End-User Tools for Grid Computing

Francisco Hernández Purushotham Bangalore Kevin Reilly

Department of Computer and Information Sciences
University of Alabama at Birmingham
1300 University Boulevard, Birmingham, AL, USA
{hernandf, puri, reilly} @cis.uab.edu

ABSTRACT

The present work describes an approach to simplifying the development and deployment of applications for the Grid. Our approach aims at hiding accidental complexities (e.g., low-level Grid technologies) met when developing these kinds of applications. To realize this goal, the work focuses on the development of end-user tools using concepts of domain engineering and domain-specific modeling which are modern software engineering methods for automating the development of software. This work is an attempt to contribute to the long term research goal of empowering users to create complex applications for the Grid without depending on the expertise of support teams or on hand-crafted solutions.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Programming Environments – *graphical environments, integrated environments, programmer workbench.*

General Terms

Design, Human Factors, Languages.

Keywords

Grid Computing, End-user Tools, Software Engineering, Domain Engineering, Domain-Specific Modeling, Visual Authoring Tools, and Automatic Programming.

1. INTRODUCTION

A recent issue of Communications of the ACM [2] included several articles about *End-user development* which addressed "Tools that empower users to create their own software solutions." Sutcliffe and Mehandjiev, in this same issue, indicate that by 2005 a small fraction of developers in the U.S. (approximate 2.75 million out of an estimated 57.75 million) will be professional developers, the huge majority (then) being end-user developers using tools such as spreadsheets, query systems, or scripting interactive websites [25]. However, these benefits have not reached the area of scientific computing and Grid computing in particular. Developing applications for the Grid remains difficult for many users.

Grid computing is a distributed computing approach that permits the aggregation of resources belonging to different administrative domains. This aggregation offers extensive processing capabilities but at the same time it increases the complexity required to develop such applications. This is due in part to the complexity of the distributed resources, where even potentially inexperienced users are exposed to all the details of the underlying Grid technologies [15]. Another reason is that current software engineering practices (e.g., reusability, modeling and rapid prototyping) have not been fully explored for the Grid model.

Traditionally, modern software engineering practices have experienced slow adoption in the area of scientific computing. This is due to the importance of efficiency that scientific computing requires [11]. Nevertheless, there are a few examples in which methodologies such as generic programming [21], domain engineering [12], and component-oriented programming [4] had been successfully applied in scientific arenas. We may expect more successes insofar as advancing hardware progress can stimulate software approaches which today may appear less than fully efficient.

This paper presents an approach for constructing end-user's tools that automate the development of applications for the Grid. Our focus is to enable inexperienced users take full advantage of the Grid infrastructure. The approach presented in this paper provides a high-level abstract layer for the construction of Grid applications. This layer is composed of visual models of specific application domains and it is constructed using concepts of domain-specific modeling. Programs that manage the application execution are generated from the corresponding visual models. Thus, users need not learn how to use the specific Grid technologies in order to develop Grid-enabled applications.

The remainder of this paper is organized as follows. Section 2 provides an introduction to Grid computing and the problems faced when developing Grid applications. Section 3 enumerates current approaches aimed at facilitating the development of these applications. The proposed methodology for creating Grid end-user tools is introduced in Section 4. Section 5 presents an example of an initial tool developed for facilitating the creation of Grid applications. Finally, Section 6 gives conclusions of our work.

First Workshop on End-User Software Engineering (WEUSE I). May 21 2005, Saint Louis, Missouri, USA.

2. GRID COMPUTING

Grid computing is a distributed computing approach which offers computing specialists and other scientists a valuable resource to access extensive processing capabilities. The distinctive feature of this approach resides in creation of virtual organizations that permit seamless aggregation of heterogeneous resources (e.g., processing units, data storage, and devices) that can belong to different administrative domains.

Middleware has traditionally been used to provide virtualization of resources. The Globus Toolkit [10] is the de facto standard. However, with the recent advent of Web services, Grid middleware has evolved and converged (with Web services) into what is now called Grid services. Grid services are Web services at the base providing interoperability, heterogeneity, and platform independence, but Grid services have added functionality that enables them to work in a Grid environment such as support for life cycle management, and notifications [22].

