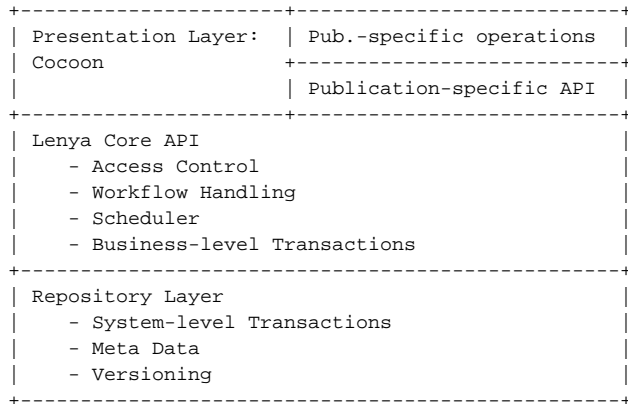


ProposalArchitecture

Some notes and random thoughts how to improve the Lenya architecture. Feel free to add your comments.

– [ApacheCocoon](#):AndreasHartmann

Overview



Resources

- A *resource* is a piece of information.
- A resource is identified by a unique *resource ID*.
- It can consist of different *versions*.
- A version has a set of meta data, e.g., language, workflow state, workflow variables, ...
- Versions can be obtained by filtering meta data (e.g., only a specific language version).
- Each version can be accessed using an *XML view*. It shows an XML representation of the resource embedded in a *resource envelope*, containing the meta data.

AccessControl

- Access control is handled on version level.
- Typically, all versions inherit the access control policies of their resource.
- It is possible to refine the policies for certain versions, e.g., for one language version.

TheLenyaProtocol

- The lenya protocol is used to identify a resource.
- A resource URL starts with the protocol name `lenya: /`.
- The name is followed by the resource ID (e.g., `/images/landscape`).
- The view is selected by a view identifier (e.g., `.xml`, `.png`).
- A version could be selected/retrieved using a uri postfix (e.g., `/images/landscape.xml/1.2`).
- Additionally, meta data can be filtered using request parameters.
- There are default filtering rules:
 - the currently configured language version
 - the latest version of this language

Examples:

- URL: `lenya://images/landscape.xml` (accessible)

```
<lenya:version resource-id="/images/landscape">
  <lenya:meta>
    <dc:creator>John</dc:creator>
  </lenya:meta>
  <lenya:asset mime-type="image/png"
    src="file:///home/john/lenya/images/landscape.png" />
</lenya:version>
```

- URL: `lenya://images/landscape.xml` (not accessible)

```
<lenya:version resource-id="/images/landscape"
  accessible="false"/>
```

- URL: `lenya://images/landscape.png` would return the binary image data
- URL: `lenya://news/index.xml?language=de`

```
<lenya:version resource-id="/news/index">
  <lenya:meta>
    <dc:creator>John</dc:creator>
    <dc:language>de</dc:language>
  </lenya:meta>
  <xhtml:html>
    <xhtml:head> ...
    <xhtml:body>
      ...
    </xhtml:body>
  </xhtml:html>
</lenya:version>
```

Collections

A collection is a specific kind of resource which contains other resources. The XML source of a version of a collection resource could look like:

```
<lenya:collection resource-id="/overview">
  <lenya:resource resource-id="/sections/news"/>
  <lenya:resource resource-id="/sections/tv"/>
  <lenya:resource resource-id="/sections/sports"/>
</lenya:collection>
```

The XML view of this version could look like:

```
<lenya:resource resource-id="/overview">
  <lenya:meta>
    <dc:creator>John</dc:creator>
    <dc:language>de</dc:language>
  </lenya:meta>
  <lenya:resource xlink:href="resource://sections/news.xml" xlink:show="embed"/>
  <lenya:resource xlink:href="resource://sections/tv.xml" xlink:show="embed"/>
  <lenya:resource xlink:href="resource://sections/sports.xml" xlink:show="embed"/>
</lenya:resource>
```

Implementation

- The resource protocol is handled by the `ResourceSourceFactory` class.
- If a resource can not be viewed by the current identity, the resource envelope is empty and contains a note about the access control protection.

The Presentation Layer

The presentation is handled by Cocoon pipelines. The XML view of a resource is requested using the resource protocol:

```
<map:match pattern="**.html">
  <map:generate src="lenya://{1}.xml"/>
  <map:transform src="page2xhtml.xsl"/>
  <map:serialize/>
</map:match>
```

A staging view of the website could be created as follows:

```
<map:match pattern="staging/**/*.html">
  <map:generate src="lenya://{1}.xml?workflow-state=staging"/>
  <map:transform src="page2xhtml.xsl"/>
  <map:serialize/>
</map:match>
```

TheRepositoryLayer

MetaData

Using a repository, metadata should be stored as properties attached to an asset/document. Different types/classes of metadata can be identified like for

- workflow
- descriptive metadata i.e. dublin core annotations
- application specific metadata, i.e. a news module would store properties when a news item is not valid anymore...

References

- [Repository](#)

Those metadata should be queried by the appropriate mechanism which is supported by the underlying repository implementation i.e. via Xpath.

Each type of metadata should be stored using its own namespace i.e. wf:state, dc:title etc.

Issues

- A repository implements its own access control system. So it knows about users and rights. How would the Lenya access control system be mapped to the one implemented in the repository?