Data and Activity Representation for Grid Computing

Amit Karnik and Calvin J. Ribbens
Department of Computer Science, Virginia Tech
Blacksburg, VA
{akarnik,ribbens}@vt.edu

1 March 2002

Abstract

Computational grids are becoming increasingly popular as an infrastructure for computational science research. The demand for high-level tools and problem solving environments has prompted active research in Grid Computing Environments (GCEs). Many GCEs have been one-off development efforts. More recently, there have been many efforts to define component architectures for constructing important pieces of a GCE. This paper examines another approach, based on a 'data-centric' framework for building powerful, context-aware GCEs spanning multiple layers of abstraction. We describe a scheme for representing data and activities in a GCE and outline various tools under development which use this representation.

1 Introduction

Computational scientists and engineers are rapidly turning to grid computing as the next source of massive computing power for their applications. But effective grid computing requires much more than raw computational power. Usability concerns dictate that high-level tools and problem-solving environments (PSEs) are needed to support large-scale scientific research on computational grids. The work reported here is part of an effort at Virginia Tech to design and build PSEs for multidisciplinary grid communities. The emphasis of our PSE and Grid Computing research group is on leveraging high-level problem-solving context to design architectures and tools which our applications collaborators can then use to build and modify their own computing environments. Our work builds on lower-level grid architectures, services, and toolkits being developed by the broader grid computing community, e.g., [3, 13, 16, 26].

The type of 'virtual organization' [12] we are targeting is the computational science research group. In working with several such groups, we have identified three themes that are crucial to effectively supporting this community. (see Ramakrishnan, et al. [25] for details). The first theme is *compositional modeling* [6, 18]. We use this term in a very general sense, referring to combining representations of parts of a computation to create a representation of a computation as a whole [11]. Thus, compositional modeling is merely an approach to problem-solving; we imply no particular implementation technology (such as distributed components, although that is one of the common ways of providing the functionality). Large-scale, multidisciplinary computational science requires sophisticated support for compositional modeling, and there are numerous research efforts underway to provide this kind of support, e.g., [2, 4, 7]. The second theme characterizing multidisciplinary grid communities is *collaboration*. The need to support collaboration is obvious,

given that we are targeting research *groups*. Collaborative computing is a large and active research community in its own right (e.g., [17, 1]) and is beyond the scope of this paper.

The third broad theme is *context*, and it is fundamental to the work described in this paper. In building computing environments for multidisciplinary grid communities, we believe there is a significant requirement for exploiting the larger context of the particular scientific application. There is also a great opportunity to leverage this context. The context we refer to is defined by the ongoing activities of the problem-solving community. Consider these simple examples: (1) a database of previous scientific results is used to improve the efficiency of current simulations or to avoid computation if a desired result is already available; (2) a simulation is being run as part of a higher-level problem-solving strategy, e.g., in an optimization loop or under the guidance of a recommender system; (3) a database of previous performance results is used by a scheduler so that grid resources are assigned intelligently; (4) a simulation is actually part of an ensemble of simulations, so that scheduling and simulation management tools can do sophisticated load balancing given greater information about the job mix. A primary emphasis of our work is to design an architecture for grid computing environments that exploits problem-solving context, broadly defined.

The remainder of this paper is organized as follows. In Section 2 we describe a high-level view of the kinds of activities that must be supported by context-aware GCEs. Section 3 discusses data and activity representation, the key issue in our data-centric approach. Finally, in Section 4 we illustrate the power of our approach by describing three tools we are developing in support of grid-based computational scientists.

2 A High-Level Architecture for Grid Computing Environments

Figure 1 shows a high-level architecture for organizing and building GCEs. This view should not be considered a complete architecture in the full sense of the word, e.g., with clearly defined interfaces between layers. Instead, it is a design framework that factors out conceptually independent layers of abstraction corresponding to different functions or modes that must be represented in an effective GCE. The layers in Figure 1 are based on three entities, defined as follows:

Model: A model is a directed graph of specific executable pieces defining the control-flow and data-flow in a computation. Note that we distinguish between a model and its representation in a GCE; the representation might involve only the model's name or it might involve a more sophisticated representation of each node in the digraph. Although models consist of ready-to-run pieces of code, these pieces may be parameterized.

