USING GWORKFLOWDL FOR MIDDLEWARE-INDEPENDENT MODELING AND ENACTMENT OF WORKFLOWS

Simone Pellegrini, Francesco Giacomini, Antonia Ghiselli
INFN Cnaf
Viale Berti Pichat, 6/2 - 40127 Bologna, Italy
simone.pellegrini@cnaf.infn.it
francesco.giacomini@cnaf.infn.it
antonia.ghiselli@cnaf.infn.it

Andreas Hoheisel
Fraunhofer FIRST
Kekuléstr. 7, 12489 Berlin, Germany
andreas.hoheisel@first.fraunhofer.de

Abstract Nowadays, the usage of workflow management systems (WfMSs) for automating complex IT processes in Grid environments is gaining more and more importance. The motivation of this trend is the demand for reducing the execution time and the organizational overhead, and for increasing the reusability and reliability of processes. Most WfMSs and corresponding workflow description languages, however, are bound to a specific Grid middleware, which limits the reusability and makes it difficult to interchange workflows between different WfMSs. This paper continues the standardization process started inside the CoreGRID project and presents an extension to the Petri-Net-based *Grid Workflow Description Language* that enables the middleware-independent definition of workflows, which can be processed by several WfMSs. A real-world example workflow using the rendering software *POV-Ray* serves as a case study to validate the interoperability of workflow descriptions over two WfMSs targeting different Grid middlewares: the *Grid Workflow Execution Service* and the *INFN/CNAF WfMS*.

Keywords: Grid Workflow, Workflow Enactment, Workflow Description Languages, Petri Nets, GWorkflowDL, gLite, Globus Toolkit 4, Web Services

1. Introduction

According to Ian Foster et al. [7], the service-oriented science paradigm is going to become one of the main topics in second generation Grids, with the ability to use third-party services, compose them into new functions and publish them as a new service. In this scenario, the composition and the execution of coupled services, which is automated by Workflow Management Systems (WfMSs), is of special importance. In the business environment, workflows are already widely used for business processes, and interoperability is made possible, e.g., by means of the BPEL4WS standard established in 2003 [5]. Unfortunately, the approaches from the business domain are not directly applicable to scientific workflows as the scientific research processes are much more dynamic than settled business processes. Therefore, interoperability is still an open issue in the scientific environment. Actually, there is no recognized standard language for scientific workflow descriptions, and even about the formalisms used to model workflows (π-Calculus [15], DAGs, Petri Nets [1], UML diagrams [6]) still exists a debate [2].

The Petri Nets formalism [1] is more suitable for modeling scientific workflows, it can formally describe both the data and the control flow making the state of the program execution explicit. Petri Nets have been improved since its original definition introducing several extensions, such as Colored Petri Nets, Timed Petri Nets and Hierarchical Petri Nets. High Level Petri Nets (HLPN) refer to a Petri Net model extended with tokens that represent high-level values, i.e., a place is marked by a multiset of structured tokens; and they are more suitable for workflow representation [11]. Inside the CoreGRID European project, a HLPN-based workflow description language, called *Grid Workflow Description Language (GWorkflowDL)*, has been proposed by Fraunhofer FIRST in [9],[3].

In contrast to other approaches from the business domain, such as BPEL [4], the GWorkflowDL has much simpler and more formal semantics, which eases the usage of formal analysis methods and enables also unskilled end users to model and modify workflows by their own. The GWorkflowDL is an XML-based language for representing Grid workflows which consists of two parts: (*i*) a *generic* part, used to define the structure of the workflow, reflecting the data and control flow in the application, and (*ii*) a middleware-specific part (*extensions*) that defines how the workflow should be executed in the context of a specific Grid computing middleware. While the generic part can be shared among different GWorkflowDL-compatible middlewares, the extensions don't because they are strongly middleware-dependent. Practically, — as far as extensions are middleware-specific — the interoperability of the GWorkflowDL is currently limited to abstract workflows without connection to functional operations.