Despite expanded availability of capabilities provided by Grid services, they are mostly being taken advantage of by academic and industrial centers. This is because developing Grid services is a complex process as can be seen from the steps required to develop a single service:

1. Implement the functionality of the service which might require the use of scientific libraries and constraints on efficiency expected for the service. Services can be as coarse as a complete simulation or as fine as a multiplication of two matrices.
2. Write a functional interface for the service so that clients know how to invoke it.
3. Define the deployment parameters to indicate the specifics of the service.
4. Deploy the service to a Grid service container and register it so that it can be found by other services.

Considering that most of the previous tasks are performed manually, the result is that few applications exist that can readily exploit the full Grid potential, and most of them have been written by Grid specialists instead of scientists or engineers [6]. In our view, this problem is but one of the factors impeding the increase of Grid users. The other major factor is the large number of existent non-Grid applications (a.k.a. legacy applications) originally designed for executing on dedicated supercomputers or parallel computers with no attention to distributed computing and yet need to be moved to a Grid environment [18].

Insofar as the process of developing services is complex, the process is aggravated when we consider that complex applications typically consist of more than one service. Then, special effort has to be employed to compose and orchestrate services (composing refers to adding the functionalities of relatively simpler services to produce a complex application while orchestrating refers to the correct sequencing of services and their outputs required to produce the desired result) which, to an extent, requires specific knowledge on Grid technologies.

To exemplify how arduous is the process of composing services, we enumerate the steps required to perform this composition:

1. Discover the service from a registry which in response indicates the location of the service.
2. Obtain the service description which indicates how to use the service.
3. Generate code needed to invoke the service according to its description.
4. Use the code generated in the previous step to connect the service to the application or service being implemented.

Since these steps have to be repeatedly performed for each composition of services, there seems to be a consensus that this process needs to be facilitated. In the following section we briefly review some prominent current directions at this facilitation.

3. CURRENT DIRECTIONS

3.1 Problem Solving Environments (Portals)

The first direction attempts to simplify use of the Grid by creating Problem Solving Environments (PSE) or portals [1]. These tools simplify the use of the Grid by supplying a repository of ready-to-use applications that were previously created and can be reused utilizing different inputs.

In order to hide the complexities of the Grid, portals appear to expedite only simple tasks (e.g., job submissions, and checking job status) [20], and seemingly lack the flexibility required to create complex applications made from composing different services.

3.2 Workflow Systems

Another direction focuses on facilitating the construction of applications by creating a workflow of services composing the application [3], [5], [8], [25], [19]. This technology is borrowed from business processing in which workflow languages like BPEL [24] have successfully been used to compose and orchestrate business related Web services.

Due to the similarities between the Grid and the Web, using workflows appeared to be a suitable methodology to facilitate the composition of complex Grid applications. However, as is the case with portals, workflow systems require services to be independently developed and stored in a repository for later use. And as seen above, this process is complex and often requires (as well) the expertise of a multidisciplinary support team.

3.3 Component Frameworks

Even though the simplification of the composition of services is of paramount importance, the development of the individual services is equally important. The final research direction attacks this problem by constructing frameworks that ease the implementation of individual components. Component frameworks are engineered to facilitate and accelerate development of applications by focusing on the reuse of individual components.

There are various examples of this solution with CCA [4] being probably the most prominent example. However, there are three problems with this approach:

1. Most component frameworks were developed before the Grid "era" which means that they have to be adapted to this new technology (wrappers being the choice most of the time).
2. Different component frameworks are not standardized, thus are not configured to be reused in frameworks other than the one of the original design.
3. Current procedures for composing components are focused at the code level where the complexities imposed by the programming languages and the component frameworks themselves impede their use for non developers for whom a higher abstract level is more advantageous.

A result of these problems is that multidisciplinary support teams are once again required for using component frameworks in a Grid environment.

4. METHODOLOGY

As seen in the previous analysis, Grid computing relies heavily on support teams. This is contrary to many other areas of computer science where there has been extensive research to create tools that empower users to create complex applications without the need of such teams. Though most Grid applications are still being developed in standard programming languages and using standard approaches, there seems to be a need of support tools that function in domains more familiar to end-users. Furthermore, for these tools to be effective it is of paramount importance that they should not be based on ad-hoc methods but instead rely on modern software engineering practices that can not only increase software quality but also improve the development of such tools.

