Java Content Repository

Tom Wheeler

Senior Software Engineer
Object Computing, Inc.

What Lies Ahead?

- An introduction
  - What is the Java Content Repository?
- Basic concepts
  - How does it work?
- Implementation
  - How do I use the API?
- Putting it all together
  - We'll dissect an application that uses JCR

What is the JCR?

- According to the JSR-170 request:
  - “A Content Repository is a high-level information management system that is a superset of traditional data repositories”
- Key points
  - JCR defines a standard way to access data
  - JCR is "content-centric"
    - Developers focus on structure
    - Not on how or where the data is actually stored

Motivation for Proposing JCR

- Many proprietary content repos exist
  - API for interaction is vendor-specific
  - There should be a standard API
- The standard API should be
  - Independent of arch., data source or protocol
  - Easy for a programmer to use
  - Relatively easy for vendors to implement

JCR, the Overlooked API

- JCR seems to suffer from bad marketing
- It is often seen as an API only for CMS
- It's a uniform interface for data access
  - As suitable for accessing a single String...
  - As it is for accessing a 10GB binary object
  - Abstracts data**source**, not just the data**base**

Application Performance

- Data ***storage*** is typically configurable *
  - For example, filesystem, database or XML
- Exact needs will vary by application
  - Amount, structure and type of data used
- To boost performance of your application
  - Can switch/tune storage scheme
  - Could also switch repository vendors

* *available data storage schemes and configuration details vary by implementation*

Why ***Shouldn't*** You Use JCR?

- Can be clumsy for some tasks
  - Maybe a poor replacement for properties files
- Extra dependencies
  - Not part of J2SE
  - Potentially a lot of extra JARs in your project
- Non-standard repository management
  - Must use vendor-specific APIS in JCR 1.0
  - This is much improved in JCR 2.0
    - But tools are still implementation-specific

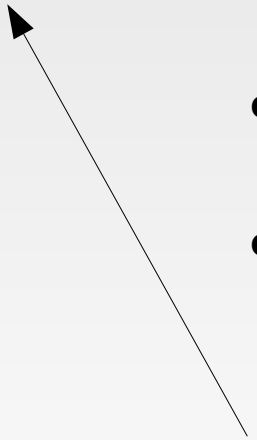
Which Applications Use JCR?

- Several portals and CMSs
  - Liferay, JBoss, Sun OpenPortal...
  - Magnolia, JLibrary, Archimede, OpenKM...
- Other interesting projects using JCR
  - InfoQ Web site
  - Freecaster.tv
  - ASC PowerLender (loan origination system)
  - Informa MapOfMedicine (clinical info app)

There are probably lots of internal corporate apps we don't know about

Who's Behind JCR?

