A GRAPHICAL MODELING ENVIRONMENT FOR THE GENERATION OF WORKFLOWS FOR THE GLOBUS TOOLKIT

Francisco Hernández, Purushotham Bangalore, Jeff Gray and Kevin Reilly
Department of Computer and Information Sciences,
University of Alabama at Birmingham,
Birmingham, AL, USA
hernandf - at- cis.uab.edu
puri - at- cis.uab.edu
gray - at - cis.uab.edu
reilly -at- cis.uab.edu

Abstract Grid computing aims at managing resources in a heterogeneous distributed environment. The Globus Toolkit provides a set of components that can be used to build applications that function in a grid computing system. Presently, applications are typically handcrafted either by using an Application Programming Interface (API) interacting through a set of command line interfaces, or by using a set of Java packages provided by the Java CoG Kit. The purpose of the work described in this paper is to automate the development of applications within the Globus Toolkit context by creating graphical workflows of applications using domain-specific modeling techniques.

The expected impact of this effort, as is often the case in automating systems approaches, is a reduction of the development time involved in generating applications for the Globus Toolkit. An additional advantage is to provide a high level view for the construction of Grid applications using the Globus Toolkit that avoids some of the intricacies and accidental complexities documented for other (current) approaches. Furthermore, users need not learn how to use the Java CoG Kit nor the Globus Toolkit to develop Grid-enabled applications.

Keywords: Domain Specific Modeling, Grid Computing, Workflows, Globus Toolkit, Software Engineering.

1. Introduction

The Globus Toolkit [10]is the de facto standard for building Grid-enabled applications. A user can choose three different approaches to construct such applications: (1) utilize a command-line, (2) exploit a C [20]API, or (3) employ a commodity toolkit such as the Java CoG Kit [22]. All of these approaches require an in depth understanding of the underlying technologies involved in constructing Grid applications. This limits the use of the Toolkit to those who are knowledgeable about the intricacies of these technologies. Traditionally, Problem Solving Environments (PSE) or portals [25]have been developed to ease the construction of Grid applications. PSE's provide a high-level view for specifying Grid-enabled applications and rely on middleware to connect with the Grid component resources [11]. This kind of tool expedites simple tasks (e.g., simple job submissions, and checking the status of a previously submitted job), but it lacks the flexibility to define a complex sequence of tasks.

Workflows have gained increasing attention for their application in composing a flow of tasks in a Grid environment [30]. Workflows describe the execution of complex applications built from individual application components, which is similar to the process used to construct applications using the Globus Toolkit. Previous workflow studies vary in complexity, ranging from the use of artificial intelligence to handle the automatic creation of workflows [6][31], to the specification of grid flows using an XML file [21],[3],[9].

Pegasus [6], and GridAnt [21]deserve special attention because they can be considered the endpoints of workflow approaches. Pegasus uses complex artificial intelligence planning techniques to generate automatically resource mappings and tasks according to application goals. GridAnt, on the other hand uses the Apache ANT tool [28]as a basis for its workflow engine. Although these two tools offer a viable solution to the workflow specification, they are rather difficult to use for a new Grid user. The level of technological complexity in Pegasus and the XML input requirement in GridAnt make it difficult to specify the workflow.

A solution is needed that removes these accidental complexities of use and embeds experimental knowledge of the domain into a code generator that can generate the complex configurations. Such a technology exists in the area of domain-specific modeling [16]. With this technology, a user focuses on higher levels of abstraction at the problem space and is able to avoid low-level details, such as Grid services and their usage.

The approach used in this paper is based on the concept that the development of Grid enabled applications can be improved by mapping the different Globus components into entities of a graphical model. This mapping is performed by using concepts of domain-specific modeling that utilizes the interfaces provided by the Java CoG Kit. By combining these graphical entities, a particular appli-

cation workflow can be generated into the Java [14]code that utilizes the Java CoG Kit API. Three research issues are exploited in this paper:

1 The creation of a meta-model that maps the Globus Toolkit's components to a graphical model. This meta-model defines the language used to construct workflow models.