As explained in the Grid Computing section (section 2) above, creating Grid applications consists of two separate issues: (1) the creation of the individual application components (or services), and (2) the deployment of those components in the distributed resources. Accordingly, the difficulty of creating Grid applications resides in acquiring a proficient knowledge in the use of the different Grid technologies (e.g., Grid middleware or Grid Services). However, in order to increase the number of individual researchers that utilize the Grid, it is an imperative to hide the accidental complexities of use (i.e., specific details of Grid technologies) from the end-users and embed this knowledge into a code generator that can generate the complex configurations. A leading technology that helps when working at this level is Domain-specific modeling (DSM) [13], which enables users to employ familiar concepts to the domain while constructing models of applications. These models can then (even) be translated into one or more representations. The benefit of creating these models is that the models can be manipulated as first class development artifacts which means that work with them can be automated [14].

The process for creating the domain models entails the following issues:

1. Analyze the domain in order to extract concepts relevant to the domain as well as knowledge on how to build applications in that domain (this process is also known as domain engineering [11]).
2. Build a meta-model with the knowledge extracted during the domain engineering steps, creating in the process a graphical domain-specific language to specify applications for the domain.
3. Create a model interpreter to generate the appropriate low-level configurations. The model interpreter provides the semantics for the visual models.

Users then interact with the graphical models which represent concepts familiar to them and the corresponding Grid applications are automatically generated by the tool. The particular code that enables use of the distributed resources can be reused because the underlying Grid technologies are the same on every application domain. This means that this code can be optimized and developed by Grid experts and then used by end-users working on different application domains.

The quality of the applications is also improved since the rules of the programs that can be created are embedded in the meta-models. This often means that only valid models can be created, illegal models being rejected at modeling time. Bugs are also minimized at earlier stages since the tool generates code that was already tested and was developed by experts in the Grid area.

Furthermore, the development of the modeling tool, in general, is facilitated by the use of MetaCASE tools which, according to Czarnecki et al. [7] are beneficial for this endeavor since they provide support for meta-model editing as well as the creation of new notations.

5. EXAMPLE

This section presents an initial exploration for creating a tool that automates work with the Grid. This example involved the creation of a general workflow system that abstracts and simplifies the development of Grid applications by hiding the low-level implementation details of the Grid middleware. The intention is to facilitate the deployment part of the Grid application construction process. The resulting tool helps the inexperienced Grid users by providing an environment in which they can graphically specify the workflow for their application and automatically generate the code that manages the execution of the application (For an in-depth explanation of this tool such as its capabilities and the range of applications that can be created the reader is referred to [16], [17]).

Applications can be created by specifying jobs on distributed resources and by specifying file transfers between those resources. Usually, jobs require one or more input files and also produce one or more output files. The following example presents a simple application using Hidden Markov Models to illustrate how this interaction is performed by this tool.

A Hidden Markov Model (HMM) was constructed to compare the differences between English and Spanish language patterns [9]. The input to the HMM is an intermingled file (parts in English and parts in Spanish) that only indicates if a letter is a

vowel or a consonant (1 or 0). The output file consists of the language prediction.

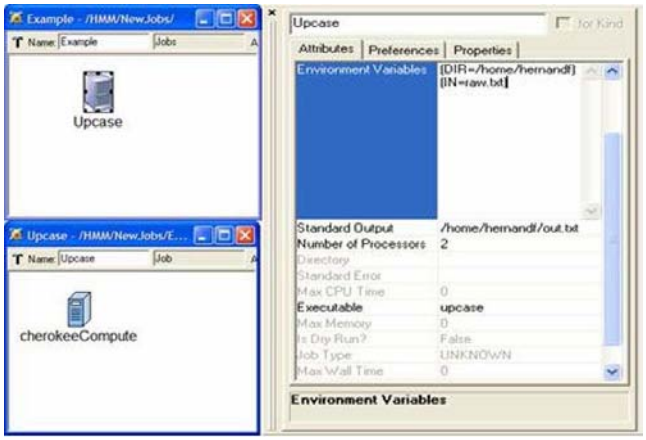

Figure 1. Model specification for Upcase task.

