ArchReviewNotesTues1

Data Model

Bitstream formats have no identifier other than the short name; this is a problem.

Changes to data model we need to consider: -

- Versioning
- Identifier system / use of identifiers
- Metadata
- Structured metadata
- Bitstream metadata
- Better typing of bitstreams
- Bitstream relationships.
- Relationships between sets of bitstreams (e.g. new series of pdfs is derived from older series of pdfs)
- Content format support
- Aggregation
- JSR170
- Terminology

Versioning

Versioning (revisioning) of collection & communities is out of scope.
Need for logical item changes that are longer than a Web UI Request response cycle. (Need for long lived transactions - will this use pessimistic / optimistic locking).
If bitstreams are removed - are old revisions retained? Policy decision.
Subversion model where item revision number increments every logical item change.
Each bitstream has two identifiers: one for 'HEAD' revision (content may change over time), one 'permalink' to the exact revision (content will not change over time).
Need use cases for variant versioning - some on the wiki. Tied up with relationship metadata.
Richard - citation context is the important aspect here - this runs into the application of identifiers.
John E: should version number be in identifier? Could persistent identifier scheme point to a record describing all the revisions?

Metadata needs to be included in revisioning. This could be achieved by modelling metadata as a resource within the Item.
MacKenzie: user interface and usability problems if every small changes prompts a revision change.
Scott: How about a description of the size of each change?
An interface issue

Identifiers

User survey indicates that we need to consider other identifier schemes than Handle.
Henry: need to encourage the commitment to persistent identifiers
Use of identifiers is a policy issue - archival uses need less functionality from an identification mechanism than an IR where content moves.
Locking in to Handle goes against manifesto - although Handle is OS, use of the system is not free. Patents apply to the RFC, but reimplementation is permitted.
John MO: Abstract persistent identifier as (service, namespace, resource id)
Henry: P2P schemes using digest id don't have namespaces.
Could be modelled using a null namespace.
Rob: services layered on top of opaque string: resolution, minting ...
Mark: JNDI is a standard Java API that provide these kind of services - could this inform our thinking about identifier schemes and resolution service design?
Each content entity will have an identifier associated (although not necessarily an actionable one).

Metadata

Structural

Use cases

- Same-time variants
- Versions that change over time
- Complex objects
- Datasets

Intellectually different content (FRBR work) should be new items? Or is this a policy decision.

Item = work. Expressions and manifestations within item.

Use of METS structure maps and bundle STRUCTURE to reference?
What metadata are we going to support in the architecture and the UI?
Metadata must be applicable to any entity in the data model.
Heterogenous metadata support: support directly or via crosswalks to DC?
Tension between flexibility and complexity in application layer and UI.
A middle ground might be to allow modules that can support other metadata formats for UI, search/browse, PMH etc.

Every item in DSpace has a canonical structure described in a METS manifest? There may also be a user defined manifest for a complex object.

Does the metadata about relationships between bitstreams belong in the DSpace manifest, or should it be in the user defined manifests. Relationship data pushed down into item specific metadata format, and a simple set of relationships supported in "core".

Henry, Richard J, MacKenzie, Rob Tansley interested in taking metadata mechanisms forward.

Half baked

Whether to keep bundle mechanism?
Important dates on items needs to be richer than Last Modified.
Requirements for a plugin framework

Interfaces / Modularity

Add-on mechanism wasn't intended to achieve all the aims of a full modularity mechanism.
Listeners to decouple Item update dependencies?
Interdependencies between a number of packages are difficult to resolve.
Storage logic in entity classes breaks layering, makes storage abstraction hard and unit testing difficult.

Modification cases

- JSPs
- Tag libraries
- Optional UI features
- Servlets (Model, Controller)
- Workflow (ingest)
- Input forms alternative metadata schemes
- Index / browse support for alternative metadata schemes
- A&A
- Extensions with persistent data storage
- Extending all 3 layers together

Pain points

- Can't extend content classes.
- Hard to extend the database in a way that isn't brittle to updates.
- HTML generation in tag libraries
- Maintenance of 3rd party extensions that aren't appropriate for the main release

How do new applications declare and effect their requirements for persistent storage?
Need to separate out user interfaces (CLI, webui etc) from business logic.
Are we replicating logic for interating over items etc?

Add-On Mechanism

DSpace is treated as an addon itself. Incremental build process, starting with dspace and adding other components. Rollback of add-on versions probably too difficult to be worthwhile.

Add-On mechanism eases pain of 1.x architecture - the worst of these should be removed by the new architecture rather than continuing to mitigate in this way.

Manakin

Addresses 3 main pain points: -

- Upgradability
  DRI Schema (developed as part of manakin development)

- Modularity
  Manakin Aspects

- Uniformity
  Manakin themes

Including both JSP and XMLUI as add-ons won't complicate build / installation process for users.

Needs for modularity mech: -

- Configuration per extension
- Persistent storage per plugin
- Versions of individual parts of API

OSGI

(Open Service Gateway Initiative, but abbreviation is anachronistic)
Ability to build complex sets of services with versioned dependencies at runtime. Runtime not needed for DSpace, but doesn't hurt.
Standardized manifests: dependencies and what services are provided.
Programming model is strictly service oriented. Using OSGI would enforce a service model on DSpace.

Maven

Could be useful for supporting uniform development of DSpace and 3rd party extensions.

Architectural issues

Is moving to an OSGI based architecture "starting from scratch"? Not really, as implementations and system scope exist currently.
Mark: Extract interfaces, refactor towards OSGI, define a roadmap.
Richard J: Functionality changes to data model etc will break compatibility anyway - changing the data model will be a "big bang" change.

MS:
manakin removes some of the pain points along with an addon mechanism
Manakin avoids some of the more difficult requirements on the addon mechanism.
Data model changes and architecture refactoring

Config handling in Addon mech in need of improvement. Commons configuration could help with this, but some rationalization of some of the existing config properties would be necessary.

Are there alternatives to Manakin?
Are there alternatives to OSGI?
Are there alternatives to Maven?

Is manakin compatible with add-ons?
Can't assume we have the mechanism because the development wasn't finished due to Richard running out of time.
Could we have a super-simple add-on mechanism that doesn't deal with config merging or any of the harder issues as an interim measure? This would be a quick way of incorporating manakin into the main release. Do this for 1.5?

Define a list of requirements for a framework this work, research and discuss appropriateness of alternatives (OSGI, Spring, homespun) over next few months. Once framework is chosen, extraction of interfaces and refactoring to framework is a resourced project.