AGWL: Abstract Grid Workflow Language *

Thomas Fahringer[1], Sabri Pllana[2], and Alex Villazon[1]

[1] Institute for Computer Science, University of Innsbruck
Technikerstraße 25/7, 6020 Innsbruck, Austria
{Thomas.Fahringer,Alex.Villazon}@uibk.ac.at

[2] Institute for Software Science, University of Vienna
Liechtensteinstraße 22, 1090 Vienna, Austria
pllana@par.univie.ac.at

Abstract. Grid workflow applications are emerging as one of the most interesting application classes for the Grid. In this paper[3] we present AGWL, a novel Grid workflow language to describe the workflow of Grid applications at a high level of abstraction. AGWL has been designed to allow the user to concentrate on describing scientific Grid applications. The user is shielded from details of the underlying Grid infrastructure. AGWL is XML-based which defines a graph of activities that refers to computational tasks or user interactions. Activities are connected by control- and data-flow links. We have defined AGWL to support the user in orchestrating Grid workflow applications through a rich set of constructs including sequence of activities, sub-activities, control-flow mechanisms (sequential flow, exclusive choice, and sequential loops), data-flow mechanisms (input/output ports), and data repositories. Moreover, our work differs from most existing Grid workflow languages by advanced workflow constructs such as parallel execution of activities with pre- and post-conditions, parallel loops, event-based synchronization mechanisms, and property-based selection of activities. In addition, the user can specify high-level constraints and properties for activities and data-flow links.

1 Introduction

In the past years extensive experience has been gained with single site applications and parameter studies for the Grid. For some 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) 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 [12, 5] and web services (BPEL4WS [2], XLANG [10], WSFL [7], SWFL [4]), this programming model is relatively new in the Grid computing area.

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

[3] © Springer-Verlag

Low-level XML-based workflow languages, like Grid Service Flow Language (GSFL) [6] and Grid Workflow [3], have been proposed to describe Grid workflow applications. However, these languages are missing some important control flow constructs such as branches, loops, split and join.

In the GridPhyN Project [1], Grid workflow applications are constructed by using partial/full abstract descriptions of "components", which are executable programs that can be located on several resources. Their abstract workflow is limited to acyclic graphs. Moreover, a reduced data-flow model (no customization is allowed) is supported, and there is no advanced parallel execution of components.

In the GridLab Triana Project [11], workflows are experiment-based: Each unit of work is executed for a specific experiment. A workflow (task graph) defines the order in which experiments are executed. Simple control-flow constructs can be used to define parallel processing, conditional execution (*if*, *switch*) and iterations (simple sequential *loops* and *while*).

In this paper we describe the Abstract Grid Workflow Language (AGWL) which allows the user to compose scientific workflow applications in an intuitive way. AGWL is an XML-based language for describing Grid workflow applications at a high level of abstraction without dealing with implementation details. AGWL does not consider the implementation of activities, how input data is delivered to activities, how activities are invoked or terminated, etc. AGWL has been carefully crafted to include the most essential workflow constructs including activities, sequence of activities, sub-activities, control-flow mechanisms, data-flow mechanisms, and data repositories. Moreover, we introduce advanced workflow constructs such as parallel activities, parallel loops with pre- and post-conditions, synchronization mechanism, and event based selection of activities. We support advanced data-flow constructs such as direct communication and broadcast communication. In addition, the user can specify high-level constraints on activities and on data-flow in order to support the underlying workflow enactment machine to optimize the execution of workflow applications.

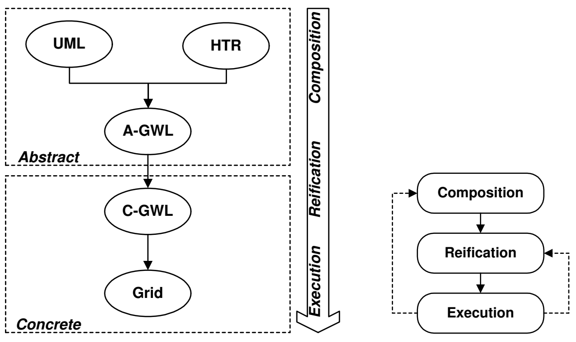
The paper is organized as follows. Section 2 depicts the workflow application development process, from the abstract and concrete level to the execution of the workflow application (*composition*, *reification* and *execution*). Section 3 introduces the basic concepts of AGWL as well as advanced constructs for control- and data-flow. Finally, Section 4 concludes the paper.

