European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

Highly Dynamic Workflow Orchestration for Scientific Applications

Tomasz Gubala
gubala@science.uva.nl
Academic Computer Centre CYFRONET AGH
ul. Nawojki 11, 30-950 Kraków, Poland

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

CoreGRID Technical Report
Number TR-0101
July 31, 2007

Institute on Grid Information, Resource and Workflow Monitoring Services
and
Institute on Grid Systems, Tools and Environments

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

Highly Dynamic Workflow Orchestration for Scientific Applications

Tomasz Gubala
gubala@science.uva.nl
Academic Computer Centre CYFRONET AGH
ul. Nawojki 11, 30-950 Kraków, Poland

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

CoreGRID TR-0101

July 31, 2007

Abstract

Scientific applications executed with modern distributed technologies tend to be computed on various, dispersed resources, and workflows become a natural method of describing them. This method of the functional application decomposition allows for fully manual design only to a certain level of complexity. However, the scientific simulations usually expose non-trivial processing logic including multiple distinct processing elements connected with complicated control patterns. Thus an environment for semi-automated and assisted composition and orchestration of such application workflows is desired. This report describes the concept and design of such a platform that analyzes user requirements regarding application results and leads the user in the process of possible solution construction, dynamic refinement and execution. The tools forming the presented system use domain-specific knowledge and employ several levels of workflow abstractness in order to deliver the functionality in a more natural way for the human user. This work presents a real-life case study from the city traffic simulation domain, in order to introduce subsequent steps of workflow orchestration. It also supplies a discussion on existing similar systems for workflow orchestration and it finishes with a conclusions section.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

1 Introduction

1.1 Scientific Workflows on the Grid

The possibility to distribute computations performed by scientific applications among many computer resources allows to approach more computationally-challenging scientific problems. The Grid is a technology that facilitates such dispersed computation model based on resources situated in different administrative domains. This solution – combined with the novel service-oriented computing paradigm [16] – constitutes a powerful basis for modern application development and execution platforms. One concept of using this approach for distributed processing in the Grid is the idea to describe the higher-level application logic in terms of a workflow. This description involves naming the parts of the distributed processing and the interdependencies that occur between them (in the form of either a control or a data flow) [21]. Such an explicit expression of application's structure makes it possible for an artificial automaton to process it and to execute the application using some input data and possibly producing some output. Such an automaton is called a workflow engine and the workflow description becomes a language to program the engine.

This approach requires an additional important step in the entire process – the preparation of a workflow description. In the Grid environment this step is often done by an expert who is able to comprehend the workflow description notation and who can use it to assemble a new Grid application using the resources available in the environment. With the application of the service-oriented architecture (SOA) the resources are accessible as services providing well-defined functionality, which is usually available through some uniform protocol and interface (the WSRF framework is an example of such a uniform interface standard [5]). In these circumstances the basic building element of a workflow becomes a single service operation. Several operations invocations may be formed into a workflow by specifying the dependencies between them – usually in a form of a data transfer activity when an output of some operation becomes an input to another. The workflow execution engine is therefore responsible for the proper invocation of operations (it is in fact a generic client for the services) and brokering the intermediate data between services.

1.2 Properties of Scientific Workflows

The traditional process of applying the workflow technology in a distributed environment involves three, mostly sequential phases: first the discovery of matching resources, second the preparation of the workflow description and third the execution and the gathering of the results. In order to assist the person that assembles a new application there exist sophisticated registries where one may look for the basic building blocks of the application, which in terms of service-oriented computing are the services and their operations [20]. The research in the field of Semantic Grid concerning semantic description of Grid services will further improve the process of service search and discovery using domain-specific knowledge to assist a user [3]. This, however, involves the development of specialized tools that are able to use the additional knowledge provided by the registries to make the workflow composition task faster and possibly less erroneous. Tools like that are frequently referred to as workflow composition and orchestration tools. Together with the execution element they form an integrated toolkit for scientific workflows designers and users.

One important feature of a workflow language and the infrastructure that surrounds it is to support more than one level of abstractness in the resource description. The reason for that is the dynamism of the Grid environment – the resources available are not guaranteed and may be switched off or altered with time. Therefore if someone is using addresses (names) of such resources to build a workflow he risks the possibility that in the future the workflow will no longer be executable (or will yield errors instead of expected results). In order to overcome this obstacle the solution must support an abstract layer in resources' identification so the names (identifiers) used by the workflow designer will not become outdated due to changes in the environment. In case of the service approach in order to abstract away from the certain instances of services published in the Grid, one rather introduces a concept of an abstract service (or an abstract service operation) which could be dynamically mapped on the realization(s) that are available at the execution runtime and not at the workflow design time.

