

Towards an UML Based Graphical Representation of Grid Workflow Applications [★]

Sabri Pllana¹, Thomas Fahringer², Johannes Testori¹, Siegfried Benkner¹, and Ivona Brandic¹

¹ Institute for Software Science, University of Vienna
Liechtensteinstraße 22, 1090 Vienna, Austria

{pllana,testori,sigi,brandic}@par.univie.ac.at

² Institute for Computer Science, University of Innsbruck
Technikerstraße 25/7, 6020 Innsbruck, Austria
Thomas.Fahringer@uibk.ac.at

Abstract. Grid workflow applications are emerging as one of the most interesting programming models for the Grid. In this paper³ we present a novel approach for graphically modeling and describing Grid workflow applications based on the Unified Modeling Language (UML). Our approach provides a graphic representation of Grid applications based on a widely accepted standard (UML) that is more amenable than pure textual-oriented specifications (such as XML). We describe some of the most important elements for modeling control flow, data flow, synchronization, notification, and constraints. We also introduce new features that have not been included by other Grid workflow specification languages which includes broadcast and parallel loops. Our UML-based graphical editor Teuta provides the corresponding tool support. We demonstrate our approach by describing a UML-based Grid workflow model for an advanced 3D medical image reconstruction application.

1 Introduction

In the past years extensive experience has been gained with single site applications and parameter studies for the Grid. For a short time, Grid workflow applications are emerging as an important new alternative to develop truly distributed applications for the Grid. Workflow Grid applications can be seen as a collection of activities (mostly computational tasks or user interaction) that are processed in some order. Usually both control and data flow relationships are shown within a workflow. Although workflow applications have been extensively studied in areas such as business process modeling [9], it is relatively new in the Grid computing area.

The Web Service Flow Language (WSFL) [11] focuses on the composition of Web services by using a flow model and a global model. Moreover, the order of

[★] The work described in this paper is supported by the Austrian Science Fund as part of Aurora Project under contract SFBF1104.

³ © Springer-Verlag

the Web services is defined. Both flow of control and data flow are represented. The Web service flow language can be considered as a language to describe concrete workflows that are based on a specific implementation of workflow activities namely Web services. However, it is not an abstract workflow language that shields implementation details. There are multiple choices to represent workflow activities. Web services represent one implementation. XLANG [15] is used to model business processes as autonomous agents which supports among others exceptions, conditional and iterative statements, and transactions. The Business Process Execution Language for Web Services (BPEL4WS) [1] specifies the behavior of a business process based on Web Services. In BPEL4WS business processes provide and use functionality based on Web Service interfaces. BPEL4WS is a merge of the concepts of XLANG and WSFL.

Commonly these languages are considered to be too complex for Grid applications with extensive language features for control flow and synchronization that in many cases make only sense in the business process modeling area. Moreover, in these models workflow is commonly defined such that interaction among Web services is done via a central workflow control engine which certainly would be a bottleneck for Grid applications.

There is also some work done to introduce languages for the specification of Grid workflow applications based on the eXtensible Markup Language (XML) [5]. Grid Workflow [4] is a language for describing Grid workflow applications developed at Sandia National Laboratories. This language is missing the advanced constructs for the control flow such as branches, loops, split and join. There is no graphical representation defined for this language. Grid Services Flow Language (GSFL) [10] is an XML based language for Grid workflow applications developed at Argonne National Laboratory. It can represent only a sequence of activities, and is missing the advanced constructs for the control flow such as branches, loops, split and join.

Triana [6] supports the graphical representations of workflows. *Triana* workflows are experiment-based: each unit of work is associated with a specific experiment. A workflow – defined by a *task graph* – defines the order in which experiments are executed. *Triana* provides support for simple control-flow constructs, but for instance advanced concurrency cannot be explicitly expressed since there is no support for synchronization conditions.

