ArchReviewIssues

This is a Wikified version of the architectural review issues identified by the DSpace
ArchReview group. This Wiki version lets us adapt this list and create additional pages to track what issues we are working on, who's working on them, and what we're doing with them.

The OLD ISSUES PAGE IS HERE OldArchitectureIssuesList

Historical Review

The current thinking about a next-generation architecture (what came to be known as DSpace+2.0) was articulated in Rob Tansley's presentation to the user group meeting in March 2004. In it he identified three 'areas for improvement', which were 'modularity', 'internationalization' and 'preservation'. Very roughly, each of these triggered a redesign proposal of the current architectural stack. From the 'top' of the stack:

*__Internationalization__: Cocoon was to replace the existing Servlet/JSP web UI affording much
easier user modification, including language and site localization.

*__Modularity__: the 'business logic layer' APIs and services were to be replaced with interfaces that more rigorously obeyed plugability and separation.

*__Preservation__: the assetstore/RDBMS(metadata) dichotomy was to be replaced with a unified 'asset store' containing AIPs with metadata and content packaged together.

Several interesting things to note:

- scalability was not a primary objective, although an (OAIS) AIP-centric storage design was held to scale better in addition to being conducive to preservation efforts.
- the Manakin project, now well under way, has goals and technology close to the 2.0 vision. Moreover, extensive work on the existing JSP UI for internationalization has resulted in a 1.4 release with 14 language packs.
- existing efforts in DSpace, notably the Plug-in Manager and the Add-On mechanism (which is still being defined), have addressed many areas of concern from the modularity proposal.

Scalability

Members expressing special interest: RobertTansley, JohnErickson, MattCockerill, RichardRodgers, GabrielaMircea, HenryJerez, RichardJones

In the meantime, new areas for improvement have been raised by the community. Perhaps foremost among them are a set of concerns which concern 'scalability'. These concerns are worth elaborating because they reflect some different underlying issues. Some examples:

- the responsiveness of the Web UI as a function of repository size. Note that this is different from responsiveness as a function of concurrent users, requests, etc (load).
- the duration of certain administrative procedures, such as batch imports, appears to increase in a non-linear fashion with respect to the size of the datasets involved, suggesting a non scale-optimal design.
- the ability of the system to operate with large individual content files. Early limitations in the RBDMS schema (4 gigabyte maximum per file) have been rectified, but the ability of the system to serve very large files via http remain.
- the responsiveness and style of the Web UI as a function of the number of container objects (communities and collections).
- the lack of functional partition or independence among parts of the system: e.g. an OAI harvest seriously degrading the Web UI performance.

DSpace 1.4 has taken a few modest steps in the direction of addressing these scalability issues:

- 'lazy' item initialization. Until version 1.4, whenever item data was requested (e.g. in search or browse result lists), all DSpace objects related to the item were retrieved from the database along with it - e.g. the submitter e-person, all the bundle and bitsteam metadata, etc. Since these were never referenced in most contexts, a lot of superfluous database lookups occurred.
- context 'flushing'. DSpace context objects (used in every operation) never released their Items, Bitstreams, etc. While this led to efficient Web UI operation (in that objects could be reused), in batch they could consume vast amounts of memory, starving the Java Virtual Machine.

Various parties have also begun to analyze using caching in the Web UI, but this has been a lower priority, given the possibility of Manakin replacing it. There is undoubtedly much more to be done, from database schema tuning to additional caching, etc., but first a more rigorous analytical structure must be devised, with a reference repository dataset (that can be scaled up and down), and tools, harnesses, etc allowing accurate metrics to be collected. Creation of such a structure is a particular challenge for an open-source project such as DSpace, since most developers must focus on local, specific needs, rather than generic, hard to quantify characteristics.

A closely related area for design work is an examination of the current architecture with regard to network distribution of components to allow greater scale. If viewed as a J2EE-like system, DSpace could allow multiple cooperating servers to acheive the functionality now confined to a single server instance.

Modularity

Members expressing special interest: Nearly everyone. We'll want to differentiate roles, aspects, and interests here to work more effectively.

