitstellt, einen RMI-Server startet und eigene NodeTypes (Listing 1) registriert, wenn sie noch nicht vorhanden sind. Grundlage dieses Service war in unserem Fall Code aus dem JBoss-Portal-Projekt. Die komplette Konfiguration des Service würde den Artikel sprengen, ist aber im Internet zu finden [4]. Die Struktur der Jackrabbit-Konfiguration ist auch auf der Jackrabbit-Website gut beschrieben [2]. Das Star-

Abb. 2: Screenshot des JCR-Explorers



ten des Repositorys kann über eine JBoss MBean erledigt werden, die in Listing 4 angeschildert ist.

### tk4jcr

Im Projekt haben wir JCR-spezifische Funktionalität in ein eigenes Subprojekt ausgelagert und dieses Open Source zur Verfügung gestellt. Das toolkit for jcr (tk4jcr [4]) enthält neben der angesprochenen MBean, der prototypischen Swing-Oberfläche vor allem auch eine generische JSF-Oberfläche zum Zugriff auf JCR (Abb. 2). Die Oberfläche kann frei im Content-Baum navigieren und Nodes und Properties entsprechend den möglichen NodeTypes anlegen. Dabei ist man bei der Bearbeitung des Contents ansonsten völlig frei, was den Explorer von CMS-Oberflächen unterscheidet (er soll auch keine Konkurrenz darstellen).

### Fazit

Der JCR 170 ist eine interessante Spezifikation, die den Wildwuchs im Content-Bereich vereinheitlichen könnte. Mit Jackrabbit steht bereits eine funktionierende und einsatzbereite Open-Source-Implementierung bereit. Der entwickelte Webshop sowie der JCR-Explorer zur Content-Pflege laufen seit mehreren Monaten produktiv und ohne Probleme, weder Last noch Performance machen sich negativ bemerkbar. Somit steht endlich eine geeignete Schnittstelle zwischen

komplexen Unternehmensanwendungen und Content-Management-Systemen zur Verfügung, die die Integration erleichtern kann und nach unserer Einschätzung in Zukunft eine bedeutende Rolle spielen wird.



**Bernd Rücker** ist Softwarearchitekt und Geschäftsführer bei der camunda GmbH. Er verfügt über mehrjährige Projekterfahrung als Architekt und Entwickler im Umfeld von Geschäftssoftware in Java EE. Er ist Autor einiger Fachartikel, Co-Autor eines EJB 3-Buches sowie Committer des JBoss jBPM-Projektes. Kontakt: [bernd.ruecker@camunda.com](mailto:bernd.ruecker@camunda.com).



**Hendrik Beck** ist Softwareentwickler bei der camunda GmbH und verfügt über mehrjährige Erfahrungen in der Integration Java EE-basierter Technologien in webbasierten Anwendungen. Im Rahmen seines Master-Studiums ist er derzeit für Konzeption und Realisierung eines Projekts der An Giang University in Vietnam verantwortlich, das u.a. auf Jackrabbit als Content Repository setzt. Kontakt: [hendrik.beck@camunda.com](mailto:hendrik.beck@camunda.com).

### Links & Literatur

- [1] JSR 170: [jcp.org/en/jsr/detail?id=170](http://jcp.org/en/jsr/detail?id=170)
- [2] Jackrabbit: [jackrabbit.apache.org](http://jackrabbit.apache.org)
- [3] Magnolia: [www.magnolia.info](http://www.magnolia.info)
- [4] tk4jcr: [www.camunda.com/toolkit\\_for\\_jcr/camunda\\_toolkit\\_for\\_jcr.html](http://www.camunda.com/toolkit_for_jcr/camunda_toolkit_for_jcr.html)
- [5] JBoss Portal: [www.jboss.org/products/jbossportal](http://www.jboss.org/products/jbossportal)
- [6] Day Software: [www.day.com](http://www.day.com)
- [7] Hibernate: [www.hibernate.org](http://www.hibernate.org)
- [8] Lucene: [lucene.apache.org](http://lucene.apache.org)
- [9] Gerd Handke: Positive Aussichten. Java Content Repository. Der Standard nimmt Konturen an, in *Java Magazin* 3.2005
- [10] JSR 283: [jcp.org/en/jsr/detail?id=283](http://jcp.org/en/jsr/detail?id=283)