Overall architecture and control flow

Ewa Deelman, Ian Foster, Carl Kesselman, Reagan Moore, Sanjay Ranka

December 2003

Workflow Representation

We define a workflow as a graph, where the vertices represent "activities" and the edges represent the precedence between the "activities". The graph may be cyclic and the graph needs not be connected. The following figure shows an example of a workflow represented as a graph.

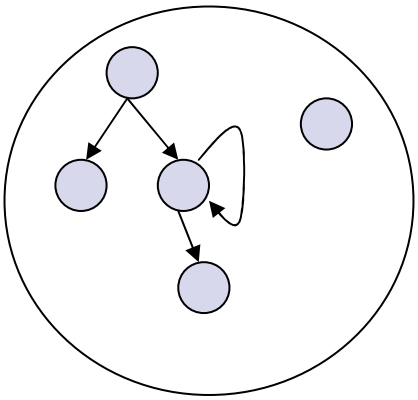

Figure 1: A Graph Representation of a Workflow.

Annotations can also be defined on the workflows, the vertices and the edges. An annotation is a set of zero or more attributes associated with a vertex, edge or subgraph of the graph. The annotations can represent the state of the workflow, the history of the operations performed on the workflow, options considered during these operations and others. The annotations thus can provide the provenance of the data computed by the workflow. The annotations defined on the edges can specify the nature of the dependency between the vertices. For instance, data produced by a precedent activity are annotated as input data to a depending one.

We can consider a vertex as a super node that contains a workflow in itself. The concept that we have defined for the workflow and annotation can be applied to the vertex as a super node recursively.

In order to execute the workflows on the Grid, the workflows need to undergo a series of refinement, of editing. The editing of a workflow continues until the workflow is annotated to be in a "done" state. There may be additional state information available when a workflow completes, for example provenance information about how the workflow was executed.

Please send comments to deelman@isi.edu

Workflow Editing

Workflows can be modified by editors. In general, editors may:

- Add vertices or edges to the graph
  - For example, information about the dependency between two vertices can be described by adding vertices and edges with annotations
  - Also, An editor can set apart activities by adding vertices and edges in a workflow.
- Delete vertices or edges from the graph
  - For example, activities that have already been executed (based on provenance information) can be deleted from the workflow. For example, the editor can delete redundant works in workflow execution by removing vertices and edges from a workflow, so called 'Graph Reduce'.
- Add, delete or modify annotations on the graph, vertices or edges
  - For instance, data- or execution-planning editor performs the modification with replacing a logical name with a physical name for data or resource location.

We note that editors can operate on portions of the graph (subgraphs) and not only on the entire graph. In general, an editor performs a transaction that maps a subgraph (*s1*) onto a subgraph (*s2*). The editor needs to preserve the dependencies that relate to *s1* in *s2*: after the mapping the edges that were directed to *s1* are directed to *s2* the edges that were directed from *s1* are directed from *s2*. Annotations on the input/output edges of *s1* are also preserved in *s2*. The annotations can include additional information for the dependencies, but can't eliminate pre-existing information as attributes. We also assume that two editors cannot edit two subgraphs at the same time if these subgraphs have common vertices or edges. For performance issues, or for reliability, we may allow two editors to make several copies of a workflow. An instance of a workflow can be submitted simultaneously to multiple execution environments. This may allow a workflow submitter to get the workflow executed with better performance.

An editor can be defined by its type, which includes the type of workflow it takes as input, the type of workflow it provides as output and the type of algorithm it uses to do the editing. Editors can perform workflow composition, planning, scheduling, execution and other functions.

Some editors may only re-write the workflow, change it from one representation to another. Examples of these editors are Chimera, Pegasus. Some editors may execute portions of the workflow, consuming resources, modifying the state of the vertices to "done" or "failed". Examples of this type of executors are DAGMan, PBS, etc.

The various types editors and their functions are described in another document [GriPhyN technical report 2004-23].

Coordination between Editors

In general, the editors communicate with each other indirectly, by annotating the workflows. We assume that not all editors need to understand all the annotations, just the ones that are relevant to them.

There are two main ways the editors can work together to refine the workflow.

1) Via a common workflow pool, where each editor picks an "appropriate" workflow, "locks it" and edits it. Upon completion of the editing process the new workflow is placed in the workflow pool and the lock is released. In this case the editors discover and select the workflows they want to edits and there is little control over which editors actually perform the refinement.
2) Via a coordinator, that decides the appropriate workflow to send to a given editor. In this case the editors are discovered by the coordinator and selected based on their characteristics.

The difficulty of the first approach is to define a control structure for the editing process, for example if they are two editors that can edit the same type of workflow, which one will get to edit it. In the second approach, we can have more control over the sequence of editors that operate on the workflow. The coordinator can choose the editors based not only on their suitability, but also performance and reliability. A more coordinated editing process can however potentially suffer from performance bottlenecks.

A third approach uses a hybrid approach with the workflow pool and the coordinator, where workflows in the pool are organized according to their types, and the coordinator classifies editors according to their types. Editors in a same class can compete for workflows in a section of the workflow pool, in which section there are workflows with the same type. The third approach gives the coordinator control over the sequence of editors for suitability and performance as well as resolving the potential bottleneck suffering.

Hierarchical Refinement Structure

A possible organization and coordination among the editors can be imposed by a hierarchical structure of the editing process. In this structure there are well defined editing levels related to the state of the workflows. The editors usually communicate between near-by levels, however skipping levels is also allowed. The nature of the communication is that of delegation, where one editor delegates the editing of a particular workflow to another editor. It is also possible to involve a coordinator in the process whose role will be to perform the delegation. As part of the delegation process, an appropriate editor needs to be chosen based on its ability to refine the workflow further. Having a workflow coordinator would allow for making global decision about the workflow editing process, whereas having the editors directly communicate with each other would provide for localized decisions.

Several strategies are possible to find appropriate editors for the delegation in addition to the workflow type consideration. For instance, Round Robin is simple and effective schedule for many environments. It will simply assign a next available editor to delegate responsibility. However, a workflow coordinator can make the editor selection utilizing complex algorithms to improve the overall performance.