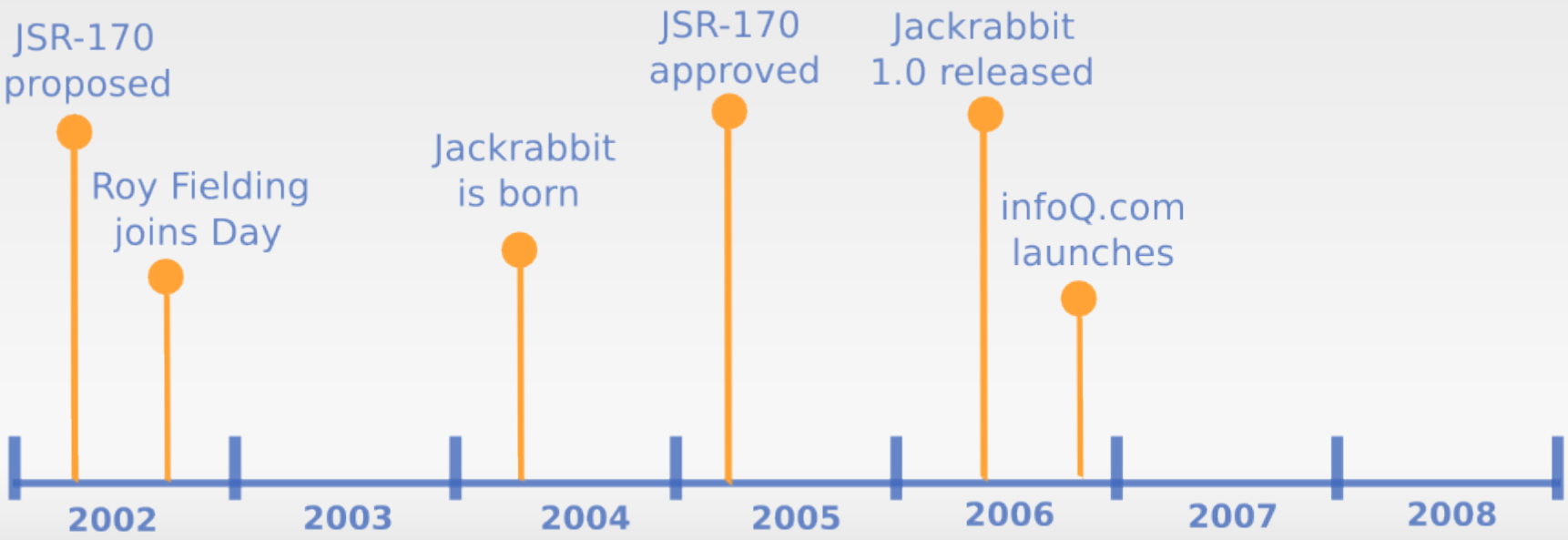
- ASF (Apache)
- ATG
- BEA
- **Day Software**
- Documentum
- HP
- IBM
- Interwoven