2 Overview

In this section we give an overview of our Grid workflow application development process from abstract representation, to the actual execution on the Grid.

Figure 1.a depicts the complete process of developing a Grid workflow application. Figure 1.b shows the flow of three fundamental activities *Composition*, *Reification* and *Execution*. The dashed lines indicate the iterative process of workflow application development.

Composition. The user composes the Grid workflow by using the Unified Modeling Language (UML) [8]. Alternatively, a High-level Textual Represen-



(a) (b)

Fig. 1. The development of Grid workflow applications. Acronyms: UML - Unified Modeling Language, HTR - High-level Textual Representation, AGWL - Abstract Grid Workflow Language, CGWL - Concrete Grid Workflow Language.

tation (HTR) may be used for the workflow composition. Our tool Teuta [9] supports the UML based workflow composition, and the automatic conversion to Abstract Grid Workflow Language (AGWL). AGWL is an XML based language for describing the Grid workflow at a high level of abstraction. At this level activities correspond mostly to computational entities and the specification of its input and output data. There is no notion of how input data is actually delivered to activities, or how activities are implemented, invoked and terminated. AGWL contains all the information specified by the user during the workflow composition.

Reification. The software infrastructure transforms AGWL to the Concrete Grid Workflow Language (CGWL). CGWL contains the information specified by the user and the additional information provided by the software infrastructure (i.e. the actual mapping to Grid resources), which is needed for the execution of the workflow. It describes the realization of AGWL by specifying how activities are actually implemented (e.g. Grid services), how they are deployed, invoked, and terminated. CGWL also describes how data is transferred and in what form. CGWL is an XML based language as well. Currently, we are in the process of the development of specification for CGWL.

Execution. CGWL is interpreted by the underlying workflow enactment machine to construct and execute the full Grid workflow application.

3 AGWL Constructs

In AGWL, an activity is a generic *unit of work*. At this level, it is not seen how the activity is implemented at the concrete level. An activity may be mapped to Grid services, software components or web services under CGWL. A *workflow instance* is the mapping of an AGWL workflow to a CGWL workflow that is actually executed on the Grid. Different workflow instances can run concurrently. The user can specify constraints that are used to tune the execution of the workflow application. An activity can be a basic *computation*, a *sequence* of activities, or a composed *sub-activity*.

The composition of the application is done by specifying both control-flow and data-flow between activities. AGWL supports basic control-flow (sequential flow, exclusive choice and sequential loops) and advanced control-flow constructs such as parallel execution of activities, parallel loops with pre- and post-synchronizations and selection of activities.

Basic data-flow is specified by connecting input and output ports between activities. The user can associate constraints with data-flow, which will be mapped, e.g., to concrete mechanisms that utilise secure communication or advanced QoS for data-transfer. In addition to basic data-flow between activities, AGWL supports advanced data-flow between activities and repositories. In the following, we describe the most essential AGWL elements in more detail.

3.1 Basic Elements

ACTIVITY: An activity in AGWL is presented as a *black-box* with input/output ports, constraints, and properties. Input and output ports are used to specify the data-flow. Control-flow is made through explicit links among activities and by using different control-flow constructs (e.g. sequences, loops, etc).

The user specifies data-flow by connecting input and output ports among activities which may not necessarily be directly connected with control flow links.

Additional information about activities can be provided through constraints and properties. Constraints provide additional information that should be honored by the underlying workflow enactment engine when executing the Grid workflow application. Examples for constraints are as follows: execute activity on Grid machines running under a specific operating system, with a minimum amount of free dynamic memory, with more than some pre-defined idle time value, deployment requirements, etc. Properties describe information that can be useful for the workflow enactment engine for instance to schedule applications. Performance information (estimated execution times or floating point operations) is frequently indicated via properties.

The XML representation[4] for the ACTIVITY construct is given as follows,

```
<activity name="name">
  <input name="name"(repository="name" query=expression)? />*
  <output name="name" (repository="name")? /> *
  (<constraints> constraints+ </constraints>)?
  (<properties> properties+ </properties>)?
  <control-in name="name"/> *
  <control-out name="name"/> *
</activity>
```

Every *activity*, *input*, *output*, *control-in* and *control-out* have a unique *name* in a workflow application. Input and output ports can be used to specify data-flow between activities and/or repositories (see Section 3.4 and 3.5).

[4] We use the notation convention from [2] for the informal syntax of XML grammar.