Progress in modularizing the Web UI for DSpace has been significant, thanks in large part to Manikin and efforts to design an add-on mechanism. Providing better guidelines to developers on how to enhance the system in low-impact ways will also help. However many parts of the system still require extensive changes to the central database or core classes of the system, and would benefit from more appropriate APIs to the core system functions. Modularity is often associated with the implementation of clear APIs or other protocol handlers to promote 'separation of concerns' in the system, but this can have a negative effect on scalability in the system, another design goal of equal importance. Examination of existing extension frameworks such as Spring, Eclipse, Excalibur, etc. would follow this analysis and lead to a recommendation of whether an existing framework would suffice, or a DSpace-specific mechanism (e.g. further work on the currently proposed add-on mechanism) will be necessary.

Information Architecture / Abstract Data Model

Members expressing interest: JohnErickson, RobertTansley, HenryJerez, RichardRodgers, GabrielaMircea, JimDowning, TimDiLauro, JohnMarkOckerbloom, Mark Diggory, RichardJones

Revisions proposed to the basic information architecture include

- versioning support: See VersioningProposal – old discussion
- revisions in the identifier system – see e.g. this old persistent identifier discussion
- more flexible metadata options (often mentioned in connection with METS)
- support for relationships between bitstreams: see BitstreamRelationships
- more robust content format support
- alignment with the JSR170 data model
- revisiting how aggregation is modeled (currently done via Communities and Collections). May dovetail with JSR 170 approach

There may be other issues here as well. Decisions about the abstract data model will affect design revisions of the Asset Repository model.

Asset Repository and Concrete Data Model

Members expressing interest: RobertTansley, HenryJerez, ScottPhillips, RichardRodgers, MattCockerill, TimDiLauro

At the level of implementation,
several preliminary efforts have been made to begin to define a new storage-layer API, and simple prototypes exist. However, less effort has been devoted to the question of the (OAIS) AIP standard itself, perhaps because METS has been assumed. This still leaves open many questions of metatdata standards (MODs, etc) within METS, and further questions of how the AIP will be managed over time, i.e. under a series of transformations that result in changed content or metadata. A draft proposal for a METS-based AIP is available (AssetStore), but a detailed discussion of it in this context is warranted. Moreover, the contentious issue of synchronization must be addressed: if the authoritative version of metadata resides in the AIP, then how are indexes, replicas, etc used at the application layer to be managed?

Lifecycle Management

Members expressing special interest: JohnErickson, RichardRodgers, TimDiLauro, RichardJones, MarkDiggory

Another large set of design questions which should be tackled are those connected with workflows, i.e. time- and role-based action sequences operating on repository content.
The area of the most frequent of user contributions is that of tools to customize the submission workflow. But that flow is only one of a large set of workflows that DSpace must support.
There are other ingest workflows, collection management workflows, preservation process workflows, etc all of which must be supported by the platform.
The design questions concern representation issues (how to encode a workflow), how to report, audit, automate.

Authentication and Authorization

Members expressing special interest: JohnErickson, HenryJerez

The granularity of authorizations permitted in DSpace is too high-level and needs better definition.
Roles and permissions are conflated in the current system.
No good management UI or other tools exist to create and maintain e-people and their roles and permissions.
The relationship of DSpace authentication and authorization to emerging standards eg Shibboleth, XACML should be defined.

Administrative/Curatorial subsystems

Members expressing special interest: MarkDiggory

The DSpace administration UI is very poor.
It doesn't expose all the curatorial functions needed for the system.
It provides no way to define local policies for the DSpace-based service or means of enforcing them. It has no serious reporting infrastructure to inform curators about the state of the archive and its contents.
It provides poor tools for editing metadata about collections, items and bitstreams, nor does it have good support for withdrawing and deleting items from collections.
Some provenance metadata is captured by the History system (which is undergoing significant work at the moment), but this metadata is largely unavailable to curators as a means to help them manage the archive. While this may not have major implications for the DSpace architecture, any significant development effort should include a review of the functionality and user interfaces provided to this critical part of the system.

Search and Browse

Members expressing special interest: ScottPhillips, RichardJones, GabrielaMircea, Mark Diggory

Something here about how search depends on Lucene (which is working out fairly well) but browse is custom built and uses the database tables and is in dire need of being completely rewritten, possibly to leverage Lucene further, or to adopt some other search engine that would be better for this. Maybe this is the right place to bring up SIMILE (RDF/Longwell) for facetted browsing?