Workflow Support in a GRACCE Metascheduling Architecture

Yonghong Yan, Barbara M. Chapman
Department of Computer Science, University of Houston
{*yanyh, chapman*}*@cs.uh.edu*

Abstract

Modeling a complex application in grid environments concerns about how to describe application modules, data sets and module relationships so that an application can be seamlessly integrated with grid middlewares. In this paper, we present a high-level abstract language for domain scientists to model distributed applications. GAMDL describes the dataflow structures of domain complex problems and allows control logics to be defined within dataflow using conditioned properties and conditioned pipes. GAMDL is intuitive and very easy to use for users without knowledge of grid computing and the use of multiple-value property makes GAMDL much more powerful than other languages to describe similar application entities. Also, driven and used by a production application, GAMDL serves as a basis for the integration between a grid metascheduler and workflow systems in GRACCE project.

1 Introduction

Grid environments [3] are increasingly being used by domain scientists with large-scale applications [1, 16, 19]. These applications are no longer being developed as monolithic codes, but incorporate multiple dependent modules, and entail the transfer and storage of a large amount of data. Enabling such an application on grid environments is much more complex than enabling an application that can be wrapped as a simple grid job. Several issues need to be addressed, such as description of application structure, grid metascheduling and workflow execution coordinations.

Modeling a domain application in grid environments concerns about how to describe application computational modules, data sets and module relationships so that an application can be seamlessly integrated with grid middlewares. Most of current workflow description languages model application control-flow on the middleware level with terms like job or service, assuming users have knowledge of grid computing. Related workflow engines also have very limited or even no grid-level scheduling functionalities. Additional extensions are required to integrate with a grid metascheduler [12].

In this paper, we present a high-level abstract language for domain scientists to describe their applications for grid deployment and integration, GRACCE Application Modeling and Description Languages (GAMDL). GAMDL describes the dataflow structures of domain complex problems, and allows the definition of control logics within application dataflow using conditioned properties and conditioned pipes. GAMDL is intuitive and very easy to use for users without background of grid computing and is much more powerful than other languages to describe similar application entities using multiple-value properties.

GAMDL design is not purely language based, but with direct application and support for grid middleware integration. GAMDL is driven, and used by a production application, UH AQF [11], and serve as a basis for the integration between a grid metascheduler and workflow systems in GRACCE project [15]. GAMDL associates job specification within application description, addressing the issue of integrating grid application and job workflow in the level of application description. GAMDL also allows job execution history to be specified in application description, which may be utilized by a grid metascheduler for resource coallocation and execution prediction.

The papers is organized as follows. In the next section, related works for application modeling and workflow description are studied, and background of GAMDL are introduced. Section 3 describes the specific features of GAMDL along with its constructs and related examples. Section 4 presents GRACCE metascheduling architecture and how GAMDL is integrated in GRACCE. Finally, conclusions and future work are outlined.

2 Air Quality Forecasting Application and GRACCE Architecture

To meet the requirements of both domain users and grid integration, we defined a set of XML schema, called GAMDL for grid application modeling and description. GAMDL is the result of enabling Air Quality Forecasting

(AQF) application on UH campus grid in GRACCE project [2].

AQF application is an integrated computational model for regional and local air quality forecasts that is composed of three subsystems: MM5, SMOKE, and CMAQ [2]. AQF execution is a computational sequence of the three subsystems with increasing resolution and decreasing geographical boundaries. Figure 1 illustrates the workflow of a nested 2-day forecasting operation over a single region of interest by a three-domain computation, 36km domain for continental USA, the 12km for south central USA, and the 4km for a smaller geographic region. Each rectangle represents a computational module and each arrow indicates the flow of data between modules.