Model Instance: A model instance is a model with all parameters specified. Note that some parameters may not be specified until runtime. Thus, while there might not exist a static conversion from models to model instances, the distinction between model instances and models is still useful. For example, using two different input data sets with the same model corresponds to two different model instances.

Simulation: A simulation is a model instance assigned to and run on a particular computational resource. It is useful to distinguish between a model instance and a simulation because, for example, a single model instance can be run (and re-run) many times using different computational resources or different random number sequences; each of these would be a new simulation by our conventions.

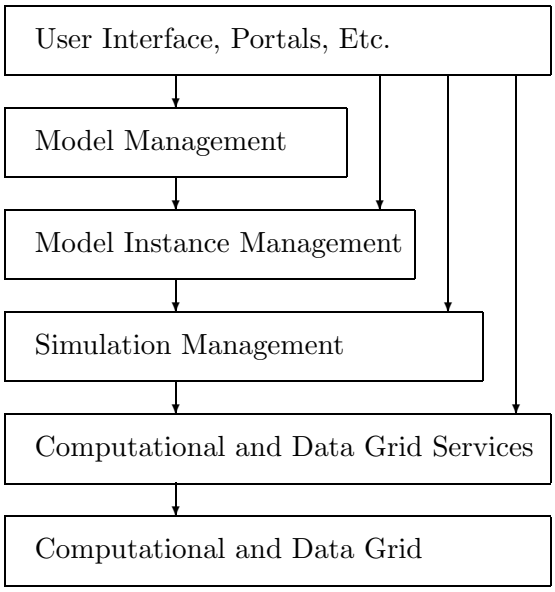

Figure 1: An architecture for data-centric computing environments for scientific grid communities.

Given these definitions, the framework summarized in Figure 1 can be used to organize the various functions which should be supported in a GCE for a typical scientific grid community. The Model Management layer is where users who need to create or modify models find tools to support this activity. Users who use existing models require only trivial support from this layer; others may require support for sophisticated compositional modeling. The primary activity in the Model Instance Management layer is parameter definition, whereby models are turned into model instances. The core service at the Simulation Management layer is assigning model instances to grid resources. (We describe many more activities in the Model Instance and Simulation Management layers below.) The lowest two levels in Figure 1 correspond to software and hardware resources that make computational and data grids possible. Since our emphasis is on high-level, application-specific issues, we omit further discussion of the architecture, protocols, and services being developed elsewhere for these levels.

Not all services or activities fit neatly into the categories shown in Figure 1, of course. For example, in 'computational steering' [21] model parameters may be modified and computational resources re-assigned at runtime; so model instance and simulation definition services are interleaved with execution in this setting. Furthermore, there are important aspects of an effective GCE that are not explicitly represented in Figure 1. For example, support for collaboration is implicit throughout. However, this high-level view of required layers of functionality helps organize and orthogonalize our efforts. In keeping with the typical end-to-end design philosophy of the grid computing [14], we have attempted to provide support for these new services as layers of abstraction over traditional low-level grid scheduling and resource management facilities. Our resulting high-level architecture 'factors out' typically blurred layers into distinct levels at which various services can be provided.

The goal of this research is to provide a flexible framework for building powerful, context-aware grid computing environments. Our approach is 'data-centric' in that we emphasize context in its broadest sense. In this view, everything is data—models, model instances, simulations, problem-solving strategies, etc.—and this data comprises the context we wish to exploit. In this paper

we focus on two layers in the above architecture, namely the Model Instance Management layer and the Simulation Management layer. Before turning to the key question of representation, we describe typical activities belonging to these two levels of abstraction.