Another important property of a scientific workflow is its dynamic nature. The dynamism is required from the workflow orchestration infrastructure to:

- support the concept of multiple levels of workflow abstraction – the flexibility is needed to concrete an abstract workflow description at runtime;
- optimize the workflow execution with respect to given criteria – the dynamism allows for as-late-as-possible decisions to allow scheduling based on the most recent infrastructure information;

- introduce a certain level of fault-tolerance – the static workflow engine can not recover from a failure having no runtime decision units built-in;
- and to allow the user to modify the workflow during runtime, e.g., dependent on intermediate workflow results.

The reasons above are even more important in the case of scientific applications as they tend to be long-running processes with complex logic. The support of the workflow description language and the workflow engine for such more advanced processing patterns like loops and parallel branching is also a very important property [17].

This report describes a scientific workflow approach that fulfils the properties described above and provides an integrated platform for using scientific workflows in the Grid environment. The document starts with a detailed top-down explanation of the system, along with the applied concepts and techniques to provide the functionality, followed by a comparison of our approach with other scientific workflows systems. The work finalizes with conclusions.

2 Workflow Orchestration

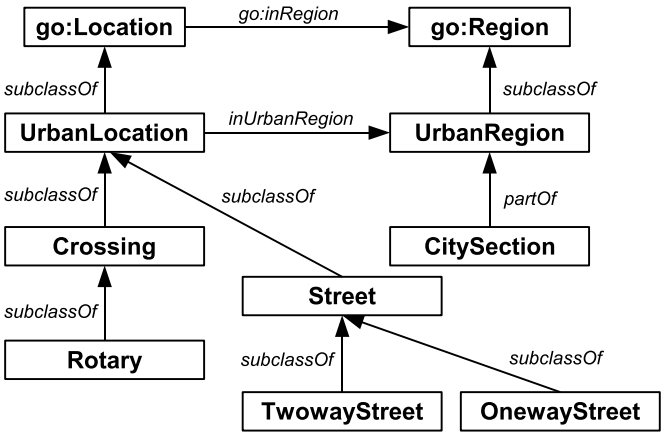

Figure 1: Partial snapshot of the example domain.

This chapter describes the whole workflow orchestration process, beginning with the specification of the user request (Section 2.1), continuing with the composition of abstract workflows that provide suitable solutions (Section 2.2), and ending with the refinement and the dynamic execution of the workflows on available Grid resources (Section 2.3).

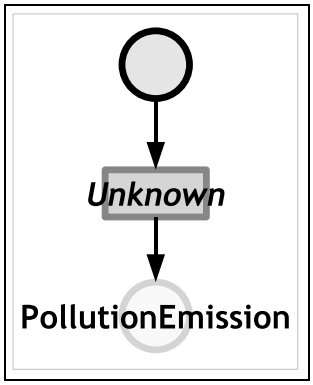

Figure 2: Initial workflow as a problem specification.

2.1 Specification of the User Request

As the workflow orchestration and execution toolkit forms a part of a larger problem solving environment, the problem specification phase is the starting point for the user. The requirement is to provide the user with a tool that assists him with the definition of the problem to be solved but that is not too constraining at the same time. The first step to the fulfilment of this requirement is to make the assistant tool *use the same language* as the user does. This involves applying of domain-related knowledge so the expert is able to specify the problem with the concepts and terms used in his area of expertise [10]. For the purpose of demonstration we use the domain of the city traffic pollution analysis (an application from that field will be also the example we refer to throughout this work). Fig. 1 shows a part of the taxonomy containing concepts and relations among them. The relations include the frequently used general relations of hyponymy (specialization) and meronymy (inclusion).

As the infrastructure needs a proper tool to design, store and use explicitly defined domain knowledge, we use the idea of knowledge transcribed with ontologies. The formal method of storing the vocabularies and taxonomies in ontologies allow for artificial systems to use it while the high level of design freedom helps the ontology modeler to precisely express the facts. However, this kind of domain-specific knowledge needs specialized repositories that are able not only to store it but also to publish it with adequate query-response mechanisms. We use the concept of Grid Organization Memory to this end [13]. This technology applies the Semantic Web standards (OWL Web Ontology Language and RDQL query language) to provide a higher-level access to the ontology-transcribed knowledge to the other tools of the infrastructure. The ontology store is accessible and dynamically alterable by the domain expert that models application-related knowledge. While this person is required to understand the concept of ontologies and description logics it is important not to ask that kind of expertise from an every-day user.

