ArchReviewNotesThur

Contents

Thursday Notes

26-October-2006, Richard Jones

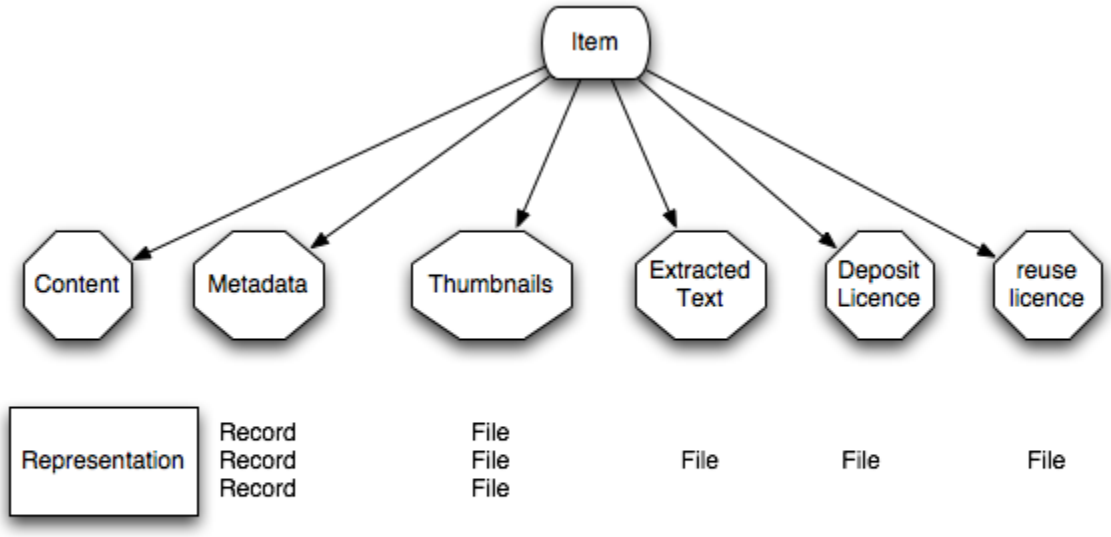
Outline of the day

- Look at authorisation approach as part of workflow study
- Data model
  - aggregation
  - bitstream relationships
- Concrete data model/storage
- History, provenance, audit
  - Admin, curatorial -> workflow

Morning Session

Data Model

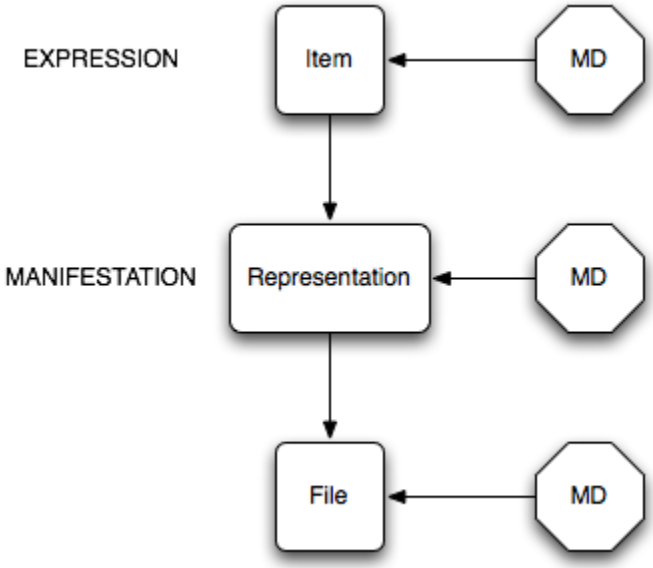
RT proposal for data model:



Discussion outline:

- RJ: This works only horizontally; / RT: Structure of representations is done by the Representation Metadata package

