JBoss DNA

02 Mai 2009

Serge Emmanuel Pagop
IT-Architect & Senior Consultant
innoQ Deutschland GmbH

serge.pagop@innoq.com

Wo initiates JBoss DNA?

Randall Hauch has been working with metadata and repositories for most of his career. He is the project lead for JBoss DNA and a Principal Software Engineer at JBoss, a division of Red Hat.

Agenda

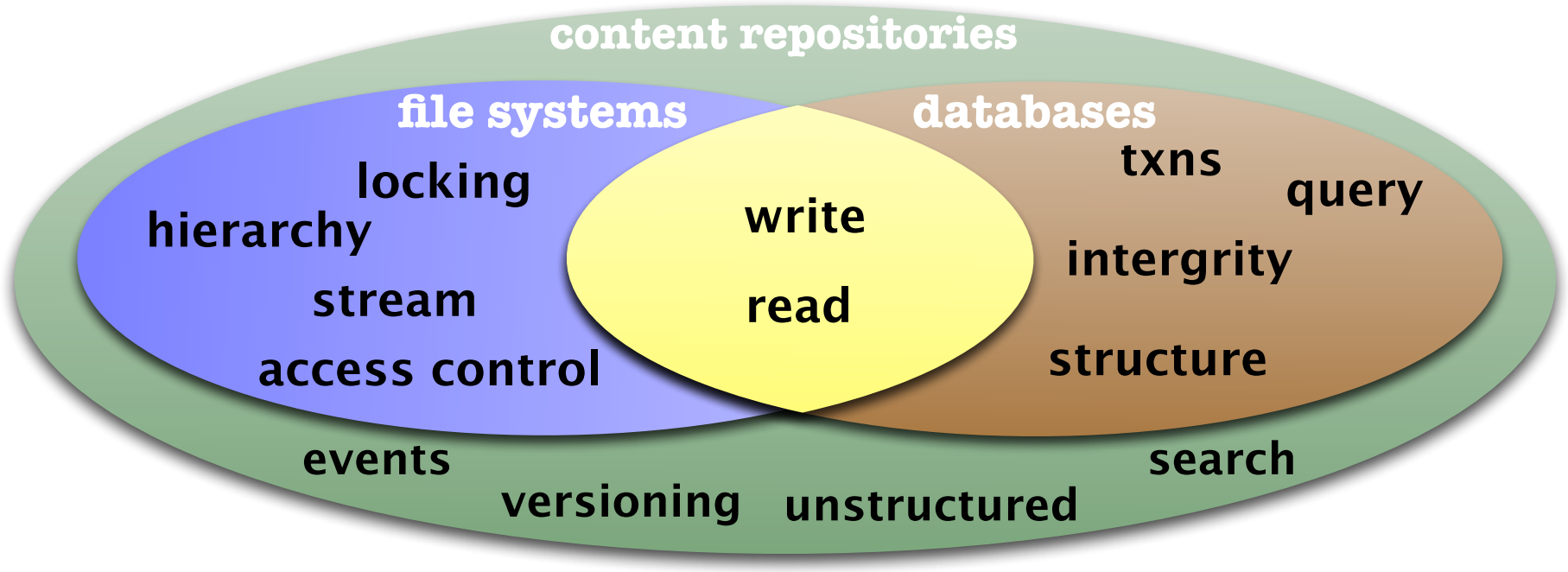
- JCR and JBoss DNA JCR in general
- JBoss DNA JCR sample code examples
- JBoss DNA architecture in a standalone & JEE App.
- JBoss DNA and Federation
- JBoss DNA Connector framework
- JBoss DNA Sequencing

Content Repositories

- **Provide**
  - Hierarchical graph-based storage
  - Flexible/extensible schema (as needed)
  - Versioning, events, and access control
  - Search and query
  - Metadata
  - Multiple persistence choices
- **Used for**
  - websites and content-based applications
  - content management
  - document storage
  - multi-media files

JSR-170 (and JSR-283)