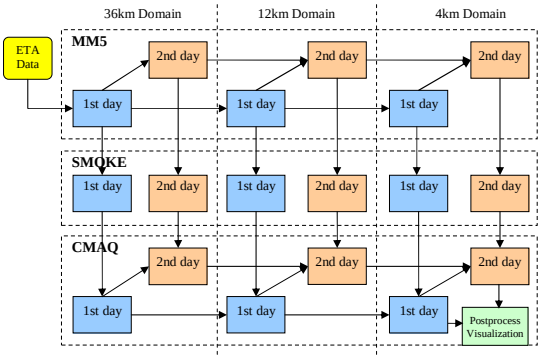

Figure 1. AQF Module Workflow

GAMDL is part of GRACCE (Grid Application Coordination, Collaboration and Execution) project, which was proposed to develop a set of grid middlewares for grid application deployment. Started from the AQF application, GRACCE aims to provide domain scientists an application-specific grid environment, supporting from the management of an application and its dataset, to the automatic execution and viewing of results. In Section 4, we will introduce GRACCE architecture and how GAMDL is integrated in GRACCE.

GRACCE metascheduler architecture has three components, a metascheduler that plans job execution and co-allocates resources for workflow tasks, GridDAG event-driven workflow system that coordinate workflow execution, and EPExec runtime execution and monitoring system [12], as shown in Figure 2.

The life-cycle of a grid workflow job in this architecture is described briefly below:

1. Grid users submit to the metascheduler a workflow job (possibly with preferred deadline) specified using the application GAMDL.

2. Metascheduler plans the execution of the tasks and the dependency handling mechanisms, and allocates resources for tasks executions. As a result, the job's EP that includes the decision details is outputted.

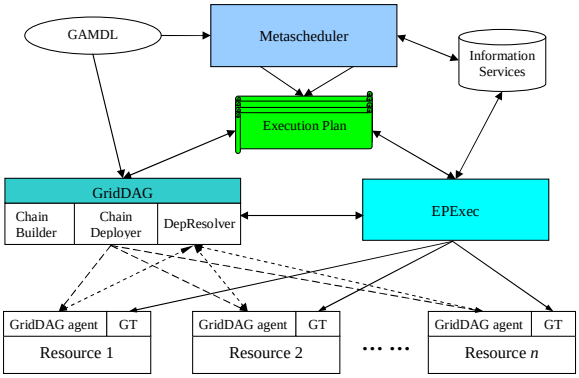

Figure 2. GRACCE Architecture

3. The job's EP is forwarded to EPExec for job execution, which submits tasks to their allocated resources and monitors their executions.

4. During execution, GridDAG workflow system handles task dependencies and decides whether task dependencies are resolved.

In GRACCE, metascheduler is defined as "a grid middleware that discovers, evaluates and co-allocates resources for grid jobs, and coordinates activities between multiple heterogeneous schedulers that operate at the local or cluster level". GRACCE metascheduling architecture separates job execution from the metascheduler, making scheduling process independent from underlying grid middleware for job execution. This allows metascheduler to work with various remote execution utilities. The separation is achieved by the concept of Execution Plan (EP) for a workflow job. The job EP contains scheduling decisions for each task and mechanisms for dependency handling, and a separate runtime system translates the EP into execution-specific scripts for job execution and controls.

Workflow Support in GRACCE

GRACCE addresses two issues in supporting workflow applications in a grid scheduling architecture, workflow application description and workflow execution and monitoring. Compared to other similar efforts as we discussed in Section 6, GRACCE solutions to these two issues have the following advantages:

- GRACCE workflow application description are much more abstract and closer to end users, relieving users from specifying so much details about workflow structure and execution.

- GRACCE workflow execution control and monitoring components are designed as an independent components of workflow description and scheduling. Different components can be developed to support different remote execution utilities, such as different versions of Globus toolkit, raw remote rsh/ssh/rpc, or direct call to local schedulers.
- Pluggable scheduling algorithm in between

3 GRACCE Application Modeling and Description Language

Compared with other similar methods, GAMDL has the following features:

- Describes both application dataflow (DAG) and control logics (loops and branches);
- The supporting workflow and runtime systems are integrated with a grid metascheduler;
- Includes the job specifications of application modules to support workflow-orchestrated metascheduling. These specifications include job resource requirements and module execution history or profiles;
- Easy-to-use and powerful, for example, similar modules can be easily described using multiple-vale properties; the description document is structured and clear by using entity Uid and Uid reference.
- Supporting partial workflow definition