2 The generation of graphical workflows between the different tasks of the application through the use of the meta-model.

3 The generation of Java programs from the graphical workflows. This is realized by using a model interpreter that traverses the graphical workflows and generates a program that manages the application execution.

The rest of the paper is organized as follows: Section 2 provides background on building applications with the Globus Toolkit & Java CoG Kit; Section 3 introduces the methodology that is used; Section 4 presents related work; Section 5 presents future work to be explored; Section 6 offers conclusions of the present work.

2. Background

The Globus Toolkit [10]provides a common middleware that considers resources as entities of a virtual organization. This facilitates the construction of Grid applications. The middleware is formed by different components (e.g., the Globus Resource Allocation Management component (GRAM) [5], and Grid Information Services (GIS) [4]), which provide services to integrate distributed resources in a Grid computing environment. Creating an application that uses this Toolkit requires the composition of several of these components. Originally, two approaches were used to generate this composition: (1) a set of command-line tools, and (2) a C API provided by the Toolkit. Considering the dynamic behavior of a Grid system, both approaches are less than satisfactory if the user is not a Globus savvy programmer [31].

The Java Commodity Grid Toolkit (Java CoG Kit) was later created to assist in the development of applications using Globus Toolkit [22]services; it was a step towards simplifying the construction of applications for the Globus Toolkit. The Java CoG Kit helps a user navigate the intricacies of the Globus components more easily by introducing a new programming model for the Grid. Furthermore, the Java CoG Kit provides many utility components that enhance the functionality of Globus. However, although the Java CoG Kit improves the interface between users and the Globus components, even the user who is Java savvy still needs to dedicate additional time in order to learn how to build applications for the Globus Toolkit. A method that incorporates these widely-used technologies in a more accessible and efficient manner can be achieved using concepts of domain-specific modeling.

In domain-specific modeling, a design engineer creates models for a specific domain using concepts and terminology from that domain [15]. The domain-specific models are developed by first creating a meta-model that specifies the ontology of the domain. The meta-model serves as a paradigm, or language, that defines the syntax and static semantics for models of that domain; the dynamic semantics are introduced by an interpreter that synthesizes the models into different representations [19](e.g., XML configuration files or source code). The Generic Modeling Environment (GME) [23]is a graphical tool that automates the creation of domain-specific models. GME allows a user to create graphical models by providing a general paradigm (language) from the meta-model definition.

3. Methodology to Support Model-Driven Generation of Workflows

Two actions are necessary to create domain-specific models for the Globus Toolkit:

1 Definition of the meta-model, defining the paradigm (language) to be used to create workflow models.

2 Implementation of the interpreter that translates the workflow models into corresponding Java code.

Both of these actions are implemented using GME. One of the advantages of using GME is that it allows a modeler to define base elements that can be reused in more complex models. This property is a major advantage because it is possible to define elements such as resources, user credentials, file transfers and job submission tasks only once and then reuse them in any specification of a workflow.

The models created can then be translated into executable specifications used to synthesize automatically various software artifacts [26]. The translation is performed by a model interpreter that recognizes the concepts from the workflow language and generates the semantic actions associated with that concept. In the following subsections, the manner in which the meta-model and its interpreter are constructed is explained.

3.1 Meta-Model

The goal of the meta-model is to define a new visual language that can be used to create specific workflow models. The design of the meta-model is based on the experimental knowledge of the particular domain. In this case, the design is based on the manner in which a user specifies the sequence of tasks in an application's workflow.

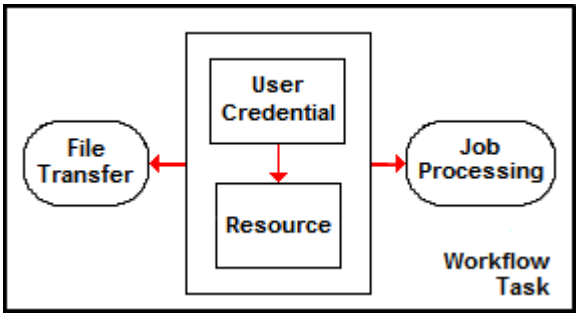

Figure 1. Workflow Task specification. A workflow task consists of File Transfers or Job processing tasks. These two tasks use a specific Grid resource. Resources require an authentication mechanism provided by the user's credentials.