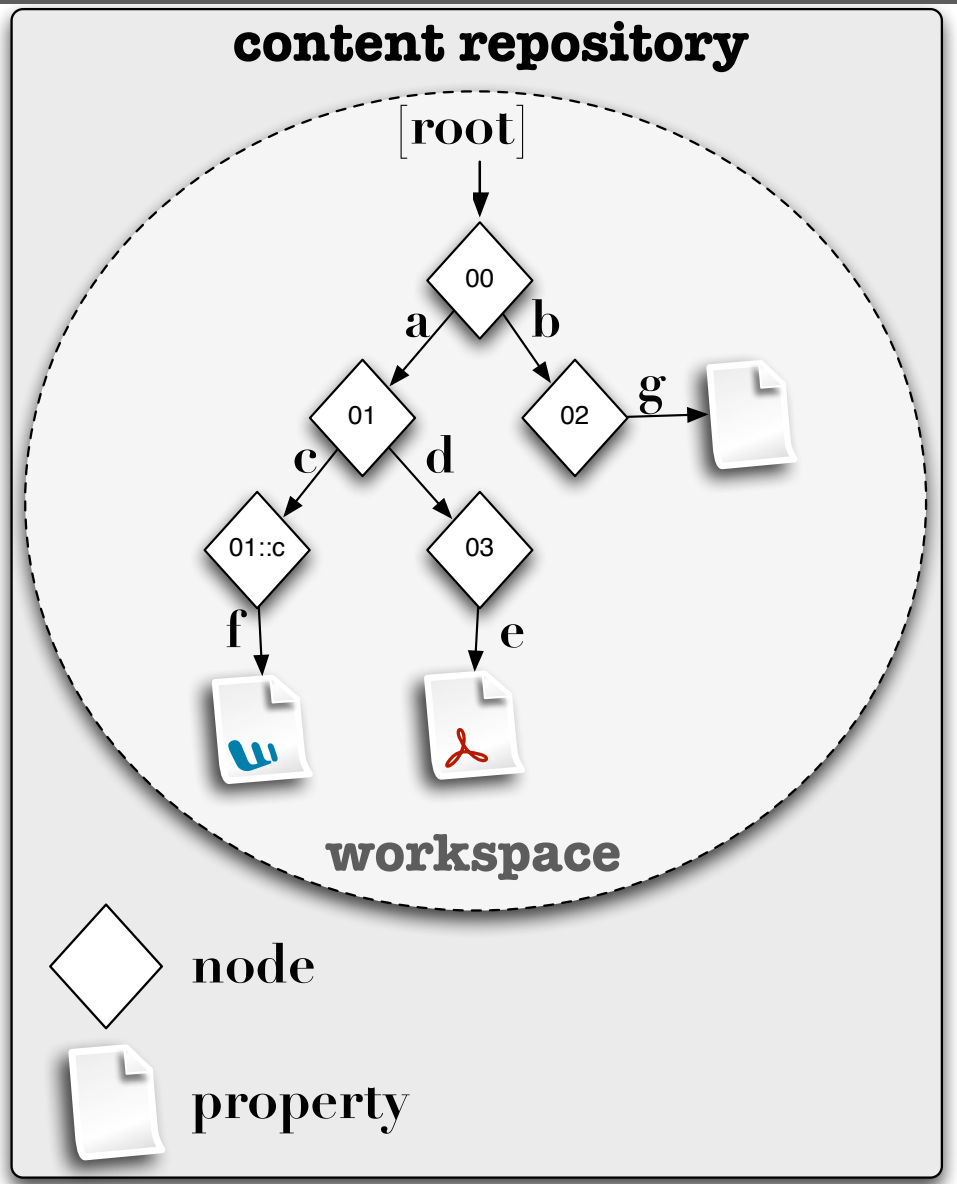
- **Standard Java API for content repositories**
  - "Content Repository API for Java" (a.k.a. "JCR")
    - javax.jcr
- **Access content while hiding persistence layer**
- **Offer best features of different layers**



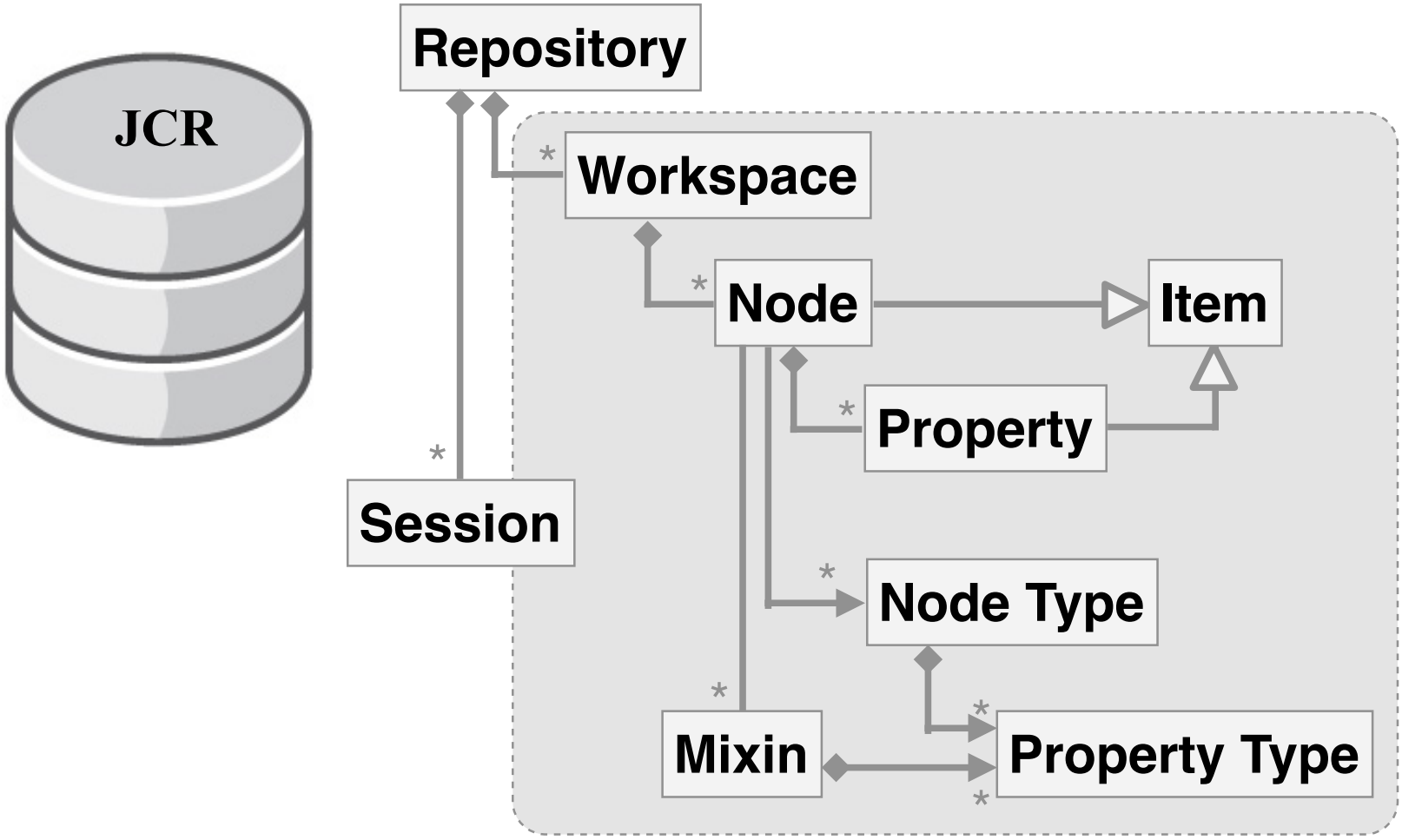
Who uses JCR?