3.1 Structures and Basic Concepts

GAMDL provides two documents to describe a distributed application and its workflow, the *Application* document and the *AppRun* document. The *Application* document defines application entities, such as executables, data files and modules, and the dependency relationships of the defined entities. It includes four major child elements, *appExecutables*, *appDataFiles*, *appModules*, and *appMdRships*. The *AppRun* document describe a workflow of an application, such as the modules that are required for the workflow, the start module(s) and the start time of the workflow. The *Application* document provides a high-level abstract description of the application from the viewpoints of end users and it should encompass all the related entities for an application. The *AppRun* document specifies an execution of the application. The use of *AppRun* document gives users more flexibilities to specifies different workflows based on their needs without each time to redefine a new application. This is especially useful for some reoccuring execution of an application.

A module represents an application component whose execution accomplishes certain application tasks. A model consumes computational resources and input data set and generates output data. A module includes one or more module jobs, each of which is able to finish the module task. **An executable** is an application binary or script. An executable could be in different modules with different execution specifications and resource requests. **Data Files** are application files for module computations, including files already available in storage resources and files that are generated during execution.

The **uid (universal id)** attribute of an entity uniquely identifies the entity, and an entity can be referenced by its *uid*. The use of *uid*'s allows users to re-use the entities which are already defined for other applications. Application entities are normally defined in several documents. The *Application* or *AppRun* document *reference*s these documents and references the defined entities using their *uid*'s.

A **Multiple-value property (mvproperty)** is a property that may have multiple values. It is defined as $mvpName = \{v0, v1, ..., vn\}$, and is referenced by $\$mvpName$. $\#mvpName$ returns the number of values defined. A reference to $mvpName$ duplicates $\#mvpName$ times the referencing sentence; and in each duplicate, the reference is replaced with one of its values. For example, if we define $sign = \{+, -\}$, and $number = \{10, 100\}$, $\${sign}\${number}$ yields all the four possible combination of $\{+, -\}$ and $\{10, 100\}$, i.e. $\{+10, +100, -10, -100\}$. Mvproperties are widely used to define similar application entities, such as one executable with several execution instances. For example, if we define $md = \{mm5\}$, $dmsz = \{36K, 12K, 4K\}$, $day = \{1d, 2d\}$, the $uhaqf - \$md - \$dmsz - \$day$ mvproperty represents all 6 instances ($\#md * \#dmsz * \#day$) of the AQF MM5 modules in Figure 1.

3.2 Application Dataflow Descriptions

GAMDL models the dataflow of a grid application using the same concept as DAG, and specifies both the dependency relationships between modules and the associated intermediate files in these relationships. Dependency relationships are defined in either parent-children (PCn) pattern or child-parents (CPs) pattern. A PCn relationship has a parent module and one or more child modules, and a CPs relationship has a child module and one or more parent modules. Intermediate files in a relationship are specified as pipes. A **pipe** has a *pipeIn* and a *pipeOut* elements; *pipeIn* specifies the piped output file of the parent task, and *pipeOut* specifies the piped input file of the child task. Each pipe is only for one intermediate file.

GAMDL dependency specification allows the decision about whether a dependency should be handled to be made

during application execution by using conditioned pipes. A **conditioned pipe** associates a pipe with an *if* boolean condition which will be evaluated on runtime. If it evaluates *true*, the pipe is handled; otherwise, the pipe is not handled. If the conditions of all pipes in a relationship are evaluated as *false*, runtime dependencies are not established and the child module will not be executed.

