

Dynamic Workflows for Grid Applications

Andreas Hoheisel¹, Uwe Der¹

¹ Fraunhofer Institute for Computer Architecture and Software Technology (FIRST)
Kekuléstr. 7
D-12489 Berlin
{andreas.hoheisel, uwe.der}@first.fraunhofer.de

Abstract

In the Grid computing community, there are several approaches to execute not only single tasks on single Grid resources but also to support workflow schemes that enable the composition and execution of complex Grid applications. The most commonly used workflow model for this purpose is the Directed Acyclic Graph (DAG). Within the establishment of the Fraunhofer Resource Grid, we developed a Grid Job Definition Language (GJobDL) that is based on the concept of Petri nets instead of DAGs to support the graph-based definition of arbitrary workflows on an abstract level. During the workflow execution, the abstract workflow must be concretized in order to be mapped onto the real Grid environment. This requires dynamic completion of the workflow based on actual information. It may be necessary to introduce new tasks – such as data transfers, deployment of software, authorization request, and data retrievals. These tasks can be represented by sub Petri nets that replace parts of the existent Petri net during runtime of the Grid application. We also propose a concept for fault management of entire job workflows by explicitly modeling the fault management within the workflow model. This fault management can be user-defined or be realized automatically by introducing new tasks enabling fault management based on fault management templates.

1 Introduction

Today, one of the main objectives of Grid computing is abstraction from the hardware infrastructure as well as hiding implementation details of software components from the user. Modern Grid infrastructure should enable the user not only to execute single tasks on specified hardware resources but also to compose and execute complex Grid applications on distributed, heterogeneous and unreliable hardware resources without taking care about lower-level details. We define the term *Grid job* as a Grid application that is composed of several Grid resources with a specified workflow. In order to specify a Grid job, we developed an XML-based *Grid Job Definition Language* (GJobDL) [6]. According to our definition, *Grid resources* are either abstract classes or concrete instances of software, hardware or data [7]. A Grid job may induce a variety of single tasks (*atomic jobs*) that are indivisible components of a Grid job.

One example of a typical Grid job is shown in Fig 1. This Grid job is defined on an abstract layer by specifying the data and control flow among software and data resources. The definition of the Grid job is independent of the hardware infrastruc-

ture, and it is up to the Grid architecture to map this Grid job onto adequate Grid resources using additional Meta data and to ensure its reliable execution. Most existing approaches hereby assume a static Grid architecture, consisting of a set of reliable distributed resources, and they use static workflow modeling based on Directed Acyclic Graphs (DAG) [1, 3, 12, 13], whereas our approach uses Petri-net-based graphs that cover a wider class of workflows. We developed a so-called Grid Job Handler that achieves dynamic completion of the workflow during runtime based on actual information in order to react to the actual situation and to include advanced fault management when executing a Grid job.

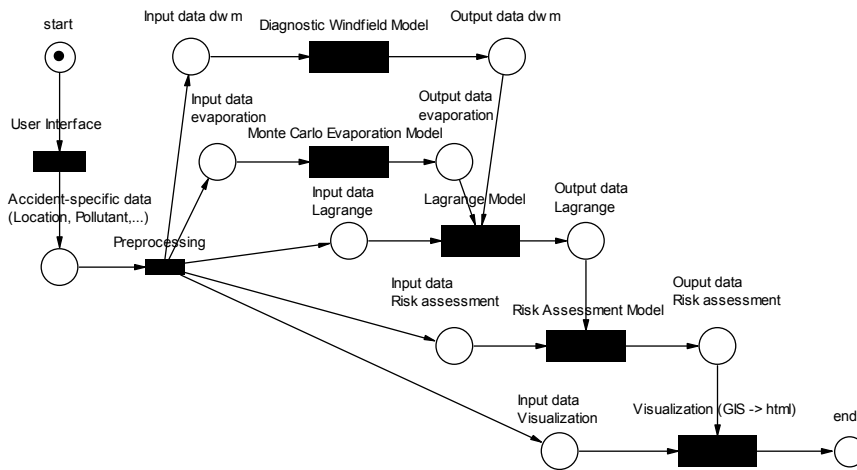


Fig 1: Example of a typical Grid job of the environmental simulation domain. This Petri net represents the workflow of a coupled grid application that is part of the Environmental Risk Analysis and Management System (ERAMAS) [2].