Figure 1 illustrates the manner in which workflow tasks (subsequently referred to as "tasks") are defined. The central rectangle indicates that each resource requires an authentication method given by the user's credentials. Rounded rectangles specify tasks consisting of file transfers and job processing. According to the requirements specified in Figure 1, four aspects need to be considered in defining the meta-model:

1 Resources for running jobs and performing file transfers, including the specification of the credentials required to authenticate the resources.

2 File transfers end-points, including resource, location on resource, and file name.

3 Jobs, including their resource and input parameters.

4 Workflows, which are a composition of the previously defined tasks.

The definition of these four aspects provides a *mapping among* the basic requirements for constructing grid applications, the services provided by Globus, and GME entities. The way in which these aspects are specified in the meta-model and their corresponding use is explained in the following subsection. Additional details can be found in [17].

3.1.1 Meta-Model Construction and Example Usage. For explanation purposes, the meta-model can be subdivided into four different parts (Figure 2), corresponding to the aspects enumerated in the previous section. All parts are specified using the same entities provided by GME's general paradigm. The distinction of the GME entities in each part is demarcated by the concept that each entity represents in the domain, and the relationship between these concepts.

The basic concepts in the domain are Resources, File Transfers, Jobs, and Workflows. These are defined in the meta-model using either an (GME) Atom or a (GME) Model entity. Model entities can contain other model entities or atoms, but atoms are indivisible. File transfers and jobs are defined as Model entities because they contain resources. Both of these concepts require state information, so (GME) attributes are associated with these entities (Figure 2.b and 2.c). For example, the definition of a job task requires the specification of its RSL (Globus Resource Specification Language) [29]parameters (Figure 2.c).