... and many more

Repository model



Primary JCR interfaces



More JCR concepts

- **Nodes**
  - have names, including same-name siblings
  - are referenced by path of names from root
- **Namespaces**
  - isolate names of nodes, properties, node types, and mixins
- **Events**
  - allow sessions to observe changes in content
  - can be filtered by change type or location
- **Versioning**
  - of nodes and subgraphs as they change
  - policy defined on each node by adding mixin (“`mix:versionable`” or “`mix:simpleVersionable`”)

More JCR concepts

- **Searching**
  - enables full-text search
  - can use special functions and XPath-like criteria
- **Querying**
  - is via SQL-like grammar
  - will change in JCR 2.0 to allow other grammars
- **Security**
  - can leverage JAAS
  - will improve in JCR 2.0 with better access controls
- **Transactions**
  - relies upon JTA and JTS

DNA-JCR supported features (L1)

- **Accessing the repository**
  - ✓ JAAS authentication (IDTrust)
- **Namespaces**
  - ✓ Session Remapping
- **Reading content**
  - ✓ Traversal Access
  - ✓ Direct Access
  - ✓ Same-Name Siblings
  - ✓ Multi-Value Properties
  - ✓ All Property Types Supported
  - ✓ Property Type Conversion
- **Exporting content**
  - ✓ System view export to XML
  - ✓ Document view export to XML
- **Node Types**
  - ✓ Inheritance Among NodeTypes
  - ✓ Discovering available NodeTypes
  - ✓ Discovering NodeTypes of a Node
  - ✓ Discovering NodeType definition
  - ✓ Property Constraints
  - ✓ Automatic Item Creation
  - ✓ Predefined NodeTypes
  - ✓ Custom NodeType Registration Namespaces
  - ✓ Session Remapping

DNA-JCR supported features (L2)

- **Writing content**
  - ✓ Create/Update/Delete Nodes
  - ✓ Create/Update/Delete Properties (through parent nodes)
  - ✓ Moving (but not copying yet)
  - ✓ Adding/Removing Mixins
- **Importing content**
  - ✓ System View Import
  - ✓ Document View Import
- **Workspace**
  - ✓ Create/Delete (delete is more DNA specific)
  - ✓ Clone Workspace (DNA specific)

Create a Repository

Create an in-memory repository source ...

```
InMemoryRepositorySource source = ... ; // specific impl.
```

Create a connection factory ...

```
RepositoryConnectoryFactory factory = ... ;  // specific impl.
```

Set up the execution context ...

```
ExecutionContext context = ... ;  // specific impl.
```

Make sure the path to the namespaces exists ...

```
Graph graph =  Graph.create(source, context)  // specific impl.
graph.create("/jcr:system").and.create("/jcr:system/dna:namespaces");
```

Create a JCR repository ...

```
Repository repository =
   new JcrRepository(context, factory,"My Repos.") ...  //  specific impl.
```

Commentary

- Getting a hold of Repository instances is implementation dependent

Create and close a Session

Ok, we already get the repository instance ...

```
Repository repository = ...  // okay, it's impl. dependent
```

Create, close a Session

```
Credentials credentials = ... ; // JAAS Credentials
String workspaceName = "My Repository";
Session session = repository.login(credentials, workspaceName);
try {
    // Use the session to access the repository content
} finally {
    if ( session != null ) session.logout();
}
```

Commentary

- Other credential options (e.g., using JAAS) are implementation dependent
- Creating sessions (or "connections") requires knowledge of credentials, making pooling difficult

Work with content

Getting nodes by path

```
Node root = session.getRootNode();
Node node = root.getNode("autos/sports cars/Toyota/2008/
Prius");
```

Getting the children of a node

```
for (NodeIterator iter = node.getNodes(); iter.hasNext();) {
    Node child = (Node) iter.nextNode();
    // Do something fun
}
```

Creating nodes

```
Node ford = root.addNode("autos/sports cars/Ford");
Node ford08 = ford.addNode("2008");
Node volt = root.addNode("autos/sports cars/
Chevy").addNode("2010").addNode("Volt","car");
```

Mixins

```
ford08.addMixin("car:year");
ford08.removeMixin("car:year");
```

Commentary

- Unfortunately JCR doesn't use generics
- Cannot create node if parent doesn't exist

Work with content

Reading a property

```
Property property = node.getProperty("engineSize");
String[] engineSize = null;
// Must call either 'getValue()' or 'getValues()' depending upon # of values!
if (property.getDefinition().isMultiple()) {
    Value[] jcrValues = property.getValues();
    engineSize = new String[jcrValues.length];
    for (int i = 0; i < jcrValues.length; i++) {
        engineSize[i] = jcrValues[i].getString();
    }
} else {
    engineSize = new String[] {property.getValue().getString()};
}
```

Setting a property

```
Property property = node.getProperty("engineSize");
property.setValue("V4 Hybrid");
// Or set directly via the node
node.setProperty("mpgCity",48);
```

Commentary

- Getting the property values could be ***way*** less verbose
- But **Value** does have methods to get the values in the Java types I want

Work with content (continued)

Reading all properties

```
for (PropertyIterator iter = node.getProperties(); iter.hasNext();) {
    Property property = (Property) iter.nextProperty();
    // Same as before
}
```

Reading some properties

```
for (PropertyIterator iter = node.getProperties("jcr:*|*Mpg"); iter.hasNext();) {
    Property property = iter.nextProperty();
    // Same as before
}
```

