ProposalSitemap2JavaApiContract

Proposal for a Sitemap <-> Java API contract for Lenya

(originally started by TorstenSchlabach - please add your comments with name)

Background

There is a high level of agreement amoung Lenya comitters that

- some sort of modularity / plugin infrastructure should be introduced
- programmatic logic should be moved to the Java layer in order to make sitemaps simpler thus easier to read, understand and maintain. This holds true for both the publication specific as well as the Lenya core sitemaps
- at some point in time, some integration to some repository other than the raw file system is planned

This relates to these two threads on the dev mailing list:

- http://nagoya.apache.org/eyebrowse/ReadMsg?listId=283&msgNo=8872
- http://nagoya.apache.org/eyebrowse/ReadMsg?listName=dev@lenya.apache.org&msgNo=8941

Note: Sometimes the threads are not properly linked together, so if you want to catch up what was been dicussed, make sure you find all mails on the subject using the subject search of your favorite archive.

The need for contracts

Contracts are good anyway, but they are most needed when it comes to publications. The impact of changes to Lenya on a publication should be kept minimal. In other words, at least the Lenya <-> Publication contract should be as stable as possible and as upwards compatible as possible to make it easy for users to upgrade to a new version of Lenya once it will be released (or even before!) without having to start a migration project on their publication. Requiring adoption of publications to new Lenya releases will hold people back from upgrading and newer is better!

There is certainly a lot of other contracts involved that should be kept stable if at all possible, for example the contract between the Lenya core and plugins (editor or ressource type). Taking a look at this drawing from one of the emails mentioned above might help to identify other contracts needed. But the purpose of this proposal is to define a contract between the Java and the sitemap layer.

Comment by AndreasHartmann

I see two separate concern areas:

Customization and implementation of CMS functionality

IMO the major concerns in this field are:

1. hide complexity from the integrator

2. provide an API which is minimal (reduce the impact of changes) and complete
3. use Cocoon concepts
4. provide simple concepts with reduced flexibility
5. allow complex customization with full flexibility (if appropriate knowledge is possessed)

Approaches:

- Provide an easy-to-use framework to implement CMS functionality (e.g., the usecase framework).
- Hide implicit functionality behind the API (e.g., versioning). A step in this direction was taken by the lightweight repository layer in 2.0.
- Hide implementation details behind the API (e.g., meta data storage and workflow history).

Presentation of content

Important points:

1. provide access to all necessary information in sitemaps and XSLTs
2. define XML-based interfaces for components (e.g., navigation elements)

Approaches:

- Plug-ins + configuration.
- Plug-ins + customization (fallback and `<xsl:import/>`). Problem: XSLT doesn't provide encapsulation like OOPLs (public and private templates). This makes it harder to declare contracts. A possible approach is:
    1. Extract the overridable templates into a separate stylesheet and include it via the `<xsl:import href="fallback://..."/>`
    2. Don't use fallback for the main stylesheet to ensure that it can't be overridden.
- Do your own (plain Cocoon + Lenya components like PageEnvelopeModule).

Proposal: Eliminate all direct file access from Lenya core and publication sitemaps

There should not be any `src="xyz"` attributes found in any Sitemaps anymore, where xyz means any URIs without a specific protocol mentioned which means the SourceResolver will default to the file resolver and and use the URI provided as a file system location which is interpreted relative to the sitemap location in the directory tree.

This includes not to use any `src="context:..."` sources as well as they basically do the same.

Current Status

The `lenya:` protocol is a very good example of determining in the Java layer where to look for the actual input. In other words, the lenya: protocol hides all the complexitiy (which belongs into the Java layer) from the sitemap. The sitemap can just go to Lenya and request the document "abc" by using something like `src="lenya:/abc.xml"`. The Java layer will use it's knowledge about the context to determine the actual location of the document. This proofed to work very well.

Links

- http://www.w3.org/TR/uri-clarification/

Open for discussion

The question will be how this might be extended for locating input other than document / content XML files, such as

- core XSLT files
- publication specific XSLT files
- schemas (I am not saying RNG here, think of on-the-fly conversion ...)
- core CSS
- plugin Javascript
- core Javascript
- what have you ...

It comes down to

- should the lenya: protocol be extended?
- should one or more new protocols be invented?
- might the fallback: protocol be obsolated by any new approach?