- Oracle
- SAP
- Sun
- Vignette



Spec. Lead

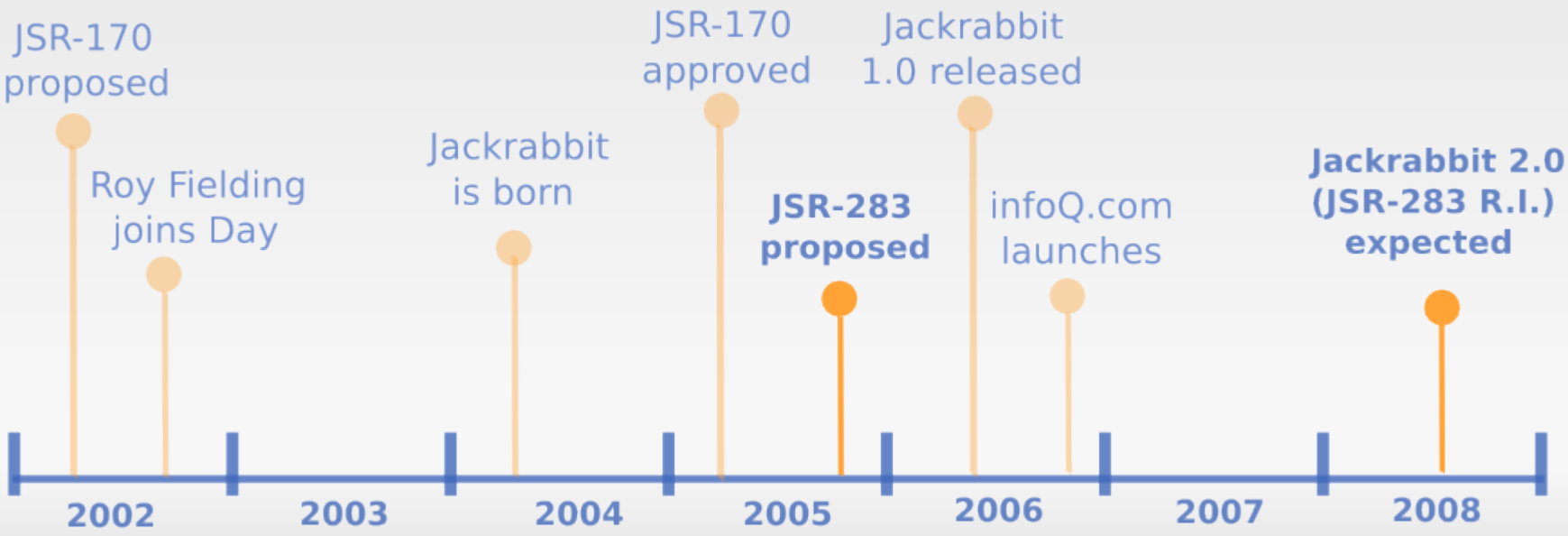
Where Did it Come From?