Visiting nodes and properties

```
node.accept( new ItemVisitor() {
    public void visit(Property property) throws RepositoryException {
        // Do something with the property
    }
    public void visit(Node node) throws RepositoryException {
        // Do something with the node
    }
});
```

Commentary

- Again, generic iterators would simplify things

David’s rules for content modeling

http://wiki.apache.org/jackrabbit/DavidsModel

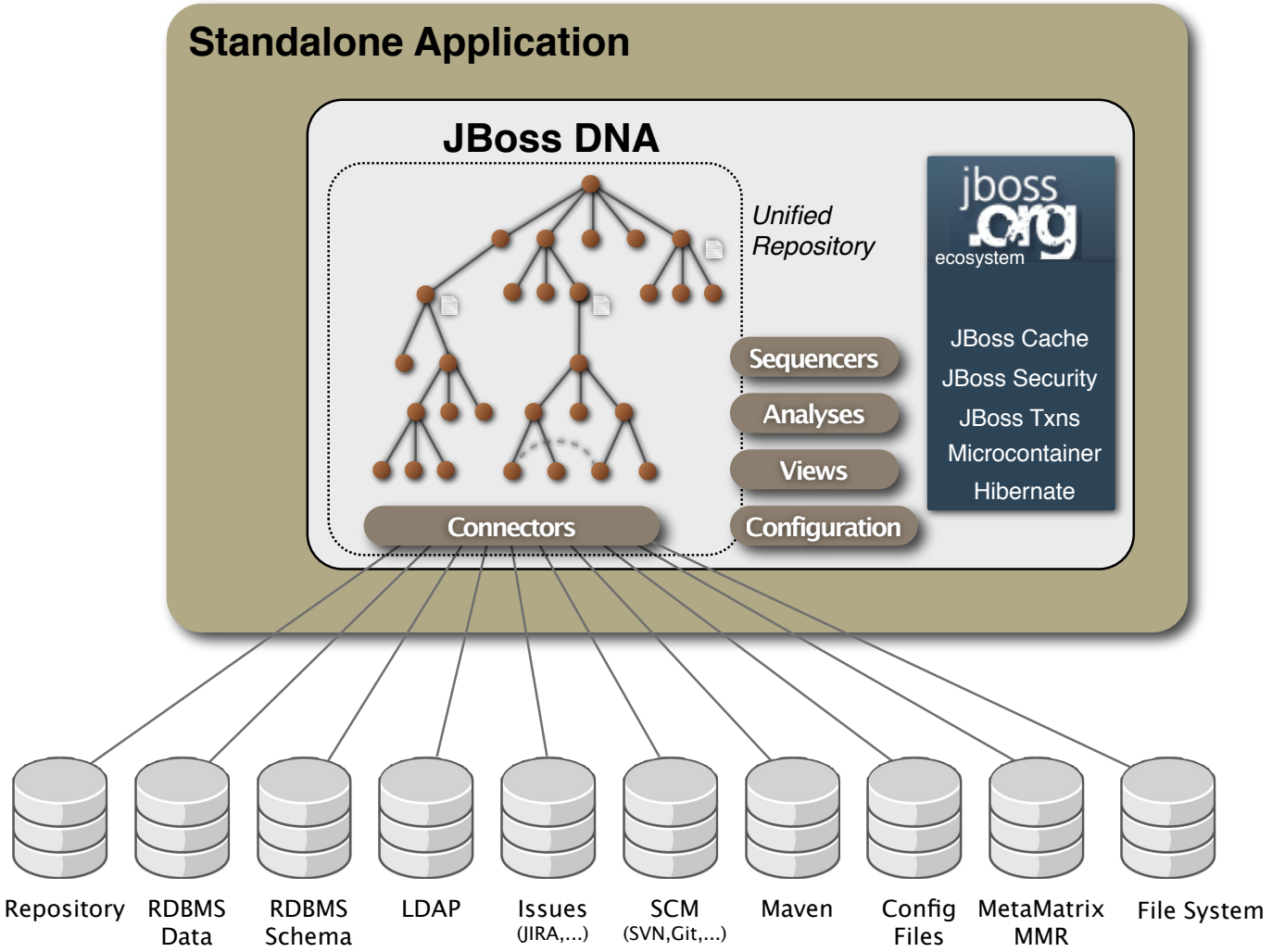
- Rule #1: Data First, Structure Later. Maybe.
- Rule #2: Drive the content hierarchy, don't let it happen.
- Rule #3: Workspaces are for clone(), merge() and update().
- Rule #4: Beware of Same Name Siblings.
- Rule #5: References considered harmful.
- Rule #6: Files are Files are Files.
- Rule #7: ID's are evil.