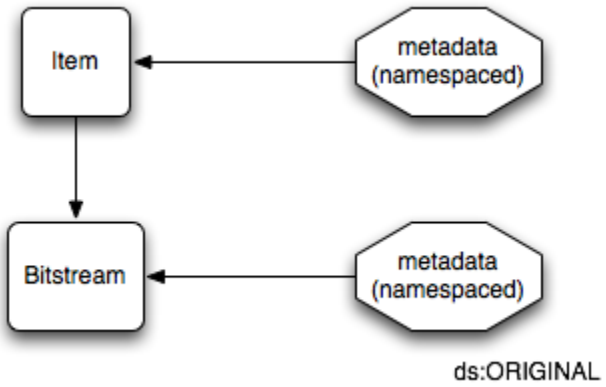
- JD: FRBR model: work->expression model / RT: departure from current system, may be confusing
- MS: attach attributes to the Files to explain to the system how to use them
- RJ: manifest file? / JD: this enforces the concept of an external metadata representation / MS: the system *requires* a metadata record / MS: lifecycle management issue requires that the data is there - doesn't file embedded metadata make the job harder / JD: yes, but that's concrete level
- RT: it's too difficult to put metadata in the relevant metadata buckets.
- JD: metadata needs to be a property at each level, not a separate entity at the top level (more FRBR) see diagram 2 below
- JD: make the model map to FRBR and let that squish; JMO: FRBR may not be appropriate in our use case at every level
- RJ: can we represent the "work" by collecting Items in metadata. MS: Item is an Expression, Representation a Manifestation, File = part of Manifestation



Summary:

We moved rapidly from the initial proposal to a more general approach. Diagram 2 shows the distilled approach where metadata is attached at each level, and the Item is equivalent to the FRBR work and the Representation is equivalent to the Manifestation. It was also articulated that we still need to represent relationships beween things in this model, and whether this sits in some sort of Item Manifest is one question that was asked and debated. The sorts of relationships that might be necessary were enumerated, and the benefits of adopting something like the DCMI terms was suggested, but largely rejected because it was too specific. Further discussion of this topic continues below ...

RR proposal for data model:



Discussion outline:

- RR: Bundle fulfills a particular software usage, such as ORIGINAL tells the system to display the files, and not other bundles. They are also Types.
- RR: Metadata is stored only at the bitstream level / MS: this means I have to look through all the bitstreams to construct the bundle / RJ: if we move this up to the manifest, then we overcome this problem / RT: How do we do ordering in this schema?
- MS: RT's model is FRBR, RR is not / RJ: actually, it depends on your interpretation of the data itself / therefore these are pretty much the same
- RR: what about files appearing in more than one Bundle?
- MD: this is a containment vs referencing problem. This is a model decision, which direction you go. JD: prefer the concrete implementation, graph based structures are too difficult for users. RR: we need to be able to surface "Representation" level information at the UI / RT: for versioning to work, we need to give the Representation level *at least* one ID. Also, cannot enumerate the future structural information models that might come to exist.

- JD: Deliverable: We have decided to create a new abstract concept, and to be able to assign identifiers to them. This is a *new* requirement for our system.

Summary:

RR presented us with a much flatter data model, where structure is managed by metadata attached at the bitstream/file level. The concern with this is how you: a) reconstruct the bundle like structure of the item and b) where you attach your identifiers for each contained thing that we might want to reference. There was further suggestion of an item manifest, but this only easily addresses issue (a). It was also finally observed that both RT's and RR's proposals were pretty much identical in terms of information architecture, but would lead to different sorts of implementations, especially that RT's proposal led to a more cocrete interpretation of the structure. At least we now understand the abstract concepts that we need to express, and our requirements for them, which is beyond what the DSpace system can currently achieve.

Use Cases:

RT takes us through the use cases from the wiki DataModelUseCases. Use cases brought up and updated have been added to that wiki page.

- MD: likes the concept of flattening the data model, to deal with large continuums of works. Example given of compound items whose quantity defies their interpretation as collections
- JMO: DSpace current architecture does not (and probably cannot) treat things like JSR-170 nodes because at different levels they are treated differently

- Some example relational models for objects (FRBR, and VRE):

FRBR

- Work
  - Expression
    - Manifestation
      - Copy/Item

VRA (for visual resources)

- Group
  - Work
    - Image
      - Surrogate

- MS: do we support these models? / RT: or are they overlay services on whatever the provided DSpace data model becomes?