The following section gives an overview of the Grid architecture that is being developed as part of the Fraunhofer Resource Grid. Section 3 describes why and how we use Petri nets to define the workflow of Grid jobs. Examples of dynamic workflows are introduced in section 4. The role of dynamic workflows in fault management is the focus of section 5. Section 6 describes our implementation of the Grid Job Handler used to process and to control Petri-net-based Grid jobs.

2 Fraunhofer Resource Grid

The Fraunhofer Resource Grid is a Grid initiative of five Fraunhofer institutes funded by the German federal ministry of education and research with the main objective to develop and to implement a stable and robust Grid infrastructure within the Fraunhofer-Gesellschaft, to integrate available resources, and to provide internal and external users with an easy-to-use interface for controlling distributed applications and services in the Grid environment [5]. The component environment supports loosely coupled software components where each software component represents an

executable file that reads input files and writes output files (atomic job). The communication is realized via file transfer. Legacy code can be integrated easily using shell scripts to encapsulate the program. Up to now, the workflow architecture does not support tight coupling schemes like CORBA, MPI, or HLA, but tightly coupled applications can be included as a whole like an atomic job.

Fig 2 depicts the Grid architecture of the Fraunhofer Resource Grid that is built on top of the Globus 2.4 toolkit [15]. The user has four different possibilities to access Grid resources within this architecture:

- (1) The user can directly use the standard Globus services like GRAM or GridFTP in order to run simple Globus jobs (atomic jobs) on a specified Grid node.
- (2) If the user wants to run a predefined Grid job, he can use the Grid Job Handler Web Service. In this case, the user must provide a GJobDL document that specifies the Grid job.
- (3) The user may use the graphical Grid Job Builder to assemble and configure the resources to form a coupled Grid job that is defined by a GJobDL document.
- (4) If the user does not know which resources to use in order to solve his problem, he may invoke the Task Mapping of the web portal. There, the user navigates through a task tree in order to constrict the application area of the problem and to map it onto a suitable set of Grid resources.

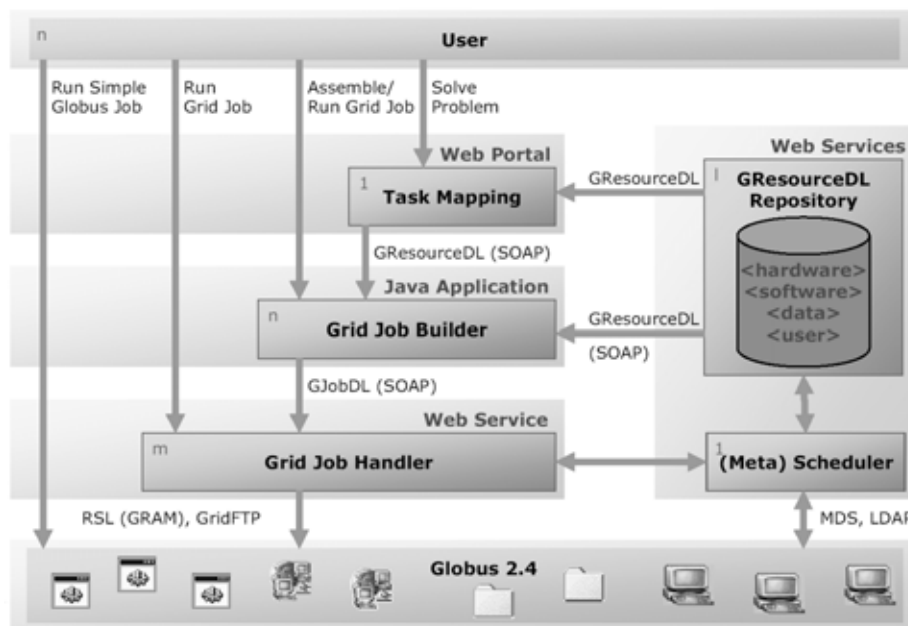


Fig 2: The layered grid architecture of the Fraunhofer Resource Grid. The numbers in the upper left corners denote the cardinality.

This paper focuses on the workflow management that is part of the Grid Job Handler. Most of the software developed within the Fraunhofer Resource Grid will be made Open Source (GPL) at the beginning of 2004 under the label *eXeGrid* [4].

