Community Content Modelling

This page summarises the approach to objects and their content models taken by a range of Hydra partners:

Partner	Contact	Notes	Governance	Generic objects	ETDs	'Simple' images	j2k images	Datasets	Journal articles	Audio	Video	Other
University of Hull	Richard Green	** cModels dataset, journalArticle etc are clones of genericContent just renamed so that they can trigger specific display options *** In the UK, ETDs have 'UKETD_DC' metadata - this cModel provides the datastream **** from DROID and PRONOM All objects have a 'properties' datastream holding such things as depositor information.	Objects are each 'isGovernedBy' a structural set - effectively an admin policy object (APO). These sets give us a hierarchical management structure (like a directory tree) and are the source from which an object clones its rightsMetadata when it is published.	Objects: Simple /compound cModels: - genericContent - compoundObject - commonMetadata	Objects: Atomistic cModels (parent): - genericParent - commonMetadata - uketdObject *** cModels (children) - afmodel: FileAsset - commonMetadata - preservationMetadata ****	Objects: Compound cModels: - genericContent - staticImage - commonMetadata		Objects: Compound cModels: - dataset ** - compoundObject - commonMetadata	Objects: Simple cModels: - journalArticle ** - commonMetadata			Objects found with an unknown cModel are processed as genericContent. This allows us to process 'new' forms of object quickly and add specific Ruby Models to handle them in more than a basic fashion at more leisure.

Penn State	Mike Giarlo		We do not have a model in place for sets or collections of objects through which to do governance. We *do* create batches transparently in the background reflecting that a set of files were uploaded as a group, but these batches are little more than identifiers which can be used in relationship triples.	Items in ScholarSphere are all instances of GenericFile, a model that we built for this application. A GenericFile is a "Simple" object, and consists of the following components: 1. Noid-style pid 2. 'content' datastream that contains the blob deposited by a user 3. 'thumbnail' datastream, a derivative of the Content datastream (if appropriate to the file format) 4. 'descMetadata' datastream, with descriptive metadata about the object, either entered by the user or extracted from the file, expressed in RDF and serialized in ntriples format 5. 'rightsMetadata' datastream, included from commonMetadata mixin 6. 'characterization' datastream, including the output of FITS, which we use to characterize every file that is deposited 7. 'properties' datastream, which we use for random bits of metadata such as the depositor (for use with apply_depositor_metadata method) and the relative_path of a file within a file set								