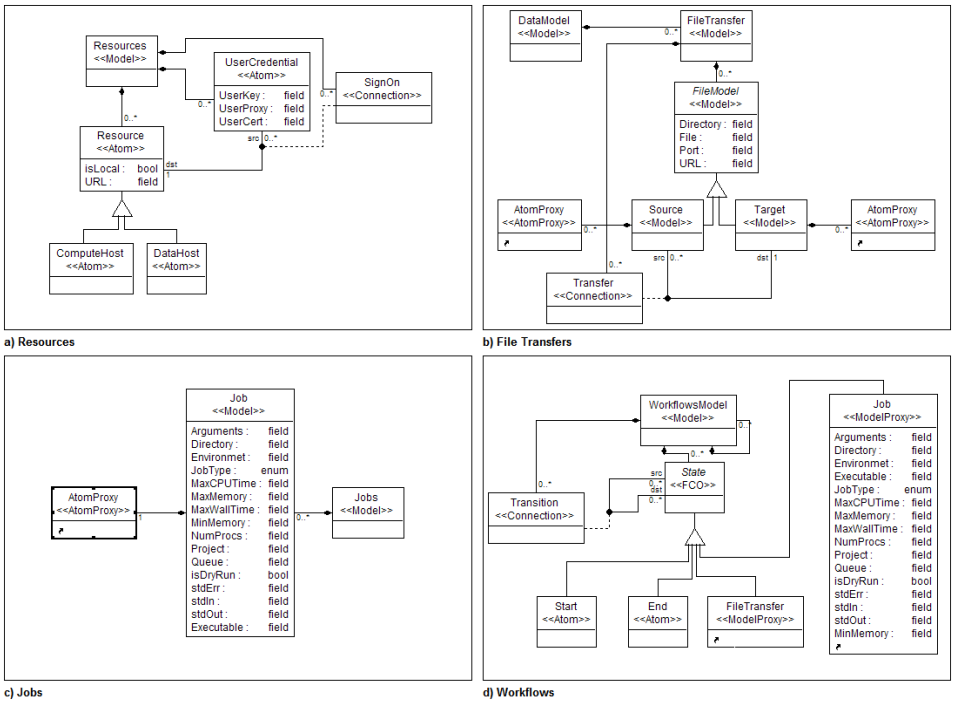

Figure 2. Meta-Model Definition. The specification of the meta-model consists of four aspects: (a) presents the definition of resources, (b) presents the definition of file transfers, (c) presents the definition of jobs, and finally (d) presents the definition of the workflow part of the meta-model.

The association between a resource and its corresponding authentication credentials is given by a (GME) connection entity. An attribute that indicates if the resource is local or remote is associated to the Resource atom. Resources that are remote need an authentication credential. Resources can be used either for computation or for data storage. As typical in UML models, the triangle of Figure 2 indicates that both kinds of resources inherit attributes and con-

nections from a basic host entity (Figure 2.a). Finally, the workflow part of the meta-model consists of the previously defined tasks (file transfers, and job specifications), and a start and end of workflow markers (Figure 2.d).

Using the meta-model, a user can define application workflows by interacting with the graphical environment provided by GME. The following example, presents a simple application using Hidden Markov Models to illustrate how this interaction is performed for a typical Grid application. A Hidden Markov Model (HMM) was constructed to compare the differences between English and Spanish language patterns [8]. 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. The subdivision of this application into different components and its integration in a Grid environment is presented in the rest of this section.

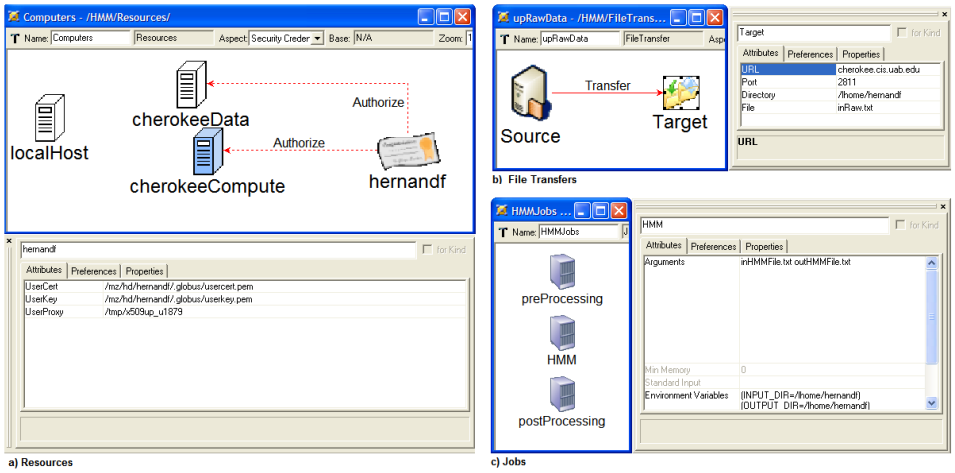

Figure 3. Model Interaction. Once the meta-model is constructed, a user graphically defines the basic elements of the workflow. Figure (a) shows how resources are defined and how a user credential authenticates the remote resources. Figure (b) shows that defining file transfers consist of specifying the location of the files on the endpoints. Figure (c) shows how the HMM job is defined by specifying its RSL attributes.

The application can be subdivided into pre-processing, HMM, and post-processing tasks. The input file is copied from the local computer to a remote host, and after the execution of the application, the output file is copied back to the local computer. The first step in defining the application involves the definition of resources. Light machines are used for data storage, while dark machines are used for computation purposes. Each remote resource needs to be authenticated with the user's credential (Figure 3.a). The next step is to

define the file transfers between the local computer and the remote host. In the case of uploading a file to the remote host, the URL and port of the host must be specified (Figure 3.b). Finally, the definition of jobs consists of the specification of the Resource Specification Language (RSL) attributes required to run the job (Figure 3.c).