- JSR-170 was proposed by Day Software
  - 21 companies represented on expert group
  - Won unanimous final approval



Where is it Going?

- JSR-170 was for JCR 1.0
- JCR 2.0 will be covered by JSR-283
  - More standardized management APIs



JCR Implementation Levels

- JCR defines two conformance levels
  - Level 1: Read-only (one-way)
  - Level 2: Read-write (bi-directional)
- Plus optional features beyond these
- This helps legacy (CMS) vendors
  - By giving them a path to compliance
- Potentially cuts costs for consumers
  - Some apps would only ever need Level I

JCR Implementation Level 1

- Level 1: Read-only (one-way)
  - Data access using any of these methods
    - Node traversal
    - Direct access
    - Query using XPath
  - Handles structured data only
  - Can export entire repository to XML
    - But cannot necessarily import one!

JCR Implementation Level 2

- Level 2: Read-write (bi-directional)
  - Includes all level 1 features
  - Import from XML
  - Add/update/delete data
  - Define/assign custom node types
  - Handles structured and unstructured data
  - Referential integrity

JCR Optional Features

- Locking
- Transaction management (JTA)
- Observation
  - Event notification for repository changes
- Versioning
  - Can retrieve previous revisions of data
- Query by SQL, in addition to XPath

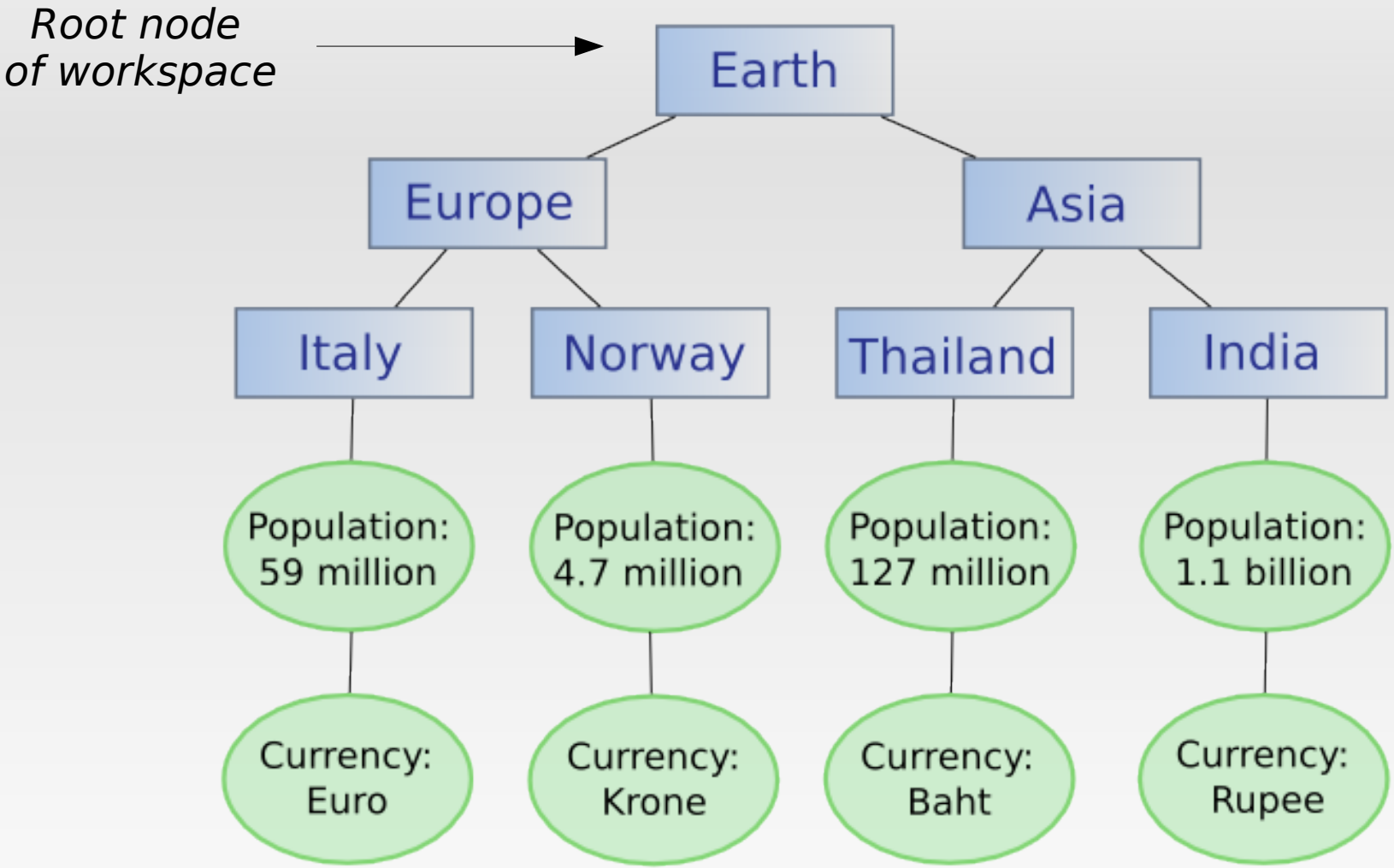
Available Implementations

- Apache Jackrabbit
  - Open source; reference implementation
- Alfresco
  - Open source; highly regarded
- eXo Platform
  - Open source
- Day Software CRX
  - Commercial, from spec. lead's company

API Overview

- JCR API defined in the `javax.jcr` package
  - 15 interfaces
  - two classes
  - 14 exceptions
- Avoid coding to implementation classes
  - Use `javax.jcr.*`
  - Don't use `org.apache.jackrabbit.core.*`
  - Can't *always* be avoided in JSR-170

