

Considerations on Constraint Modeling in Grid Application Workflow Descriptions

*Greg Graham, Anzar Afaq, David Evans, Gerald Guglielmo, Eric Wicklund
Fermi National Accelerator Laboratory, Batavia, IL, 60510-0500, USA*

*Peter Love
Lancaster University, Lancaster, United Kingdom*

Introduction

The Grid is emerging as a specialized distributed computation standard of unprecedented power and scope, promising to turn commodity networks and computers into commodity computation. The Grid concept has already been proven useful for science in many applications [1] [2] and substantial infrastructure already exists [3] or is being planned [4]. Data processing on the Grid ranges from tightly coupled computation within a single application instance using standard interfaces such as MPI [5] to multiple filtering or data processing applications with large data flows between them. At the same time, the requirements on such aggregate processing jobs must be coordinated among many individual researchers or research groups within a Virtual Organization (VO) [1] or between multiple Grids. Especially in the case of workflows containing a moderate number of application steps and service invocations, it becomes a daunting task to check that all of the input parameters and software configurations conform to decisions made at the collaboration level. It is useful to have a language that is able to specify constraints on the parameters of the individual workflow steps that bring them into line with collaborative decisions coherently across the entire workflow, possibly even dynamically as late decisions are being made about execution environment. In the paper, it will be shown that:

- 1) Collections of constraints can be gathered into documents called contexts that function as operators on existing workflow graphs. An algebra of contexts supporting composition can help different subgroups within a VO work together through constraint sharing. Decomposition of contexts can allow for variance of constraints simultaneously across several different categories.
- 2) Constraint expressions and contexts form an interesting and hitherto largely unexplored area of data provenance. Knowledge of the constraints implies that it is possible not only to know the values of application input parameters, but also why they were set in particular ways.
- 3) A web services infrastructure supporting the distribution of constraints and allowing for delayed operation of constraints to workflows so that constraint resolution can be delayed and can become part of the job planning process.

The techniques developed here will find fruitful application in the aspects of organizing input parameters and constraints and in the aspects of sharing and enforcing collaborative decisions about those constraints. A partial implementation of these ideas already exists in a workflow building tool called MCRunjob [6] for the Compact Muon Solenoid (CMS) experiment [7], an High Energy Physics experiment based at the European Center for Nuclear Research (CERN) in Geneva, Switzerland[8].

In the following, we will focus both on application level constraints and constraints on physical resources. We will motivate how multigraphs including new arrow types called metadata flows can be used to express constraints. And we will outline a general procedure for reducing multigraphs into fully constrained workflow descriptions suitable for execution by a workflow manager such as DAGMan[9].

Objects and Operations in a Workflow Constraint Language

Workflow specifications are often expressed as directed graphs. Let $G=(N,A)$ be a general workflow graph. For simplicity, we assume a non-cyclic workflow as represented by a DAG, but the extension to cyclic workflows is straightforward. Each node N in N corresponds to an application and the set of arrows A corresponds to a partial sequencing of the nodes, perhaps generated by data flow or control flow relationships¹. The nodes may have attributes specifying input parameters for or results from the corresponding application. In order to express constraints on the attributes in G , one can add both nodes and arrows resulting in a multigraph², where the extra arrows will correspond to constraint relationships between specific node attributes. These extra arrows will be called **metadata flows** (as opposed to “data flow” or “control flow”) and the set of all metadata flow arrows constraining a graph G will be denoted F . In addition to metadata flows, special nodes may be added whose only purpose is to serve as sources or sinks for metadata flows. These extra nodes are called **metadata terminals**, and the set of these added to G will be denoted M . The multigraph containing G and its constraints expressed as in F and M will be denoted M_G . (See Figure 1.) Several types of arrows are evident in M_G including at least conventional workflow sequencing arrows and metadata flows. Special arrows that contain extra instructions or even nodes to handle cases like file transfer associated with a data flow relationship are an example of a special case.

Operations in the constraint language will consist mainly of reduction operations that satisfy the metadata flows and gradually reduce M_G to a fully constrained and specified workflow graph G' . Let $F(N::n,M)$ denote a metadata flow, where N and M are nodes in N , and n is an attribute in N . The first argument is the target of the constraint while the second argument is the domain. A reduction R_F over F is an operation that replaces the value of attribute $N::n$ by some value which satisfies the constraint as computed from the domain M . For example, the simplest such operation is just the assignment reduction $=_F$: the $N::n = M::m$ constraint on $N::n$ is that it be equal to $M::m$ where m is some attribute

¹ Alternatively, the nodes may correspond to data products and the arrows may correspond to data transformations. These two pictures are equivalent and we will assume for the purposes of this paper that a transformation graph can always be converted into an application sequence graph, have constraints applied, reduced according to those constraints, and then converted back into a transformation graph.