JBoss DNA ecosystem

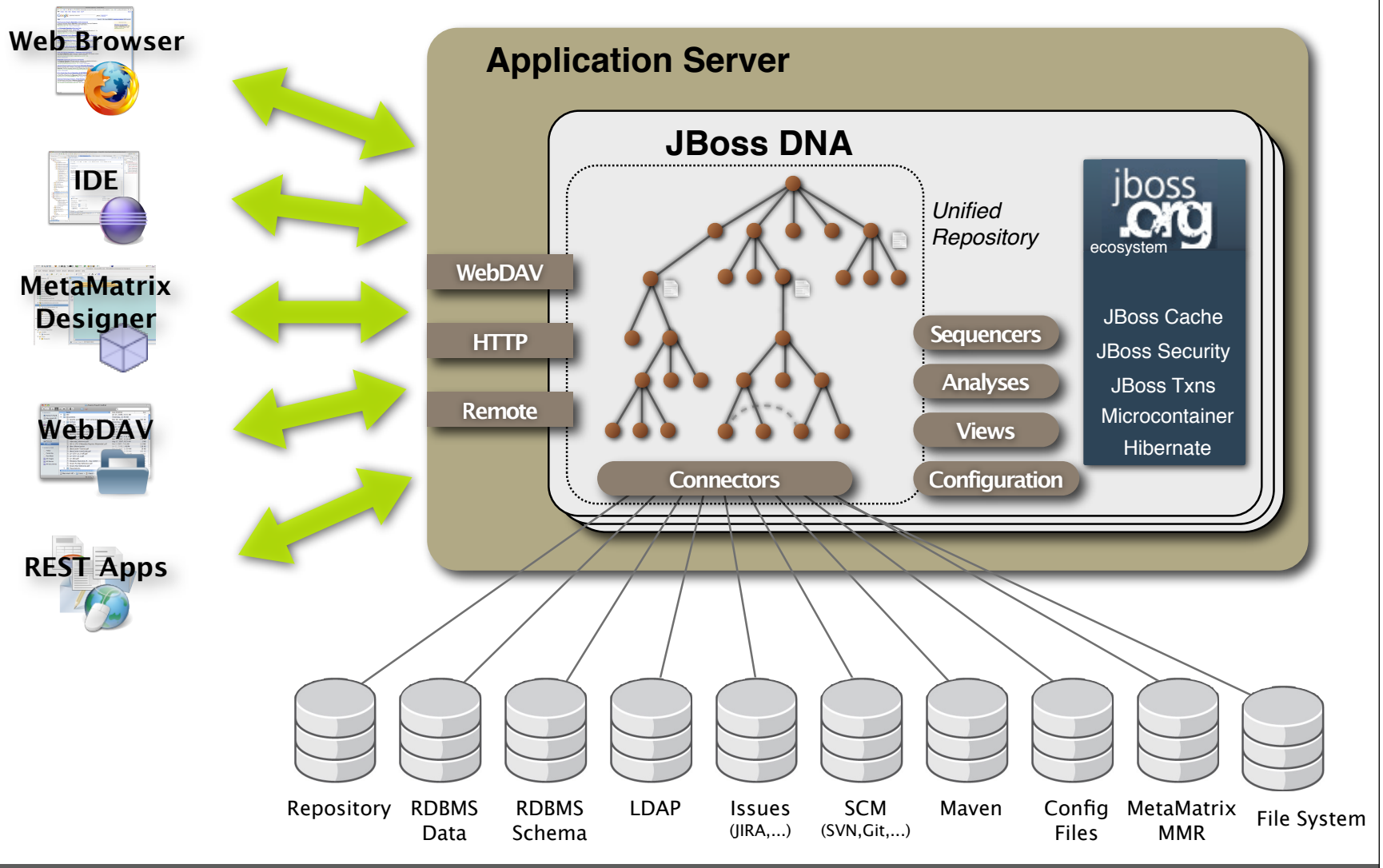
JBoss DNA

- **New JCR implementation that**
  - looks and behaves like a regular JCR repository
  - unifies content from a variety of systems
  - extracts the most benefit from the content
- **So what's different?**
  - where the content is stored (lots of places)
  - federation!
  - use of existing best-of-breed technology
    - cache, clustering, persistence, deployment
  - enterprise-class repositories
  - micro-repository for embedded use