In this paper we introduce a new approach that employs the Unified Modeling Language (UML) [12] for graphically modeling and describing Grid workflow applications. Workflow modeling is strongly supported by UML through activity diagrams. Moreover, UML is a widely used standard that is easier to read and understand by human beings than XML documents as commonly used by many workflow specification languages. In order to facilitate machine processing of UML diagrams, UML tools support the automatic transformation of UML diagrams into XML Metadata Interchange (XMI) [13]. We tried to incorporate all important features presented by previous workflow languages including WSFL, eliminated unnecessary complexity and introduced new concept that were missing. Our UML modeling approach covers some of the most important modeling

constructs for workflow hierarchical decomposition, control and data flow, synchronization and notification. Moreover, we included workflow constructs that are not covered by existing workflow languages including parallel loops and broadcast communication. Existing work can express parallel activities which must be specified one by one through a fork mechanism. Grid applications often exploit large numbers of independent tasks which should be naturally expressed by parallel loops. Moreover, some workflow languages support some kind of message passing. But the important concept of broadcast is not provided by existing work. Our UML-based graphical editor Teuta provides the corresponding tool support. We demonstrate our approach by describing a UML-based Grid workflow model for an advanced 3D medical image reconstruction application [3].

The paper is organized as follows. Section 2 briefly describes the subset of the UML that we use in this paper. A set of concepts that may be relevant for describing Grid workflow is presented in Section 3. A case study for an advanced 3D medical image reconstruction is presented in Section 4. Finally, some concluding remarks are made and future work is outlined in Section 5.

2 Background

In this paper, UML activity diagrams are used for representing computational, communication, and synchronization operations. We use the UML1.x notation [12], because at the time of writing this paper the UML2.0 specification is not officially accepted as a standard.

An activity diagram is a variation of a state machine in which the states represent the execution of actions or subactivities and the transitions are triggered by the completion of the actions or subactivities. An action state is used to model a step in the execution of an algorithm, or a workflow process. Transitions are used to specify that the flow of control pass from one action to the next action state. An activity diagram expresses a decision when guard conditions are used to indicate different possible transitions (see Figure 1(a)). A guard condition specifies a condition that must be satisfied in order to enable the firing of an associated transition. A merge has two or more incoming transitions and one outgoing transition. Fork and join are used to model parallel flows of control (see Figure 1(b)). The initial and final state are, respectively, visualized as a solid ball and a solid ball inside a circle. Figure 1(c) shows the UML notation for the object flow, which we use for the representation of the data flow in a workflow.

Figure 1(a) illustrates how to model a loop by employing an activity diagram, whereas Figure 1(b) shows one option for modeling the parallel execution of two activities.

3 Graphical Representation of the Grid Workflow

In this section we describe a set of UML elements and constructs that may be used for the graphical representation of the workflow. Due to space limitation we are not able to present all relevant aspects, but nevertheless we can illustrate

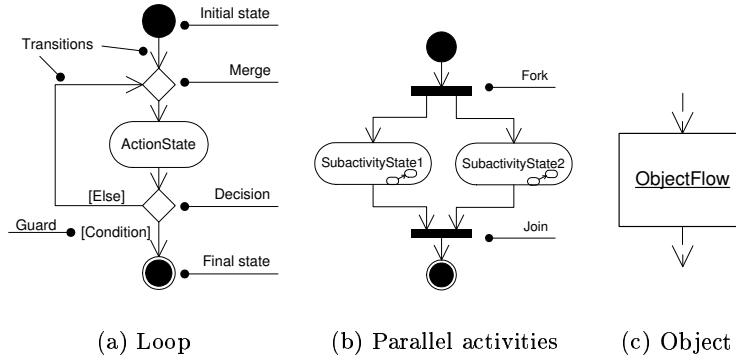


Fig. 1. UML activity diagram notation

the expressiveness of the UML activity diagram for specification of the workflow with a set of examples.

3.1 Workflow Hierarchical Decomposition

The Workflow Management Coalition [16] defines workflow as:

The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

A Grid workflow could be defined as a flow of a set of activities. Usually, these activities represent CPU intensive computation or the transfer of the large files.