² A graph comprises some set N of nodes and a set A of arrows such that for any two nodes N and M in N , N and M can have at most one arrow a in A between them. In a multigraph, the restriction on the number of arrows is dropped.

in M . Categorically, the unifying character of this picture can be seen by considering that general constraint reductions targeting an attribute value in some workflow node are equivalent to constant assignment reductions emanating from a metadata terminal node[11]. This equivalence should be reflected in the constraint language.

Reduction always results in the removal of a single arrow from F and possibly the alteration of at most one attribute in the target node of the flow. Reduction can continue until M_G has been transformed into a usable workflow graph G^3 . (For help with the symbols being introduced, please refer to Table 1 below.)

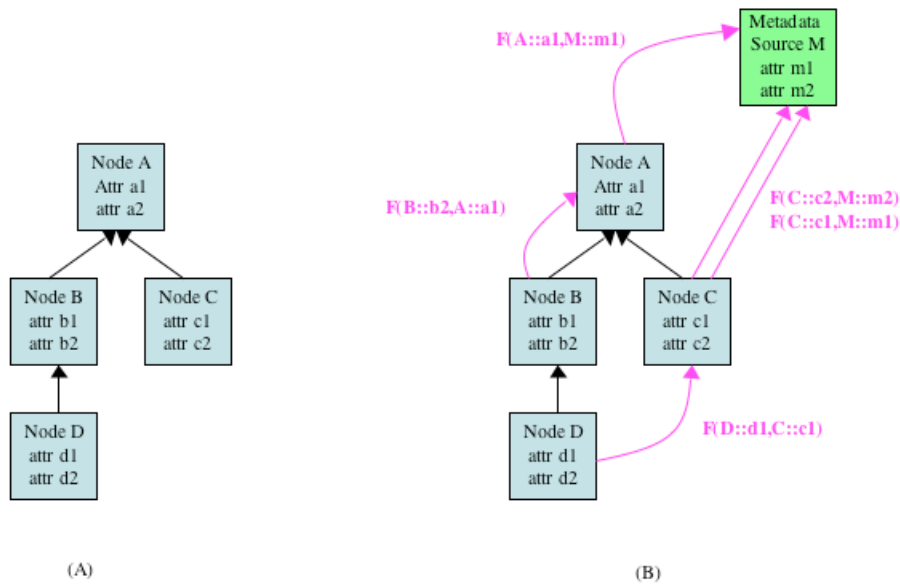


Figure 1: (A) A simple four application (non cyclic) workflow graph. In order to use this graph, all node attributes must be set. (B) The simple graph in part A has been augmented with metadata flows (in pink) specifying various constraint relationships. In addition, a metadata source has been added. The constraint augmented workflow graph in a multigraph.

Let M_G' be the metadata flow subgraph of M_G such that M_G' contains all of the nodes of M_G but only the arrows in F . It should be noted that this subgraph should be acyclic so that at least one partial ordering for the reduction process exists and is well defined. Metadata flows exist essentially to carry metadata to a finite number of graph nodes in a possibly cyclic G , and each node in G has a finite number of attributes. Thus it is always possible to find a finite spanning tree.

³The original workflow graph G in M_G did not have constraints applied, and the final G after reduction does. It is in fact an iterative process so that there is a whole sequence of reduced “ G ’s” between the first one and the final one. This goes for F as well.

Nonetheless, there are many such reduction partial orders and some optimization may be gained by grouping some reduction operations together. The language should therefore have some facility to favor a given reduction partial order over others, such as specifying that some constraints belong to groups.

Graph	Name	Nodes	Arrows
G	Workflow Graph	N	A
F	Metadata Flows	-	F
M	Metadata Terminals	M	-
C	Context	M	F
M_G	Constrained Unreduced Workflow	M+N	A+F
M_G'	Metadata Flow Subgraph	M+N	F

Table 1: Some help with terminology. Contexts will be introduced in the next section.

An example of a metadata source terminal is a node that holds a query result from some catalog. For example, it may contain metadata about calibrations that need to be synchronized across all application nodes in a workflow, or it may iterate over input filenames in a dataset in a multiple workflow preparation phase. An example of a metadata sink is a node that may consume metadata merely to record it, such as a tracking system or provenance recorder or a node that will transform the workflow into a shell script.

