JBoss DNA

Project Status - September 2008

Randall Hauch
Project Lead

JBoss DNA timeline

Feb 2008 - Project Announced

Focus: sequencer framework

May 2008 - Release 0.1

Focus: federation, connectors, more sequencers

TODAY

Sept 2008 - Release 0.2

Oct 2008

Nov 2008

Dec 2008

More frequent releases with smaller but incremental features, including: configuration, persistence, clustering, search, more sequencers, more connectors

JBoss DNA community

- **Randall Hauch**
  - Project lead
- **John Verhaeg**
  - Federate Tools contributor
- **Dan Florian**
  - Federate Tools contributor
- **Stefano Maeste**
  - JBossWS contributor
- **Serge Pagop**
  - jBPM contributor
- **Michael Trezzi**
  - New to JBoss.org
- **Alexandre Porcelli**
  - Drools contributor

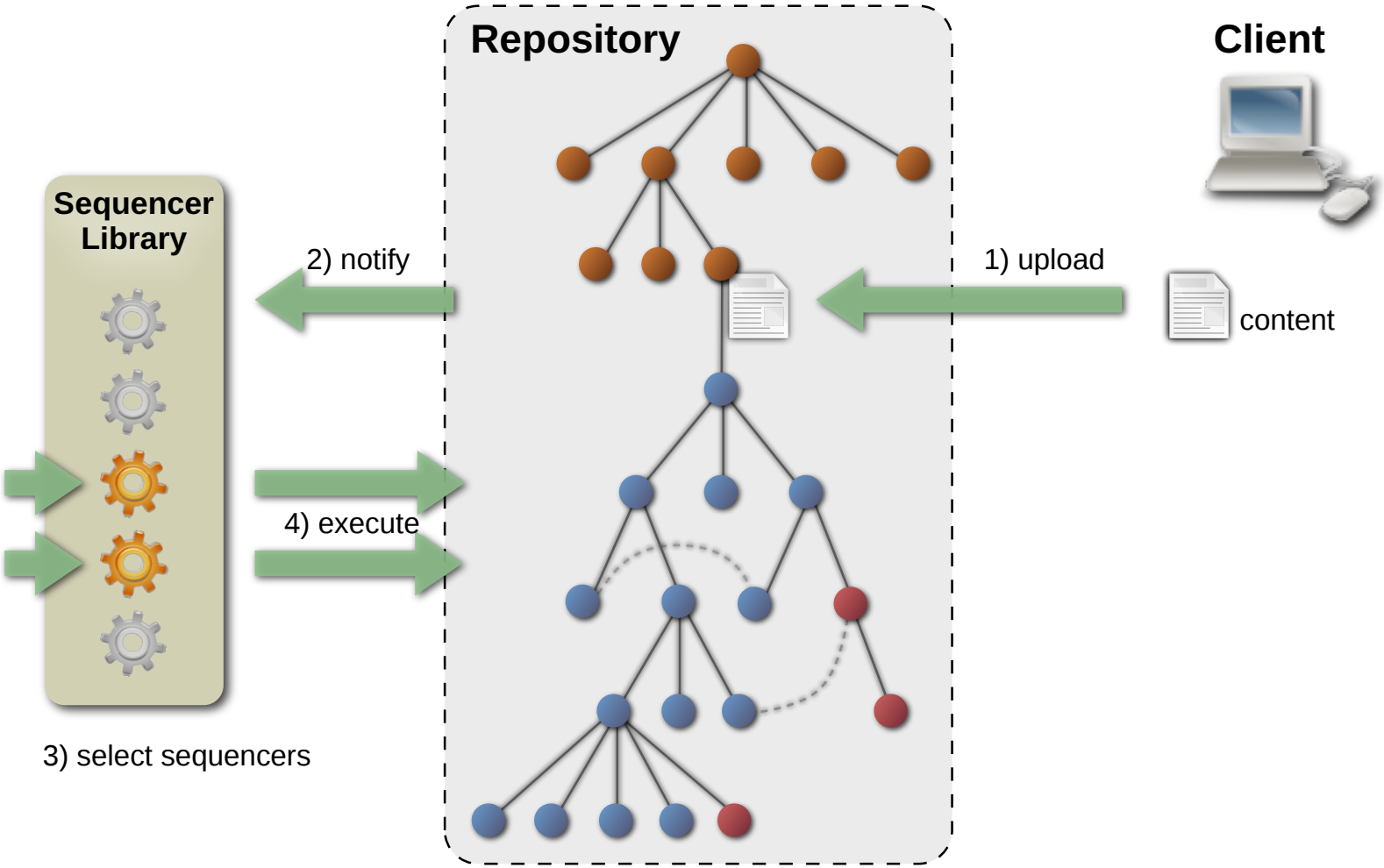
JBoss DNA information

- **Project: www.jboss.org/dna**
- **Downloads (0.1)**
  - Binary with DNA source, jars, and Getting Started doc
  - Source
  - Getting Started example application
- **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
- **Blogs:** jbossdna.blogspot.com
- **IRC:** irc.freenode.net#jbossdna

Using JBoss DNA 0.1

- **Automatically extract content added to JCR**
  - "Sequencers" read content streams and output structure
  - DNA determines which sequencers apply, runs them while handing them the content stream, and saves the output
  - Generated output can be written to the same repository (where the content is stored) or to a different repository
- **Requires JCR-compliant repository**
  - DNA listens to 1 or more JCR repositories
  - Tested with Apache Jackrabbit
- **Sequencers provided out of the box**
  - Image sequencer - extracts metadata from image files (e.g., PNG, GIF, JPEG, etc.)
  - MP3 sequencer - extracts track title, album, etc. from MP3s