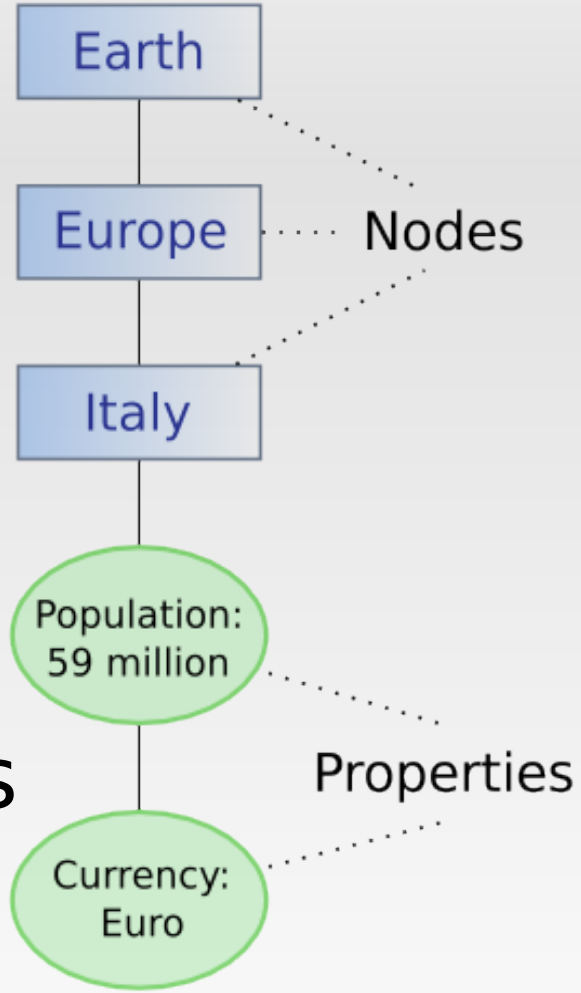
Concepts: Data is Hierarchical



NOTE: Any node can hold properties, not just a leaf node

Concepts: Nodes and Properties

- Nodes organize the data
- Properties ***store*** the data
- Think of a UNIX filesystem *
  - Roots and paths
  - Nodes are like directories
  - Properties are like files
- Any node can have properties
  - Not just leaf nodes



* *conceptual hierarchy does not match actual storage format on disk*

Concepts: Node Types

- Nodes, like Java objects, have a type
  - Types specify what data is allowed
- There are many primary types
  - All inherit from a base type (`nt:base`)
  - `nt:unstructured` is the most flexible
- You can also define custom types *
  - Actually *needing* to do this is less common than you'd think

* *Implementation-specific in JSR-170, but standardized in JSR-283*

Concepts: Mixins

- A node can only have one primary type
- But can have multiple “mixin” types
  - Added via `Node.addMixin(String name)`
  - Common ones include:
    - `mix:lockable` supports locking
    - `mix:referenceable` supports UUIDs
    - `mix:versionable` supports versioning

Concepts: Workspace/Session

- `javax.jcr.Session`
  - Provides access to repository content
  - Provides access to root node
  - Allows access of node by path or UUID
- `javax.jcr.Workspace`
  - Represents a view of the repository
  - 1:1 mapping to a Session object
  - Accessed via `Session.getWorkspace()`

Access Control

- You authenticate using some variation of
  - `Repository.login(Credentials cred)`
- Authentication scheme is pluggable
  - But JAAS implementation is typical default
- Access control is implementation-specific
  - Jackrabbit uses `AccessManager`
    - Built-in `SimpleAccessManager` (3 access levels)
    - Can also plug-in custom implementations

Basic Steps for Using JCR

- *Configure repository*
- *Start or create repository*
- Log into repository
- Get a Session
- Work with data (add/delete/etc.)
- Log out of session
- *Shut down repository*

these are implementation-specific, but they happen infrequently

Using JCR: Configuration

- Implementation-specific
- For Jackrabbit, a big ugly XML file
- Specify repository options
  - Access control
  - Repository data storage location
  - Persistence Manager (DB, XML, FS, etc.)
  - Search and indexing options

Using JCR: Starting a Repository

- If first usage, must create repo first
  - Otherwise, you can start an existing repo
- Details are implementation-specific
  - But the simplest case for Jackrabbit:

```
// TransientRepository is mainly used for development.
// It starts up when the first session is opened,
// and shuts down when the last session is closed.
Repository repository = new TransientRepository();
```

- In production, you'll probably use JNDI

Using JCR: Log in/Get Session

- Authenticate using some form of
  - `Repository.login(Credentials cred)`
  - Exact type of `Credentials` may vary
  - Only `SimpleCredentials` is defined by API
  - There are four overloaded `login` methods
    - Can specify credentials – or not
    - Can specify workspace – or not
- Return value is a `Session` object

Using JCR: Adding Data

```
try {
    Node root = session.getRootNode();

    Node earthNode = root.addNode("Earth");
    Node europeNode = earthNode.addNode("Europe");
    Node italyNode = europeNode.addNode("Italy");

    italyNode.setProperty("Population", 59000000L);
    italyNode.setProperty("Currency", "Euro");

    session.save();
} catch (RepositoryException e) {
    // TODO: handle the exception
}
```

Using JCR: Accessing Data

- Data is contained in properties
  - To get a property, you must first get the node
- There are three ways to access a node
  - Direct access
  - Traversal from another node
  - From the result of a query
- Examples of each coming right up...