In our interaction with computational scientists, we find that much of the interesting work takes place at the Model Instance Management level. In the simplest case, users assign parameter values to a model and ask a simulation management tool to execute the simulation. (Note that we are using 'parameter' in a very general sense. A parameter can be a problem or algorithm parameter, an input file, or any problem-solving module.) This level also includes tools for generating sets of model instances, e.g., ensemble generators and parameter sweep tools [8, 9]. High-level problem solving strategies and activities belong at this level as well. For example, scripting languages are used to 'program' model instance generation in the context of an outer optimization loop, or adaptation and caching strategies are employed, e.g., adjusting algorithm parameters or using a recently computed result as an initial guess for a subsequent computation. Another high-level strategy found at this level is a 'database query' mode, where scientists describe sets of model instances in an SQL-like language, and previously computed results are retrieved or new results computed, as needed. Finally, model instance management includes tools that analyze or reason about model instances. An emerging example is the use of 'recommender systems' [23, 24] which use data-mining strategies to help users select problem or algorithm parameters, based on a database of previous results.

The Simulation Management level also includes a wide variety of activities, when distinct concerns are factored out. In the most straightforward case, users define a simulation by selecting resources, and use grid and data services to interact with those resources. Middleware such as Globus [13] provides the tools needed to perform these tasks, including authentication, staging, running, and retrieving of results. More interesting possibilities for simulation management arise when we consider assigning compute resources to sets of model instances. For example, better scheduling and load balancing strategies are possible when we know more of the context (e.g., number of model instances, dependencies between instances, and requirements of each) or when we have performance results from previous simulations. More sophisticated modes of interacting with running simulations also belong at this level, including collaborative monitoring and steering of simulations, fault tolerance, and job migration. Also found at this level are services that allow interaction with a data grid, e.g., insertion, retrieval, and mining.

3 Data and Activity Representation

It is clear from even the brief discussion above that a wide variety of context-aware tools and environments will be needed to support grid computing communities. The challenge will only increase as grid computing gains in popularity among scientists outside the current generation of grid 'power users.' Flexibility, usability, and extensibility will be of paramount concern. Building vertically integrated, problem-specific environments from scratch for each problem-solving context is not a good idea; it requires too much effort and results in complex and inflexible systems. Hence, many research groups are working to define abstractions, architectures, and frameworks that allow environments to be built out of simpler, pre-existing components. Projects such as CCA [2], Harness [4], Cactus [7], and many others are making good progress in this direction. These approaches rely on carefully defined interfaces, strictly enforced via object oriented technology. Such a strategy is certainly useful *within* each of the layers illustrated in Figure 1, e.g., in defining complicated models by composing simpler models. However, we believe a different approach is

needed in order to allow easy construction of GCEs spanning several of the layers in Figure 1.

Our approach does not attempt to define precise interfaces and component models for each layer of abstraction. Instead, we take a more flexible, data-centric view. By 'data-centric' we mean that *everything* in the problem-solving context can be viewed as data. We are building tools which treat as 'data' things such as a name (e.g., an executable, a machine, a user's data file), a model of a physical system, an ensemble of simulations, a binding between simulation components, a load-distribution, or a set of performance results. Data can also include activities, such as constructing a parameterized model, searching a database for previous results, running a set of simulations. By taking this approach, we believe a wide variety of tools can be built and combined to form flexible, powerful, context-aware GCEs.

A data-centric strategy implies that representation is a central concern. We require representation schemes that are sufficiently powerful, expressive, and extensible, but which are also relatively concise. In this section we describe a scheme we are defining for representing data and activity at the Model Instance Management and Simulation Management levels of abstraction.

Simulation Definition Language

Our current effort in representing the data and activities occurring at the Simulation Management level in a GCE is motivated by the most basic of these activities, namely running a piece of code on a grid resource. This task involves staging input files and executables onto a grid resource, executing the code and getting back the results.

A uniform manner to represent and share data and information is through Extensible Markup Language (XML) [10], which allows data elements to be marked-up using tags. XML allows designers to define the tags themselves, resulting in a very flexible but powerful tool to represent any data. Bivens, et al. [5] use a similar XML-based approach to representing workflows in a grid environment. We have defined a Simulation Definition Language (SDL), based on XML, which supports the basic scenario outlined above, but is easily extended to a wide range of data and activities required for model and simulation management. Several of the basic SDL tags are illustrated in Figure 2. The meaning of the tags is self-explanatory.