AGWL supports the notion of hierarchical decomposition of activities. An activity containing other activities (sub-workflow), is also an activity. In that case, all the input ports of the incoming activities of the sub-workflow are mapped to the input ports of the composed activity, and the output ports of all the outgoing activities are mapped to the output ports of the composed activity. Analogously (re)connections are handled for the control-flow.

3.2 Basic Control-flow

AGWL supports basic control-flow among activities as in conventional programming languages. The control-flow constructs are: the standard sequential flow (SEQUENCE), exclusive choice (SWITCH, IF-THEN-ELSE), and sequential loops (FOR, WHILE and DO-UNTIL). AGWL can be used to specify multiple exits from loops. The BREAK construct can be used to exit the execution of a loop construct. We also support an EXIT construct to terminate the execution of the entire workflow instance.

3.3 Advanced Control-Flow

In addition to basic control-flow constructs that are commonly found in workflow languages, AGWL supports advanced control-flow elements that are highly useful for scientific Grid workflow applications. These constructs provide the user with enough abstraction to compose advanced and complex Grid workflow application and specify concurrency in a very natural way (parallel execution of activities, parallel loops).

PARALLEL: The PARALLEL construct is used to execute activities concurrently. The XML representation of the PARALLEL construct is as follows,

```
<parallel syncCondition=IdentifierSet>
  (<fork preCondition=expression > activity </fork>)+
  (<default> activity </default>) ?
</parallel>
```

Before executing the activities specified in `<fork>`, the *preCondition* is evaluated for all the activities. Only those that evaluate to true, are executed concurrently. If no *preCondition* is specified, all the activities are executed in parallel. The user can also specify a synchronizing post-condition (*syncCondition*) in order to wait for specific activities at the end of the PARALLEL construct, and then continue the control-flow, e.g. this feature can be used to select the *fastest* activity. Other examples of post-conditions include "wait for all activities to finish", "wait for one", and so on.

If no *syncCondition* is specified, the PARALLEL construct will wait for all activities to be finished. By default, the same input is passed to each parallel activity. However, different input sources can also be specified, e.g. from other activities or from repositories (see Section 3.5 for advanced data-flow).

Finally, the optional `<default>` activity inside the PARALLEL construct is used to specify an activity that is executed if all the pre-conditions of all activities of a parallel construct evaluate to false.

PARALLEL-FOR: The PARALLEL-FOR construct can be used to specify parallel for-loops. The XML representation for PARALLEL-FOR construct is,

```
<parallel-for index="i..j">
  <body> activity </body>
  <gathering> activity </gathering>
</parallel-for>
```

A PARALLEL-FOR consists of two parts: *body* and *gathering*. The body describes the activities of a loop. A PARALLEL-FOR defines that each of its iterations can be executed in parallel. It does not specify any parallelism within a loop iteration. Parallelism within a loop iteration can be achieved through the parallel construct. Each activity inside of a `parallel-for` loop will receive the loop *index* as an implicit input value. If there is a need for other input values they need to be explicitly defined within the body for each activity. Once all concurrent running activities have finished, the *gathering* part will be executed. Gathering is a special activity provided by the user that utilises the output of all the concurrent activities, and potentially produces some combined output.

The PARALLEL-FOR construct may be used to model parameter studies on the Grid. The index can be used to access different input data for all concurrent running activities. In the gathering phase, all results can be e.g. visualised or combined to create a final result of the parameter study.

SELECT: AGWL introduces the SELECT construct which selects one activity from a set of activities based on their *properties* and a specific *criteria*. Activity properties may define, for instance, information about performance characteristics, quality of service parameters, etc. The selection criteria is an expression defined over properties to determine minimum, maximum, average, threshold values, etc. The XML representation for the SELECT construct is:

```
<select property="aProperty" criteria="aCriteria">
   activity+
</select>
```

The SELECT contruct is frequently used to determine an activity with the longest or shortest predicted execution time.

3.4 Basic Data-Flow

As described in Section 3.1, the input and output ports of activities are used to model basic data-flow. If the activity refers by name to the output port of another activity, a data-flow link is established. Data-flow can be expressed among arbitrary activities not necessarily connected by any control flow link at all. For example, a data-flow link between activities `A` and `B` is simply established by defining the output of `A` as `<output name="oA">`, and the input of `B` as

`<input name="oA">`. If more than one activity declares `oA` as a desired input, then the output will be sent to all those that *subscribed* to it. This can be seen as some kind of simple *notification* or *broadcasting* mechanism.