Using JCR: Traversing Data

```
try {
    Node root = session.getRootNode();

    // walk down from the root
    Node earthNode = root.getNode("Earth");
    Node europeNode = earthNode.getNode("Europe");
    Node italyNode = europeNode.getNode("Italy");

    Property currProp = italyNode.getProperty("Currency");

    String currName = currProp.getString();
    System.out.println("Italy's Currency: " + currName);
} catch (RepositoryException e) {
    // TODO: handle the exception
}
```

Using JCR: Direct Data Access

```
try {
    String path = "/Earth/Europe/Italy/Population";
    Property popProp = (Property) session.getItem(path);

    long population = popProp.getLong();
    System.out.println("Italy's population: " + population);

} catch (RepositoryException e) {
    // TODO: handle the exception
}
```

can also use `session.getNodeByUUID` if using referenceable mixin

Using JCR: Updating Data

```
try {

    // retrieve italyNode as before
    italyNode.setProperty("Population", 9000000L);
    italyNode.setProperty("Currency", "Lira");

    session.save();
} catch (RepositoryException e) {
    // TODO: handle the exception
}
```

Using JCR: XPath Queries

```
try {
    Workspace ws = session.getWorkspace();
    QueryManager qm = ws.getQueryManager();

    // get all countries trading with the Euro
    String qs = "//*[@Currency='Euro']";
    Query qry = qm.createQuery(qs, Query.XPATH);

    QueryResult res = qry.execute();
    NodeIterator resIter = res.getNodes();

    while (resIter.hasNext()) {
        // do something with found nodes
        Node node = resIter.nextNode();
    }
} catch (RepositoryException e) {
    // TODO: handle the exception
}
```

Using JCR: SQL Queries

```
try {
    QueryManager qm = session.getWorkspace().getQueryManager();

    String qs = "select Population, Currency "
              + "FROM nt:unstructured WHERE "
              + "Currency='Euro'";

    Query qry = qm.createQuery(qs, Query.SQL);
    QueryResult res = qry.execute();
    RowIterator resIter = res.getRows();

    while (resIter.hasNext()) {
        Row row = resIter.nextRow();
        long pop = row.getValue("Population").getLong();

        // do something with the data
    }
} catch (RepositoryException e) {
    // TODO: handle the exception
}
```

Using JCR: Deleting Data

```
try {
    String path = "/Earth/Europe/Italy";
    Node italyNode = (Node) session.getItem(path);

    // to delete a property...
    Property currProp = italyNode.getProperty("Currency");
    currProp.remove();

    // to delete a node...
    italyNode.remove();

    session.save();
} catch (RepositoryException e) {
    // TODO: handle the exception
}
```

Using JCR: Repository Shutdown

- Details are implementation-specific
  - But for Jackrabbit

```
JackrabbitRepository jr =
        (JackrabbitRepository) myRepository;

jr.shutdown();
```

For More Information

- JSR 170 (Original JCR Specification)
  - http://jcp.org/en/jsr/detail?id=170
- JSR 283 Site (JCR 2.0 Specification)
  - http://jcp.org/en/jsr/detail?id=283

For More Information

- Jackrabbit (Open Source JCR / Ref. Impl.)
  - http://jackrabbit.apache.org/
- Alfresco Site (Open Source JCR)
  - http://www.alfresco.com/

For More Information

- eXo Platform (Open Source JCR)
  - http://wiki.exoplatform.com/
- Day Software, A.G. (Commercial JCR)
  - http://www.day.com/

Conclusion

- The Java Content Repository...
  - is a powerful data access API
    - Is a Java standard
    - Is easy to use and understand
    - Can potentially replace JDBC, XML, etc.
  - Has strong open source support
  - Is worth considering for your next application