After all of the tasks are defined, the application can be constructed by specifying the required sequence of tasks (Figure 4). File images indicate file transfers, and computer images indicate jobs to execute. The star in the far left indicates the start of the workflow, and the sphere on the far right indicates the end of the workflow.

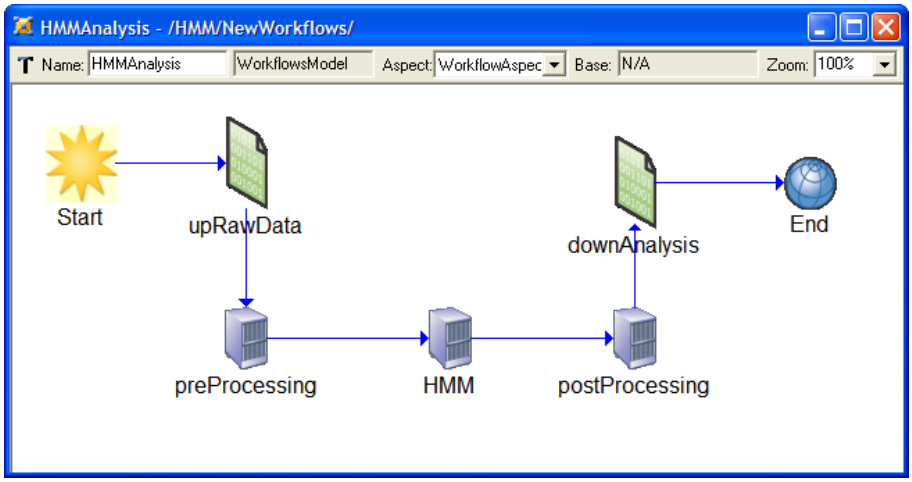

Figure 4. Definition of the workflow in the model. This figure presents the workflow for the example. The input file is copied to the remote host (upRawData). A preprocessing job is executed on that file and its output is analyzed by the HMM job. The output of the HMM job is then modified in the postProcessing step. Finally the output of the postProcessing job is downloaded to the local computer (downAnalysis).

3.2 Interpreter

After the workflow is specified, a model interpreter traverses the internal representation of the model and generates the control code that manages the application execution. The interpreter first gathers all the information from resources, jobs, and file transfers. This information, along with the specification of the application workflow, constitutes the interpreter's input. The interpreter then executes the semantic actions associated with each workflow task. The output of this step is a Java program that manages the application execution. The Java program uses a set of supporting classes, or adapters [13], implemented

to standardize the interaction with the Java CoG Kit API. The Java CoG Kit API is used as a bridge to communicate with the enabled back-end resources managed by the Globus Toolkit (Figure 5).

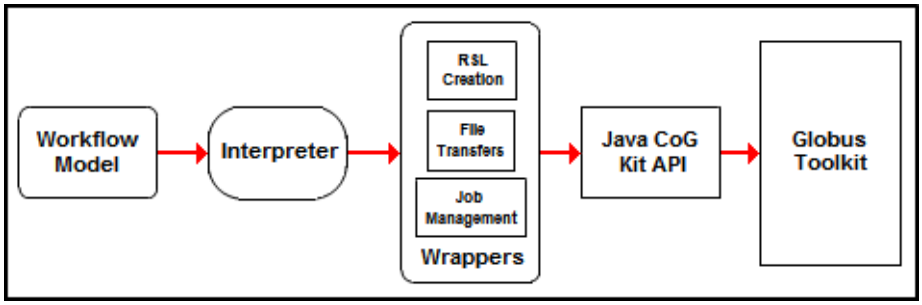

Figure 5. Structure of the Code Generation. The Interpreter traverses the workflow model and generates a Java program that interacts with a set of wrappers. Creation of RSL strings, specification of file transfers, and remote execution of jobs are the facilities provided by the wrappers. The wrappers use the Java CoG Kit API to communicate with the Globus Toolkit.