Constraints and Contexts

Basic structures in the constrained workflow language are the multigraph and the metadata flow. The basic operation in a constrained workflow language is the reduction operation that gradually reduces the multigraph by removing metadata flows while causing constraints to be satisfied, resulting in a conventional workflow graph. In the above section, we introduced the metadata flow subgraph **M_G'** that contains all of the nodes in **G** together with any metadata terminal nodes and the metadata flows. **M_G'** is however unsatisfactory as a referenced object in the language, because it mixes the workflow nodes of **G** with the constraints explicitly. Much more useful is the context **C**, defined here as consisting of the metadata terminals alone and the metadata flows. By partitioning **M_G** into context parts **C** and application workflow parts **G**, we gain the possibility that a given set of constraints in **C** agreed upon by some large organization can be applied to multiple application workflow graphs written by users. In other words, a given application workflow graph should be able to run in a variety of contexts and acquire automatically the constraints therein.

Given the infinitely wide variety of workflow graphs and contexts possible, it is a non-trivial task to design an algebra by which these sets can be combined in a meaningful way. A further problem is that **C** by itself is not even a well-defined graph because some of the arrows properly in **C** may point to nodes that are in **G** but not in **C**! The approach taken here to deal with these problems is to assign types to the graph nodes in the application workflow graph. By surveying the set of all possible application nodes in an

organization, it is possible to come up with a super-context document D_C that, rather than being a graph subset, is a collection of rules for how to apply metadata terminals and metadata flows in a real application workflow graph as workflow nodes are added into the super-context. The rules are indexed by node type and are applied under one of two kinds of semantics: “only-once” semantics or “for-each” semantics. For metadata terminals, the only-once semantics are generally used. For metadata flows, the for-each semantics are generally used. (See figure 2.)

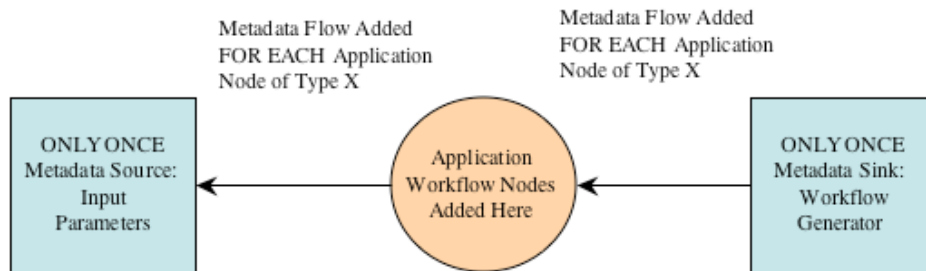


Figure 2: A context document. The application workflow node type (added at center) is used to lookup rules for adding metadata flows and/or metadata terminal nodes. Generally, flows are added with “for-every” semantics and nodes are added with “only-once” semantics.

This approach, though it restricts the kinds of metadata flows to something less than full generality, works for a surprisingly large number of cases. Also, the approach can be easily extended to switch rules based on tuples of node types rather than just singleton nodes as is currently worked out.

An algebra for combining different super-context documents is being developed. This is somewhat more difficult to do in complete generality not only since the documents are not graph descriptions, but also because rules must be developed to handle metadata collisions; when metadata flows share the same target. The final metadata flow that gets applied may depend upon the order in which the context documents are processed. However, there is still a large class of problems for which the metadata flows do not collide and are yet useful; and there is also another set of problems for which the “shadowing” behavior of metadata flows is actually desirable, such as for simple replacement of site dependent variables.

Much existing work on Context Oriented Programming (COP) [12] is being done for mobile computing. Systems are being developed to exchange the actual code that gets run in different locales. The present work is different in that it is effectively exchanging data and constraints and not actual code. Also, the definition of ‘locale’ here is anything of relevance to the VO: physics group, personal role, etc.

Constraints and Contexts as Provenance

Provenance deals with the problem of collecting all of the information needed in order to recreate a data product. The transformation graph approach of GriPhyN is useful here[13]. To briefly recap, a data product in GriPhyN is logically represented as the final result of a sequence of transformations where each transformation is a template for an actual application run that processes the data. Given an initial data product, together with a concrete specification of transformations (called derivations) it is possible to exactly reconstruct all of the application processing steps that went into creating the final product. This picture has the useful side effect that a data product can exist *virtually*, that is, before any of the actual processing steps have been run. This gives rise to the possibility of performing an extra optimization step in which the possibility of playing off the cost of transfer of an existing data product to the end site versus recalculating it there. Detailed provenance is therefore built into the system.

The prospect of saving metadata flows and metadata terminal nodes before the process of reduction begins on a constrained workflow graph offers the possibility of saving a new kind of provenance. Namely, in addition to saving the flat values of all of the parameters that go into creating a data product, one can also save the constraints and relationships among those parameters and, by extension, *why* the parameters in a conventional provenance have the values that they do: which calibration set is being used, is something really constrained across all workflow steps, which collaborative group signed off on a set of constraints as a whole and not just considering each constraint one by one.