Figure 1 illustrates the manner in which a job is specified by the users. The name of the executable file, environment variables, name of the output file, number of processors required, and the machine in which this job is to be executed (in this case cherokeeCompute) are required to specify this particular task. Figure 2 shows the corresponding Java code that is generated from the model information.

```
1  ……
2  GlobusRSL UpcaseRSL = new GlobusRSL();
3  UpcaseRSL.setEnvironmentVariables
4            ("(DIR=/home/hernandf)(IN=raw.txt)");
5  UpcaseRSL.setExec("upcase");
6  UpcaseRSL.setNumProc(2);
7  UpcaseRSL.setStdOut("/home/hernandf/out.txt");
8  ……
```

Figure 2. Code generated by the model interpreter for the Upcase job.

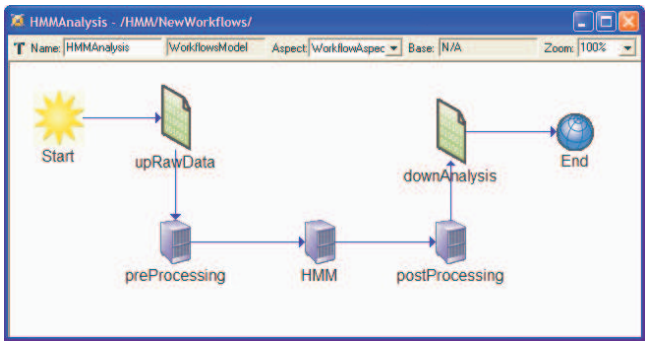

Figure 3. Definition of the application as a model.

After all of the tasks are defined, the application can be constructed by specifying the required sequence of tasks (Figure 2). File images indicate file transfers, and computer images indicate jobs to execute. The star in the far left indicates the start of the Grid application, and the sphere on the far right indicates its end. The input file is copied to the remote host (upRawData). A preprocessing job is executed on that file and its output is analyzed by the HMM job. The output of the HMM job is then modified in the postProcessing step. Finally the output of the postProcessing job is downloaded to the local computer (downAnalysis).

After the model is specified, a model interpreter traverses the internal representation of the model and generates the control code that manages the application execution. With this tool, end-users need only to know the particulars of their applications and are not required to learn the manner in which the Grid technologies operate. Nevertheless, they are able to specify complex applications and execute them in distributed resources.

6. CONCLUSIONS

The goal of the research described in this paper is to improve the development of tools that automate the creation of Grid applications from particular domains. Tools created using this approach permit the graphical definition of models and the automatic generation of the code that controls the execution of the Grid applications. Our current focus has been on the deployment aspect of the construction process but, as noted in the introduction, the development of the individual components is an integral part of the process. Our future work considers this aspect.

This research is based on domain-specific modeling techniques. The benefits of using these techniques which motivated this study were:

1. Domain modeling focus on higher levels of abstraction at the problem space *rather than solution space*, such as specific Grid middleware and their usage. End-users have a better understanding of the applications by working at the problem space.
2. Modeling tools and their code generators facilitate the more rapid ability to change the application's details. That is, it is easier to manipulate and change domain models rather than the associated code. Furthermore, the domain knowledge is embedded in the rules that govern the visual models (meta-model) as well as in the model interpreters.
3. Quality of the systems is improved since the high level models only permit the specification of correct models and the low level implementations are coded by Grid experts.
4. The use of MetaCASE tools facilitates the rapid development of Grid tools for different application domains. They do this by providing different facilities such as a language by which new meta-model notations can be specified and overall graphical support for interacting with the models.

Using these modeling techniques, an example tool was constructed. This tool abstracted the Grid domain and permitted the specification of applications that were able to run in distributed resources. Users were able to submit their applications by graphically specifying the details of their application.

End-users are able to better understand models that are expressed in their day-to-day language rather than in

cumbersome and often extraneous programming languages. This kind of tools will help enable end-user developers to gain access to the processing capabilities of the Grid without depending on the expertise of support teams resulting in end-users' ability to create complex applications by themselves.

7. REFERENCES

[1] Special Issue: Grid Computing Environments. *Concurrency and Computation: Practice and Experience*, 14:1035-1593, 2002.

[2] End-user development: tools that empower users to create their own software solutions. *Communications of the ACM*, 47(9), September 2004.

[3] E. Akarsu, F. Fox, W. Furmanski, and T. Haupt. WebFlow – high level programming environment and visual authoring toolkit for high performance distributed computing. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, pages 1-7, 1998.

