OverviewRepository

This page is totally out-of-date wrt to Lenya 2.0. – JörnNettingsmeier <<DateTime(2007-07-30T14:11:20Z)>>

Lenya 2.0 provides a basic repository implementation, which will probably be replaced by JCR (see ProposalRepository).

Overview

The repository is accessed via URLs using the `lenya://` protocol. The root of the URL space is the servlet context directory, which means that the URL

```
lenya://lenya/pubs/mypub/content/authoring/index.xml
```

accesses a content resource inside a publication.

Internals

When a URL of the form `lenya://...` is resolved using the `SourceResolver`, the `LenyaSourceFactory` creates a `RepositorySource` object. The repository source accesses the repository via a `o.a.l.cms.repository.Node`, which provides the actual repository functionality. Currently, the `Node` interface is implemented by the `SourceNode` class, which accesses files using `context://` sources. The `Node` interface extends the `Transactionable` interface, which means that a repository node is designed to be read and manipulated in transactions. It supports locking and versioning.

Locking

Imagine the following scenario:

- Bob opens document A and makes some changes.
- Alice decides to make some changes as well, and saves document A immediately.
- Now Bob clicks the "save" button.

Without special precautions, Alice's changes are lost. Locking is used to determine if a persistent item has been changed since it was read for the first time during a transaction. To use the item, it has to be loaded as an object. Before the first read access, a lock is placed on the object. The lock contains the current version number. When Alice saved the item A, Bob's transaction can now compare the current version of A with the lock version by calling `Transa ctionable.hasChanged()`. This way, it can determine if A was changed during the transaction and output a warning message. This policy is called *opti mistic offline lock*, because Bob's transaction was optimistic that no changes would occur during editing.

Another scenario:

- A website contains some articles A^1, ..., A^{20}.
- Alice wants to create an overview document, which is assembled using the abstracts of A^1, ..., A^{20}.
- The *Create Overview* usecase presents her with a page containing checkboxes for all articles.
- She browses all articles to make sure that the abstracts are OK. Whenever she finishes an article, she checks the corresponding box, which advances the usecase. The usecase places a lock on the article node.
- When Alice has finished browsing all articles - which can take a considerable amount of time - she can be sure that no abstract has been changed, because each article had been locked after she had checked it.

Check-In and Check-Out

Check-In and Check-Out are used to prevent a repository node from being changed by more than one client at the same time. A complex transaction might not want to use optimistic offline lock, because it would be too expensive to lose all changes if the affected node was meanwhile changed by another user. In this case, the node can be checked out in the beginning of the transaction. If a node is checked out, it cannot be locked. This means that no other transaction may begin which would affect the node - the node is protected from being changed by another user.

Versioning

Whenever a change is made to a repository source, a new version is created. This is achieved using Lenya's revision controller.