

# Status Report

Ed Frank, University of Chicago

6 May 02

**Task:** We were given the task of exploring the relationship between the data handling language contained in ADB and the various Atlas Grid components. The plan was to quickly field a strawman design to elicit input from the larger group.

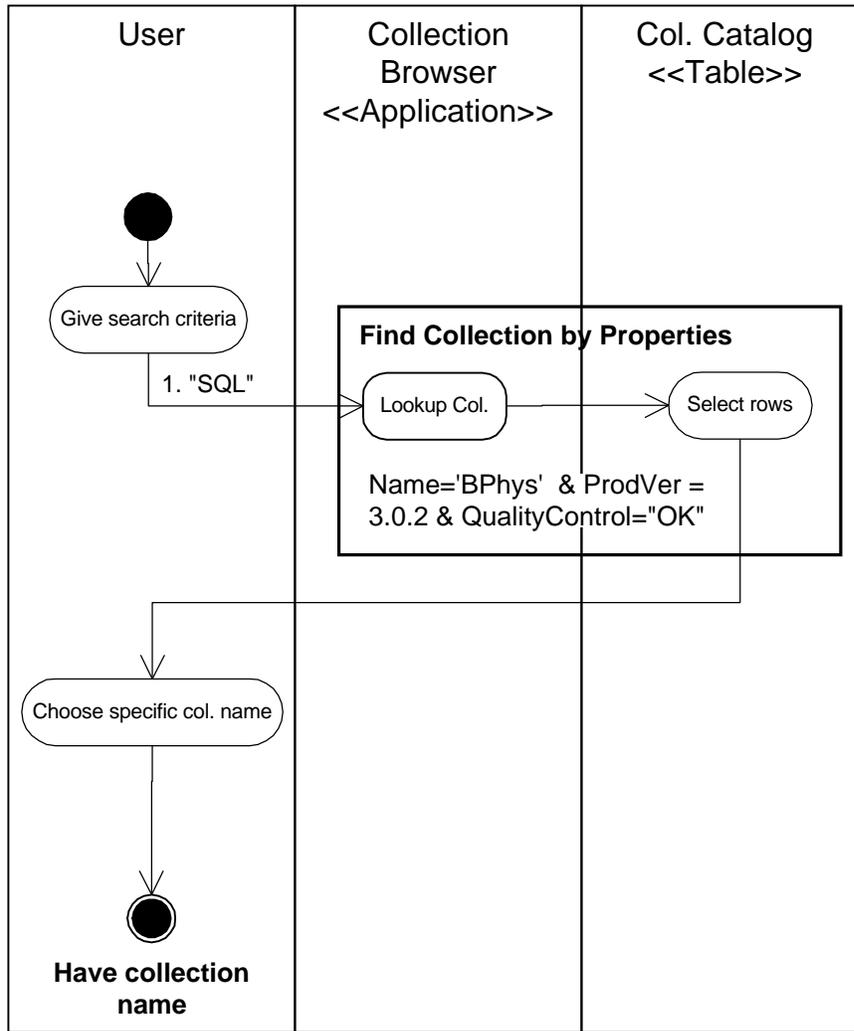
**People involved:** Greg Chisholm, Ed Frank, Rob Gardner, David Malon, Mike Wilde.

**Status:** With other activities and geographic separation, we found it difficult to gather the requisite people needed for progress. We abandoned the plan of a strawman and went directly to soliciting comments on the Atlas Grid mailing lists but received no feedback. My personal interpretation of this is that people really do need the strawman in order to frame their thoughts.

**This document:** This document has a few high level sketches that are likely to be wrong. The purpose of presenting them is to capture (and extend) the few discussions that have happened. The diagrams try to enumerate the entities involved in a few use cases in terms of UML activity diagrams and try to sketch their responsibilities. The aim is several fold.