[4] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a common component architecture for high performance scientific computing. In *Proceedings of the 8th. IEEE International Symposium on High Performance Distributed Computing*, 1999.

[5] H. Bivens. *Grid Workflow.* Grid Computing Environments Working Group Document, 2001. http://zuni.cs.vt.edu/grid-computing/papers/draft-bivens-grid-workflow.pdf. [February 8, 2005].

[6] C. Boeres and V. Rebello. EasyGrid: Towards a framework for the automatic grid enabling of legacy mpi applications. *Concurrency and Computation: Practice and Experience*, 16(5):425-432, April 2004.

[7] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker. Generative programming for embedded software: An industrial experience report. In D. Batory, C. Consel, and W. Taha, editors, *Proceedings of ACM SIGPLAN/SIGSOFT Conference, GPCE 2002*, volume 2487 of *LNCS*, pages 156-172. Springer-Verlag, 2002.

[8] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1:25-39, 2003.

[9] J. Fisher, F. Hernandez, and A. Sprague. Language patterns: Comparison and prediction using hidden markov models. In Proceedings of the 41st Annual ACM Southeast Conference, pages 246-250, 2003.

[10] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputing Applications*, 11:115-128, 1997.

[11] J. Gerlach. *Domain Engineering and Generic Programming for Parallel Scientific Computing*. Elektrotechnik und Informatik**,** Doktor der Ingenieurwissenschaften, Technishen Universitat Berlin, Berlin, Germany, 2002.

[12] E. Giloi, M. Kessler, and A. Schramm. PROMOTER: A high level object-parallel programming language. In *Proceedings of International Conference on High Performance Computing*, 1995.

[13] J. Gray, T. Bapty, S. Neema, and J. Tuck. Handling crosscutting constraints in domain-specific modeling. *Communications of the ACM*, 44(10):87-93, October 2001.

[14] J. Greenfield and K. Short. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley Publishing, Inc., 2004.

[15] T. Haupt, P. Bangalore, and G. Henley. Mississippi computational web portal. *Concurrency and Computation: Practice and Experience*, 14:1275-1287, 2002.

[16] F. Hernández, P. Bangalore, J. Gray, Z. Guan, and K. Reilly. GAUGE: Grid automation and generative environment. *Concurrency and Computation: Practice and Experience*, to appear. 2005.

[17] F. Hernández, P. Bangalore, J. Gray, and K. Reilly. A graphical modeling environment for the generation of workflows for the globus toolkit. In V. Getov and T. Kielman, editors, *Component Models and Systems for Grid Applications. Proceedings of the Workshop on Component Models and Systems for Grid Applications held June 26, 2004 in Saint Malo, France*, pages 79-96. Springer, 2005.

[18] P. Kacsuk, A. Goyeneche, T. Delaitre, T. Kiss, Z. Farkas, and T. Boczko. High-level grid application environment to use legacy codes as ogsa grid services. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, 2004.

[19] M. Lorch and D. Kafura. Symphony – A java-based composition and manipulation framework for computational grids. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2002)*, pages 136-143, 2002.

[20] J. Novotny. The grid portal development kit. *Concurrency and Computation: Practice and Experience*, 14:1129-1144, 2002.

[21] J. Siek and A. Lumsdaine. The matrix template library: A generic programming approach to high performance numerical algebra. In *Proceedings of ISCOPE 1998*, volume 1505 of *LNCS*, pages 59-70. Springer-Verlag, Santa Fe, NM, 1998.

[22] B. Sotomayor. The globus toolkit 3 programmer's tutorial. http://gdp.globus.org/gt3-tutorial/. [February 8, 2005].

[23] A. Sutcliffe and N. Mehandjiev. Introduction. *Communications of the ACM*, 47(9):31-32, 2004.

[24] S. Thatte. Business Process Execution Language for Web Services.http://www106.ibm.com/developerworks/webservices/library/ws-bpel/. [February 8, 2005], May 2003.

[25] G. von Laszewski, K. Amin, M. Hategan, N. Zaluzec, S. Hampton, and A. Rossi. Gridant: A client-controllable grid workflow system. In *Proceedings of the 37th Hawaii International Conference on System Science*, pages 210-219, 2004.