The UML stereotyping extension mechanism provide the possibility to make the semantics of a core modeling element more specific. In addition, stereotyping mechanism improves the readability of the model, by associating a specific name to a model element. The stereotype *workflow*, which is defined based on the UML modeling element *subactivity*, is used to represent the workflow (see Figure 2(a)). The compartment of the stereotype *workflow* named *Tags* specifies the list of tag definitions which include *id* and *type*. The tag *id* is used to uniquely identify the element *workflow*, whereas the tag *type* is used to describe the element *workflow*. By using tags it is possible to associate an arbitrary number of *properties* (such as *id*) to a modeling element. Analogously are defined the stereotypes that are presented in the remaining part of this paper, but because of the space limitation the stereotype definition process is not shown.

The UML modeling element *subactivity* supports hierarchical decomposition. This makes possible the description of the workflow at different levels of abstraction. Figure 2(b) shows the workflow *SampleWorkflow1*. The content of the workflow *SampleWorkflow1* which comprises a sequence of workflows *SampleWorkflow2* and *SampleWorkflow3* is depicted in Figure 2(c).

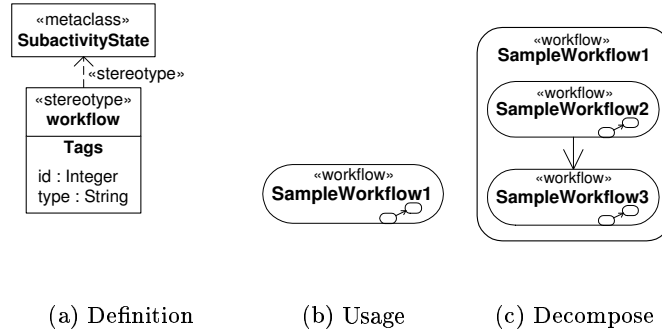


Fig. 2. The representation of the workflow hierarchical decomposition with the stereotype *workflow*

3.2 Elemental Operations

Compute, *TransferData*, and *View* are some of the elemental operations of the Grid workflow (see Figure 3). Element *Compute* (see Figure 3(a)) represents a computational intensive activity of the workflow. Element *TransferData* (see Figure 3(b)) represents the transfer of data. Properties of this element may specify the source and destination of the data, and the type of the data transfer. Figure 3(c) depicts the element *View*, which represents an activity of the visualization of the data.

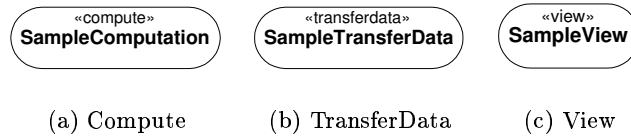


Fig. 3. Elemental operations

3.3 Control Flow

UML activity diagrams provide a reach set of elements for representing the control flow (see Section 2). Figure 4(a) shows an example of the branch. If the value of the logical expression *condition* is *True* then the activity *SampleComputation* is executed. An example of the loop is represented in Figure 4(b). The body of the loop is executed if the *condition* is true.

Existing work can express parallel activities which must be specified one by one through a fork mechanism (see Figure 1(b) in Section 2). However, Grid applications often exploit large numbers of independent tasks which should be naturally expressed by parallel loops (see Figure 4(c)). The '*' symbol in the upper right corner of the activity denotes dynamic concurrency. This means that multiple instances of the activity *SampleComputation* may be executed in parallel. The parameters of the parallel loop are specified by using the properties of the element *parallelloop*. The element *parallelloop* may be used to represent

