ProposalContentModel

Lenya 2.0: Proposal for an object model to support composed documents and plugins

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

Introduction

The premise of this proposal is that extending the core API with several additional explicit types (meaning interfaces, abstract classes) can provide a basis for implementing several key functionalities.

Note that

- this proposal is not about the question "how is this knowledge accessed (and created)" - that is more an issue of the ProposalSitemap2JavaApiContract.
- this proposal tries to take a fresh look at things; whereas some existing interfaces and classes in Lenya already contain parts of what is discussed here.

This proposal addresses the following issues:

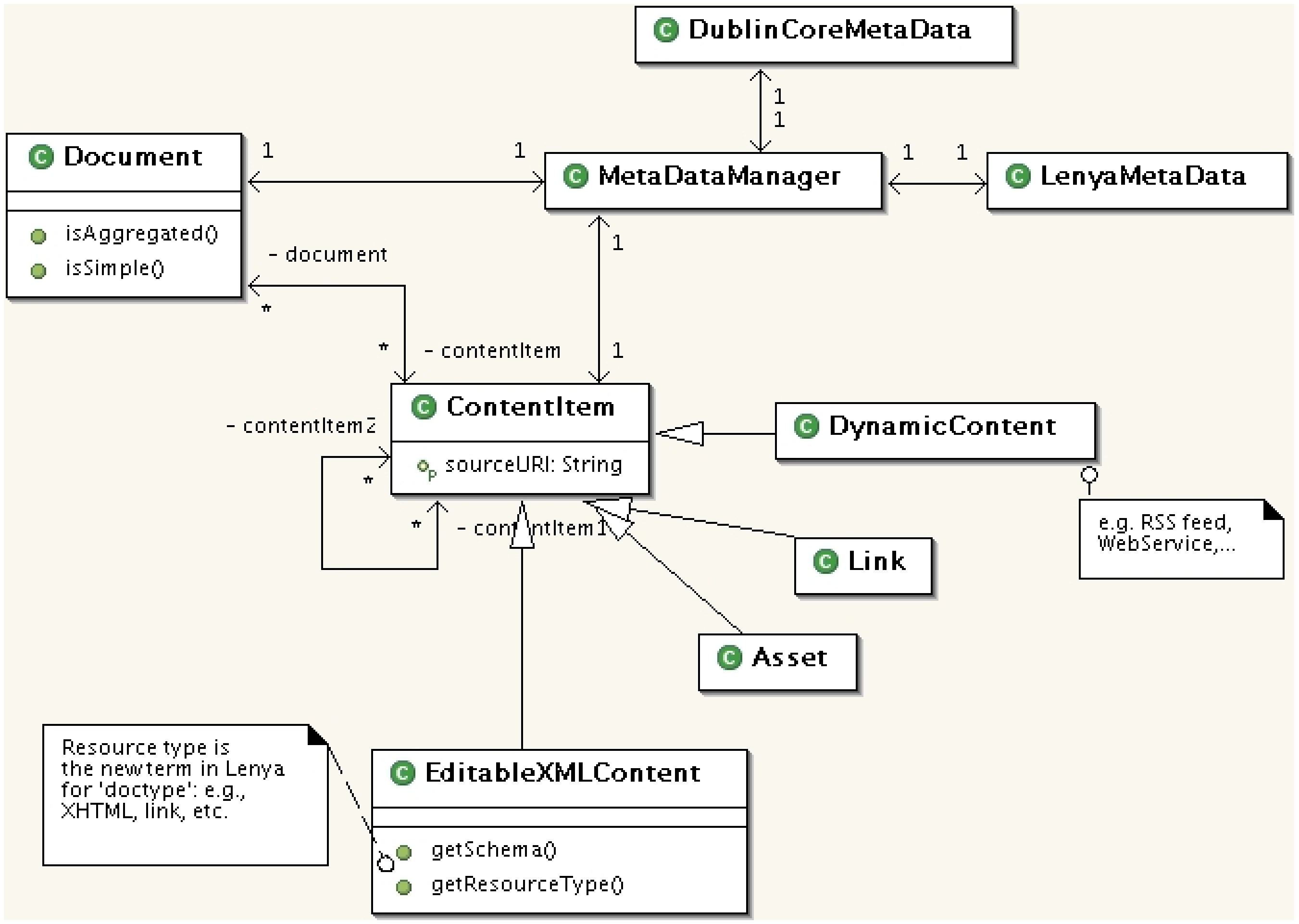
- the assumption about documents currently is that they are made of a single part (the page). However, real documents may consist of several, independently managed content parts: some parts may be static, others dynamic; some parts may have other access rights then others (example: news part in the homepage of a publication). Currently in Lenya, this is only possible by customized sitemap manipulation, but not supported out-of-the-box; i.e. the core has no 'knowledge' about composed documents, nor can it offers GUIs to create documents by composing them.(See also ProposalXIncludeAggregation)
- the above issue also means that content can not (easily) be reused in different documents. Solving the issue above would solve this issue at the same time.
- further, we need a support for centralized asset management. Currently, an area for "central" assets can be created, and individual documents can refer to them by URL. But there is no knowledge within the CMS about such relations, so no support for consistency (publishing together, deleting, ...)
- we probably also want a centralized link management, as a typical feature expected from a CMS
- for future development, a plugin architecture is desired. A richer core model can facilitate the 'intelligent' integration of plugins; i.e., what types of 'things' is a plugin for; so we can distinguish between types of plugins (editor plugins, renderer plugins), and dynamically offer the user a plugin depending on what she is doing