Another aspect of Petri-Net-based workflows regards the potentially high number of nodes (places and transitions) often involved in complex workflow descriptions. As far as the Petri Net formalisms are graph-based, the high number of nodes becomes an important issue during the design time of the workflow; the higher the number of nodes — the more complex it is to debug and change it.

In this paper, we introduce some improvements to the GWorkflowDL in order to achieve what in mainstream programming languages is known as the "*write once and run everywhere*" paradigm. We are going to define an *abstract* kind of operation, called `operationClass`, that GWorkflowDL compatible WfMSs *must* understand and implement; and mechanisms used to map such abstract operation into concrete services, such as the invocation of Web Service method calls or the remote execution of command line programs. Also a mechanism for *sub workflow* invocation will be defined, that as happens with local method calls, it allow code reuse and simplify the overall design process of workflows. The concept of abstract operations applied to the GWorkflowDL solves some interesting interoperability issues and opens several scenarios in workflow execution; for example, a platform might demand the execution of an independent part of a workflow (a sub-workflow, for example) to a different WfMS, or furthermore a repository of workflow descriptions can be managed and shared among several WfMSs.

Sect. 2 describes in detail the extensions introduced in the GWorkflowDL, defining operation language elements for the web service invocation and the remote program execution. In Sect. 3 we will show a concrete example of such interoperability running a workflow description on two different platforms (based on gLite and on Globus Toolkit 4), concluding with the description of how we intend to progress with this work in the future in Sect. 4.

2. Concept

In order to better illustrate the rationale behind this paper, we first consider a trivial workflow example compound by a single operation. Using the Petri Net formalism, it can be expressed as shown in Fig. 1. The transition `T` performs a single workflow operation, and the input and output places are used to hold the tokens which represent the input and output data involved in the operation.

The operation associated to the transition `T` can be formally described as $(b_1 \dots b_M) = f(a_1 \dots a_N)$ and, according to the Petri Net theory, it is executed when all its input variables can be bound to a token from the input places and if the output places have not reached their capacity. In this case, $a_1 \dots a_N$ variables are bound, at runtime, respectively to the tokens $t_1 \dots t_N$, and the results of the operation, temporary available in variables $b_1 \dots b_N$, are stored as tokens into the output places. For more details about how to apply the Petri

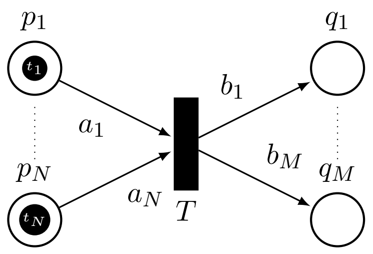

Figure 1. A *generic* single operation modelled as a HLPN-based workflow

```
<workflow>
  <place ID="p1">
    <token><data><t1 xsd:type="type">value</t1></data></token>
  </place>
  <place ID="p2">
    <token><data><t2 xsd:type="type">value</t2></data></token>
  </place>
  <place ID="q1" />
  <transition ID="T">
    <inputPlace placeID="p1" edgeExpression="a1"/>
    <inputPlace placeID="p2" edgeExpression="a2"/>
    <outputPlace placeID="q1" edgeExpression="b1"/>
    <operation /> <!-- generic operation -->
  </transition>
</workflow>
```

Figure 2. GWorkflowDL description of the Petri Net in Fig. 1 considering $N = 2$ and $M = 1$

Net theory to Grid workflows, including conditions ("transition guards") and XPath edge expressions, please refer to [11].

Using the GWorkflowDL, the structure of the Petri Net can be described as depicted in Fig. 2. The description relies on the generic part of the GWorkflowDL, actually the structure of the network is completely described (in terms of places, transitions, and edges); however, a workflow also consists of operations to be performed and they cannot be represented at this level of the language.