JBoss DNA architecture



JBoss DNA architecture



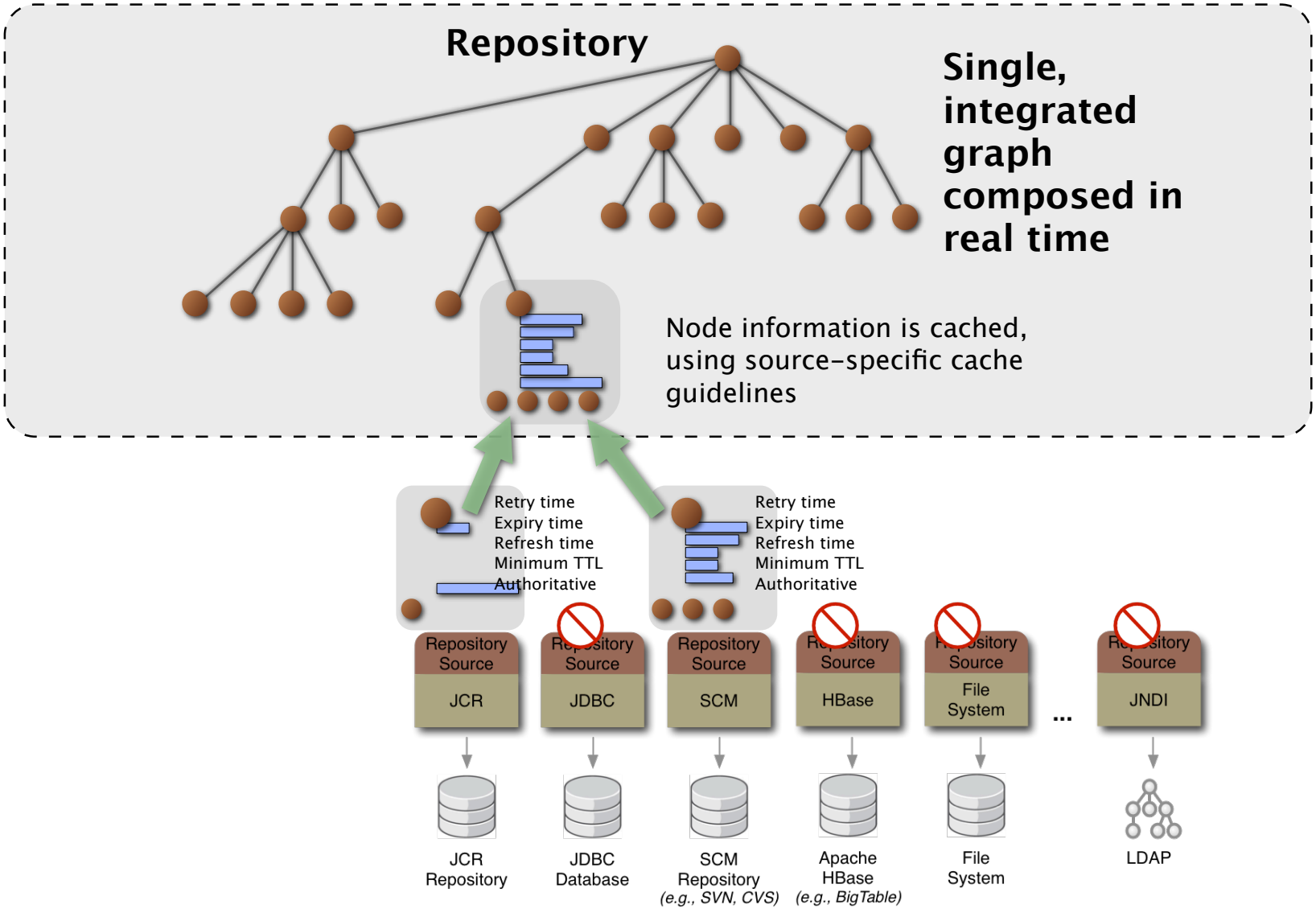
Configuring JBoss DNA

- **Configuration repository**
  - contains content describing the components
  - observed so updates are reflected in components
  - just a regular repository
- **Enables clustering**
  - Processes use the same configuration repository
  - Add and remove processes as required
- **Repository management**
  - One repository containing configurations for multiple repositories
  - Manage configuration simply as content (edit, copy, etc.)
  - Versioning supports rolling back to previous configuration

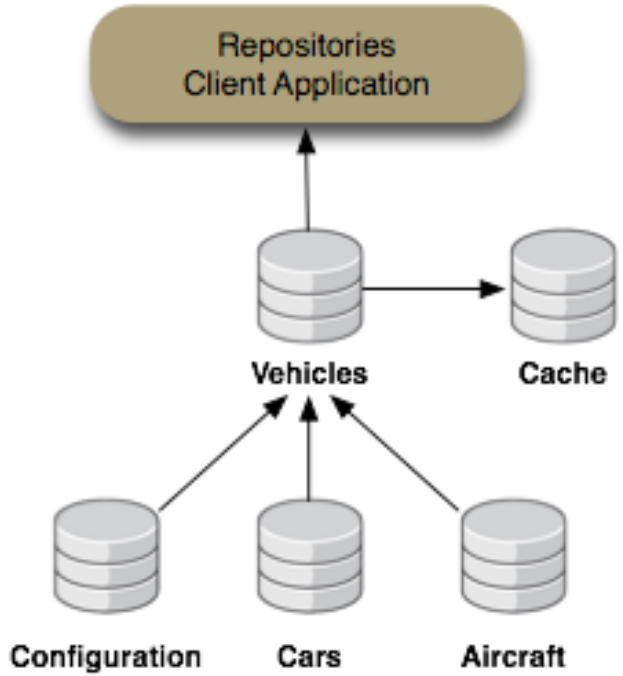
Federation Use Cases

- **Unify content in multiple external systems**
  - Content still managed in current *system of record*
  - Benefits of a single repository
- **Local caching repository**
  - Remote repository is the master
  - Application wants a local copy/cache of data it uses
- **Images, large files for web content**
  - Store in JCR (versioning, events, auditing, access control)
  - Copy latest to file system (direct access by web server)
- **Segregating data by type**
  - Images in one repository, user info in another, etc.
  - Application still uses one repository
- **Segregating data by region/owner**
  - Multiple repositories structured similarly
  - Each region owns its data, but reads other regions' data

Federation and integration



Federated Vehicles content



Example: Federation configuration

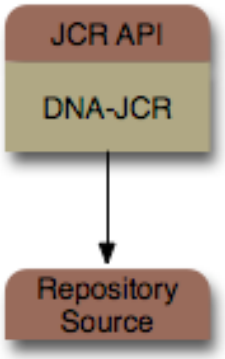
```
<jcr:system ...>
    </dna:namespaces>
    <!-- sources from which content ... -->
    <dna:sources jcr:primaryType="nt:unstructured">
        <dna:source jcr:name="A" dna:name="Cars".../>
        <dna:source jcr:name="B" dna:name="Aircraft" ... />
        <dna:source jcr:name="C" dna:name="Vehicles" ..../>
        <dna:source jcr:name="D" dna:name="Cache" ... />
    </dna:sources>
    <dna:federatedRepositories jcr:primaryType="nt:unstructured">
        <dna:federatedRepository jcr:name="Vehicles"... ">
            <dna:workspaces jcr:primaryType="nt:unstructured">
                <dna:workspace jcr:name="default" ...">
                    <dna:cache dna:sourceName="Cache" .../>
                    <dna:projections ...>
                        <dna:projection jcr:name="Cars" dna:projectionRules="/Vehicles => /" .../>
                        <dna:projection jcr:name="Aircraft" dna:projectionRules="/Vehicles => /" .../>
                        <dna:projection jcr:name="Configuration" dna:projectionRules="/ => /" .../>
                    </dna:projections>
                </dna:workspace>
            </dna:workspaces>
        </dna:federatedRepository>
    </dna:federatedRepositories>
</jcr:system>
```