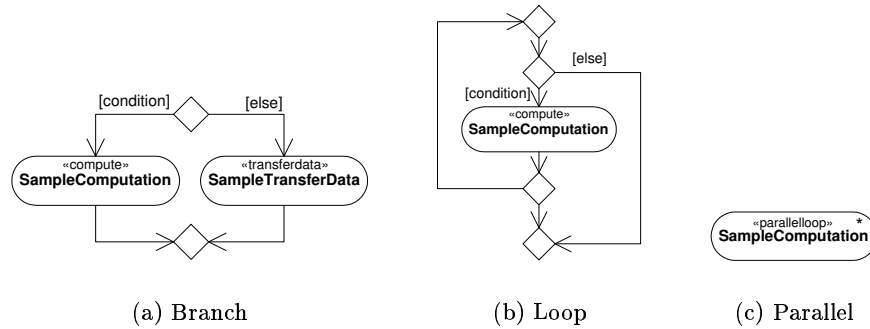


Fig. 4. Control flow

parameter studies on the Grid. The index can be used to query different input data for all concurrent running activities. In the gathering phase, all results can be for instance visualised or combined to create a final result of the parameter study.

3.4 Data Flow

One of the specific features of the Grid workflow is the moving of large amounts of data among workflow activities. Figure 5 shows an example of the graphical representation of the data flow. The output file *SampleFile* of the activity *SampleComputation1* serves as input for the activity *SampleComputation2*.



Fig. 5. Data flow

3.5 Synchronization

For the realization of the advanced synchronization workflow patterns we use *events*. Figure 6(a) shows the synchronization of parallel flows of control via events. The activity *SampleTransferData* will not begin the execution before either activity *SampleComputation1* or activity *SampleComputation2* is completed. The actions send and receive *event* are associated with transitions. An alternative notation for representing send and receive of an event is shown in Figure 6(b).

3.6 Notification

A Grid service can be configured to be a notification source, and a certain client to be a notification sink [14]. Multiple Grid clients may subscribe for a particular

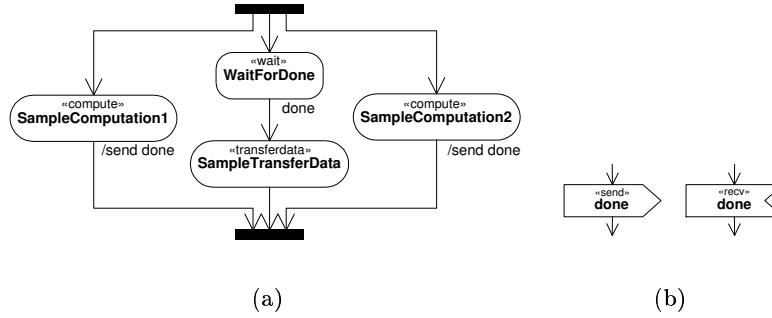


Fig. 6. Synchronization via events

information in one Grid service. This information is broadcasted to all clients that are subscribed. Figure 7 shows the graphical representation of the concept *broadcast*.

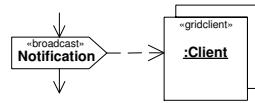


Fig. 7. The *broadcast* concept

Grid service notification mechanism may be used for directly sending large amounts of data. This avoids the communication via a central workflow control engine, which certainly would be a bottleneck for Grid applications.

3.7 Constraints

The information specified by the user in the form of *constraints* is important to enable the workflow repair and fault tolerance. If during execution some constraints are no longer fulfilled, then the execution environment should be able to find alternative mapping. A UML constraint specifies the condition that must be true for a modeling element. A constraint is shown inside the braces ({ }), in the vicinity of the element with which is associated.

Figure 8 shows an constraint, that is associated with the activity *SampleComputation*, which specifies that execution time of the activity should be less than 10 time units.

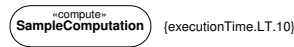


Fig. 8. Constraint specification

4 Case Study: Advanced 3D Medical Image Reconstruction

In order to illustrate the work described in this paper we will develop a workflow for a real-world grid application for 3D medical image reconstruction, which is developed by the Institute for Software Science of the University of Vienna in cooperation with the General Hospital of Vienna [2]. For this case study we use our UML-based editor Teuta [7, 8], which provides the tool-support for the graphical workflow composition of Grid applications.