Aggregation Discussion summary:

- JD: eradicate community; does not have useful function / general agreement - the semantics for the DSpace community may be useful, but at an information model level is meaningless
- Attach the metadata at every level
- MD: don't use collections (in the previously mentioned context), instead do with metadata / RT: will DSpace use a simple data model that people will be able to understand, or will it be more complex? Until now we have been erring towards the simple, but we are moving in the direction of the complex.
- SP: Bundles do not exist in the interface; a more complex schema which can do the representation, but which is hidden for the UI as bundles are. Gives the opportunity to extend when the situation arises, but is hidden from everyday users. Instead, a bigger mechanism: recursive Items and Representations / MS: is this done in Metadata? / RT: do we replace Item with Object? / JD: ... / JE we are debating the general data model and what the default for DSpace needs to be - the balance is what we need to achieve; JD +1. MD: JSR-170 *is* the super abstract model.
- RT: This is leading towards a complete re-implementation / cf Manifesto 8 / MD: common class that everything extends - this becomes a Node / GT: the representation of the abstract model is our three level implementation, but which does not preculde extension / SP: DSpace *is* the data model - we should not change it, only change it once, and never plan to change it again; do not need to use the super general node mechanism, because that is not the DSpace product. JE: DSpace model is an *application* of the nodes and properties model. RR: JSR-170 is not the only N+P framework, it's just the latest. RJ: we're not getting anywhere - N+P is pretty much implementation, we really just need the DSpace abstract information model.

Summary:

For the most part this consisted of the formulation of the primary recommended data model for DSpace. We looked into the balance between extremely general ways of managing hierarchical information in Nodes and Properties (exemplified in this case by JSR-170), and the extremely specific requirement of providing a useful DSpace information model which could be implemented easily, and quickly communicated and understood without too great a departure from the existing information model.

Making a decision:

Here we attempted to make the final decision on the recommended information model to be worked on. Below summarises the main points raised:

JE: must be possible to support the future options of DSpace implementers, irrespective of our current data model

MS: more turn-key than fedora, less so than greenstone. We can't be so abstract, but we can't always hand-hold

MS: we must make our recommendation on the concreting of the N+P, so that we can write the application, implement inside the data model, and so it can be explained to people to ensure that it can be appropriately applied.

JMO: this is close enough to the current structure: easily explained, adopted, but has more flexibility so that people can

RT: we should go with this right now, and worry about generalising more later on.

MD: how will this model be abused?

JMO: should be possible to misuse if you know what you are doing

RT: versioning in the container model

MD: if versioning is at the item level; for a work with many images, the versioning must be at the images; don't want to version the entire work of the item

JMO: we are converging on diagram 2. Metadata and content are managed together.

MS: do we switch to more FRBR speak? Representation -> Manifestation (agreed)

GT: File doesn't work, because you can't necessarily put every asset in the assetstore

GM: Can EPeople fit into this scheme?

JMO: name changing bitstreams implies functional change, even if it is not. Can we just live with bitstream?

SP: everyone knows what file means

JMO: nominations 1) bitstream, 2) datastream, 3) file, 4) resource

We do a preliminary round of voting for Bitstream name change, where we can each vote as many times as we like:

```
Aggregate Votes: File: 6; Bitstream: 4; Resource: 4; Datastream: 1 (JE wants to be on the record for that being him!)
```

SP: nominate: Bits; gets 4 votes

Next we vote on whether first to switch from Bitstream to something else, then second vote for our other options, with just one vote per person:

```
Single Vote: Change vs Not Change: 9 to Change
Single Vote: File, Resource, Datastream, Bits: 6 for File
```

Concrete Data Model (making the data model more real):

- RR: how do we propose architecturally to enforce the use of Manifestation for their correct use?
- RT: lets take a couple of use cases, and fill out the data model for those cases

Use cases:

1) A PDF (article), with deposit licence, extracted text

Submit: PDF + metadata, agree to a licence. Create a manifestation containing the single file. The deposit licence and reuse licence are metadata, but they may be represented as files ...