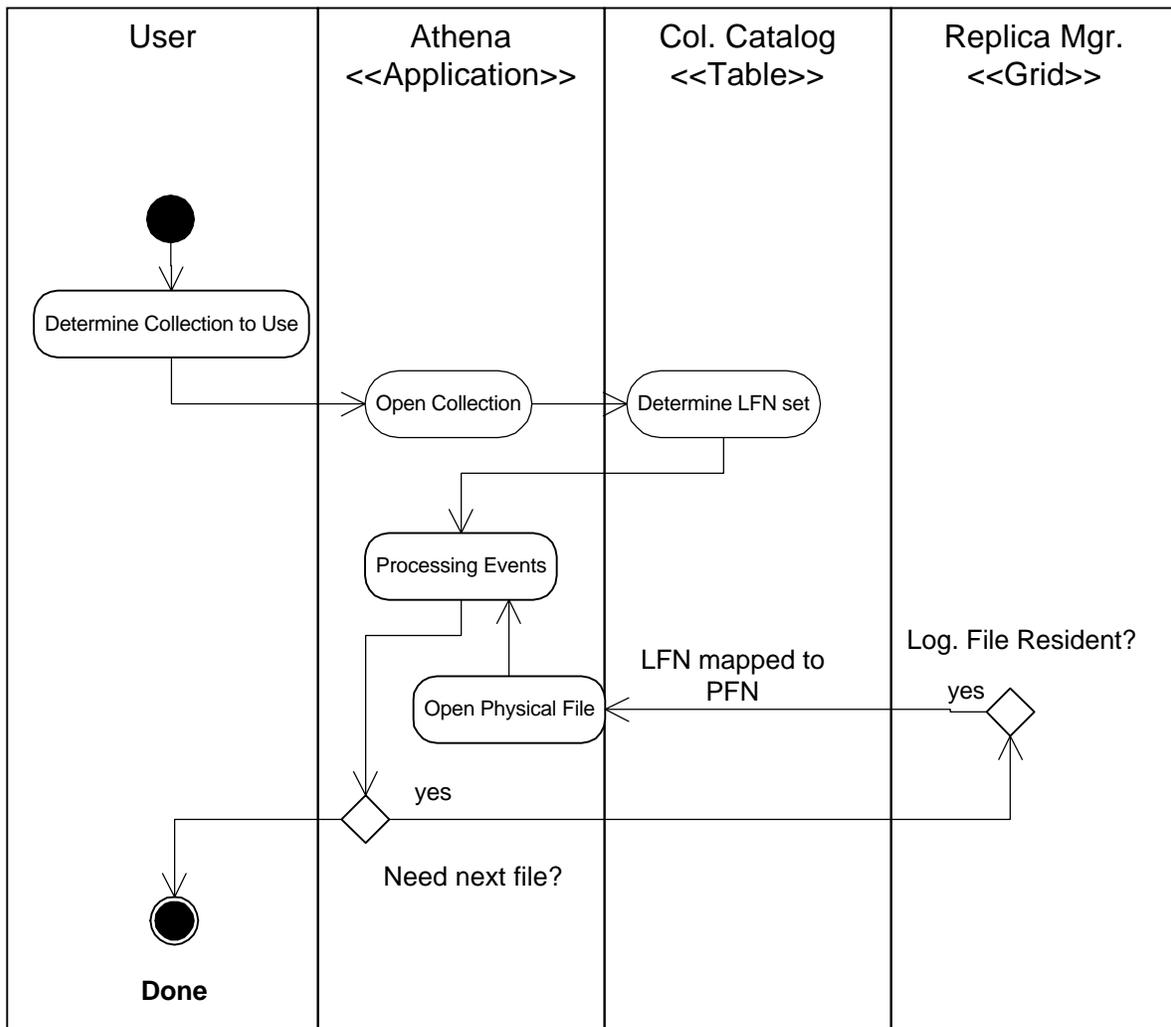
1. Motivate discussion to elicit the concepts and input needed by a user in order to execute the use cases. These are to be compared with the stipulated language in ADB (and elsewhere).
2. Make clear what input is needed from the group for us to proceed. We need to know that the entities identified are the correct ones and that the assigned responsibilities are correct.
3. Search for design consequences. For example, I believe we learn that replication must be tracked at both the file level and at the collection level. We also learn that the map from collection to logical filename-set is not unique in the presence of virtual data.