Sequencing content



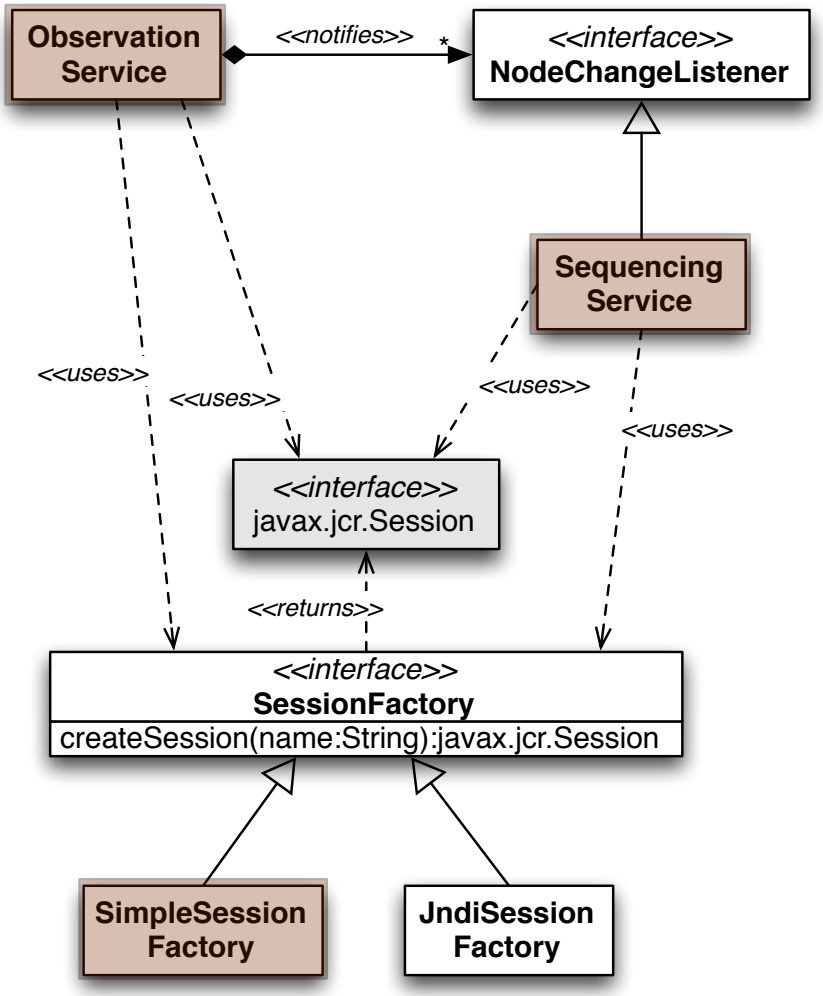
Configuring Sequencers

- **Sequencers are chosen based upon path expressions**
- **Basic form:**

  inputPath => outputPath

- **Input path defines criteria for matching content**
  - Matches node names (and same-name-sibling indexes)
  - Use wildcards in names and/or indexes
    - "//" means any sequence of nodes
    - "*" means a node with any name
  - Can capture groups using parentheses (like regular expressions)
  - Can prefix with name of workspace
- **Output path defines where the output will be written**
  - Default (".") is to write to the same location as the input node
  - Can use named groups captured from input path
  - Can prefix with name of workspace

DNA services



Set up environment for DNA

Create the SessionFactory

```
SimpleSessionFactory sessionFactory = new SimpleSessionFactory();
// Register JCR repository implementation
final javax.jcr.Repository myJcrRepository = ...
sessionFactory.registerRepository("Main Repository", myJcrRepository);
// Register JCR workspaces with login info so DNA can establish connections
Credentials credentials = new SimpleCredentials("jsmith", "secret".toCharArray());
sessionFactory.registerCredentials("Main Repository/MyWorkspace", credentials);
```

Create and start the ObservationService

```
ObservationService observationService = new ObservationService(sessionFactory);
observationService.getAdministrator().start();
// Monitor the named JCR workspace for new nodes or new/changed properties ...
observationService.monitor("Main Repository/MyWorkspace", Event.NODE_ADDED
                                                        | Event.PROPERTY_ADDED