AQF Example

The GAMDL document for AQF application in Figure 1 is shown in the following. The included mvproperty file `uhaqf.mvproperties` defines three mvproperties:

```
md={mm5, smoke, cmaq}
dmsz={36K, 12K, 4K}
day={1d, 2d}
```

Another mvproperty, `mdName($md,$dmsz,$day)`, is defined as `uhaqf-$md-$dmsz-$day`. `MdName` will be forked into 18 (which is `#md*#dmsz*#day`) values to represent all the AQF computational modules. AppExecutables, appDataFiles and appModules reference all the required entities by their Uids. **The complete description document is appended in Appendix A.**

Implementation Issues For appMdDeps element, we want to note here that the duplicating of mvproperty referencing sentences is per-element based in an XML document by a GAMDL preprocessor. When the preprocessor encounters a reference to a mvproperty, it duplicates the nearest outer element that contains the reference. This element is called the containing element of a mvproperty reference. The processor does not recursively process the same mvproperty references in the child element of the containing element, instead, instantiates all references to the mvproperty in a duplicate element with the same value. For nested mvproperty like `mdName($md,$dmsz,$day)`, the processor will not instantiate the referenced mvproperties in its definition. Instead, the processor first replaces it with its value string without duplicating the containing element, and then processes the document. Following these rules, the mvproperty references in the `uid` attribute of CPsRship element cause the duplication of an entire CPsRship element.

Based on the above rules, it is much easier to explain the appMdDeps element in AQF gridApp. AppMdDeps has nine CPsRship elements and here we just show one of them. It specifies the CPs relationships between the corresponding SMOKE and MM5 modules and will be forked into six (`#dmsz*#day`) CPs relationships.

Finally, we note that in specifying dependency relationships, mixing CPsRship and PCnRships are not allowed. The preprocessor checks the consistencies of these relationships and the intermediate files.

3.3 Control Logics Descriptions

Although GAMDL is mainly a dataflow definition language, it allows the specification of control structures, such as loops and branches. These structures are required for the workflow in which the module dependency relationships and the intermediate files can only be decided based on some runtime parameters. For example, one module generates different output files in different executions and these files are needed by different child modules. The PCn relationship and the associated intermediate files can only be decided when the parent module finishes. Some mechanisms similar as if-else programming statement are required to handle such situation.

GAMDL allows the specification of control logic by using conditioned pipes and variables. A **variable** is a <*name*, *value*> pair associated with an *if* condition. When assigning a new value to it, only if the *if* condition is evaluated as *true*, the assignment can be made. An assignment without an *if* condition is always made. If the *value* being assigned is in the format of *value1:value2*, *value1* is assigned if the *if* condition is evaluated as *true* and *value2* is assigned if the *if* condition is evaluated as *false*. A condition can reference system environment variables, variables defined in other modules and condition functions. A module may assign values to conditioned properties before its execution (in *preAssign* element) and/or after its execution (in *postAssign* element).

An Example of Workflow with Loops and Conditional Branches

In the workflow of Figure 3, module md2 generates different outputs (F1, F2 or others) in different loops and the outputs are processed by module md3, md4 or md5. The loop count is 100.

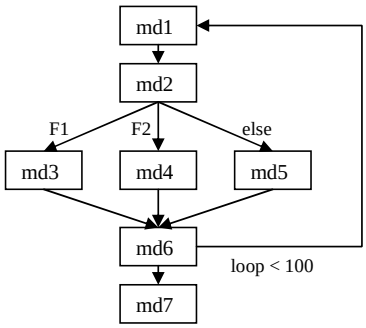

Figure 3. Workflow with Loop and Condition

In the GAMDL specification showing below, module md1 post-assigns a property “loop”, whose initial value is 100 and stride is -1. Module md2 post-assigns two properties, F1recent and F2recent. F1recent is set as “true”

if file F1 is generated by md2 in last execution, otherwise F1recent is set as "false"; and so as for F2recent. The pipe condition for `md3-md2-CPs` CPsRship is set as `pipe(F1)&&F1recent`, which is evaluated as "true" if F1 is generated in last execution and is available for piping in. The setup for F2-pipe in `md4-md2-CPs` CPsRship and and else-pipe in `md5-md2-CPs` CPsRship are similar. Loop control is specified in CPs relationship of md1 and md6; and a null pipe is created with condition `loop<100`. **See Appendix B for the complete document description.**

To conclude this section, we note that although GAMDL can specify complex workflow structures by using conditioned properties and conditioned pipes, their usage introduce additional complexities in reasoning application logics. For dataflow applications that can be modeled as DAGs, it is not necessary to use such features.