## Determine Collection to Use Activity Diagram



1. This diagram illustrates the basic procedure for determining what collection to process.
2. The collection catalog is a set of tables that relate collection names to collection properties. A user may issue queries based upon these properties to find interesting collections. Saying, "SQL" above is figurative, but perhaps not a bad guess.
3. Having determined the collection to study, the user works in terms of collection name henceforth.
4. We have introduced an agent (application) in the system called a Collection Browser.
5. Question: Does the collection browser only consider data in local tables or is it itself grid aware?
6. Comment: everything here is vanilla ADB stuff.
7. This process will be collapsed to "Determine collection name" in future diagrams.

## Local Athena Reading (Activity)



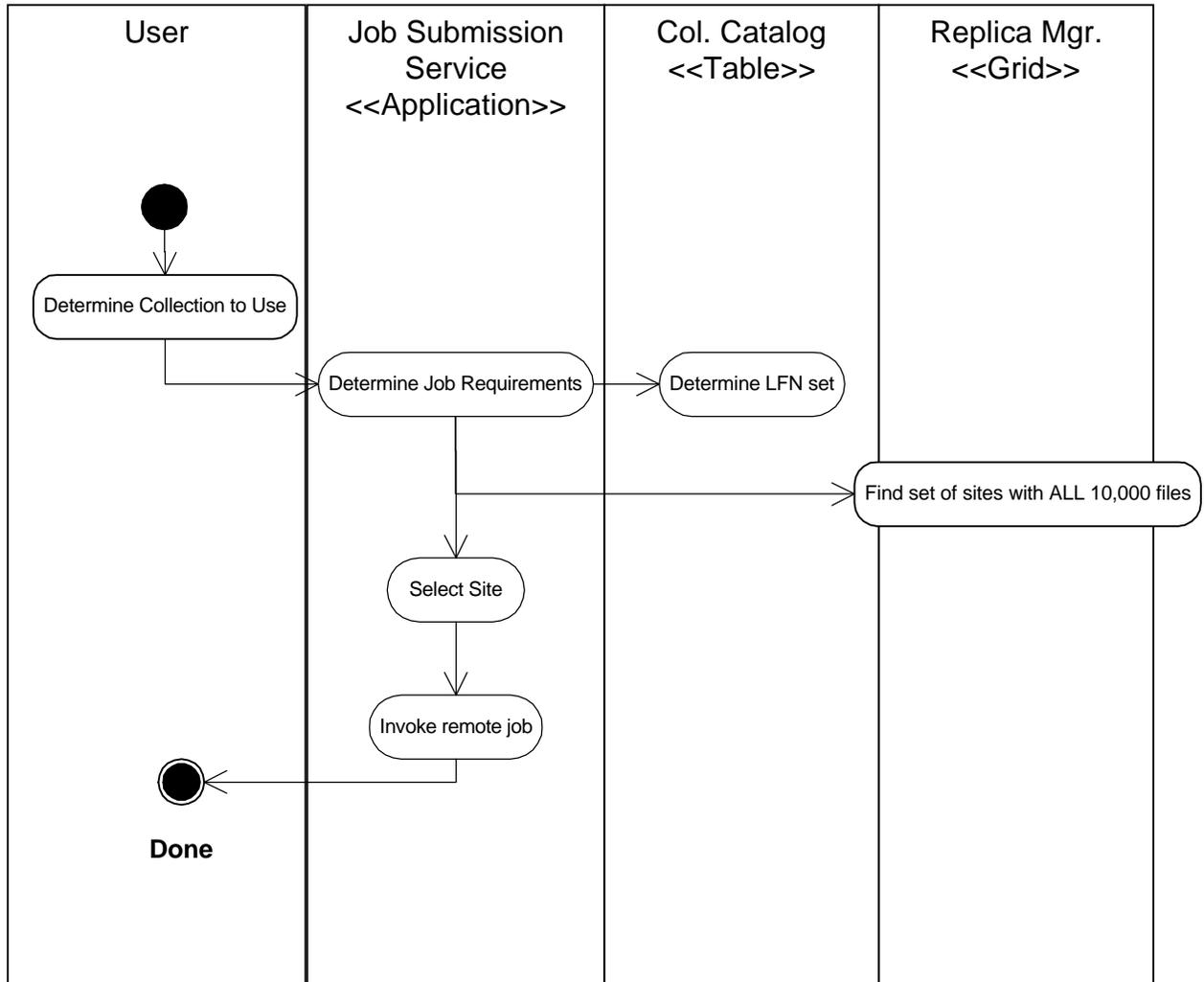
1. This diagram explores how a local job maps collections to logical file sets and then maps logical filenames to physical via the replica manager. Remapping of filenames was stipulated in ADB but a mechanism was not provided. We should decide if the Grid replica manager is appropriate or not for local jobs. Note that the use case here is the support of LOCAL remapping of files, e.g., even without grid motivated replication, a system administrator may need to relocate a file to do disk maintenance, for load balancing purposes, etc.