In order to solve this interoperability problem, the GWorkflowDL has been extended with the concept of an *abstract operation*, which, due to its abstract nature, can be part of the generic workflow description. Therefore, we introduced the `operationClass` element, depicted in Fig. 3 (*i*), which contains all necessary information for the WfMS to automatically map the abstract operation onto a concrete operation and which at the same time remains completely middleware-independent.

```
<transition ID="T">
  ...
  <operation>
    <op:operationClass name="f">                                  (i)
      <!-- concrete service providers' list -->
      <op:execOperation software="software:g"                    (ii)
        hardware="hardware:localhost" quality="0.8"/>
      <op:subwOperation gwdl="file://workflows/g.gwdl"          (iii)
        operationName="g" quality="0.9" selected="true"/>
    </op:operationClass>
  </operation>
</transition>
```

Figure 3. The `operationClass` element `(i)` including two child elements `(ii)` and `(iii)` representing concrete implementations of the abstract operation *f*

The mapping of the abstract operation f onto concrete services implementing f is done by the WfMS's *resource matching* and *scheduling* process, and bases on the following information: The name of the abstract operation (f), the edge expressions of the incoming $(a_1 \ldots a_N)$ and outgoing edges $(b_1 \ldots b_M)$ and the values of the input tokens $(t_1 \ldots t_N)$. Therefore, we can write:

$$(g_1 \ldots g_L) = f_{map}(f, a_1 \ldots a_N, b_1 \ldots b_M, t_1 \ldots t_N) \tag{1}$$

L represents the number of matching service candidates (g). In a more sophisticated approach, the mapping could also depend on larger workflow regions, taking into account the previous and following operations within the workflow, e.g., to optimize the communication between operations.

Fig. 3 (*ii*) and (*iii*) depicts an example result of this mapping process with $L = 2$. The abstract operation f, represented by the `operationClass` element, has two concrete implementations; the first one (*ii*) is the remote execution of the program identified by the name `software:g` (e.g., representing the program `/usr/bin/g`) demanded to the remote machine with the identifier `hardware:localhost`, the second one (*iii*) is the invocation of a Sub-Workflow whose definition is located at `file://workflows/g.gwdl`. The mapping algorithm itself is strictly related to a specific middleware implementation, it could, for example, just be a simple lookup-table which maps abstract functions to concrete services or a more complex solution involving ontologies, based on equation 1.

In order to demonstrate how this mechanism could work in a heterogeneous Grid environment, two different types of concrete workflow operations have been considered in our example. In fact, the majority of the scientific workflows rely on few types of operations which are mostly represented by either Web service invocation or remote program execution. The UML di-

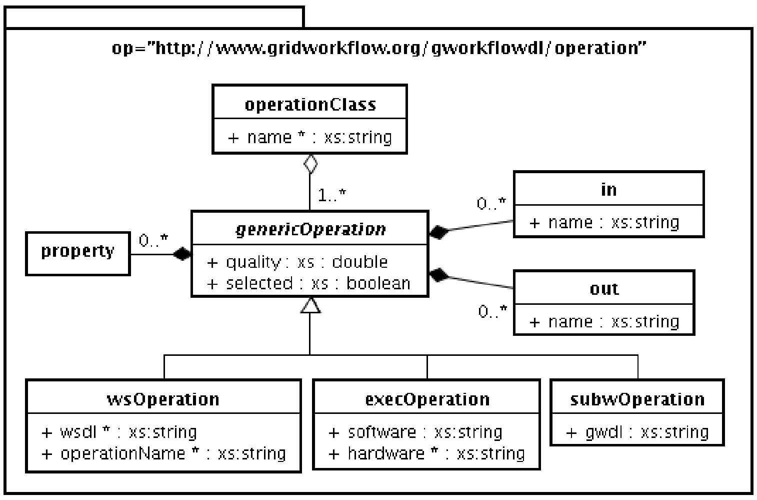

Figure 4. UML representation of the `operationClass` XML Schema. (Attributes with the '∗' character represent *required* XML attributes)

agram depicted in Fig. 4 shows the relations among the `operationClass` XML element and the concrete operations `wsOperation`, `execOperation` and `subwOperation`. The diagram in Fig. 4 represents the XML Schema of the namespace "op". XML elements are considered as UML classes, and XML attributes as UML attributes (where the ∗ character in UML attributes identifies *required* XML attributes). The full XML schema will be soon available for download at `http://www.gridworkflow.org/gworkflowdl/operation`.