3.4 GAMDL Module Job Specification

In GAMDL, job specification is associated with module description so that job workflow can be easily constructed from module dependencies. A module have parent modules and child modules. In an *AppRun* document of an application, which specifies an execution, the included modules, their dependency relationships and the start module(s) constructs a module graph. The associated job(s) for these modules construct the execution workflow of the *AppRun*. Which job to chose to construct the workflow is left to the metascheduler to make the decisions.

GAMDL module job specification is similar as Globus RSL [20] or GGF Job Submission Description Language (JSDL) [18] in terms of specifying job submission details. But in GAMDL, resource allocation decisions, such as job execution hosts are not specified, which will be made by a grid metascheduler. After such decisions are made, GAMDL job specification can be translated to an RSL for Globus job submission.

Another difference between GAMDL job specification and RSL is GAMDL provides much more flexibilities than RSL for application-specific grid scheduling. In GMJSL, job resource specification are associated with application workflow. When allocating resources for module jobs, a scheduling system makes resource allocation decisions based on the module dependencies relationships, for example, sibling module jobs are allocated concurrent resources. Using RSL, users are required to specified the resource multirequests for the purpose of resource co-allocation [6, 7] based on application workflow. RSL itself cannot associates the workflow when specifying the resource requests, and scheduling systems have to refer to another workflow descriptions document to coordinate the application-level scheduling.

GAMDL also introduces job profile specification, thus allows a metascheduler to utilize the historical information of module executions to help resource allocation decision making. For applications like AQF that run everyday with similar scenarios, it is very easy to predict the execution scenario of a module on the resources on which the module has been executing. Based on such predictions, metascheduler can make much better decisions of resource co-allocation. Also statistical analysis, data normalization and scaling may also be performed on the history executions for other purposes in scheduling, such as the coordination between applications.

4 Workflow Execution and Monitoring in GRACCE

GAMDL is part of GRACCE metascheduling architecture. In GRACCE solutions, Domain users are only required to provide application descriptions and specify their resource requirements using GAMDL. GRACCE metascheduler is responsible for allocating grid resources for application modules, place module jobs on resources for execution and monitor them, and return the results back to users.

4.1 GRACCE Appdesc

GRACCE Appdesc is a set of utilities and GUI tools being developed for application users to use GAMDL and manage GAMDL applications. Currently, Appdesc includes a GAMDL preprocessor and parser, and a GUI interface to view an application description, and to launch and monitor an AppRun.

Appdesc GAMDL preprocessor is a utility to manipulate the GAMDL document defined by users. The major two functionalities of the preprocessor are to validate a GAMDL document, and to instantiate mvproperty references. In validation, the preprocessor validates the GAMDL XML document against its schema, and check the consistence of the document. Specific consistence checking includes DAG graph checking, dependencies relationships consistence checking, and entity reference, file reference and mvproperty reference checking. The preprocessor instantiates mvproperty reference following the rule we describe before.

Appdesc GAMDL parser translates a GAMDL document into Java objects as the internal representation of an application and its AppRuns. To support the integration between GRACCE metascheduler, workflow system and execution system, only the GAMDL parser is allowed to modify these objects.

Appdesc GUI is a Java graphical interface for end users to manage their applications and appruns. From Appdesc

GUI, users are able to load *Application* and *AppRun* documents and display them in a tree or graph view. See Appendix x for a snapshot of the Appdesc GUI. Appdesc GUI also provides users interfaces to generate execution schedule for an AppRun workflow, to launch the AppRun using one of the generated scheduler and monitor it.

Currently, Appdesc GUI does not support editing an application or appRun and users have to modify the GAMDL document to change the application or appRUn. But GUI and drag-and-drog editing feature is currently under development. Also, GRACCE itself only impleneted local schedule for an AppRun.