Figure 6 presents the code that is generated for the HMM job submission task. The process presented here is repeated for each job submission and similar code is generated. The references to the GlobusRSL on line 10 and the GRAMJob on line 20 are the wrapper classes that communicate to the Java CoG API. Line 7 reads the user's proxy into a byte array. This is done to authenticate the user on the specific resource. Lines 10 through 17 create the corresponding RSL string. Finally, lines 20 through 33 submit the job to the specified resource and waits for its completion. As can be seen in this code, if some problem occurs then exceptions are caught and a corresponding message is displayed. File transfers operate in a similar way by using a different wrapper.

4. Related Work

The idea of composing applications from reusable components is not new. For example, WebFlow [1]introduces a platform-independent system that dynamically composes new applications from reusable components by clicking and dragging icons. The Job model of UNICORE [7]uses a set of directed acyclic graphs, and also permits the use of conditional and iterative execution of job groups or tasks. DAGMan [12]also maps a direct acyclic graph specification onto a physical environment. The Symphony framework [24]uses a graphical user interface for rapid collaborative development of grid applications following a data flow paradigm. Triana [27]also offers a visual programming model for the dynamic composition of predefined software components. Other works propose languages to specify Grid workflows. For example, Grid Workflow [3]focuses

```
1:// ------------------------------------------------------------
2://  This code is generated by the interpreter when it
3://  finds a Job model in the workflow
4://  Generating Code for Job: HMM
5: try {
6:  // load the user proxy certificate
7: byte[] hmmProxy = getByteArray("/tmp/x509up_u1879");
8:
9:  // create the rsl string
10:  GlobusRSL hmmRSL = new GlobusRSL();
11:
12:  hmmRSL.setArg("HMM inHMMFile.txt outHMMFile.txt");
13:  hmmRSL.setEnvironmentVariables("(INPUT_DIR=/lhome/hernandf)
    (OUTPUT_DIR=/lhome/hernandf)");
14:  hmmRSL.setStdOut("/lhome/hernandf/sttOutHMM.txt");
15:  hmmRSL.setNumProc(2);
16:  hmmRSL.setDir("/usr/bin");
17:  hmmRSL.setExec("java");
18:
19:  try {
20:    GRAMJob hmmGRAM = new GRAMJob();
21:    // submit the job
22:    String hmmID = hmmGRAM.submitJob("cherokee. cis.uab.edu",
     hmmProxy, hmmRSL.toRSL());
25:    // wait for its completion
26:    String hmmCond = hmmGRAM.checkStatusOfJob(hmmID, hmmProxy);
27:    while (!hmmCond.equals("Job DONE."))
28:    {
29:      System.out.println(hmmCond);
30:      Thread.sleep(1000);
31:      hmmCond = hmmGRAM.checkStatusOfJob(hmmID, hmmProxy);
32:    }
33:    System.out.println("hmmGRAM Done.");
34:  } catch (Exception e) {
35:    System.err.println(e.getMessage());
36:    System.exit(-1);
37:  }
38:} catch (Exception e) {
39:  System.out.println("There was an error loading the user proxy.");
40:  System.exit(-1);
41:}
```

Figure 6. Code Generation. Code generated by the interpreter for the HMM job submission task.

on proposing a standard for the sequence of complex high-performance computational tasks within a Grid. GridAnt [21]uses an XML-based language to

specify client-side workflows. GridAnt is also able to submit the executions of tasks or file transfers by using a workflow engine based on the Apache ANT tool [28]. The construction of the workflow base aspect of the environment has been influenced by these projects.