This discussion has currently ceased because of an argument over where metadata "files" live. It will be returned to later

Lunch

Afternoon Session

Event Mechanism

- Model
- History
- Listener/Callback

RR introduces the general framework for an Event system

```
Content Model ---> Event ---> Dispatcher ---> Consumer ---> Browse/Search/JMS ...
```

- JMO: how does this work with the submission workflow?
- RR: transactions work differently in this, because Events may violate transaction safety. Therefore, Events that are published are only cleared from the queue once the transaction has been released from the database
- RR: consumers can be either synchronous or asynchronous. Uses a JMS consumer, which places events in a persistent queue, which can then be run by some other process, or by cron at night. / JD: if a consumer raises an exception, does this stop the transaction? / RR: no. The transaction has completed.
- JMO: how does this work with the framework/modules planned / RR: great. It makes it possible to push services out of the content model, then it is just a case of writing the consumer
- MD: what's the content of the event?
- RJ: in an asynchronous environment the transaction termination ruins the context being passed in
- RR: History system will be a strong synchronous consumer, because of the need to be sure of the data chain

- RT: something like this is essential, especially for scalability
- JD: you need different mechanisms for interceptors and watchers
- JMO: if this looks good, we should make the recommendation, even if we leave further research as to how this is developed aside / JD: is what has already been developed by LS and RR ready to go into 1.5 / RT: there is no way currently of modelling what high-level events there are in the current code.
- JMO: doesn't have to be smart enough to work within the context, just needs to be smart enough to understand what has happened leading to the Event
- MD: Two models: single dispatcher that everyone subscribes to; all objects generating events, and listeners decide which they listen to. That is, single point of contact, or multiple points.

LS arrives

Summary: general consensus that this is good, and should be got into the source code as soon as possible. This will allow us to build a History system as consumer, and therefore this enables the advancement of that system.

- Update from LS: rip out the old history system. Search code, updates and browse can go into even consumers. Goal: event system that can be asynchronous: choose whether your consumer is sync or async. At db commit, events are fed into the dispatcher (no consumers get events until after this). Assumption: any modification worth mentioning happens at the database. Every place that makes a change to the database calls "add Event". Future modifications to the data model have to keep up to that, otherwise bugs will creep in. Events sit in the context, then at commit passes them to the dispatcher. We can have any number of dispatchers. This mechanism is atomic - if transaction is aborted, no events are fired.

- Timeline:
  - Context created (includes db connection)
  - Operations cause changes to happen to the db connection
  - events are stored
  - context.complete() - transaction with database is completed
  - events are then fired to dispatcher
  - DSpace context goes out of scope

- LS: The event *cannot* affect the outcome of the transaction
- Some debate over the pros and cons of sync vs async, and how it is proposed to work. General agreement that this can probably go into 1.5, and also prep it up for 2.0 - it is patchable into the 1.5 now, but needs testing / RT: raises continued concerns about sync vs async, especially how this will affect batch processing / concerns over whether JMS is too heavy weight? LS: configure is a bit tricky but not so bad; also option to run as separate daemon, which brings with it other concerns (e.g. keeping the daemon up, and bringing it back up when it goes down)
- Proposed message types on wiki: EventMechanism
- Terminology discussion: Event vs Message?

DECISION: yes! recommend that this will go into 1.5 initially, and become "core" for 2.0. Exact design for 2.0 will depend on framework review, and upcoming event prototype, and there will be a new logical event.

QUESTION: MS: Do we implement a new History system or not? No one is pressing for it, but it could be important. RR: is the History system a detachable, optional thing? We have an architecture that permits this for the first time, so pull it out of the core. General consensus that that is a good idea.