4 Example Tools

We find that SDL is sufficiently powerful to represent a wide variety of simulations and activities involving simulations. This section illustrates the power of our data-centric framework for constructing context-aware GCEs by briefly describing three tools we are building to support collaborations with computational scientists at Virginia Tech. Each tool supports a model instance management or simulation management activity, as defined in Section 2; each relies on SDL to represent simulations.

4.1 Job Submission Tool

The most basic simulation management activity is submitting and retrieving the results of a single simulation. Figure 3 summarizes our core tool for running a simulation using grid services supplied by the Globus toolkit. In fact, the figure shows several tools in a generic architecture for connecting the Simulation Management and Grid Services levels of abstraction. Although Globus is assumed here, the architecture is easily generalized to support other grid service layers, e.g., Legion [19].

```
<simulation>
  <host nprocs="50">anantham.cs.vt.edu</host>
  <working_directory>/raid/home/akarnik</working_directory>
  <executable>my_app</executable>
  <stdin>st_in</stdin>
  <stdout>st_out</stdout>
  <input_files nfiles="4">
    <file>in1</file>
    <file>in2</file>
    <file>in3</file>
    <file>in4</file>
  </input_files>
  <output_files nfiles="2">
    <file>out1</file>
    <file>out2</file>
  </output_files>
</simulation>
```

Figure 2: Sample SDL file.

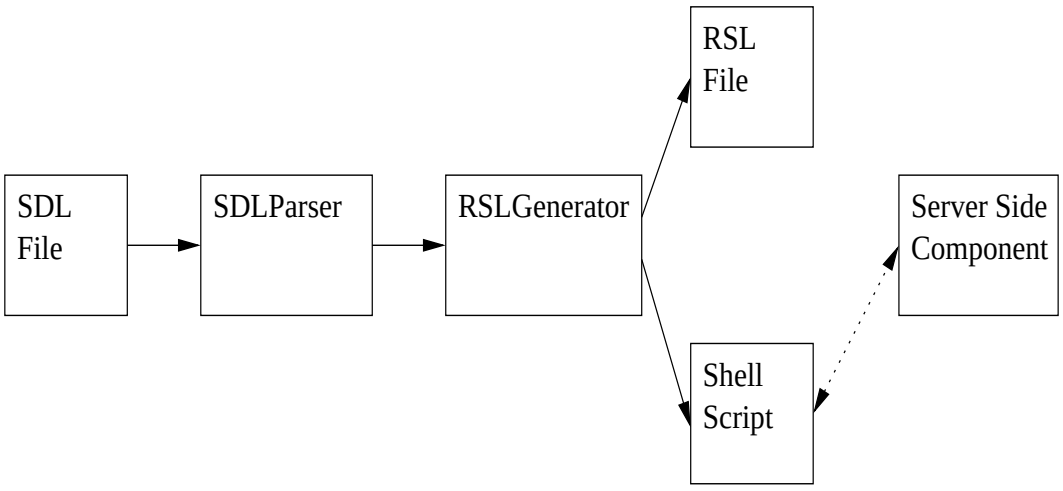

Figure 3: An SDL-to-RSL framework for running simulations

Globus provides a Resource Specification Language (RSL) [15] to specify details of job execution over the grid. RSL is based on attribute-value pairs and allows users to describe resource requests and job definitions. In order to run a simulation represented in SDL on a Globus-based computational grid, we must first convert the SDL representation into the corresponding RSL definition and then submit the RSL definition to Globus. Since SDL is based on XML, we can extend the XML parser Xerces [27] to build the SDLParser tool, which parses an SDL file and generates an intermediate representation (currently just a flat file). The RSLGenerator then takes this intermediate representation and produces two output files: an RSL file containing attribute-value pairs that enable Globus to run the simulation on a grid resource, and a shell script containing commands to setup environment variables, transfer input files, submit the RSL file to Globus, and retrieve output files.