4.2 GRACCE Workflow System

The GRACCE workflow system is called **GridDAG**. GridDAG is not a complete workflow execution or enacting engine that is able to execute and monitor workflow tasks. Instead, GridDAG is an event-driven workflow coordination system that is responsible only for handling and coordinating tasks dependencies. EPExec runtime system executes and monitors workflow tasks based on the events sent from GridDAG. A standalone runtime system for executing and monitoring workflow tasks provides much more flexibilities in integrating workflow system, metascheduler and various grid utilities. Each of them can be self-developed and integrates together with defined interfaces. Various remote job execution system, such as Globus GRAM, ssh, rsh can be used in EPExec.

As shown in Figure 2, there are four components in GridDAG to support the eventing mechanisms, event chain builder, chain deployer, GridDAG agent, and DepResolver. The chain builder reads job execution plan forwarded from metascheduler and generate the event chains according to the EP. An event chain is a sequence of the events flowing between the event producer and consumers in the predefined order. The chain builder also decides who are the producer and consumer; and what events are going to be generated by each producer and to be received by each consumer. GridDAG agents installed on each grid resources coordinate the runtime event activities by playing the role of both event producers and consumers. Another GridDAG module, DepResolver is configured to received all event notifications and keeps track of the states of tasks' dependencies (a task may have more than one dependencies). When all dependencies of a tasks are resolved, DeResolver takes certain actions, which are typically sending requests to EPExec for job submission or control.

4.3 Pluggable scheduling module

For each appRun workflow, several execution schedule can be generated.

5 GRACCE Workflow in UH AQF Application

In GRACCE architecture, the scheduling and execution of AQF application is modeled as follows:

- AQF application is modeled and described in a GAMDL *Application* document.
- An AQF execution (could be a full execution or a partial execution) is specified in an *AppRun* document.
- Using GRACCE Appdesc, end users load AQF *Application* and *AppRun* documents.
- End users generate a schedule for an AppRun by specifying certain scheduling parameters, such as start time, reoccuring, etc.
- End users launch an AppRun using one of the generated schedule and monitor it.

In the tree layout of the AQF workflow in Figure 4, some modules can be concurrently executed.

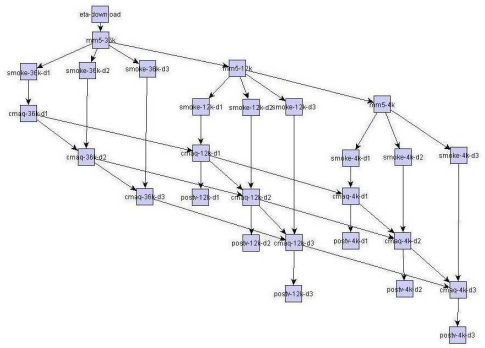

Figure 4. AQF Workflow Tree

5.1 Generate A Local Schedule

Workflow scheduling and enaction is often concerned with how to launch workflow modules in the proper order and how to handle module dependencies. The commonly-used policies is to launch a module or submit a job to launch the module when the parent modules are completed and their dependencies are properly handled. In a distributed or grid environment, it is often hard to find an immediately available resources for a ready module and a randomly submitted module job is often held in local queue. So, to schedule workflow for performance purpose (or to schedule workflow application with user perferred deadline), additional policies or techniques must be expolited. For example, how to schedule independent modules for the best concurrency, how to plan the execution of workflow modules,

or reserve resources in advance for module execution. Also, the processing of intermediate files should be considered in making the scheduling decisions.

To study the behavior of a workflow scheduler, in GRACCE, we first consider a local workflow scheduler, i.e., the workflow is executed on a single resource, such as a cluster or a SMP box, which often has shared file system. In such setup, if the intermedidate files between dependent modules are stored in the shared file system and accessible to all the modules, a child module can be started immediately as long as its parents modules generate the required intermediate files. So when schedule workflow modules, there is no need to schedule file transfers.