                                                        | Event.PROPERTY_CHANGED);
```

Create and start the SequencingService

```
ExecutionContext executionContext = new SimpleExecutionContext(sessionFactory);
SequencingService sequencingService = new SequencingService();
sequencingService.setExecutionContext(executionContext);
sequencingService.getAdministrator().start();
// The sequencing service wants to listen for these events
observationService.addListener(sequencingService);
```

Tell DNA what to sequence

Create the SequencerConfig instances and add to the SequencingService

```
// Set up the Image sequencer ...
String name = "Image Sequencer";
String desc = "Sequences image files to extract the characteristics of the image";
String classname = "org.jboss.dna.sequencer.images.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);


// Set up the MP3 sequencer ...
name = "Mp3 Sequencer";
desc = "Sequences mp3 files to extract the id3 tags of the audio file";
classname = "org.jboss.dna.sequencer.mp3.Mp3MetadataSequencer";
String[] mp3PathExpressions = {"//(*.mp3)[*]/jcr:content[@jcr:data] => /mp3s/$1"};
SequencerConfig mp3SequencerConfig = new SequencerConfig(name, desc, classname, classpath, p3PathExpressions);
sequencingService.addSequencer(mp3SequencerConfig);
```

Of course we can add, remove, and update sequencer configurations at any time

Any number of sequencer configurations can use the same sequencer implementation

Writing a Sequencer

- **Implement this 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 the output structure using SequencerOutput method:

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

- **Add it to the classpath**
  - Or implement a ClassLoaderFactory
- **Add 1 or more configurations that use it**