3 Describing Grid jobs with Petri Nets

The GJobDL description of a Grid job contains the resource descriptions of the basic resources that are required to define the Grid job and the model of the Grid job workflow using the concept of Petri nets [11]. In many other grid projects, the workflow of coupled Grid applications is modeled using Directed Acyclic Graphs (DAG) (see Fig 3). One example for this approach is UNICORE, where so-called AbstractJobs are defined on the basis of DAGs [3]. Other prominent grid projects using DAGs are Condor [1, 12], Cactus [13], and Symphony [10]. DAGs are widely spread due to their simple structure, they possess, however, some relevant disadvantages: As DAGs are directed, it is impossible to define bi-directional coupling schemes between software components. DAGs are acyclic, so it is not feasible to explicitly define loops (while . . . do). A DAG only describes the behavior, but not the state of the system. DAGs generally describe only the workflow, and not the dataflow. Because of these limitations we decided to use Petri nets (see Fig 3) instead of DAGs. The idea to use Petri nets to control the workflow of complex applications has been borrowed from the Graphical Simulation Builder that is being developed by the Potsdam Institute for Climate Impact Research (C. Ionescu, pers. comm.).

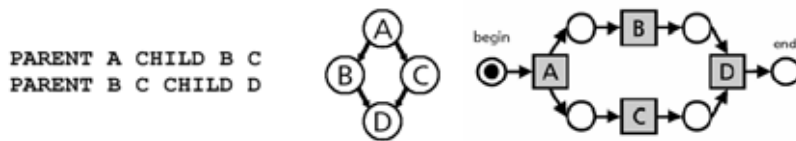


Fig 3: Example of a Directed Acyclic Graph (DAG) (left) and the equivalent Petri net (right).

The type of Petri nets we introduce here corresponds to the concept of Petri nets with individual tokens (colored Petri net) and constant arc expressions which are composed of places, denoted by circles (\circ), transitions, denoted by boxes (\square), arcs from places to transitions ($\circ \rightarrow \square$), arcs from transitions to places ($\square \rightarrow \circ$), individual and distinguishable objects that flow through the net as tokens (\bullet), an initial marking that defines the objects which each place contains at the beginning, and an expression for every arc that denotes an individual object. A place p is called input place (output place) of transition t if an arc from p to t (from t to p) exists. A brief introduction to the theoretical aspects of colored Petri nets can be found, e.g., in [8].

We use Petri nets not only to model the workflow, but furthermore to control the workflow of grid jobs. In most cases, the workflow within grid jobs is equivalent to the dataflow, i.e., the decision when to execute a software component is taken by means of availability of the input data. Therefore, the tokens of the Petri net represent real data that is exchanged between the software components in the grid. In this case, we use Petri nets to model the interaction between software resources represented by

software transitions, and data resources represented by data places. In some cases, however, the workflow is independent from the dataflow, and in addition to the data places and software transitions we have to introduce control places and control transitions. The corresponding tokens contain the state of the process (e.g., done, failed). Control transitions evaluate logical conditions. For further details about the Petri net approach of the Fraunhofer Resource Grid refer to [7].

4 Dynamic workflows

The refinement model of the Petri net theory allows substituting parts of a Petri net by new sub Petri nets. The Grid Job Handler takes advantage of this feature and supplements the workflow during runtime by introducing additional tasks that are necessary to complete the Grid job. The user is not required to model every detail of the workflow – he just has to include the essential transitions and places that are related to the software components and the data he wants to include in his Grid job. Additional tasks that have to be invoked due to specific properties of the Grid infrastructure (e.g., network topology) are detected by the Grid architecture and considered by automatically introducing additional transitions and places before or during runtime of the Grid job.

In the actual version of the Grid Job Handler, data transfer tasks and software deployment tasks are automatically added to the workflow if they are missing in the initial Grid job definition provided by the user. A data transfer task may be introduced to transfer files that are not available on the remote computer (Fig 4). A software deployment task may be introduced to install software components on a remote computer (Fig 5). Further Petri net refinements could concern authorization, accounting, billing and fault management tasks (see next section).

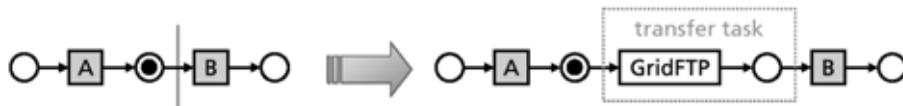


Fig 4: Software components *A* and *B* are scheduled to different Grid nodes. A data transfer task is introduced to transfer the output files of *A* to the location where *B* will be executed.

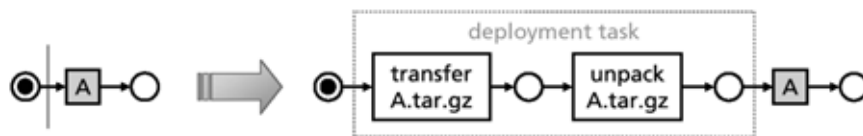


Fig 5: Software component *A* should be executed on a Grid node, where it is not yet installed. A software deployment task is introduced to install the software on the corresponding Grid node.

