ArchReviewSynthesis

Synthesis of architectural review discussions

- Version: **26 Feb 2007** (create 31 27 Oct 2006)

This page has a summary of recommendations, group action items, and proposals from the group. It was populated based on the notes from the weeklong architectual review meeting in October 2006. For the full architectural review group recommendations, see the report published in January 2007.

Working principles: We recommend the working principles stated earlier, with some small changes made on Monday.

1.x branch: We recommend that support be given for Manakin out of the box in later 1.0 releases, using a very lightweight add-on mechanism.

User interface: We recommend that version 2.0 not depend on the JSP mechanism for user interface, but use systems that allow more decoupled composition of business logic and UI, such as Manakin.

Interoperability: In the interests of interoperability, we recommend:

- A published data model that can be fully imported and exported
- A core interface capable of supporting application with a wide variety of present and future protocols
- A standard distribution that includes applications for certain widely used protocols.

Scalability: Our goals: We recommend that DSpace be designed to scale to at least 10 million items. We recommend that it not impose limits on the size of content files. (The protocols and third-party software used by default DSPace might impose some limits of their own.) We recommend that ingest should be fast, and that it should take no more than 1 second of overhead (not including transfer time) to import an item in a 10-million item repository. (Concurrency: 10 concurrent updating users, 100 concurrent reading users.)

We do not believe that performance issues in the current DSpace implementation are at their heart architectural, but may require implementation changes, such as improving indexing, caching, and performance for browsing.

We believe that the event mechanism we recommend below will improve scalability as well.

We intend the system should continue to be implementable across load-balanced clusters of servers, and we intend to keep the design compatible with this goal.

Data model: We recommend that the basic item data model be revised so that items contain multiple manifestations (a concept that replaces bundles, and is meant to correspond more closely to manifestations in the FRBR model), and that each of those can contain multiple content files (a renaming of the bitstream), and that items, manifestations, and content files all have persistent identifiers, and can all have associated metadata records (possibly more than 1).

We also recommend that the data model support a sequence of revisions of Items known as *versions.* Versions can be used to model corrections, format migrations, and other modifications of essentially equivalent semantic content. Identifiers exist for specific versions of items, and for manifestations and content files within those versions. Identifiers without a version designation refer to the latest version of the item.

To date, identifiers have been Handles. For now, we recommend that this continue as the default persistent identifier mechanism, but that it be possible for other types of persistent identifiers to be used instead. (I.e. we should not make architectural specifications that intrinsically tie the system to Handles.)

EPeople should also be persistently identifiable via valid URIs. (These could also be Handles in the out-of-the-box version.) The URIs do not necessarily have to be resolvable.

We note that Communities and Collections are very similar, and may be worth thinking of as two different forms of a more general Container concept. We have thus far stopped short of simply replacing them with a Container-- right now Communities only include Containers, and Collections only contain Items, and we weren't sure at the meeting whether breaking this assumption might break things in bad ways-- but the architecture may move in that direction (it may be useful to support Containers that can include both Containers and Items).

Concrete data model recommendations: Metadata is maintained in the persistent store, and can be extracted in serializable form. The system will maintain default metadata schemas at all 3 levels, and other metadata can also be added. Views of the metadata will be cached (e.g. in the relational database) that are efficiently and intelligibly accessible by the user interface and other applications. Issues of how these views are to be maintained is a matter of further design, but we propose that the event mechanism may be a useful way to maintain consistency as content and metadata gets updated. There should, at the very least, be a way to provide crosswalks to basic Dublin Core (not necessarily including all metadata, but enough to satisfy basic requirements of browsing, OAI DC data, etc.)

For more extensive changes in expressions, we recommend that relation metadata be more centrally supported in DSpace applications and user interfaces.

Events: We recommend an Event mechanism be included as a central component of DSpace. Events get raised in response to changes to data, and possibly other phenomena. Listeners can be registered to respond to different Events, in a publish-subscribe model. (The History mechanism would be one such listener, for example.) Larry Stone has a prototype that looks interesting; the the framework extension and workflow systems we recommend may also be involved in Event management. More details to come.

Plugin/Extension framework We recommend that DSpace adopt a general framework for adding extensions to the DSpace core. Since there are already a variety of third-party open source frameworks available, we recommend that DSpace adopt one of these instead of building one from scratch. We have designed a subgroup to review and recommend possible options (see Action items below.)

Among the requirements and desiderata of the framework are:

- Fine-grained version dependency support (to ensure compatibility with core and other extensions)
- Support for separate update of different modules
- Compatibility with our open source license
- Access to persistent storage (including extension-specific persistent storage)
- Compatibility with distributed environments
- Java support
- The ability to reconfigure without recompilation (reconfiguring during runtime could be nice too, but might nto be realistic)
- Support of relevant and applicable open standards
- Ease of use and configuration. (It should be easy to integrate with Manakin, for instance.)
- If the framework is also used in related library / archiving / higher-ed applications, so much the better.

(More details in Friday's notes.)

Workflow: We recommend adopting a third party API to implement a generic workflow system in the backend, then support a serious of tailored XML UI aspects to leverage the generic API for specific workflow needs.

Among the requirements and desiderata of the workflow engine are:

- Ability to support full curation lifecycle. This includes both supporting ingest workflows supported in Dspace 1, but also workflows for later lifecycle operations supported by DSpace (e.g. creation of new versions)
- Compatibility with our open source license
- Support for changing workflows at run time
- Ability to have different workflows for different communities and collections
- Compatibility with distributed environments
- Ability to receive and react to DSpace+Events
- Dovetailing with the modularity/extension framework
- The engine itself does not need to have its own UI, as long as it has a well-defined API that can be built on.

(More details in Friday's notes.)

API review and documentation As per the manifesto call for a stable core, the APIs of the core should be carefuly reviewed and documented in the new design. The choice of framework and workflow managers will affect the exact composition of these APIs.

Action items:

- **John** will shepherd the production and publication of a roadmap document. (See the report.)
- **Rob** will produce an anonymized version of the survey for public consumption and citation from roadmap.
- (ds-framework)ds-framework **Mark**, **Richard Rodgers**, **Jim** and **Graham** will produce a recommendation for a framework for integrating implementation modules (for DSpace core, applications, and third-party addons). This could be something along the lines of OSGI, for example. If Manakin is standard in 2.0, it would use this mechanism rather than the simple ad-hoc addon mechanism developed for 1.x. The group will produce a recommendation based on the requirements list and on discussions with related projects. The group would also produce challenging use cases on which a prototype implementation would be tested, and then reviewed by the forthcoming architecture oversight group.
- There would be a follow-on architecture oversight group that would review and check the detailed designs and specifications produced by the DSpace+2.0 implementors based on the recommendations of this group.
- (ds-workflow) ds-workflow **Scott**, **Richard Jones**, **Gabriela** and **Mark** will produce a recommendation for a workflow management system.

Proposals:

- Fleshing out the concrete data model (**Rob** will be happy to work on this idea)
- A follow-on **architecture oversight committee** would review and check the detailed designs and specifications produced by the DSpace+2.0 implementors based on the recommendations of this group.