The `genericOperation` element in the diagram doesn't represent any concrete operation but allows to factorize some common properties of underlying operations. Each concrete operation has a `quality` attribute used to describe the quality of the provided service, which is essential for providing certain QoS and for optimizing the resource selection. The `quality` attribute can depend on various dynamic monitoring properties, such as load, free memory, reaction time, security constraints and reliability of the resources. The optional elements `in` and `out` express a specific mapping between the input and output places on the one hand and the input parameters and return values of the associated operation on the other hand. More details about concrete operations representation is provided in the next sections by means of examples.

2.1 Web Service Invocation

Today the main Grid middlewares used in the scientific community (Globus Toolkit, UNICORE, and gLite) expose Grid middleware functionality as Web

```
<operation>
  <op:operationClass name="f">
    <!-- b1 = f(a1,a2) -->                                      (i)
    <op:wsOperation wsdl="http://localhost:8080/f?wsdl"
        operationName="f" quality="0.5">
    </op:wsOperation>
    <!-- b1 = f(a1,a2) = g(a1/x,a2*a1+15,35) -->               (ii)
    <op:wsOperation wsdl="http://localhost:8080/g?wsdl"
        operationName="g" quality="0.9" selected="true">
      <in name="input1">a1/x</in> <!-- child "x" of element "a1" -->
      <in name="input2">a2*a1+15</in>
      <in name="input3">35</in>
      <out name="output1">b1</out>
    </op:wsOperation>
  </op:operationClass>
</operation>
```

Figure 5. Example of two concrete Web Service operations. `(i)` shows the default mapping scheme, `(ii)` a more complex mapping between the abstract and the concrete operation and its parameters, using XPath 1.0 syntax.

Services (*Grid Services*); so it is possible to think of workflows as a set of Web Service invocations executed in respect of their interdependencies. Furthermore, Web Service helps to achieve interoperability as far as it is a software technology designed to support interoperable Machine-to-Machine interaction over a network based on message exchange according to the SOAP protocol.

Considering the Petri Net structure described in Fig. 2, the abstract operation f could be implemented by two concrete Web Service candidates as depicted in Fig. 5.

The WSDL interface description of the first concrete Web Service operation (*i*) is located at `http://localhost:8080/f?wsdl`. In general, this WSDL document also specifies the binding URL of the target Web Service (e.g., `http://localhost:8080/f`). The attribute `operationName` declares the name of the Web Service method `f` to be invoked by the operation. As far as there is no additional information, it means that the input tokens (consumed from the input places of the transition) must be used in the invocation argument list considering the order of the message part names specified within the WSDL and matching it with the incoming edge expressions. The return value, which is unique, is associated to the unique output variable `b1`.

In the second service candidate (*ii*) the `in` XML elements are used in order to specify a custom mapping for the input parameters (and return value) of the Web Service method invocation. In this case, the f function has been mapped onto the concrete service `g` which receives four input values. Within the `in` element, XPath expressions can be expressed as input parameters and the mapping between the values; their position in the web service method call is established in two ways: *positional* if the `name` attribute is not specified (not shown here); and *WSDL-based* if the `name` attribute is matched with the SOAP message part names of the WSDL declaration of the method (as shown in Fig. 5). The `out` XML elements are used analogically to describe mappings for output parameters.