2. The diagram indicates an initial mapping of collection name to file set followed by iteration over the logical file names (LFN) in the file set. In fact, one might determine the logical filenames while reading the collection rather than reading them all up-front.

3. This example, most importantly, shows the replica catalog operating at the file level.

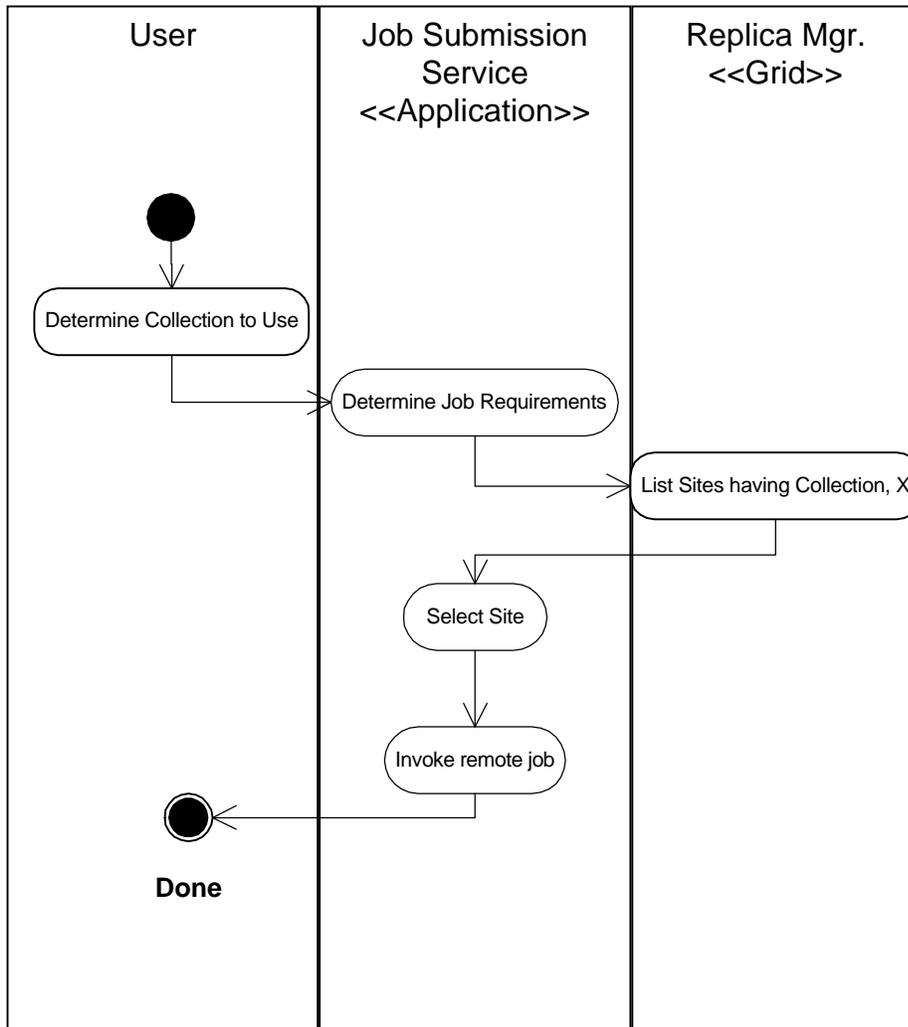
4. Notice that the user perspective of the system is at the Collection level, not file level.