The application for our case study, which is also part of the GEMMS project [3], is based on a service oriented architecture to provide clients with advanced fully 3D image reconstruction services running transparently on remote parallel computers. As a service layer we will use a Generic Application Service (GAPPService). GAPPService is a configurable software component which exposes a native application as a service to be accessed by multiple remote clients over the Internet providing common methods for data staging and remote job management: the upload method uploads the input files to the service, the start method executes the start script of the application and the download method downloads the output files. Additionally, a status method can be specified which returns the current status of the application.

The particular tasks necessary to execute the image reconstruction service are done manually using the web-based ClientGUI. In the most simple use cases the manual invocation performed by the client is sufficient (e.g. if the medical doctor wants to reconstruct 2D projection data). But in some cases an additional workflow would be able to automate the execution and facilitate the handling of the reconstruction service (e.g. for instrument calibration or for testing purposes). Therefore the next step will be to develop the workflow which will allow the automatic execution of the service tasks. In the following we will model such a workflow.

In the following we consider the abstract workflow which hides the details of the implementation and the actual workflow describing the execution details.

On the right-hand side of Figure 9 is depicted the abstract workflow which can be used in almost all use cases as general workflow. First of all the user has to upload an input file. Next the user has to execute the start script which starts the reconstruction software. Finally after the reconstruction has finished the download method can be executed and the output files can be downloaded. Now the received 3D image can be visualized.

On the left-hand side of Figure 9 is shown an activity diagram with the details hidden from the user. The workflow described below describes a very specific use case where the user can validate the quality of the images and start a new reconstruction with new parameters. As described in the previous activity diagram the first step is to upload the input file to the host where the GAPPService is installed. In the second step the user can execute the StartScript. The image reconstruction starts after the execution of the StartScript. If the StatusScript is specified, the user can invoke the getStatus method as long as the reconstruction