Hategan et al., [2]proposes a technology and architecture-independent abstraction layer to provide interoperability across multiple Grid implementations, resulting in the Open Grid Computing Environment (OGCE). The main function of OGCE is to serve as a technology-independent, open, and extensible framework for client-side Grid development. This concept is similar to the idea presented in this paper of using meta-models to abstract the underline Grid technologies. Because the models and the interpreter that translates those models are two different components of the modeling environment, with a change in the interpreter, the same models can be reused for different Grid architectures. This is an attempt to abstract the Grid environment into a high-level layer such that the essence of the workflow is not bound to a specific Grid environment. Furthermore, the abstractions provided by OGCE are comparable to those introduced in this paper. For example, the task concepts presented in [2]contain concepts similar to those involved in the job specification (Figure 2). However, the main difference between the studies is in the level of abstraction. In this paper the abstraction layer is realized at a domain-model level, but in [2]the abstraction layer is at a programming language level (Java).

5. Future Directions

Work on the modeling environment is in its initial phase. The current implementation of the environment can handle only a limited number of sequential tasks in the workflow. At present, the generated applications communicate directly with the Java CoG Kit (as seen in Figure 6). This causes scalability problems due to the generation of specific code for each workflow task. A solution to this problem is currently under investigation and consists of developing a reusable workflow engine and generating appropriate configurations from the graphical models. In addition to improving the scalability of the generated applications, current efforts are aimed at three different areas: (1) allow parallelism of tasks in a workflow, (2) allow third party transfers, and (3) allow the definition of hierarchical workflows. A modified meta-model that considers these capabilities has already been implemented, but the corresponding interpreter is still in progress. Nevertheless, even without these capabilities, the initial experience with the environment is promising. In addition to this work in progress, future directions that will be considered involve three aspects:

1 In order to further simplify the use of the environment, integration with GIS [4]is the logical next step. This integration will provide feedback to

users so they can decide which resources are more appropriate for their applications.

2 The current trend in Grid computing is moving towards a service architecture. To make the environment capable of moving in that direction, future work will be focused in two aspects: (1) the utilization of web services as workflow tasks, and (2) the capability of generating web services from workflows. The latter will allow non-web service applications to run and cooperate in a web service environment.

3 GridAnt [21]is another tool that allows a user to specify workflows for the Globus Toolkit. The difficulty posed by that tool is in the use of an XML file as an input. One of the major advantages of using the approach presented in this paper is that more than one interpreter can be implemented for a particular meta-model. Because of this, the environment can serve as a front-end and by changing the interpreter, the required GridAnt's input file can be generated. The ability to generate multiple artifacts from the same model is a key benefit of model-driven techniques [23].

6. Conclusion

The goal of the research described in this paper is to improve the development of applications within the Globus Toolkit by creating graphical workflows of applications using domain-specific modeling techniques and the Java CoG Kit API. The benefits of using domain-specific modeling techniques which motivated this study were:

1 Domain modeling removes the accidental complexities of creating workflows in a Grid by focusing on higher levels of abstraction at the problem space *rather than solution space*, such as specific Grid libraries and their usage.

2 When exploring various workflows scenarios, modeling tools and their interpreters facilitate the more rapid ability to change the workflow details. That is, it is easier to manipulate and change domain models rather than the associated code.

3 Model-driven techniques possess the ability to generate multiple artifacts from the same model. Thus, with the same domain knowledge different output representations can be generated.

Using these modeling techniques, a meta-model for the Globus Toolkit was created, as well as an interpreter that automatically generates Java code from the workflow models. With this approach, a programmer manipulates graphical

models that represent the different components provided by the Globus Toolkit. From these models, the programmer is able to generate the corresponding Java programs that manage the execution of the application.

The potential impact of this study is the reduction of the development time involved in generating applications for the Globus Toolkit. Furthermore, users are not required to learn how to use the Java CoG Kit nor the Globus Toolkit to develop Grid-enabled applications. Rather, they construct graphical models that are at a more appropriate level of abstraction for describing the essence of the problem for a specific domain.

References

[1] 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 Supercomputing*, 1998.

[2] K. Amin, M. Hategan, G. von Laszewski, and N. Zulezec. Abstracting the Grid. In *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing* (PDP 2004), La Coruña, Spain, 2004.

[3] H.P. Bivens. *Grid Workflow*. Grid Computing Environments Working Group Document, 2001.