General design goals

- enrichen the object model of Lenya trunk; define interfaces / classes for major types of content and plugins. The primary goal is to enable (more) user-friendly management of different content types. Also, to provide an architecture for developers to implement 'stuff' without having to know all the internals.
- maintain current functionality of Lenya trunk, with minimal code changes
- plugins declare what functionality they support:
    - editor plugins: when the CMS user navigates to a content item, Lenya automatically determines the plugins that can work with this type. Suppose the CMS user looks at a document consisting of a single piece of content, an XHTML file. Lenya, in the edit menu, lists all registered editors for this type, e.g. Bitflux, Kupu, FCKEditor, whatever.
    - renderer plugins: the default rendering mechanism stays as it is now: a document which is simply XHTML (or other XML) is transformed with the XSL stylesheets. If a document consists of several XML content items, Lenya sequentially applies a stylesheet relevant for each resource type. In addition or in replacement of the default rendering mechanism, plugins might be used e.g. to sequence content parts within a page according to some other algorithm.
- the danger of a more explicit object model is clearly to lose some flexibility. So the goal is to explicitly define in the core only those types (or categories of types) that the CMS should definitely support; by categories I mean things like `EditableXMLContent`, `Asset`, ... Whereas resource types as currently understood (xhtml vs links vs whatever) would not be affected, meaning they can still be configured as is currently done.

Content model

Class diagram:



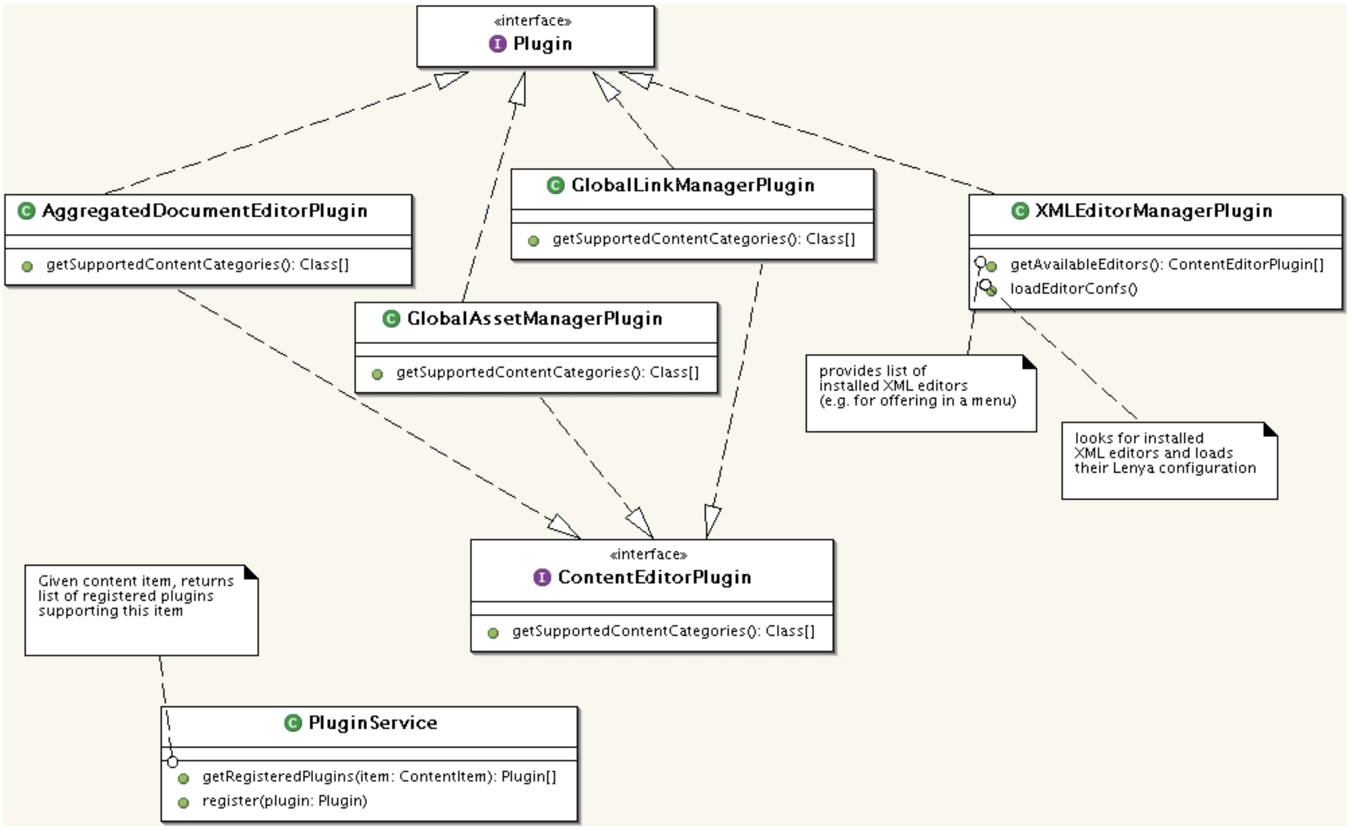
Comments:

- a document can consist of a list of several content items. That is, a document is an aggregation of 1 or more content items. The document itself thus no longer directly refers to content; it contains references to content parts.
- a `ContentItem` instance can refer to several other `ContentItem` instances, with several implications:
  - the current mechanism where an XTHML document refers to Assets is supported;
  - the site/info area can show all related items to a given item (equivalent of today's 'asset' tab). The advantage is, we can now also show what is currently termed "global assets"
  - this knowledge can be used in use-cases such as 'Publish' -> related items may also need to be published.
- if a document consists of a single `EditableXMLContent` item (such as an XHTML page), the user does not see any difference with the current Lenya. She can choose an editor such as BXE or Kupu to directly edit the content.
- if a document consists of several content items, an aggregation manager GUI is presented first. This manager allows to piece together a document from existing content items, or to define new placeholders for content items within the document. When the user chooses to edit an individual item in this document, as above she has the choice of appropriate editors.
- Note that there is no notion of 'position' in the API, that is the API says simply which 'things' are related, not how they should be rendered within a live page. The default mechanism would be to render them sequentially; and for XHTML documents the current mechanism can still be used: that is, the xhtml source has a `<lenya:asset>` tag at the place where an asset should be shown within the page.

Relationship with plugins

This section contains some initial thoughts about plugin interface definitions - certainly requires more thinking.

Class diagram:


«interface»
Plugin
AggregatedDocumentEditorPlugin
getSupportedContentCategories(): Class[]
GlobalLinkManagerPlugin
getSupportedContentCategories(): Class[]
XMLEditorManagerPlugin
getAvailableEditors(): ContentEditorPlugin[]
loadEditorConfs()
GlobalAssetManagerPlugin
getSupportedContentCategories(): Class[]
provides list of installed XML editors (e.g. for offering in a menu)
looks for installed XML editors and loads their Lenya configuration
Given content item, returns list of registered plugins supporting this item
«interface»
ContentEditorPlugin
getSupportedContentCategories(): Class[]
PluginService
getRegisteredPlugins(item: ContentItem): Plugin[]
register(plugin: Plugin)

Comments:

- this model thus far only supports finding an appropriate plugin dynamically, for a content item. How exactly the plugin then interoperates with Lenya is still open. My guess: depending on the purpose of the plugin (e.g. "for editing"), a more precise contract can be defined (e.g. such a plugin must be able to receive a piece of content, and have a callback registered for saving it)
- what does a plugin implementation consist of ? Presumably each plugin will have a subdirectory, with sitemap, xsls, and configuration files. Depending on the plugin, it may also have Java classes etc. For individual XML editors, ideally no Java classes are required, as the `XMLEditorManagerPlugin` would be the service to access information about them.
- which requests are to be handled by the plugin will be presumably determined by the URI.
- the plugins must register themselves with the `PluginService` (presumably an Avalon service). How should this happen ? One possibility: they are configured within the `PluginService` Avalon configuration; so the `PluginService` at initialization can read which plugins exist, and can dynamically ascertain what they are for. Note that this would not be required for the individual XML editors, as they would encapsulated by the `XMLEditorManagerPlugin`

Relationship with repository

It might be a good idea to have a sub-branch in the repository for each category of content: `EditableXML`, `Asset`, `Link`, `Dynamic`; plus a subbranch for documents. The site-tree will be constructed from the document nodes. The global asset manager will provide access to the `Asset` sub-branch, and so forth.

Open questions

- Regarding `Link` and `DynamicContent`, in the current model these categories don't really yet seem useful. Should there be some explicit structure in the API regarding what a `Link` is ? Should there be explicit classes for common types of dynamic content, e.g. `RSSFeed`, `WebServic`
.
- do the different categories of content have different structures in the repository ? e.g. a link, if it exists as an explicit type, should be explicitly structured in the repository, and not a formless XML String, as opposed to a piece of XHTML content.

Comments by AndreasHartmann

- I woudln't model a document as a collection of content items. IMO a document should be dynamically assembled, based on a specific content item. The document is generated by resolving references to other content items and external resources.
- I wouldn't introduce specific classes for different types of content items. There should be special components (content types) which implement the behaviour. The actual content item is just a storage facility.