Writing a Sequencer

- **Image sequencer implementation**

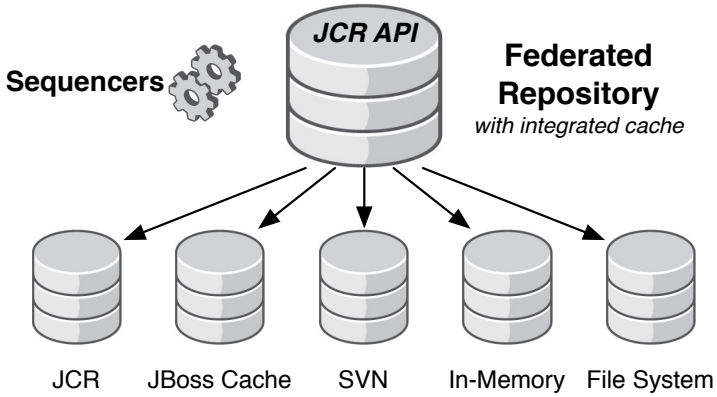
```
1  public void sequence( InputStream stream, SequencerOutput output, ProgressMonitor progressMonitor ) {
2
3      progressMonitor.beginTask(10, ImageSequencerI18n.sequencerTaskName);
4
5      ImageMetadata metadata = new ImageMetadata();
6      metadata.setInput(stream);
7      metadata.setDetermineImageNumber(true);
8      metadata.setCollectComments(true);
9
10     // Process the image stream and extract the metadata ...
11     if (!metadata.check()) return;
12     progressMonitor.worked(5);
13
14     if (progressMonitor.isCancelled()) return;
15
16     // Generate the output graph if we found useful metadata ...
17     NameFactory nameFactory = context.getFactories().getNameFactory();
18     PathFactory pathFactory = context.getFactories().getPathFactory();
19     Path metadataNode = pathFactory.create(METADATA_NODE);
20
21     // Place the image metadata into the output map ...
22     output.setProperty(metadataNode, nameFactory.create(IMAGE_PRIMARY_TYPE), "image:metadata");
23     output.setProperty(metadataNode, nameFactory.create(IMAGE_MIME_TYPE), metadata.getMimeType());
24     output.setProperty(metadataNode, nameFactory.create(IMAGE_FORMAT_NAME), metadata.getFormatName());
25     output.setProperty(metadataNode, nameFactory.create(IMAGE_WIDTH), metadata.getWidth());
26     output.setProperty(metadataNode, nameFactory.create(IMAGE_HEIGHT), metadata.getHeight());
27     output.setProperty(metadataNode, nameFactory.create(IMAGE_BITS_PER_PIXEL), metadata.getBitsPerPixel());
28     output.setProperty(metadataNode, nameFactory.create(IMAGE_PROGRESSIVE), metadata.isProgressive());
29     output.setProperty(metadataNode, nameFactory.create(IMAGE_NUMBER_OF_IMAGES), metadata.getNumberOfImages());
30     output.setProperty(metadataNode, nameFactory.create(IMAGE_PHYSICAL_WIDTH_DPI), metadata.getPhysicalWidthDpi());
31     output.setProperty(metadataNode, nameFactory.create(IMAGE_PHYSICAL_HEIGHT_DPI), metadata.getPhysicalHeightDpi());
32     output.setProperty(metadataNode, nameFactory.create(IMAGE_PHYSICAL_WIDTH_INCHES), metadata.getPhysicalWidthInch());
33     output.setProperty(metadataNode, nameFactory.create(IMAGE_PHYSICAL_HEIGHT_INCHES), metadata.getPhysicalHeightInch());
34     progressMonitor.done();
35 }
```

This does the work of reading the stream

The remainder generates the output

What's coming in 0.2 (Sept 08)

- **JCR API implementation (partial)**
- **Sequencers**
  - Image sequencer
  - Audio files
  - MIME types
  - Java source
  - ZIP files
  - MS Office files
  - XML files (import)
  - JCR CND files
  - ESB Messages
- **Federated repository with connectors**



Where we're going

- **Ready for use in JBoss (and other) products**
  - Self-contained repository persistence (via connectors)
  - Manage sequencers and connectors in repository
  - Remote connectivity, search, transactional updates
- **Use Microcontainer**
  - Configure, set up, and manage DNA services
  - Compatible with other components: JNDI, JTS, JMS, ...
- **"Micro-repository"**
  - Small configuration of DNA with limited dependencies
  - Useful for managing configuration of other systems
  - Connectors can read info from configuration files
- **More sequencers and connectors**

How we'll get there

- **Release cycles to date have been 3-4 months**
  - Added core frameworks (sequencers, federation, connectors)
  - Initial implementation of JCR
- **Now we can move towards shorter cycles**
  - Cycles about 3-4 weeks long
  - Each release will incrementally add to features
  - Rapid and steady addition of features/capabilities
  - More easily adapt to what users need

Look for several new releases by year end

Interested?

Join the team!

www.jboss.org/dna