Configure Repository Service

```
// Create the execution context that we'll use for the services. If we'd want to use JAAS, we'd
// create the context by supplying LoginContext, AccessControlContext, or even Subject with
// CallbackHandlers. But this example doesn't use JAAS in this example.
ExecutionContext context = new ExecutionContext();

// Create the library for the RepositorySource instances ...
RepositoryLibrary sources = new RepositoryLibrary(context);

// Load into the source manager the repository source for the configuration repository ...
InMemoryRepositorySource configSource = new InMemoryRepositorySource();
configSource.setName("Configuration");
sources.addSource(configSource);

// Now instantiate the Repository Service ...
RepositoryService service = new RepositoryService(sources, configSource.getName(), context);
service.getAdministrator().start();
```

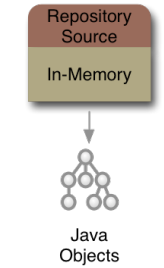
Connector framework

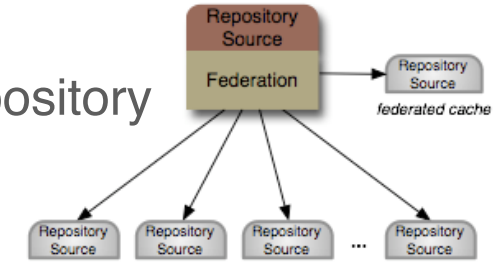
- **RepositorySource**
  - represents a connectable external system
  - creates connections
  - a JavaBean that's analogous to JDBC DataSource
- **RepositoryConnection**
  - represents a connection to a source
  - process requests by translating to source language
  - adapts content changes into events
- **RepositoryService**
  - manages RepositorySource instances
  - maintains pools of connections for each source
  - can reflect what's defined in a configuration repository

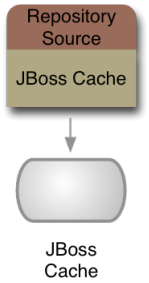


JBoss DNA connectors

- **In-memory**
  - simple transient repository
- **Federated connector**
  - Merges content from multiple other sources
  - Each projects its content into “federated” repository
  - Strategies for merging nodes
  - Uses another source as the cache
- **JBoss Cache**
  - support for distribution, clustering, replication
  - ability to persistent information in databases
- **Connector to JPA Persistence Store**
  - persists graph content using JPA

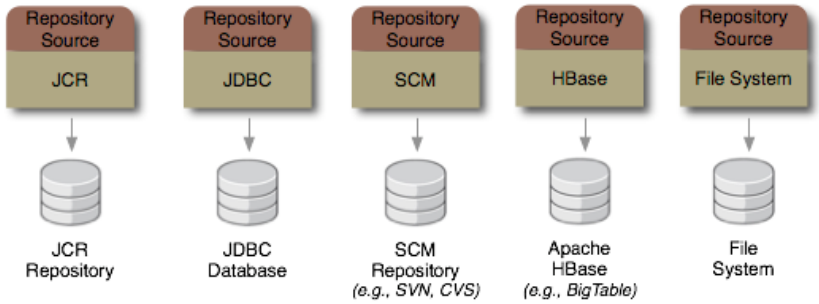






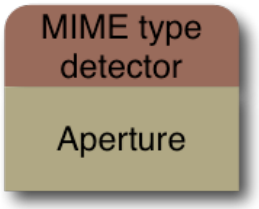
JBoss DNA connectors

- **Relational databases**
  - schema information (metadata)
  - data
- **File system connector**
  - expose files and directories
  - store content on file system
- **JCR repositories (scheduled)**
- **SCM systems**
  - SVN, CVS, Git
  - maps directory structure into `nt:folder` and `nt:file` nodes
  - includes version history
- **Maven repositories (scheduled)**
- **JNDI/LDAP (scheduled)**