## Using Job Submission Service (Activity)



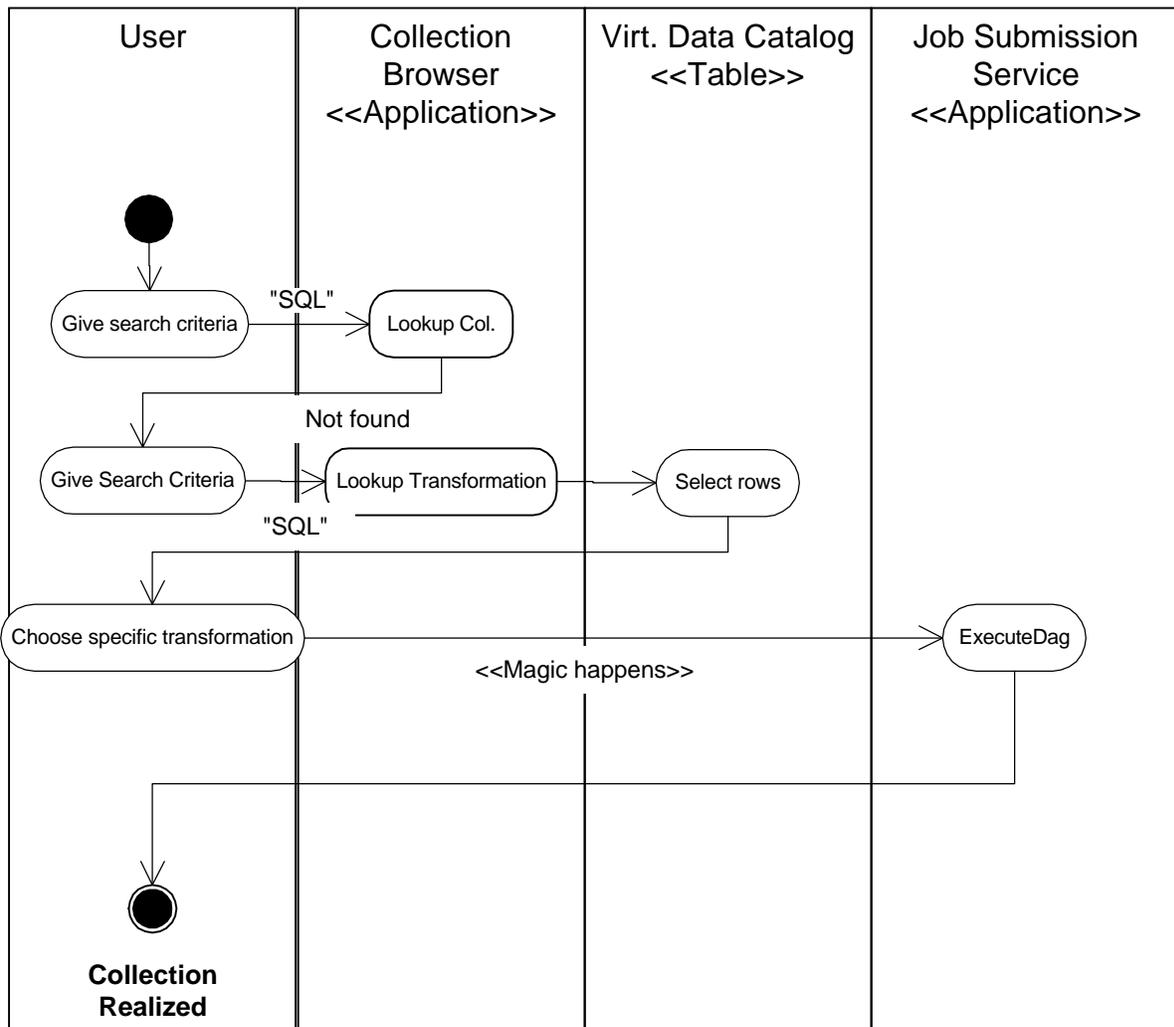
1. The example presumes the user wishes to migrate the job to a site holding all needed data.
2. The only point of the diagram is that, if the replication service only tracks at the file level, then the expensive operation, "do the following 10,000 files reside at site X?" will need to be done. The 10,000 files corresponds to the file set for the user's collection. Computing the intersection of a set with 10,000 members with the set of all installed files (perhaps millions) is inefficient and repetitive compared to simply recording the residency of the collection itself.
3. Thus, the example attempts to show a requirement that the replication manager track at the collection level as well as the file level. Actually, perhaps this should not be pinned on the replication manager, but the ADB notion of specifying input via collections implies that some grid entities need to operate at the collection level as well as at the file level. This conclusion will be strengthened by the example using virtual data.