AQF modules have been executed on a 48-cpu Xeon cluster, and we have performed thorough profiling of the execution of each AQF module, including the wall-clock execution time using different number of cpus and the sizes of generated files. Most of the AQF parallel modules have the best performance and scalability around 16 cpus. The profiles for module execution time (in minutes) are shown in the following table:

Module	*T(m)*	*Module*	*T(m)*
ETA Download	20	CMAQ 36k d2	22
MM5 36k	48	CMAQ 36k d3	6
MM5 12k	48	CMAQ 12k d1	22
MM5 4k	220	CMAQ 12k d2	22
SMOKE 36k d1	24	CMAQ 12k d3	6
SMOKE 36k d2	24	CMAQ 4k d1	22
SMOKE 36k d3	6	CMAQ 4k d2	22
SMOKE 12k d1	5	CMAQ 4k d3	6
SMOKE 12k d2	5	POSTV 12k d1	18
SMOKE 12k d3	1	POSTV 12k d2	18
SMOKE 4k d1	4	POSTV 12k d3	5
SMOKE 4k d2	4	POSTV 4k d1	18
SMOKE 4k d3	1	POSTV 4k d2	18
CMAQ 36k d1	22	POSTV 4k d3	5

Table 1. AQF Execution Profile

In a 48 cpus machines, we partition them into three 16-cpus partitions, which is p1, p2, p3, which simulate three clusters.

5.2 local schedule performance

6 Related Works

A general-purpose application modeling and workflow description language for distributed and grid applications should satisfy below requirements from user point of views:

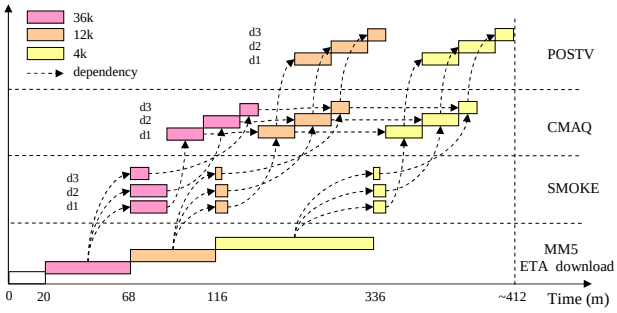

Figure 5. An AQF Execution Schedule

1. Able to describe application dataflow logics and advanced control structures, such as loops and branches. Dataflow constructs, for example, Directed Acyclic Graph (DAG), are the fundamental elements of a workflow language. The control logics are required for workflows with dependencies that cannot be decided in advance and can only be decided based on some runtime parameters;

2. Description should model the original logics of an application, requiring no additional translation from users that can be done by software tools. For example, if an application is defined in dataflow structure, description should not require users to extract the control-flow in order to use the description language;

3. Defined languages should be easy-to-use for domain users, requiring at most introductory knowledge of grid computing.

6.1 Related Works

Workflow related works have been extensively studied in [4], and there have been a number of efforts to provide a modeling method to define the workflow of distributed applications. Condor DAG (Directed Acyclic Graph) [14] allows the description of parent-child relationship of jobs, thus provides the basics for building application workflows. Gridbus workflow [5] describes application logics as DAGs using XML-based workflow language (xWFL). But both Condor DAG and Gridbus workflow do not handle runtime control structures such as loops and branches, and work for the applications whose workflow can be predefined statically. Business Process Execution Language (BPEL) [13], which combine IBM WSFL [22] and Microsoft XLANG [8] standard, is an XML-based workflow definition language to describe enterprise business processes. BPEL is in low-level web service level; additional extension and wrapping development are needed to make it easy of use by grid application owners. XScufl is a specific workflow definition language for Taverna project [21], but XScufl is too

fine grained and detailed for describing scientific applications. Abstract Grid Workflow Language (AGWL) [10] "describes" application control flow using constructs of imperative programming style and it is powerful enough to describe various complex application logics. But users are required to specified the details of the control flow, such as which modules can be parallel or sequentially executed. For dataflow applications, users have to translate the dataflow into control-flow to use AGWL.

Karajan of Globus CoG Kit [17] is a powerful workflow system that includes a workflow description language and a workflow engine. Similar as AGWL, Karajan workflow language provides constructs for users to specify grid related tasks, such as job execution and file transfer, and the execution flow of these tasks. Karajan workflow engine controls the execution of these tasks based on the specification. But using Karajan, users must know in advance the execution hosts of the computation tasks, and the source and destination hosts of file transfers. Such scheduling-related information should be provided by a global-level metascheduler, which is not addressed in Karajan. Requesting users to make decision about resource allocations in a dynamic changing environment is impossible. So Karajan is more suitable for a low-level workflow execution control, instead of for high-level application modeling and metascheduling.