5 Fault management

In the Grid computing domain, we distinguish between *implicit* and *explicit* fault management. **Implicit fault management** is inherently included in the Grid middleware and is invoked either by lower-level services regarding fault management of atomic jobs or by higher-level services considering the workflow of the Grid job. This type of implicit fault management can be achieved by Petri net refinement as shown in Fig 6, where a fault management task is introduced automatically if the submission or execution of an atomic task fails. **Explicit fault management** in our definition refers to user-defined fault management. Within the Petri net workflow model, the user defines the fault management explicitly by including user-defined fault management tasks in the Petri net of the Grid job. Two examples of user-defined fault management are shown in Fig 7.

We propose that Grid architectures should provide mechanisms for both, implicit and explicit fault management. The implicit fault management guarantees a basic fault tolerance of the Grid system whereas explicit fault management is needed to support arbitrary, user-defined fault management requirements.

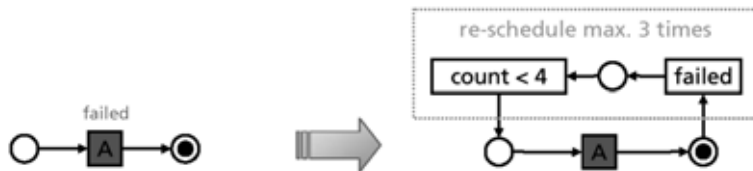


Fig 6: Example of implicit fault management. If the execution of software component *A* fails, a fault management task may be introduced into the Petri net. Here, the fault management task re-schedules the software component maximum three times.



Fig 7: Two examples of explicit, user-defined fault management. Left side: If *A* fails, *B* will be executed; if *A* completes successfully, *C* will be executed. Right side: If *A* does not complete after a specified time out, *C* will be executed. If *A* completes in time, *B* will be executed.

6 Grid Job Handler

The Grid Job Handler is responsible for the execution of each Grid job on a set of suitable hardware resources. Therefore, the Grid Job Handler parses the Grid job description, resolves the dependencies between the Grid resources, and searches for sets of hardware resources that fulfill the requirements of each software component. A

Meta scheduler (see Fig 2) is used to select the best-suited hardware resource of each set of matching hardware resources, according to a given scheduling policy (fastest, cheapest, etc.). The Grid Job Handler maps the resulting atomic jobs onto the Globus Resource Specification Language (RSL) [14] and submits them via GRAM to the corresponding Grid nodes. For the communication between the Job Handler and the Globus grid middleware, we use a patched version of the Java Commodity Grid Kit [9]. The Grid Job Handler itself is deployed as a Web Service with possibilities to create, run and monitor Grid jobs remotely. The desktop version of the Grid Job Handler includes a graphical user interface (see Fig 8) and additional command line tools.