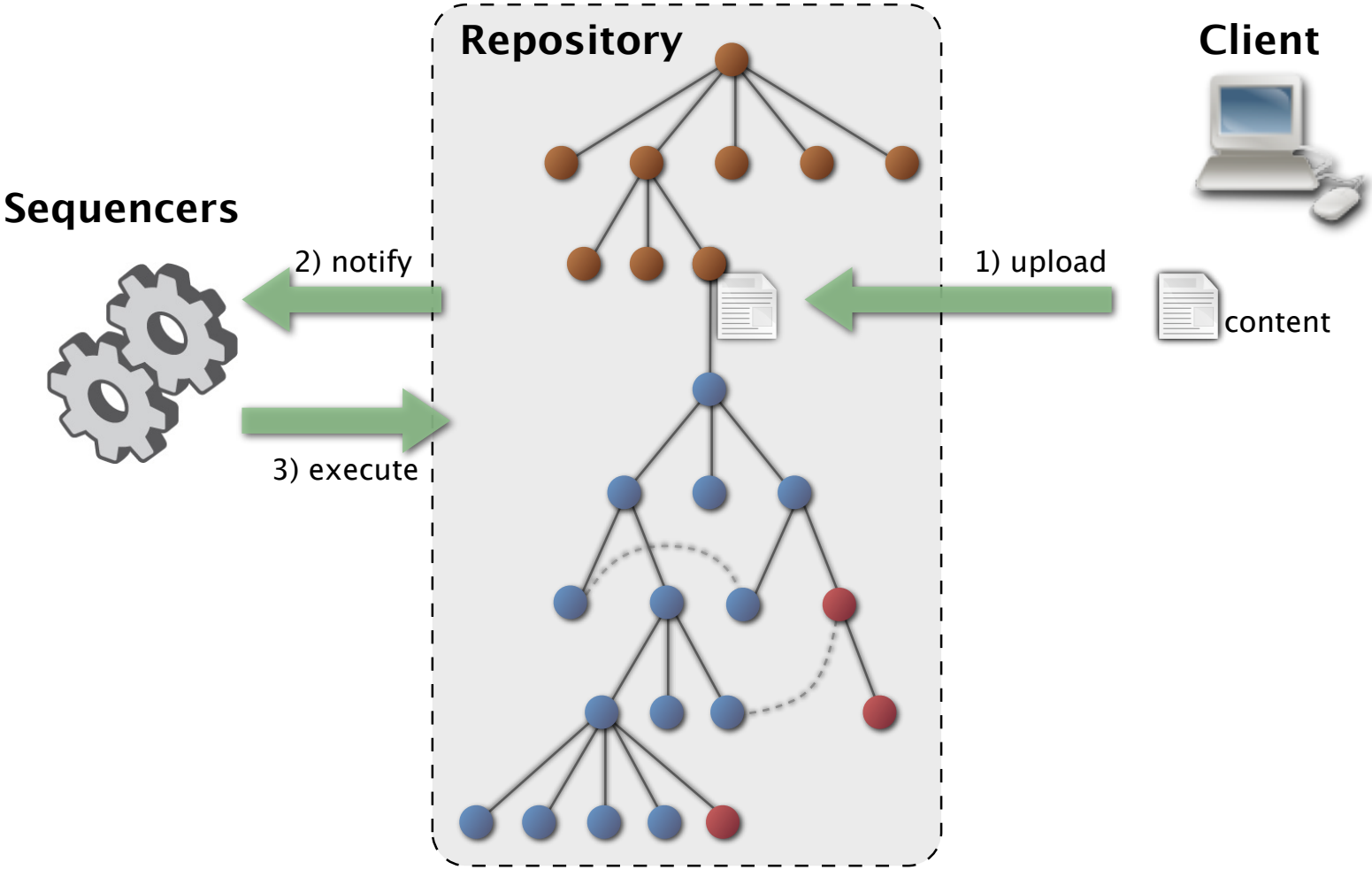


Detecting media types

- **Content often includes files**
- **Often want correct MIME type as metadata**
- **Typical approaches**
  - map extensions
  - interpret content
- **JBoss DNA uses MIME type detectors**
  - extensions that determine MIME type given filename and/or content
  - default implementation uses the Aperture open-source library



Sequencing content



Configure sequencers

- **Path expressions describe**
  - paths of content to be sequenced
  - path where to put generated output

```
inputRule => outputRule
```

  - examples:

```
//(*.(jpg|jpeg|gif|bmp|pcx|png))[*]/jcr:content[@jcr:data] => /images/$1
//(*.mp3)[*]/jcr:content[@jcr:data] => /mp3s/$1
//(*.(doc|ppt|xls))[*]/jcr:content[@jcr:data] => ./
```

- **Register a sequencer configuration**
  - Name, description, classname, and path expressions
- **Make available at runtime**
  - put sequencer implementation on the classpath
  - or use a `ClassLoaderFactory`

Configure SequenceService

Instantiate and configure the SequencingService ...

```
SimpleSessionFactory sessionFactory = new SimpleSessionFactory();
sessionFactory.registerRepository("Repository", this.repository);
Credentials credentials = new SimpleCredentials("jsmith", "secret".toCharArray());
sessionFactory.registerCredentials("Repository/Workspace1", credentials);
JcrExecutionContext context = new JcrExecutionContext(sessionFactory,"Repository/Workspace1");

// Create the sequencing service, passing in the execution context ...
SequencingService sequencingService = new SequencingService();
sequencingService.setExecutionContext(context);
```

Start the SequencingService ...

```
sequencingService.getAdministrator().start();
```

Configure SequenceService

… sequencers that it will use.

```
String name = "Image Sequencer";
String desc = "Sequences image files to extract the characteristics of the image";
String classname = "org.jboss.dna.sequencer.image.ImageMetadataSequencer";
String[] classpath = null; // Use the current classpath
String[] pathExpressions = {"//(*.(jpg|jpeg|gif|bmp|pcx|png)[*])/jcr:content[@jcr:data] => /images/$1"};
SequencerConfig imageSequencerConfig = new SequencerConfig(name, desc, classname,
                                                           classpath, pathExpressions);
sequencingService.addSequencer(imageSequencerConfig);
```

Configure observation service …

```
this.observationService = new ObservationService(sessionFactory);
this.observationService.getAdministrator().start();
```

observation service is started, listeners can be added

```
observationService.addListener(sequencingService);
```

Shutting down DNA services …

Writing your own sequencer

Implement interface

```
public interface StreamSequencer {
    /**
     * Sequence the data found in the supplied stream,
     * placing the output information into the supplied output map.
     */
    void sequence( InputStream stream, SequencerOutput output,
                   ProgressMonitor progressMonitor );
}
```

- Read the stream
- Create output structure using SequencerOutput parameter:

```
output.setProperty(path, propertyName, propertyValue);
```

JBoss DNA sequencers (as of 0.4)

ZIP archives

Java source

Microsoft Office documents

MP3 audio files

JCR Compact Node Definition

jBPM PDL (scheduled)

Images (JPEG, GIF, BMP, PCX, PNG, IFF, RAS, PBM, PGM, PPM & PSD)

For more information

- **Project: www.jboss.org/dna**
- **Downloads (0.4)**
  - Binary, source, documentation, examples
- **JBoss Maven2 Repository**
  - “org.jboss.dna” group ID (several artifacts)
- **Documentation**
  - Getting Started describes the design, the different components, and how to use them with a trivial example application
  - Reference Guide describes how JBoss DNA works internally from a developer perspective
- **Blogs:** jbossdna.blogspot.com
- **IRC:** irc.freenode.net#jbossdna

Q&A