The additional XML element "`property`" defined in the `genericOperation` as depicted in Fig. 4 can be used in order to specify additional information about Web Service invocation. A typical usage of this element could be related to authorization issues; but those aspects, which are strongly related to a particular middleware, are beyond the scope of this paper.

2.2 Program Execution

Since its birth, the Grid has been defined as a *batch system*. Many of the Grid-based WfMSs allow to express workflows as a collection of Grid jobs which are submitted to a Grid middleware or a local batch/queue system (such as PBS or Condor) respecting the dependencies among them. Job submission, i.e., the remote execution of command line programs, is a Grid primitive which is provided by all well-known Grid middlewares, the operation, however, cannot be generalized, as happens with Web Services, because of the middleware-dependent job submission interfaces. As a consequence, the program execution operation could have several implementations depending on the number of Grid middlewares addressed by a specific platform. The usage of the `in`, `out` and `property` XML elements is similar as described in the previous section; a concrete example will be discussed in the next chapter.

2.3 Sub-Workflow execution

The sub-workflow invocation mechanism allows to define wokflows as simple components, which can be combined together into higher-level workflows. The interface of a sub-workflow is defined by two sets of places considered as input and output places of the sub-workflow. A sub-workflow can be associated to a transition if compatible, it means that the number (and type) of the input and output places must be equal to the sub-workflow interface. The definition of a sub-workflow can be done using the GWorkflowDL without any extension, just by linking a `subwOperation` to another GWorkflowDL document, which represents the sub-workflow as shown in Fig. 3.

3. Case Study

This section describes a real-world application of the above mentioned concept in order to validate the interoperability of the proposed workflow language extensions regarding existing Grid infrastructures. In the following, we describe two implementations of a GWorkflowDL execution engine, one implemented in C++ supporting Web Services and actually working on top of the gLite middleware; the other one, called *Grid Workflow Execution Service* (GWES) implemented in Java supporting Globus Toolkit 4 and standard Web Services.

3.1 The INFN/CNAF Workflow Management System

The Workflow engine developed by the INFN/CNAF research center within the CoreGRID project is mainly written in C++ and *formally* implements the HLPN formalism [14]. The main idea behind the engine design is that every kind of task can be described composing a finite number of simple operations into a new workflow modeled with the Petri Net formalism. The engine relies on the fact that operations of Petri Net transitions can be one of the following: (*i*) Web Service invocation, (*ii*) local method execution, or (*iii*) sub-workflow invocation. This makes it possible, for example, to define a workflow that describes how to submit a job to the gLite (or the UNICORE) Grid middleware; and, using the Sub-Workflow invocation mechanism, such workflows can be mixed up in order to represent more complex tasks.

The advantages of this design choice are manifold: the support of a new Grid middleware can be provided at runtime simply by supplying a new workflow description; because of its simplicity, the engine's core is fast and extremely reliable. The workflow engine is still in prototyping stage; however, in the current state of development it is capable of running simple workflows on the gLite middleware.

3.2 The Grid Workflow Execution Service (GWES)

The Grid Workflow Execution Service (GWES) [12],[8] is a workflow enactment engine developed and enhanced under the leadership of Fraunhofer FIRST within various projects, such as the Fraunhofer Resource Grid, K-Wf Grid, Instant-Grid, MediGRID, Enterprise Grids, and CoreGRID. The GWES enables automation and interactive monitoring of complex processes executed in Grid environments. A unique feature of the solution is the completely virtualized resource allocation based on abstract modelling of the process structure and the powerful resource description formalism. The GWES controls the execution of workflows and maintains a persistent image of their state in an XML database. It is implemented as a standard Web Service thus easing its integration into existing IT infrastructures and business processes. An intuitive user interface is available as a JSR 168-conformant Java portlet which can be smoothly integrated into existing portal solutions (e.g., GridSphere). The Grid Workflow Management System is compatible to several Grid middleware

