Don't call it a 'content model'!

The problem

There are at least four different potential meanings to the term 'content model' in the Hydra/Fedora context. This has the potential to create confusion and ambiguity when it comes to content modelling discussions.

1. Content types, like books or ETDs, that users want to manage with a repository
2. Ruby models (the "M" in MVC) that are integral to Hydra
3. Fedora content models (cModels in the Fedora content model architecture available in Fedora 3.x)
4. Atomistic v. compound objects: content modelling patterns for Fedora objects

1. Content types

Hydra will use the term 'content type' to describe a type of digital object that Hydra and a repository should support. This includes both general content types (e.g., ETD's, archival images, articles) and specific implementations of those content types (e.g. Stanford ETDs, UVa images, Hydrangea articles).

Primitives

There is also the notion of content type primitives. A primitive is a fundamental atom object that bears an actual file payload. They are single, (near-) universal content types which may either stand-alone or be incorporated into higher order content types (in a book or ETD, e.g.).

- A standard set of metadata, required and optional (a commonMetadata cModel)
- Generic simple content (corresponding to the genericContent cModel)
- Static image (staticImage cModel)
- jpeg 2000 (jp2 cModel)
- File asset (aka downloadable, aka blob, aka file, aka 'other') -- (the fileAsset afModel, part of the activeFedora implementation of these models. It is a combination of commonMetadata and genericContent and rules to facilitate the management of atomistic objects in Hydra)

Other primitives are being developed for such things as audio and moving image content.

Higher-Level Models

These are models that represent higher-level, molecular objects that may have primitives and/or other Higher-level objects as children.

Eg.

Hydrangea Datasets
    Hydrangea Articles
    ETDs
    Book

2. Ruby Models

The content types described in section (1) are realized in a piece of Ruby on Rails code which we call a Ruby model. This is the 'M' in the Ruby on Rails MVC (model-view-controller) structure consisting of:

- a Ruby 'model' which declares the data streams, metadata, and custom methods in Ruby on Rails
  - must explain
  - * where active-fedora models fit in
  - * primitives
  - Blacklight (Rails views) and object-specific views
  - Rails controllers to extend the Hydra REST API

3. Fedora cModels as part of the content model architecture

Fedora's Content Model Architecture (CMA) implemented as of Fedora 3.x, defines four types of Fedora object:

- data objects: containers for content - the 'usual' type of Fedora object in a repository
- content models: containers for content models that describe the characteristics of one or more data objects
- service definitions: containers that list service calls which can be used with objects conforming to a particular content model
- service deployments: a container for service deployment bindings used to implement service calls

A Fedora content model object is often referred to as a cModel and Hydra will use that term to distinguish it from other types of content model that we might reference.

An extensive description of the Fedora CMA can be found here.

4. Atomistic v. Compound objects: content modelling patterns

The Fedora community has identified three prominent constructional content modelling patterns: simple, compound and atomistic.

- simple data objects contain a single content-bearing datastream
- compound data objects contain multiple content-bearing datastreams in a single object
- an atomistic (also called complex) object has one or more 'child' objects each normally bearing a single content-bearing datastream. All these child objects are linked by a single 'parent' object which is able to identify them and express any relationship (for instance ordering) between them.
These are general strategies for modelling digital objects in Fedora, supported by Fedora cModels, and Hydra will refer to these as object modelling ‘patterns’.

For the purpose of Hydra end users and content contributors, the choice of pattern is invisible, and abstracted away from the user interface via the Fedora REST API, which sits below the Hydra Ruby model, which is manipulated by Hydra Ruby controllers, which communicate via the Hydra REST API with Hydra Ruby views.

The Hydra / Fedora stack is...

Note (1) that Solrizer (the Hydra/nga indexing tool) leverages the Hydra Ruby models. If you don't have a Hydra Ruby model, the best way to use solrizer is to write one or find an existing one (neither of which is overly technically challenging).

Note (2) that disseminators are a shorthand for a Fedora sDef with an sDep against a cModel.

Ruby Code or Fedora Disseminator?

We expect some functionality may migrate from Hydra code to Fedora disseminators over the course of time.

A function may best be coded in Ruby if it...

* traverses many objects, many relationships
* is very complex
* is immature - and may need more time to mature
* is more readily addressed by Ruby code to start with

A function may best be coded as a Fedora disseminator if it is...

* a simple web service that might move out of Fedora
* a basic, universal function, such as support for content type primitives (e.g., image)
* more readily addressed by a Fedora-savvy developer