[4] K. Czajkowski, S. Fitzgerald, I. Foster, and C.Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*. pages 181-184. IEEE Press, August 2001.

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

[6] 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**,1:25-39, 2003.

[7] D.W. Erwing. UNICORE - a Grid Computing Environment. In *Concurrency and Computation: Practice and Experience*. **14**, 13-15:1395-1410, 2002.

[8] J. Fisher, F. Hernández, and A. Sprague. Language Patterns: Comparison and Prediction using Hidden Markov Models. In *Proceedings of the 41st Annual ACM Southeast Conference*, pages 246-250. ACM Press, Savannah, GA, 2003.

[9] *Flow Editor*. http://www-unix.globus.org/cog/projects/floweditor/

[10] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*. 11:115-128, 1997. http://www.globus.org

[11] G. Fox, D. Gannon, and M. Thomas. Overview of Grid Computing Environments. In *Grid Computing: Making the Global Infrastructure a Reality* (F. Berman, G. Fox, and T. Hey eds.), pages 543-554, John Wiley and Sons Ltd, Chichester, 2003.

[12] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*. **5**, pages 237-246, 2002.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts, 1995.

[14] J. Gosling, B. Joy, and G. Steele. *The Java language specification*, Addison-Wesley, 1996.

[15] J. Gray, T. Bapty, S. Neema, and J. Tuck. Handling Crosscutting Constraints in Domain-Specific Modeling. *Communications of the ACM*. pages 87-93, October 2001.

[16] J. Gray, M. Rossi, J.P. Tolvanen. Preface: Special Issue on Domain-Specific Modeling. In *Journal of Visual Languages and Computing*. **15**, 3-4:207-209, June/August 2004.

[17] F. Hernandez. *Domain-Specific Models and the Globus Toolkit*. Tech. Rep. UABCIS-TR-2004-0504-1,Department of Computer and Information Sciences, University of Alabama at Birmingham, 2004.

[18] *JLex: A Lexical Analyzer Generator for Java(TM)*. http://www.cs.princeton.edu/ appel/modern/java/JLex/

[19] G. Karsai, M. Maroti, A. Lédeczi, J. Gray, and J. Sztipanovits. Composition and Cloning in Modeling and Meta-Modeling. In *IEEE Transactions on Control System Technology (special issue on Computer Automated Multi-Paradigm Modeling)*, pages 263-278, March 2004.

[20] B. Kernighan and D. Ritchie. *The C programming language.* Prentice Hall, 1988.

[21] 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*, Island of Hawaii, Big Island, Jan 5-8, 2004.

[22] G. von Laszeweski, I. Foster, J. Gawor, and P. Lane. A Java Commodity Grid Toolkit. *Concurrency and Computation: Practice and Experience.* **13**, (8-9):643-662, 2001, http://www.cogkits.org/

[23] A. Lédeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, G. and Karsai. Composing Domain-Specific Design Environments. *IEEE Computer.* pages 44-51, 2001.

[24] M. Lorch, D. and 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, Berlin, Germany, 2002.

[25] J. Novotny. The Grid Portal Development Kit. *Concurrency and Computation: Practice and Experience.* **14**, (13-15), 2002.

[26] J. Sztipanovit and G. Karsai. Model-Integrated Computing. *IEEE Computer*,pages 110-112,1997.

[27] I. Taylor, M. Shields, I. Wang and R. Philp. Grid Enabling Applications Using Triana. In *Workshop on Grid Applications and Programming Tools*. Seattle, 2003.

[28] *The Apache ANT Project*. http://ant.apache.org/

[29] *The Globus Resource Specification Language RSL v1.0*. http://www.globus.org/gram/rsl_spec1.html

[30] *Workflow in Grid Systems Workshop*. http://www.extreme.indiana.edu/groc/Worflow-call.html

[31] B. Yeo and B. Khoo. An Agent-based Grid Flow Manageme2004. nt Framework for the Problem Solving Environment (PSE). In *Scientific Workflow Management Mini-Symposium*, GlobusWorld, 2004.