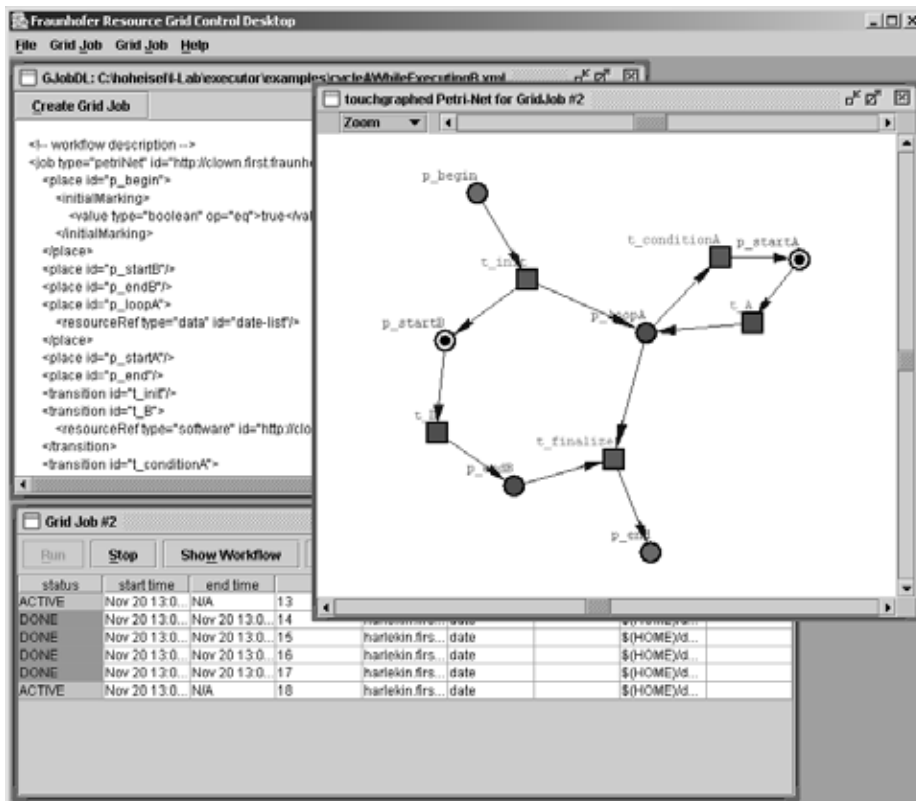


Fig 8: Screenshot of the graphical Grid Job Handler user interface. The upper left panel displays an excerpt of the GJobDL document. The right panel shows a graphical representation of the corresponding Grid job workflow. The lower left panel lists the atomic jobs that are induced by the Grid job with their actual status.

7 Conclusions and future work

In order to support the user in composing Grid applications, it is necessary to have suitable higher-level services that hide the complexity of lower-level details. One

example of such a higher-level service is the Grid Job Handler, which we established within the Fraunhofer Resource Grid on top of the Globus toolkit. The most innovative part of the Grid Job Handler is the Petri-net-based workflow model that allows the definition of arbitrary workflows with only three different components: transitions, places and arcs. This enables the easy orchestration of complex workflows, including conditions and loops and regarding the dataflow as well as the control flow of Grid applications. The dynamic workflow model introduced here takes advantage of the Petri net refinement theory, which allows adding additional tasks to the workflow during runtime, such as transfer tasks, software deployment tasks or fault management tasks.

Future work that we planned in the domain of workflow management for Grid applications considers the achievement of the OGSi standard. We will study how the Petri-net-based workflow management will adapt to OGSi and if it is feasible to relate transitions to the invocation of Web or Grid Service method calls. Another interesting point that requires future work is the simulation and prediction of Petri nets for advanced reservation of Grid resources and additional fault management on the workflow level enabling workflow check pointing and recovery of whole Grid jobs.

References

1. Condor: The Directed Acyclic Graph Manager. <http://www.cs.wisc.edu/condor/dagman/> 2003.
2. ERAMAS homepage: <http://www.erasmas.de> 2003.
3. D. W. Erwin and D. F. Snelling. UNICORE: A Grid Computing Environment. In LNCS 2150, 825–834. Springer-Verlag, 2001.
4. eXeGrid homepage: <http://www.exegrid.net> 2003.
5. Fraunhofer Resource Grid homepage: <http://www.fhrg.fhg.de> 2003.
6. Fraunhofer Resource Grid: XML schema of the Grid Job Definition Language: <http://www.fhrg.fhg.de/de/fhrg/schemas/gadl/gjdl.xsd> 2003.
7. Hoheisel, A., Der, U.: An XML-based Framework for Loosely Coupled Applications on Grid Environments, P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2657, 245–254, © Springer-Verlag, http://www.andreas-hoheisel.de/docs/Hoheisel_and_Der_2003_ICCS.pdf 2003.
8. Jensen, K.: An Introduction to the Theoretical Aspects of Coloured Petri Nets. LNCS 803. 230–272, Springer-Verlag, 1994.
9. von Laszewski, G., Foster, I., Gawor, J., Lane, P.: A Java Commodity Grid Kit. Concurrency and Computation: Practice and Experience 13, 643–662, 2001.
10. Lorch, M., Kafura, D.: Symphony — A Java-based Composition and Manipulation Framework for Computational Grids. In: Proceedings of the CCGrid2002. Berlin, Germany 2002.
11. Petri, C. A.: Kommunikation mit Automaten. Ph.D. dissertation. Bonn 1962.
12. Thain, D., Tannenbaum, T., and Livny, M: Condor and the Grid. In Grid Computing: Making the Global Infrastructure a Reality. Fran Berman and Geoffrey Fox and Tony Hey, editors. John Wiley and Sons Inc., 2002.
13. The Cactus Project: <http://www.cactuscode.org> 2003.
14. The Globus Project: The Globus Resource Specification Language RSL v.1.0. http://www-fp.globus.org/gram/rsl_spec1.html 2000.
15. The Globus Toolkit 2.4. <http://www.globus.org/gt2.4/download.html> 2003.