7 Conclusion and Future Work

In this paper, we presented GRACCE Application Modeling and Description Language (GAMDL), a high level abstract language for domain users to describe a grid-ready application. GAMDL, motivated by AQF application, describes application dataflow structure so that users do not need to "program" the application control-flow. GAMDL is the basis for the integration between a grid metascheduler and workflow systems in GRACCE project. GAMDL specifies a module job within application description, thus job workflow could be easily constructed. The job specifications provide a rich set of information to help grid metascheduler make resource-allocation decisions. Other features of GAMDL include the use of mvproperty to easily describe similar entities, and the ability to describe complex control logics using conditioned properties and conditioned pipes.

We are currently using GAMDL for other applications and recommend users who have grid-ready applications to try it. Revising is possible based on users' comments and our metascheduler development progress in GRACCE. We are also investigating related semantic web standard [9] to make GAMDL more expressive and powerful.

References

[1] B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, A. Sim, A. Shoshani, B. Drach, and D. Williams, *High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies*, Proceedings of the ACM/IEEE SC2001 Conference, 2001

[2] B.M. Chapman, P. Raghunath, B. Sundaram, and Y. Yan, *Air Quality Prediction in a Production Quality Grid Environment*, Engineering the Grid: status and perspective, edited by J. Dongarra, H. Zima, A. Hoisie, L. Yang and B.D. Martino, Spring 2005

[3] I. Foster, C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001.

[4] J. Yu, and R. Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, Technical Report, GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, March 10, 2005

[5] J. Yu, and R. Buyya, *A Novel Architecture for Realizing Grid Workflow using Tuple Spaces*, Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, Nov. 8, 2004, Pittsburgh, USA

[6] K. Czajkowski, I. Foster, and C. Kesselman, *Resource Co-Allocation in Computational Grids*, Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8), pp. 219-228, 1999.

[7] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, *A Resource Management Architecture for Metacomputing Systems*, Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.

[8] S. Thatte, *XLANG-Web Services for Business Process Design*, Microsoft Corporation, 2001

[9] T. Berners-Lee, J. Hendler, and O. Lassila, *The Semantic Web*, Scientific American, May 2001

[10] T. Fahringer, J. Qin, and S. Hainzer, *Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language*, Proceedings of Cluster Computing and Grid 2005 (CCGrid 2005)

[11] W.F. Dabberdt, M.A. Carroll, D. Baumgardner, G. Carmichael, and R. Cohen *Meteorological research needs for improved air quality forecasting*, Report of the 11th Prospectus Development Team of the U.S. Weather Research Program, 2004.

[12] Y. Yan, B.M. Chapman, and B. Sundaram, *Air Quality Forecasting on Campus Grid*, Workshop on Grid Applications: from Early Adopters to Mainstream Users, GGF14, Chicago, IL 2005

[13] BPEL4WS: Business Process Execution Language for Web Services, http://www.106.ibm.com/developerworks/webservices/library/wsbpel

[14] DAGMan (Directed Acyclic Graph Manager), http://www.cs.wisc.edu/condor/dagman.

[15] Grid Application Coordination, Collaboration and Execution, http://www.cs.uh.edu/~yanyh/gracce

[16] Grid Physics Network, http://www.griphyn.org

[17] Java CoG Kit Karajan/Gridant Workflow Guide, http://www.cogkit.org/release/4_0_a1/manual/workflow.pdf

[18] Job Submission Description Language, GGF WG, https://forge.gridforum.org/projects/jsdlwg

[19] Linked Environments for Atmospheric Discovery, http://lead.ou.edu

[20] The Globus Resource Specification Language RSL, http://www-fp.globus.org/gram/rsl_spec1.html

[21] The Taverna Project, http://taverna.sourceforge.net

[22] Web Services Flow Language (WSFL), http:///www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf