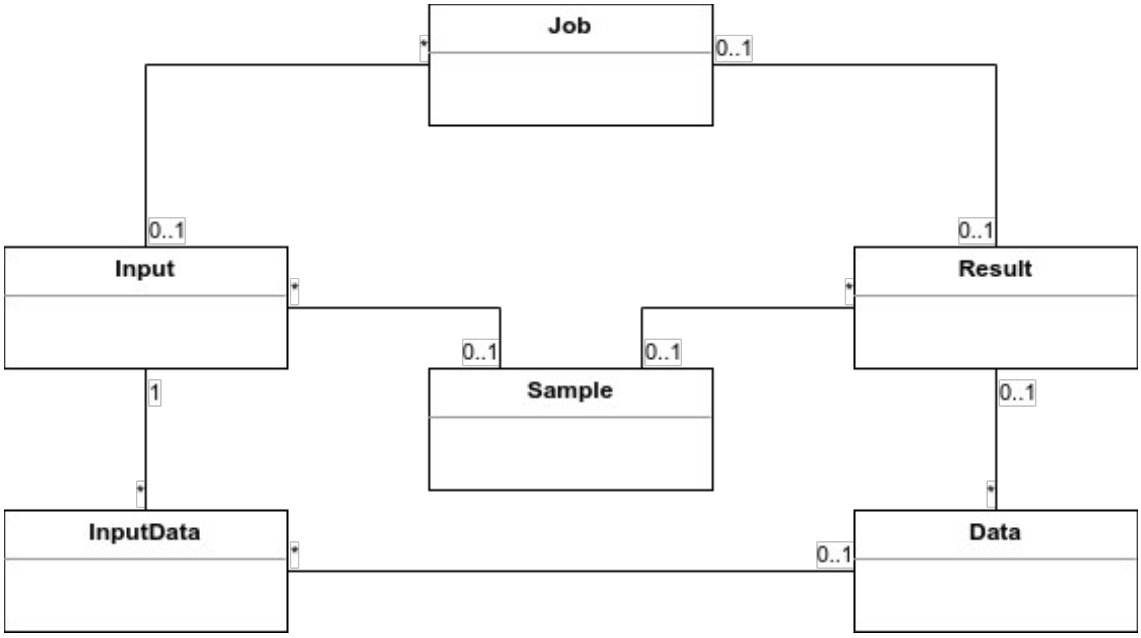
Initial draft proposal, version 0.0.2

Model Structure

Version 0.0.1 was supposed to start a discussion – but did not. Looking back, it was rather underdeveloped, and we should probably continue to ignore it. The idea behind *this* version is to start agreeing on at least the overall structure of the model we want to build. The key point is about the minimum number of objects and crosslinks between them that we need in order to accommodate our data. Class names and details can change, classes can be split and merged, and we may want to add more complex data structures (like unit cells or reflection shell data) but I think we shall need at least this much structure.
For a relevant use case, I am thinking about a workflow-driven multi-sweep crystallography experiment. For such experiments it is crucial that data input and processing result can be given in terms of a *set* of selected sweeps. Organisation in terms of single sweeps is obviously inadequate for the purpose, and organisation in terms of Samples not only precludes straightforward representation of multi-sample processing results, but also lacks a way of distinguishing which of the multiple sweeps, mesh scans, or experiments performed on a sample actually are (or should be) processed together.

We should remember that the model is meant to specify an *interface* between programs and LIMS systems. People are free to use any internal organisation they find practical, as long as they are able to read and re-export data according to the model.



Abstract classes defining the model structure and the links between them. For those unused to UML note the cardinalities: ‘1’ (mandatory), ‘0..1’ (optional), and ‘*’ (0 to many)

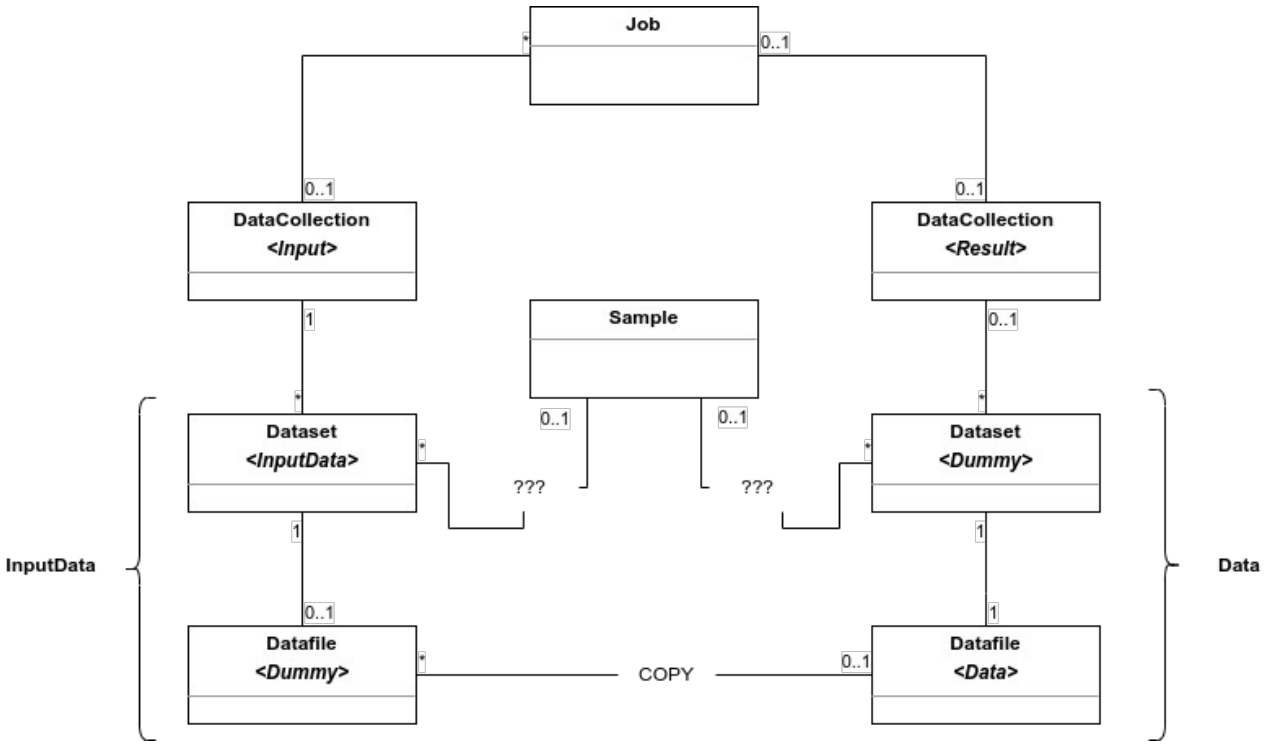
- These are abstract classes - actual classes will be subtypes. The **Job** class is used both for describing actual experiments and for processing runs (calculations), with **Input** and **Result** collecting the input resp. output for one **Job**. Everything we think of as data are subtypes of the **Data** class, which might be Sweep, Mesh scan, Line scan, Model, Map, MTZ file etc. ... The **InputData** class serves only to hold parameters that describe how a particular **Data**

object fits in as input to a particular Job. Note that only the **Data** class will have links to files.

- All classes should have a slot for program-specific name-spaced data.
- Any **Job** (experiment or processing) will have some parameters or results that refer to the entire **Job** rather than to any individual input or output file. Accordingly, each **Job** is connected to one **Input** and one **Result** object that can hold these parameters, as well as grouping multiple data sets. Note that **Results** are only connected to one **Job**, and that the same **Input** in practice will rarely be used more than once.
- Each **Data** object is produced by a single **Job** (of whatever type), and so can hold parameters that are only defined relative to that **Job**, like the distinction between characterisation sweeps and acquisition sweeps, or overall maps and early-minus-late maps.
- **Data** objects can serve as input for many different **Jobs**, so we need the **InputData** class to sit between the **Data** and **Input** classes and specify parameters that describe how the data fit into this particular **Job**. There may not be that many attributes here, but we might need e.g. a role (characterisation versus acquisition, input versus reference mtz files, …), information on included/excluded image ranges, possibly a weight, or program-specific parameters. Note that **InputData** are specific to a given **Input** object, and so the link between the two is mandatory.
- We should limit the number of different subtypes by using the same classes in different contexts. E.g. a Sweep **Data** type could be used as part of a diffraction plan (partially populated and without a file Url), as part of a message to the acquisition queue (with a Url but no data file as yet), to store the data collection result, and as part of the input for a processing program (linked to an **InputData** object).
- As currently drafted, the model assumes that each **Job** (experimental or processing) is connected to (at most) one single **Sample**. For single-crystal experiments, the sample would be equivalent to the crystal. This could be extended to multi-crystal experiments later, as long as the composition, history, expected crystal form etc. is the same for all the crystals in the same well / chip / vial. Where it is appropriate, we can link a Sample also to **Result** objects or to processing **Input**. Processing jobs are not in principle limited to using data from a single sample, but there is no explicit support for processing multi-sample data in the model, except by following inter-object links back to the Sample record. The alternative of connecting the Sample record to the **Data** object would give full support for multi-sample experiments and processing but would severely complicate the more common case of doing an experiment or processing on a single sample and wanting to find the relevant Sample record.

ICAT mapping

The ICAT model objects do not fit exactly with what we need to do here, so storing these data in ICAT would require a certain amount of crafting. One problem is that ICAT links the sample to the Dataset rather than the DataCollection. A more basic problem is that ICAT does not have crosslinks between Datasets but only between Datafiles, and that the ICAT Datafile strictly describes a file object, with location, checksum, etc. Considering that our **Data** object can correspond to a file, to a set of mini-cbf files, to only part of the data inside an HDF5 file, or to no file at all, this does not suggest an obvious mapping. The diagram below shows one possible way that the model presented might be stored in ICAT.



Possible ICAT implementation of the draft model. **InputData** and **Data** are handled by a combination of a Dataset and a Datafile with a 1:1 link between them. To reproduce the draft model would require a hack that somehow connected each DataCollection with a single, optional Sample.