A final component of the architecture represented in Figure 3 is a resource-side component responsible for any pre- and post-processing steps required to run the job on that particular resource. The ServerSide tool is site-specific, reflecting the software environment of the particular resource. For example, we have implemented a resource-side component for the Portable Batch System (PBS) [22] and Maui Scheduler [20] running on a 200-node Linux/Myrinet cluster. In this case the component is aware of PBS-specific issues such as job-submission scripts and available queues. It generates the required files and directory structures for running a job in the PBS environment, and interacts with PBS and Maui to monitor job progress and retrieve results. Although, the ServerSide component does not interact directly with SDL, having a resource-aware component underneath the Grid Services layer is consistent with our approach of separating independent concerns. It allows us to keep SDL quite general and does not require us to expose PBS-specific details in the SDL or RSL specifications. In this way, the same representation of a Globus job, namely the '(RSL, script)' pair, is portable to other computing resources.

4.2 Parameter-Sweep Tool

Computational grids are frequently used for experiments expressed as a parameter sweep. We are implementing a tool which allows users to perform a parameter-sweep experiment conveniently. The tool consist of three sub-systems: an XML Generator, a Parameter Sweep Definition tool, and a Sweep-Engine. A high-level architecture is shown in Figure 4.

The XML Generator takes as input a sample application input file and produces an XML representation of a typical input file, identifying the various parameters in it. The Parameter Sweep Definition tool allows the user (interactively) to indicate parameters and ranges defining an experiment, and uses the XML file to produce a parameterized input file. The parameterized input file is then given to the Sweep-Engine, along with the XML file defining the generic structure of an input file, and a set of individual simulations and appropriate application input files is generated. For each of these simulations a SDL representation is produced, which can then be submitted to a computational grid.

4.3 Simulation Lookup Tool

A third tool made possible by our approach to simulation representation is a database 'lookup' tool. This is a simple example of the idea of taking advantage of previously computed results. All simulations run in the GCE can be stored using their SDL description in a database, currently implemented as a collection of files in an organized directory structure. We are developing a simulation-lookup tool which allows users to search the database for simulations matching certain

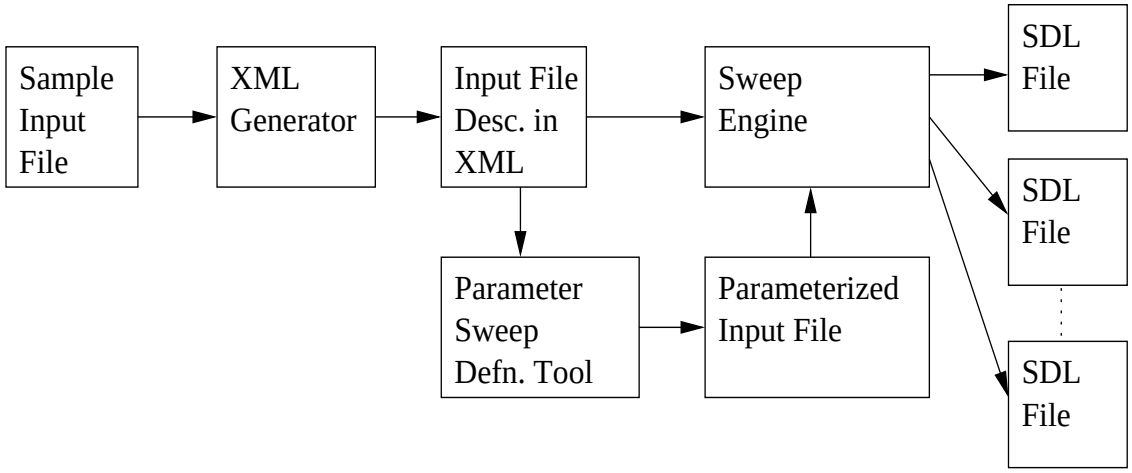

Figure 4: Parameter Sweep tool

criteria. The obvious merit of such a tool is that in a collaborative GCE, expensive simulations done by others need not be re-run. For example, in a parameter sweep, it may be that a subset of the requested simulations has already been completed. The results from the previous run can simply be presented to the user. This tool will also support a more general 'database query' tool, where sets of model instances are described through an SQL-like interface; the tool retrieves previous results or generates new simulations as needed. We are also exploring problem-specific query-optimization techniques in this context.

References

[1] D. A. Agarwal, S. R. Sachs, and W. E. Johnston. The reality of collaboratories. *Computer Physics Communications*, 110:134–141, 1998.