Infrastructure Components

Due to space constraints, the infrastructure components needed to support a constraint language will be treated briefly here. Obviously, an implementation of a constraint language satisfying the considerations given here can be done so that all reductions are done before execution time. In this mode, a workflow specification is given along with a super-context, all reductions are done offline in order to obtain an executable workflow, and then this workflow is run on an execution resource.

However, in order to optimize some of the reductions, it is often desirable to delay them. This corresponds to a particular mode of abstract planning. In order to achieve this abstract planning, it is useful to group the reductions by some partial order as described above. Some group or other of reductions can be done by the appropriate remote procedure calls or database lookups. In the infrastructures being planned, these could be exposed as a simple Web Service.

We alluded to special metadata flows above that may come along with helper nodes. For example, a dataflow relationship (non-streaming) often comes along with a requirement for a transfer of one or more data files. This metadata flow then would be more complex in that more than one application or atomic service is involved, and it lasts for a non-negligible amount of time. Therefore a service handle is needed, and the Web Services Resource Framework (WSRF) will be employed.

Conclusion and Relation to Other Work

This paper outlined some considerations for constraint modeling in Grid application workflows. We focused mainly on semantic constraints and not constraints on physical resources. We have shown how multigraphs and metadata flows can be used to express constraints, and outlined a general procedure for reducing multigraphs into fully constrained graph workflow descriptions.

DAGMan[9] has an excellent language for graph workflow description, and a manager for executing the graph nodes in sequence with fault tolerance. It is integrated with ClassAds[10] which provide an excellent mechanism for expressing constraints in terms of external quantities, such as physical or machine parameters.

Many of the concepts discussed here have been prototyped in a production system for the CMS experiment called MCRunjob. MCRunjob is a macro script driven tool for producing jobs for Monte Carlo simulation of the effects of high energy physics events in the CMS detector at CERN. MCRunjob contains metadata flows with assignment reduction, framework style grouping of reduction operations, metadata terminal nodes for catalog lookup and job building, and super-context documents. In MCRunjob, a demonstration partitioning of contexts have been achieved that allow for independently swapping among different grid environments (“US” versus “EU” grids versus “Chimera”[15]) and among different physics environments (“official CMS” versus “private” production) and executing the same application workflow in all six environments.

Traditional context oriented programming substitutes actual executable code depending upon the locale in which it is executing. This work focuses mainly on metadata flows among parameters in graph nodes and how those metadata flows change depending upon what context is used. This is essentially equivalent to operating on method footprints, but effectively only by varying default parameter values based on a logical locale with meaning in a VO. A formal investigation in this direction may bear fruit with respect to heavily parallel programming, for example with MPI. In particular, in a data streaming environment, if constraints are changing then a Web Services infrastructure that supports the style of semantic constraints described here could be able to support dynamic constraint switching in real time.

References

(Web pages given below will be replaced with print references where possible in the paper.)

1. I. Foster and C. Kesselman, ed. **The Grid: Blueprint for a New Computing Infrastructure, 2nd Edition**, Morgan Kaufmann (2004)
2. F. Berman, G. Fox, and T. Hey. **Grid Computing: Making The Global Infrastructure a Reality**, John Wiley & Sons, (2003)
3. Up to date information about Grid2003 can be found at: <http://www.ivdgl.org/grid2003>
4. Up to date information about the European Grid EGEE to be available at <http://egee-intranet.web.cern.ch/egee-intranet/gateway.html>
5. Up to date information about MPI can be found at: <http://www3.niu.edu/mpi>
6. G.E. Graham, et al. **McRunjob: A High Energy Physics Workflow Planner for Grid Production Processing** Proceedings of CHEP 2003 (TUCT007) San Diego
7. Up to date information about the CMS Experiment can be found at: <http://cmsinfo.cern.ch>
8. Up to date information about the CERN Laboratory can be found at: <http://www.cern.ch>
9. Up to date information about DAGMan can be found at: <http://www.cs.wisc.edu/condor/dagman>
10. Up to date information about ClassAds can be found at: <http://www.cs.wisc.edu/condor/classad>
11. B. Pierce. **Basic Category Theory for Computer Scientists**. MIT Press (1991)
12. R. Keays. Context Oriented Programming. PhD Thesis, University of Queensland (2002)
13. Up to date information about the GriPhyN project can be found at: <http://www.griphyn.org>
14. G.E. Graham. **The CMS Integration Grid Testbed**. International Symposium on Grid Computing (ISGC 2003) Taipei, Taiwan
15. I. Foster, et al. **Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation** 14th International Conference on Scientific and Statistical Database Management (SSDBM 2002)