I. Review of Data Model (Rob)

See also: DSpace Data Model

- Communities
- Collections
- Items
  - all items have a Submitter who is an ePerson
  - MDRRecord (flat: name, value pairs)
  - fields
  - metadata schema
- ePersons
  - permissions for items
- bundles
  - name
- bitstreams
  - size
  - checksum
  - name
  - description
  - format
  - sequence number
- bitstream format
  - name
  - long name
  - mime/type
  - file extension

II. Problems with the Data Model

1. Versioning (Rob)

- Rob's versioning idea (item-level down)
  - "snapshot" of an item at a instant of time
  - coarser-grained notion of "transaction"
    - "logical item change"
    - makes dealing with "events" easier
    - sort-of wiki "locking" model (i.e. very short-term)
  - concept: items are immutable
    - previous "versions" never go away (maybe policy driven)
    - encapsulating higher-level item object
  - What about e.g. Pre-/Post-print versioning?
    - discussion about identification syntax for versions...
      - also, Handles and versioning
    - (ms) issues of complexity presented to users
      - policies concerning displaying major and minor changes
      - keeping around all copies, etc

2. Identifiers

- Q: should DSpace dictate the identifier system?
  - if so, should that be HS?
- Rationale: concern over long-term interoperability
  - avoiding "Tower of Babel"
- separate issue: if the org has a HDL prefix, it needs to be a HDL
- concern over long-term, exit strategies, etc...
- (hj) HS is an RFC and CNRI patent protects implementations thereof
- (jmo) (service, namespace, resource_in_namespace)
- (Rob) if we treat everything as uri or opaque string...

*RECOMMENDATION*: Each CONTENT COMPONENT should have some sort of PERSISTENT IDENTIFIER associated with it

3. Metadata Flexibility Options

- Structural (e.g. METS)
  - today, no way to specify structure and relationships
- Descriptive
  - today, item-level descriptive MD
- Representational
- Binding metadata to structure

- Use cases (high level):
  - Versions (alternatives)
  - Versions (versioning)
  - Complex file structures

- (MS) from a library perspective, whether something is a unique work or not
- Lengthy discussion of metadata as bitstreams
  - and TYPES of bundles
- (hj) should DSpace allow item-specific md models?
  - reaction in room: wow, huge implications
  - (Rob) crosswalks required
  - problems with user interface
  - e.g. have subsystem whose job it is to deal with MD in specific way
- (Rob) all of this is possible with the current architecture
  - media filter to convert whatever the scheme is into DC (e.g. for OAI)
- (MS) ultimate model would be RDF, someday

**RECOMMENDATION:** Always must be able to CROSSWALK to DC

- mechanism should be the default (DC)
- but MD typing mechanism/bundling needs to be extensible
- currently there are examples (oai, mit 'dwell') that allow asking for specific
- JD: Right now, flat metadata structure is the bottleneck
- (RJ) Perhaps we need to think about what it means to be an item in DSpace
  - a specification we put out
  - (MS) Larry Stone's "IP" proposal
    - canonical components of a "DSpace Item"
    - i.e. a manifest of the AIP, with structure map
    - then, arbitrary complex objects
    - MIT's purpose is for interop with SRB

**RECOMMENDATION:** Put "whether to keep bundles or not" on Half-Baked list

4. Relational Metadata

- inter-item relationships
- intra-item relationships
- between bitstreams, bundles, etc
  - sets of bitstreams to sets of bitstreams
- between objects

- Q (Rob) Bundles and bitstreams?

5. Content Format Support

6. Aggregation

7. JSR-170

  - versioning?

8. Terminology

  - bitstream vs datastream vs...