[2] R.C. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a Common Component Architecture for High-Performance Parallel Computing. *HPDC '99*, 1999.

[3] M. Baker, R. Buyya, and D. Laforenza. The Grid: International Efforts in Global Computing. In *Proceedings of the International Conferences on Advances in Infrastructure for Electronic Business, Science and Education on the Internet(SSGRR 2000)*, Italy, 2000.

[4] M. Beck, Jack Dongarra, Graham Fagg, Al Geist, Paul Gray, James Kohl, Mauro Migliardi, K. Moore, T. Moore, P. Papadopoulos, S. Scott, and Vaidy Sunderam. Harness: A next generation distributed virtual machine. *Special Issue on Metacomputing, Future Generation Computer Systems*, 1999.

[5] Hugh P. Bivens and Judy I. Beiriger. GALE: Grid Access Language for HPC Environments. Available at sass3186.sandia.gov/ hpbiven.

[6] J.C. Browne, E. Berger, and A. Dube. Compositional Development of Performance Models in POEMS. *International Journal of High Performance Computing Applications*, pages Vol.14(4):pages 283–291, Winter 2000.

[7] Cactus. www.cactuscode.org. Visited 12/10/2001.

[8] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLes Parameter Sweep Template:User-Level Middleware for the Grid. In *Proceedings of the Supercomputing Conference(SC' 2000)*, 2000.

[9] A. DeVivo, M. Yarrow, and K.M. McCann. A comparison of parameter study creation and job submission tools. Technical Report NAS-01-002, NASA Ames Research Center, Moffet Field, CA, 2000.

[10] Extensible Markup Language. www.w3.org/xml/. Visited 12/10/2001.

[11] K.D. Forbus. Qualitative Reasoning. In A.B. Tucker, editor, *The Computer Science and Engineering Handbook*, pages 715–733. CRC Press, 1996.

[12] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 2001.

[13] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Supercomputer Applications*, 11:115–128, 1997.

[14] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, Orlando, FL, 1999.

[15] Globus Resource Specification Language. www.globus.org/gram/rsl_spec1.html. Visited 12/10/2001.

[16] Grid Portal Development toolkit. dast.nlanr.net/features/gridportal/. Visited 12/10/2001.

[17] Jonathan Grudin. Computer-supported cooperative work. *CACM*, 32:30–34, 1991.

[18] K. Keahey, P. Beckman, and J. Ahrens. Ligature: Component Architecture for High Performance Applications. *International Journal of High Performance Computing Applications*, pages Vol.14(4):pages 347–356, Winter 2000.

[19] Legion. www.legion.virginia.edu. Visited 2/4/2002.

[20] Maui Scheduler Project. mauischeduler.sourceforge.net. Visited 2/28/2002.

[21] S.G. Parker, C.R. Johnson, and D. Beazley. Computational Steering Software Systems and Strategies. *IEEE Computational Science and Engineering*, pages 50–59, October–December 1997.

[22] Portable Batch System. www.openpbs.org. Visited 2/28/2002.

[23] N. Ramakrishnan, E. Houstis, and J.R. Rice. *Working Notes of the AAAI-98 Workshop on Recommender Systems*. American Association for Artificial Intelligence, Cambridge, Massachusetts, 1998.

[24] N. Ramakrishnan and C.J. Ribbens. Mining and Visualizing Recommendation Spaces for Elliptic PDEs with Continuous Attributes. *ACM Transactions on Mathematical Software*, pages Vol.26(2):pages 254–273, June 2000.

[25] N. Ramakrishnan, L. T. Watson, D. G. Kafura, C. J. Ribbens, and C. A. Shaffer. Programming Environments for Multidisciplinary Grid Communities. *Concurrency and Computation: Practice and Experience*, 2002. To appear.

[26] S. Verma, M. Parashar, G. von Laszewski, and J. Gawor. A Corba Community Grid Toolkit (CoG). In *Proceedings of the 2nd International Workshop on Grid Computing*, Denver, Colorado, 2001.

[27] Xerces XML parser. xml.apache.org/xerces-c/index.html. Visited 12/10/2001.