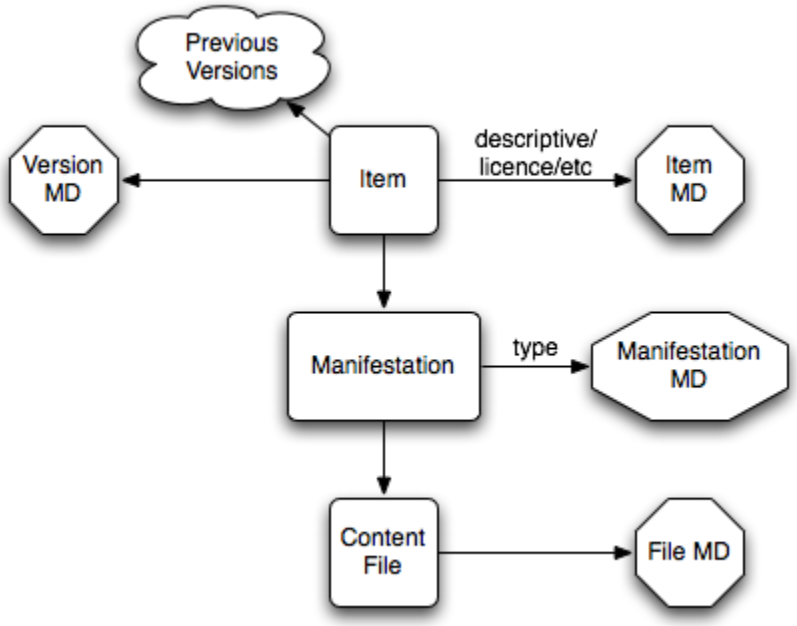
LS: every archivable object needs to expose a URI to be used successfully in any History system
MS: PREMIS is not flexible enough for us to use as an events framework

Question: do we ship with a history system, or leave it out. RR: should do so, and MIT will be on the hook to provide it. JD: provenance is so important these days.

- LS: what do we really want to do with the History system? Do we want to be able to publish this in the AIP?

Return of the Data Model

- Files -> Content Files to avoid confusion between logical and physical
- RT: Versioning was done via identifiers, and is therefore orthogonal to the model in diagram 2. See the below diagram for an updated model:

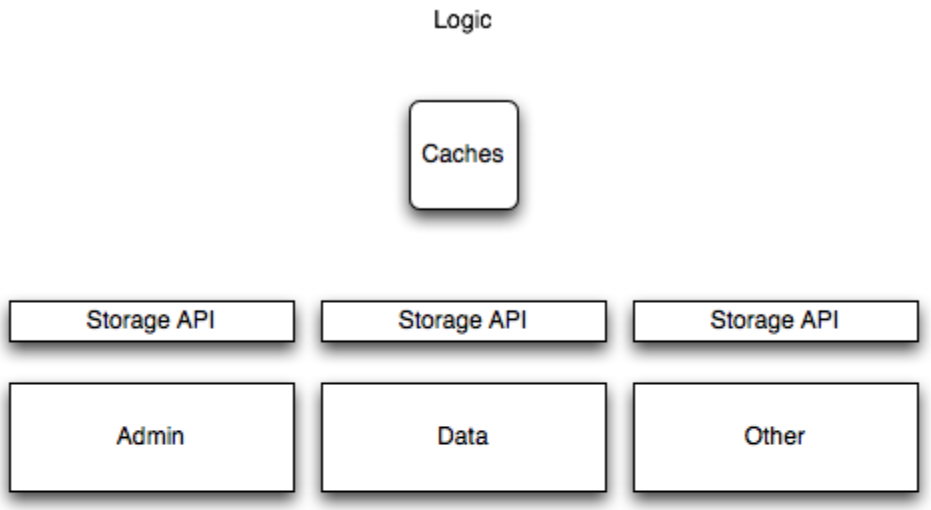


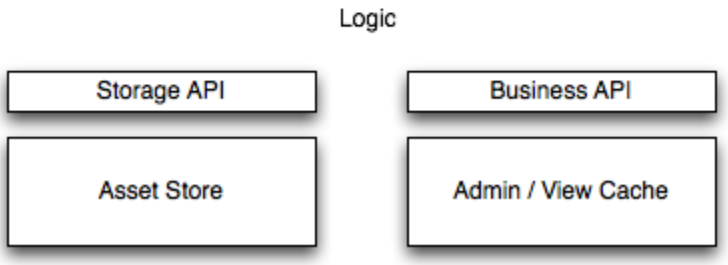
- MS: in addition to storing content files in the asset store, we must also be able to store serialisations of metadata / JMO: is the stored copy the canonical, copy of record / LS: copy of record should be in the asset store for backup / GT: we've had problems with the assetstore, but the database is fine. / RJ: metadata in the asset store can be in a format which cannot trivially be represented in the database / GT: does this have to be an architectural decision, instead make it a policy approach / JD: 2.0 has a storage API - database or assetstore is not the point: this is a storage mechanism question. JMO (by proxy): could be up to the storage mechanism to choose storage, and just to provide files on request. JMO (for real): there are certain types of metadata which you can get to quickly / JD: then we formalise the database as the "cache", and leave the application to sort out how to store. The cache can be built by modules as they need them (i.e. a certain view of the metadata as needed). RT: event mechanism to deal with updating and invalidating cache.

JMO summary: we have a storage layer, which *may* be in a database, but not necessarily, AND we have a database cache which is a view of that data as appropriate.

- MD: what do we do about Groups and EPeople
- RT: 3 types of data: 1) archival data in assetstore 2) Ephemeral Administrative Data (epeople, permissions) 3) Cache of the assetstore / JD: why discriminate at all on 1 and 2? The storage interface can talk about storing epeople just as it does with items - it does not affect the information model / LS: 2 uses of epeople 1) access control and admin metadata 2) belonging to items in an archival way minor repeat of person identifier debate
- JD: exit strategy should include all system data which would allow us to do migration well. / MS: export and hand-off are different. We could support both
- RJ: if admin data is not in assetstore, and is not a cache, then we need a third storage area
- SP: two asset stores, one for admin, one for storage, both with a cache.
- LS: has rights management been covered properly. Must be able to hand off course grained access control/rights metadata RJ: e.g. embargoed items
- GT: rule should be that you ask for the data, and you get it - it is not your concern whether it comes from the cache or the assetstore

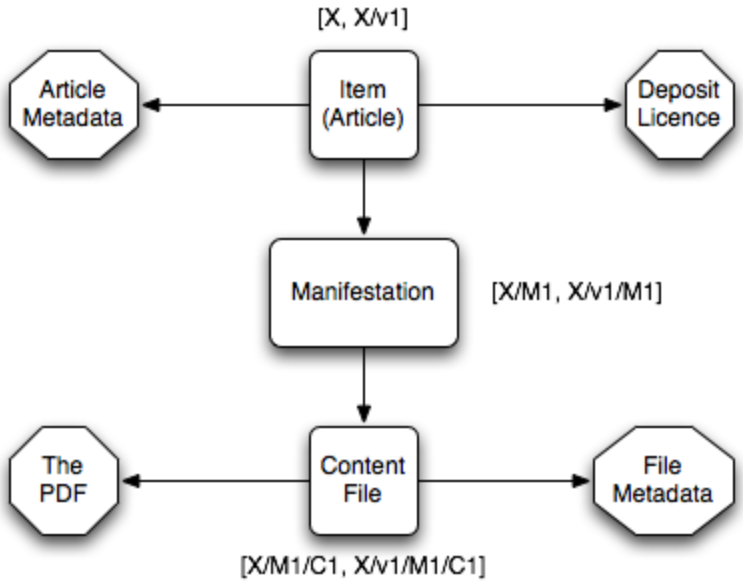
Two possible structures for storage are presented, as presented in the below diagram:





back to use cases

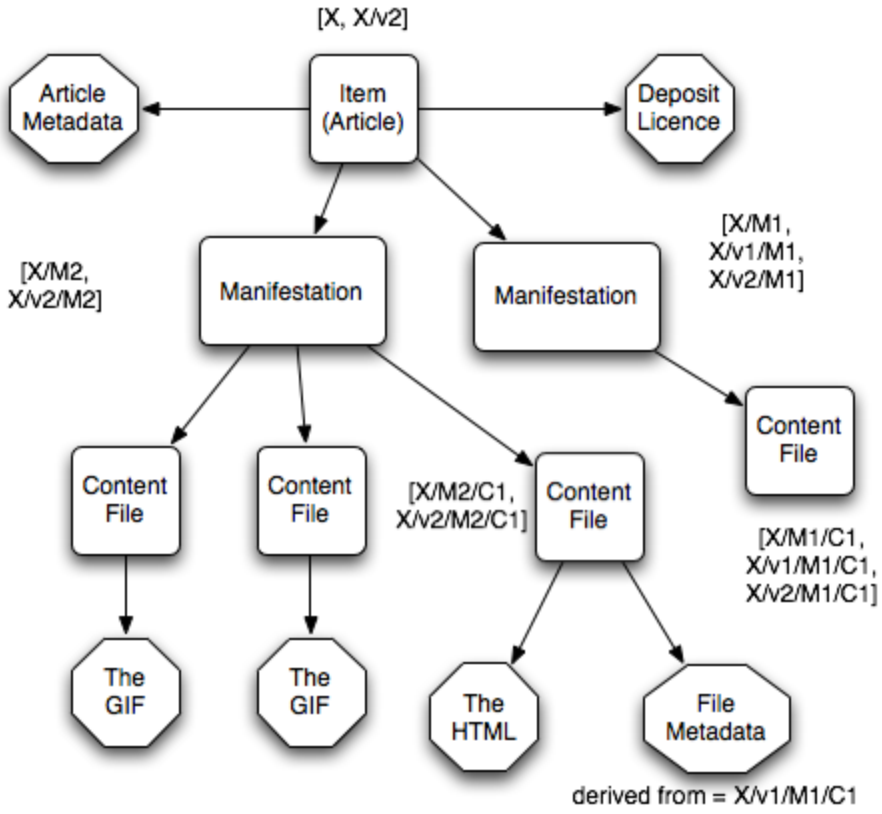
1) as before. see above



- Versions problem: revisions vs variants

2) The PDF is converted to HTML with GIF

- JMO: Do we need to create a new version if we create a new manifestation? / JD: Revision number increments when there is any change.
- JD: note: the manifestations belong to two separate versions of the Item
- Versioning is addressed again; see the diagrams for results



ISSUE: (GT): "derived from:" does not tell you whether the source that it has been derived from (using the version identification schema) has been changed since, given the global version identifier update that happens when a item update happens.

NB (GT): IMHO this is something that needs to be enshrined in the data model. While it's theoretically OK to call this curatorial, the system has to be able to efficiently identify when derived manifestations are potentially out of date - not doing so threatens the preservation model ('dumb' version links would make every existing derived manifestation for an item appear to be potentially out of date whenever any revision is made - such as adding a new derivation). However, we did appear to have a consensus on 'smart' version links that update with revisions providing the manifestation pointed to hasn't been altered - which would resolve this issue.

RESPONSE: This is not the burden of the data model, this is a curatorial operation. Perhaps the Event system can help here?

- JMO: what happens now when we do a media-filter?
  - RT: new manifestation, derived from is set, use = extracted text, or some value

- SP: when to generate new versions. It should be a curatorial decision: i.e. a button on the item page "version this item" / JD: we do need a concept of a working copy / MS: this means that a single version cannot be immutable / JD: the working copy solves this by requiring the commit of an archival copy which is guaranteed immutable / MS: doesn't that mean that you can't cite the most recent version (i.e. the "working copy") / JD: yes, it means exactly that. (but it is addressable by the system) / MS: this opens up options for collaboration / JD.SP: if there is a typo, we still need the concept of a minor edit, so we have a minor/major edit process. New versions are cheap.

IMPORTANT: revisions are cheap; have to have a way to hide them in the UI. Must also be able to distinguish major or minor versions.

- LS: can we create a Site level container to put communities/collections in

DECISION: generalise containers, and investigate the long term needs to have containers that contain different sorts of objects