## Using Job Submission Service (Activity)



1. This is as in the previous example using a job submission service, but this time we allow the Job submission service (JSS) to determine the set of candidate sites with a single query by collection name. Note that this example and the previous has elided all other aspects of selecting a site, e.g., CPU, available staging space, release version installation, etc.

## Virtual Data via Collection Browser (Activity)



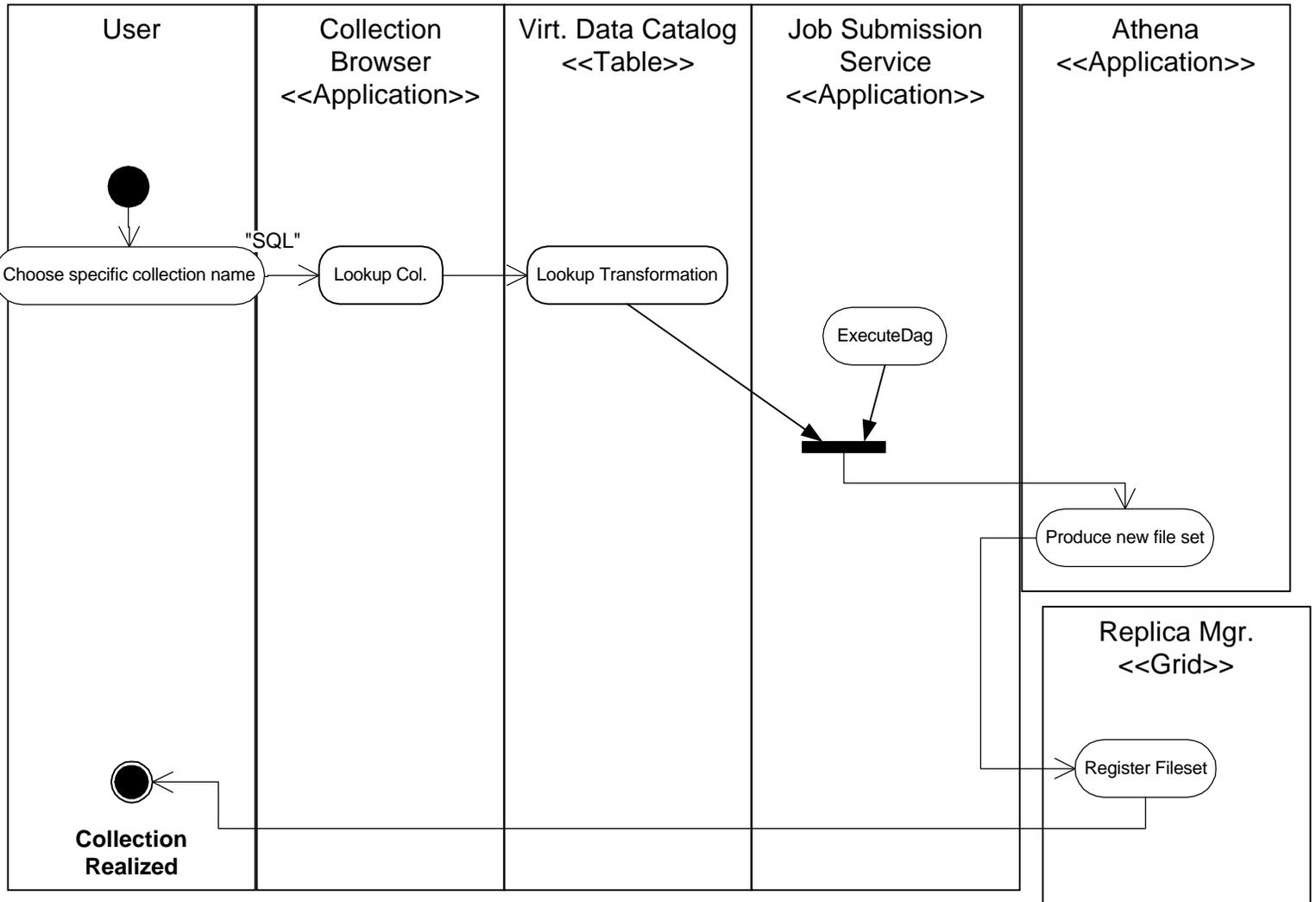
1. Here we explore how we might do virtual data. The example has the user heavily involved. The use case would be "Allow a user to use a transformation in the Virtual Data catalog to realize data locally." So, imagine the JSS to be a local job submission service.