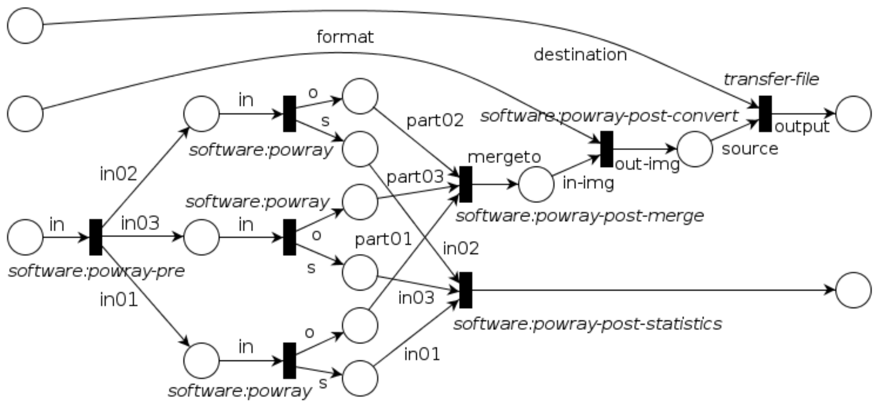

Figure 6. Representation of the POV-Ray example workflow for rendering a short animation of 24 frames. The tokens (with the number 24) represent the 24 input 3D-scenes, the output formats and the target destination for each image. Each input 3D-scene is splitted (*povray-pre*) into 3 POV-Ray concurrent jobs (*povray*). The partial images are merged afterwards (*povray-post-merge*). The *povray-post-convert* operation converts the rendered images into the desired format. Finally, the resulting images are transferred to the destination (*transferFile*).

stacks (such as Globus Toolkit, Condor, PBS, Web Services) and operating systems (Windows, Linux, Unix, Mac-OS).

The GWES interprets the GWorkflowDL supporting different abstraction layers of transitions or operations, respectively. The system is currently being used by a number of projects in various domains, such as bioinformatics, traffic management, flood forecasting, environmental risk analysis, and in enterprise resource planning. The licensing allows free use for scientific or educational purposes.

3.3 Interoperable Workflow Example

This section presents a real-world example workflow from the digital media production domain, which uses distributed installations of the *Persistence of Vision Ray-Tracer (POV-Ray)* [16], in order to speed up the rendering process for images or animations. POV-Ray creates three-dimensional, photo-realistic images using a rendering technique called ray-tracing. It reads in a text file containing information describing the objects and lighting in a scene and generates an image of that scene from the view point of a camera also described in the text file. In order to speed up the rendering process, the Grid workflow presented in this section splits up the rendering job into several sub-jobs which are either responsible for a certain image region (e.g., lines $161 - 320$ of an 480×640 image), or for a certain set of frames within an animation.

Fig. 6 depicts an example workflow for validating the interoperability between the INFN/CNAF WfMS (with focus on gLite) and the GWES (with focus on Globus Toolkit 4) using the extended GWorkflowDL. The workflow automates the rendering process for 24 images in Full-HD 1080p format (resolution 1080×1920), each of them splitted into three sub-processes (i.e., for image lines $1-360$, $361-720$, $721-1080$). So in total, this workflow could be distributed to maximum $24 \times 3 = 72$ concurrent Grid operations. The abstract workflow description itself is middleware-independent but contains all information required to find suitable service implementations for each workflow operation, such as the pre-processing step splitting the input data (*povray-pre*), the actual rendering step (*povray*), the two post processing steps for merging the results (*povray-post-merge* and *povray-post-statistics*) and the conversion of the final images (*povray-post-convert*). Finally, the images are transfered to a user-defined location by means of the operation *transferFile*. The methodology for mapping these abstract operations onto suitable Web Service method calls or remote/local program executions is left to the specific WfMS.