3.5 Advanced Data-Flow

REPOSITORY: In addition to the basic data-flow among activities, AGWL supports data-flow between activities and special entities called *repositories*, which are abstractions for data containers. They are used to model, for instance, saving intermediate results or querying data resources without bothering about how a repository is actually implemented (file servers, databases, etc.).

A repository is not an activity. It has a unique name. The insertion and retrieval of information is made by using the repository name without specifying any explicit input/output ports, i.e. data-flow between activities and repositories is specified by using the input/output ports of activities and the name of the repository. If an activity should store its output in a repository, then it should specify the name of the repository and attribute of the `<output>` tag (see Section 3.1). For instance, an activity `A` that wants to store its output (named `oA`) in a repository `R`, must simply define the following output: `<output name="oA" repository="R"/>`.

Enhanced Data-Flow: In addition to basic data-flow among activities, AGWL extends the basic data-flow links with optional user-defined *constraints* and *properties* which can be used by the underlying workflow enactment engine to tune the data-transfer mechanism and behavior. Constraints again imply requirements that should be fulfilled by the enactment machine whereas properties is simply extra information. The user may have experience about the kind of output and input that is required by the activities. He thus may define constraints such as "store locally", "use high-speed network", "use-cache", "use-replica", "secure transfer", etc. by specifying specific mnemonics. Input and output data types, estimated data volumes, etc. can be described through properties.

The extension for the specification of constraints and properties are associated with the `input` and `output` ports of activities (here we show only the `output` port). The XML representation is,

```
<output name="name" (repository="name") ? >
  (<constraints> constraints+ </constraints>) ?
  (<properties> properties+ </properties>) ?
</output>
```

3.6 Synchronization with Events

AGWL supports synchronization of activities based on *events*. An illustrative example is the AGWL construct FIRST.

FIRST: This construct selects one activity from a set of activities based on the occurence of a specific event. Every activity A_i in a set of activities $\{A_1, .., A_N\}$ is associated with an event e_i in a set of events $\{e_1, .., e_N\}$. FIRST construct waits for one of the events from the set of events to occure. The first event that occures determines which activity is selected for execution. The XML representation for the FIRST construct is,

```
<first>
  (<event id="eventId">
      activity
  </event>) +
</first>
```

4 Conclusions and Future Work

In this paper we have described the abstract Grid workflow language (AGWL) to simplify the specification of Grid workflow applications at a high level of abstraction without being distracted by low level details (e.g. how to start and stop tasks, how to transfer input and output data, what communication protocol to use, etc.). AGWL is XML-based and has been carefully crafted to cover the most essential workflow constructs including sequence of activities, sub-activities, control-flow/data-flow/event mechanisms, and data repositories.

We have developed a UML interface to AGWL called Teuta [9], in order to graphically orchestrate Grid workflow applications. We are currently in the process to develop a Grid workflow enactment engine to execute Grid workflow applications based on AGWL.

References

1. GriPhyN: Grid Physics Network Project. www.griphyn.org.
2. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, 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.
3. H. Bivens. Grid Workflow. Sandia National Laboratories, http://vir.sandia.gov/∼hpbiven/, April 2001.
4. Y. Huang. SWFL: Service Workflow Language. Technical report, Welsh e-Science Centre-Cadiff University, 2003. http://www.wesc.ac.uk/projects/swfl/.
5. Business Process Management Initiative. Business Process Modelling Language. www.bpmi.org/bmpi-downloads/BPML-SPEC-1.0.zip, June 2002.
6. S. Krishnan, P. Wagstrom, and G. Laszewski. GSFL : A Workflow Framework for Grid Services. Preprint ANL/MCS-P980-0802, Argonne National Laboratory, August 2002.
7. F. Leymann. Web Services Flow Language (WSFL 1.0). Technical report, IBM Software Group, May 2001.
8. OMG. Unified Modeling Language Specification. http://www.omg.org, March 2003.
9. S. Pllana, T. Fahringer, J. Testori, S. Benkner, and I. Brandic. Towards an UML Based Graphical Representation of Grid Workflow Applications. In *The 2nd European Across Grids Conference*, Nicosia, Cyprus, January 2004. Springer-Verlag.
10. S. Thatte. XLANG: Web services for Business Process Design. Technical report, Microsoft Corporation, 2001.
11. Cadiff University. The Triana Project. http://trianacode.org/triana/.
12. The Workflow Management Coalition. http://www.wfmc.org/.