2. The sketch comes back to the user to allow him/her to choose which of several possible transformations to use. For example, the search specification given by the user may have located a set transformations used to produce a number of Monte Carlo jobs and the user proceeds to refine the selection to a specific one. This is Virtual Data in the extended sense of recording our recipes for producing data rather than the more limited sense of "a recipe for reconstituting a *specific* piece of data." (We will explore that example next.)

3. The "ExecuteDag" bubble is a place holder that must be elaborated by consulting with GriPhyn.

4. The key point of this diagram, with respect to reconciling ADB with Grid, is that the user's point of view is entirely in terms of collections and collection meta data. Actually, we have added a new concept to the users' view: a "production recipe," or what GriPhyn calls a Transformation.

## Virtual Data via Collection Browser - Reproducing a Previously Made Collection (Activity)



1. This example shows a scenario for reproducing a **specific** collection via virtual data mechanisms. Thus, in this example, the collection browser directly goes into the data realization phase. This may not be the right choice, but there is a point to make regardless of the choice...
2. The diagram shows a job eventually being run that produces a new fileset. Remember, a single invocation of athena can produce multiple files. Although the virtual data transformation is identical in two jobs, the two sites may have different rules for allocating resources to jobs and this can lead to two identical jobs producing different numbers of files on output for the same logical physics sample.
3. To be robust against the possibility that a transformation can produce different numbers of files, or the same number of files but with different content, it appears we need a level of indirection. It appears that replication should be tracked at the collection level as well as at the file level, as discussed in an earlier example **and** we may need to introduce the notion of a collection alias or logical collection. The idea is that a request for collection "JetSet" may invoke the execution of a recipe that yields a new, non-identical fileset. We give that collection a name, e.g., "JetSet-Rep-0" and enter it in the replica manager as a valid match to a query for collection "JetSet."
4. Therefore we have three reasons for tracking within the replication manager at collection level. First, the need to support queries at the collection level to reduce the complexity of determining collection residency. Second, because Virtual Data ought to make sense at the per-collection level since that is our most common level of book keeping. Third, to give robustness against variation in collection representation (in terms of files) that may occur when re-runs of the transform occur.

## Final Remarks

This status report has tried to show some high level activity diagrams that explore the interaction of our data handling entities. Sketches like this help identify responsibilities or the entities (not the same as requirements!) and so form a foundation for interface specification. The diagrams should be possible with no more than a handful of swim-lanes (entities) per page. Otherwise, there is too much detail or too much flowing between the entities.

A number of important issues for ADB have not been explored here. They are the hard ones. For example, ADB uses placement categories not just to cluster data, but as the rendezvous mechanism between an executing program and a set of resource allocations made by a database administrator. ADB refers to these allocations as resource pools. In the context of a grid, where a user configures a job at one site but runs it at another, we need to figure out what this means.

Another issue skipped here, but alluded to, are various resource registration steps. ADB recognizes that a job can produce hundreds or thousands of files in an execution. The user does not specify output filenames. Only placement categories are given and the system maps these to storage pools and then assigns filenames as they are needed by the running job. The filenames are bound to collections in the meta-data. That is the ADB view. What does it look like on the grid? When do things get registered with grid-aware agents? What names need to be unique globally? How are they assigned?

"Extract and transform" is the central theme of ADB. We need to give an example of it operating on the grid.