The focus of our work is to show the interoperable description and execution of workflows using the GWorkflowDL which we validate using the above example workflow. The discussion of the parallelization itself and the achieved speed-up (as well as the contents of the nice animation which we rendered) is beyond the topic of this paper.

4. Conclusions and Future Work

In the recent years, several workflow management systems have been established in the scientific Grid community, however, almost all of them are targeting a single Grid middleware and the workflow description languages are more or less bound to concrete execution platforms. In order to overcome this drawback, we extended the *Grid Workflow Description Language* (GWorkflowDL) by introducing an abstract operation element, which is middleware-independent, and can be interpreted and executed by several workflow management systems targeting on different Grid middleware.

We are currently validating this approach using an interoperable example workflow which automates the parallel rendering of a 3D animation using POV-Ray on two different target workflow management systems: the *INFN/CNAF WfMS* and the *Grid Workflow Execution Service* (GWES).

As future work we plan to implement a hierarchical approach, which eases the reusage of certain workflow regions as sub-workflows. Another topic which demands continuous research is the mapping of the GWorkflowDL from and to other workflow description formalisms and to enable the cross-delegation of workflow parts or whole workflows from one WfMS to another in a productive system.

Acknowledgments

Our work is supported in part by the European Union through the IST-2002-004265 Network of Excellence CoreGRID.

References

[1] W. van der Aalst. The application of Petri Nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[2] W. van der Aalst. Pi calculus versus petri nets: Let us eat humble pie rather than further inflate the pi hype. *unpublished discussion paper*, 2003.

[3] M. Alt et al. Using High Level Petri-Nets for Describing and Analysing Hierarchical Grid Workflows. In *Proceedings of the CoreGRID Integration Workshop 2005, Pisa*, 2005.

[4] A. Alves et al. Web Services Business Process Execution Language Version 2.0. OASIS, 2007.

[5] T. Andrews et al. Business process execution language for web services version 1.1. Technical report, BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems, 2003.

[6] M. Dumas and A. H. M. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. *Lecture Notes in Computer Science*, pages 76–90, 2001.

[7] I. Foster et al. The physiology of the grid: An open grid services architecture for distributed systems integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, jun 2002.

[8] Fraunhofer FIRST. Grid Workflow Execution Service home page http://www.gridworkflow.org/gwes, 2007.

[9] A. Hoheisel and U. Der. An XML-based framework for loosely coupled applications on grid environments. In P. Sloot, editor, *ICCS 2003*, number 2657 in Lecture Notes in Computer Science, pages 245–254. Springer-Verlag, 2003.

[10] A. Hoheisel et al. Documentation of the Grid Workflow Description Language toolbox. http://fhrg.first.fraunhofer.de/kwfgrid/gworkflowdl/docs/, 2005.

[11] A. Hoheisel and M. Alt. Petri Nets. In I.J. Taylor, D. Gannon, E. Deelman, and M.S. Shields, editors, *Workflows for e-Science – Scientific Workflows for Grids*, Springer, 2006.

[12] A. Hoheisel. Grid Workflow Execution Service – Dynamic and Interactive Execution and Visualization of Distributed Workflows. In *Proceedings of the Cracow Grid Workshop 2006*, Cracow, Poland, 2007

[13] K. Jensen. An introduction to the theoretical aspects of Coloured Petri Nets. In J. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency*, volume 803 of *Lecture Notes in Computer Science*, pages 230–272. Springer-Verlag, 1994.

[14] S. Pellegrini et al. A Practical Approach to a Workflow Management System. *Proceedings of the CoreGRID Workshop 2007*, Dresden, Germany, 2007

[15] F. Puhlmann1 and M. Weske1. Using the π-Calculus for Formalizing Workflow Patterns. *Lecture Notes in Computer Science*, pages 153-168, 2005.

[16] POV-Ray - The Persistence of Vision Raytracer. Website of the POV-Ray Project. http://www.povray.org/, 2006.