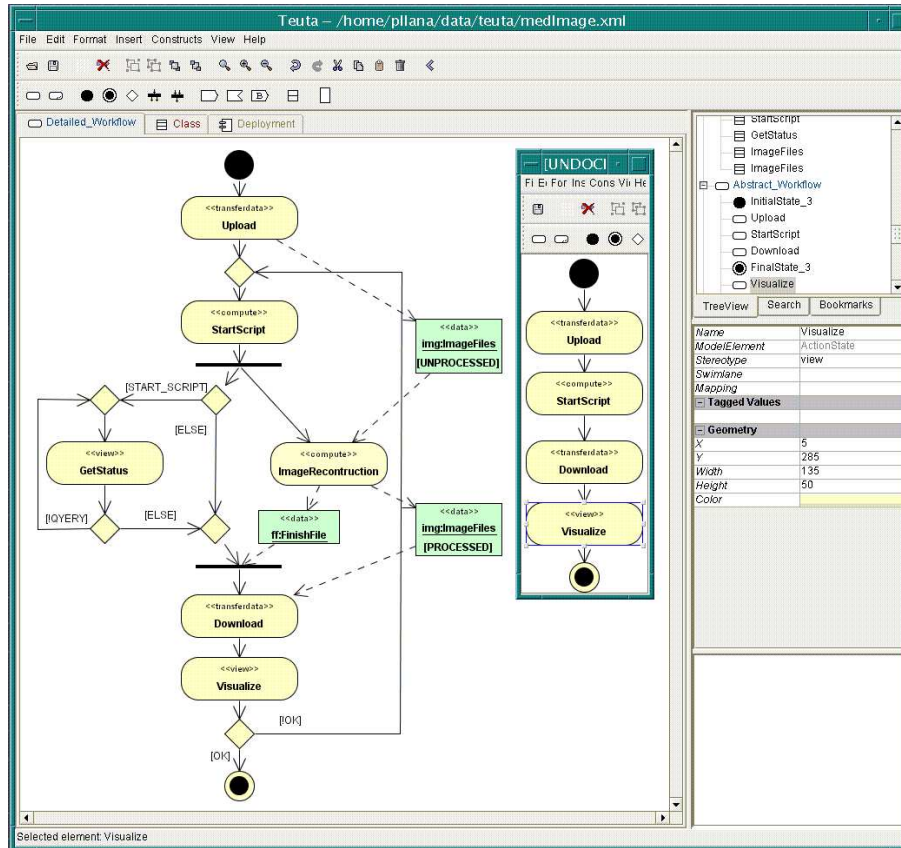


Fig. 9. The workflow composition of the 3D image reconstruction application

job is executing and obtain the current status of the application. After finishing the processing, the reconstruction application has to generate the finish file which is especially necessary if the StatusScript is not specified. The generation of the finish file informs the client that the application has finished processing. After the reconstruction has finished, the output file can be downloaded and visualized. If the reconstruction result is not satisfying, the reconstruction could be started again with new reconstruction parameters. This could be repeated until the visualized image has the acceptable quality.

5 Conclusions and Future Work

In this paper we have described an approach for graphically modeling and describing Grid workflow applications based on the UML standard. We have presented a set of concepts for modeling control flow, data flow, synchronization, notification, and constraints. We have demonstrated our approach by describing a Grid workflow model for an advanced 3D medical image reconstruction application.

The upcoming UML2.0 specification provides a richer set of concepts for behavioral modeling. We plan to extend our tool Teuta for supporting of the new concepts and notation of UML2.0.

References

1. T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services. Version 1.1, BEA, IBM, Microsoft, SAP, and Siebel, May 2003.
2. W. Backfrieder, M. Forster, S. Benkner, and G. Engelbrecht. Locally Variant VOR in Fully 3D SPECT within A Service Oriented Environment. In *International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, Las Vegas, USA, June 2003. CSREA Press.
3. G. Berti, S. Benkner, J. Fenner, J. Fingberg, G. Lonsdale, S. Middleton, and M. Surridge. Medical Simulation Services via the Grid. In *17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, France, April 2003. IEEE Computer Society.
4. H. Bivens. Grid Workflow. Sandia National Laboratories, <http://vir.sandia.gov/~hpbiven/>, April 2001.
5. T. Bray, J. Paoli, C. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). <http://www.w3.org/TR/REC-xml>, October 2000.
6. Department of Physics and Astronomy, Cardiff University. Triana, 2003. <http://www.triana.co.uk/>.
7. T. Fahringer, S. Pllana, and J. Testori. Teuta. University of Vienna, Institute for Software Science. Available online: <http://www.par.univie.ac.at/project/prophet>.
8. T. Fahringer, S. Pllana, and J. Testori. Teuta: Tool Support for Performance Modeling of Distributed and Parallel Applications. In *International Conference on Computational Science. Tools for Program Development and Analysis in Computational Science.*, Krakow, Poland, June 2004. Springer-Verlag.
9. Business Process Management Initiative. Business Process Modelling Language. www.bpmi.org/bmpi-downloads/BPML-SPEC-1.0.zip, June 2002.
10. S. Krishnan, P. Wagstrom, and G. Laszewski. GSFL : A Workflow Framework for Grid Services. Preprint ANL/MCS-P980-0802, Argonne National Laboratory, August 2002.
11. F. Leymann. Web Services Flow Language (WSFL 1.0). Technical report, IBM Software Group, May 2001.
12. OMG. Unified Modeling Language Specification. <http://www.omg.org>, March 2003.
13. OMG. XML Metadata Interchange (XMI) Specification. <http://www.omg.org>, May 2003.
14. B. Sotomayor. The Globus Toolkit 3 Programmer's Tutorial. <http://www.casa-sotomayor.net/gt3-tutorial/>, July 2003.
15. S. Thatte. XLANG: Web services for Business Process Design. Technical report, Microsoft Corporation, 2001.
16. The Workflow Management Coalition. <http://www.wfmc.org/>.