III. Interfaces and Modularity

1. Review of the APIs (Rob)

2. **Pain Points:** Enumerating the reoccuring types of mods that break things

  - JSPs
  - Servlets
  - Ingest workflow
  - metadata extensions
    - esp. adding new fields
    - downstream indexing
• browsing
  • Authentication
    • issue: synching with ePerson database
  • Authorization
  • Code Protection on content classes
    • i.e. for extensions on content classes
  • Persistent data store for extensions

3. Are we going to decide to stay with servlets and JSPs
   • or discard and move toward e.g. Manakin?
   • To be discussed (below)

4. Much discussion of the current layering ("Application"/"Business Logic"/"Storage")
   • (MD) We need to understand why certain code keeps getting replicated

5. Overview of the AddOnMechanism (RJ)
   • See [AddOnMechanism Wiki](http://wiki.dspace.org/index.php/AddOnMechanism)
   • See [AddOnMechanism presentation](https://bora.uib.no/bitstream/1956/1156/2/presentation-1.0.pdf)

6. Summary: What should we be able to do without changing code? (Rob)
   • Add persistent storage for customizations
   • Add new UI pages, link to new pages from existing pages
   • Modify existing UI pages
   • Modify workflow

7. DSpace Manakin Overview (SP)
   See: [DSpace Manakin Wiki](http://wiki.dspace.org/index.php/Manakin)
   • Pain points
     • Upgradability
     • Modularity
     • Uniformity
   • Aspects and Themes
     • Aspects contain Java source code, static resources, Cocoon's sitemap
   • Manakin solves these pain-points:
     • JSPs
     • Servlets
     • MD extensions (certain cases)
     • workflow (UI aspects)
   • Draft Recommendation (SP)
     • first, embrace the AddOnMechanism
     • proposed road map:
       • 1.5: JSPUI full support, initial version with XMLUI
       • 1.6: JSPUI full support, XMLUI full support & rec'd
       • 1.7: JSPUI depreciated, XMLUI full support & rec'd
       • 2.x: XMLUI only

“What are the alternatives to Manakin?”
   • Are there potential incompatibilities with other of these frameworks?
   • (sp) Could put Manakin into the same source tree as JSPs
   • (Rob) Is there a really-really dumbed-down version of the AddOnMechanism that could be put in to the *Main tree*
     • (sp) “A day's worth of work...”

8. OSGi Overview (RR)
   • Open Services Gateway Initiative
   • See esp. OSGi Technology web site
   • See OSGi Technical Whitepaper

“What are the alternatives to OSGi?”
   • Spring Framework
   • RJ's AddOnMechanism,*but it isn't complete*
     • (ms) Strategy: Choose framework, get resources, do analysis to identify APIs, implement...

9. What about Maven? (gt)
   • “Maven is about the application of patterns in order to achieve an infrastructure which displays the characteristics of visibility, reusability, maintainability, and comprehensibility...”
   • “Maven uses a declarative approach, where the project structure and contents are described, rather then the task-based approach used in Ant or in traditional make files ... This helps enforce ... development standards and reduces the time needed to write and maintain build scripts...”
   • Another option to consider for making the add-on build process a “little easier to do”

“What are the alternatives to Maven?”
• Ant
• (jd) Maven does what it wants to do
  • (rr) It enforces "patterns" across an org (see above)

10. What is our recommendation?

  • where to focus resources?
  • should the focus be on major refactoring?
  • should the focus instead be on key pain points (e.g. persistent storage)?
  • (rob) multiple trajectories
  • (md) need some reorg of code base
    • keep dependencies separate and isolated
    • once isolated, then refactor/define the interfaces
    • fix what breaks
    • use the tools (e.g. Eclipse)
  • need to experiment!
  • (rj) But if we refactor the information model, we'll need to refactor the core anyways
  • (jd) we're not starting from scratch!
    • need to define the goal and go there

11. Break-time discussion of what level of difficulty refactoring should take on, and how it might be managed...

12. (MS) Attempt at summary

  • Manakin with an AddOnMechanism addresses a lot of the pain points
  • See above: Short-term solution putting simplified AddOnMechanism in*main tree
  • A refactoring will be required with the refactored information model
  • There will be a 2.0 with an AddOnMechanism more like OSGi (see JSE questions above)
  • (MS) RJ's approach is a short-term but not long-term
  • NEED: "Plug-in framework" (e.g. OSGi) plus "build framework" (e.g. Maven)