Finally, in order to facilitate the easy use of the toolkit we apply the knowledge-based assistance technique to lead the user through the problem specification phase. The assistant tool employs the context-based advice system that is able to use both structured (i.e. expressed through ontology) knowledge and free-text user notes in order to propose the possible results the user may obtain using the workflow system [14]. After the user finally decides on the type of problem he wants to tackle with, the initial form of workflow sketch is generated.

Fig. 2 shows an example of an initial workflow. In this case the user asks for the results of the pollutant emission due to the city traffic. The workflow that is generated states what kind of outcome the user expects from the workflow execution (the bottom circle in the diagram). The following section brings the description of the used formalism and notation to define workflows.

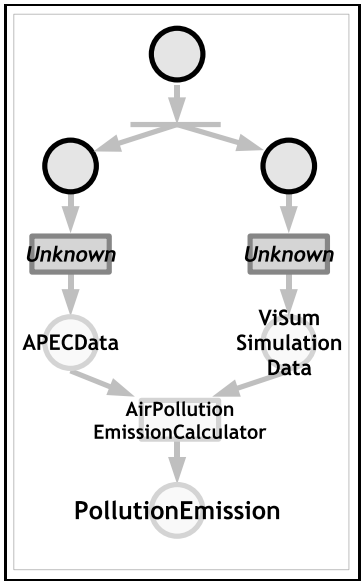

Figure 3: An example of workflow after one step of composition.

2.2 Abstract Solution Composition

The language for the description of workflows which we use within the examples in this paper is based on the Petri Nets formalism [12]. A typical Petri net has a set of transitions (visualized as rectangles or bars) and a set of places (depicted with circles) with possible directed connections that go from a place to a transition or vice versa. The execution mechanism uses the notion of tokens that reside in places and that drive the workflow with the general rule that a transition *fires* by consuming one token from each input place and producing one token in every output place. Absence of a token in any of the incoming places precludes the execution of the transition. In our case, the transitions are related to service operations (method calls) and the tokens to the workflow data. Both, operations and data, may possess several levels of abstractness. In order to support these multiple levels of abstractness in the workflow description we apply the idea of the High-Level Petri Nets (HLPN). With this approach we denote the additional information on the current level of abstractness of a workflow element with its color. In the Fig. 2 one may see the light bottom place which denotes an abstract but semantically identified data. This means that it is known what kind of data may be produced there while there is no real data at the moment. The darker gray transition above the place is an unknown part – the element we know nothing about except the fact that is should eventually yield *PollutionEmission* result. This construction with the data type of not known producer defined inside a workflow is called an unsatisfied dependency. The next tool involved in the process of workflow composition is responsible for turning such unknown parts into a proposed solution in the form of a workflow.

For every dependency that needs to be resolved the tool contacts the ontological registry in order to find suitable service operations that may produce the required result. The registry is therefore another source of knowledge (besides the domain-specific vocabularies mentioned in Sect. 1) for the workflow orchestration environment [2]. The services are described in an ontological form with statements regarding the service operations' inputs, outputs, preconditions and effects (the IOPE set). Through these notions the tool that composes workflows is able to match different operations into a workflow. By associating the required data with the produced output the tool constructs a data flow between operations (it also uses a specific notion of effect that may bind two operation together with non-data dependency).

As may be noticed in Fig. 3, a newly introduced operation (the light-grey bottom transition) is described in an abstract form. It specifies an operation type in a meaningful way for a domain expert and identifies a certain operation

class. An operation class is a logic set of all the operation instances that implement (or realize) given functionality. Therefore a single operation class may also be understood as a counterpart to an *interface* in the object-oriented design. This level of abstraction of a workflow ensures that the application logic is valid (as the correct data flow is secured) while it lacks any implementation-specific details, such as specific service endpoints addresses. This feature makes the workflow description invulnerable to the frequent changes in the Grid environment and allows for the workflow reuse capability. Another property of the operation is a set of two input data. The composition assistant using the semantic description of the operation may identify the necessity for further inputs to be provided – every such an input constitutes another unsatisfied dependency that needs to be resolved. Following this reverse traversal approach the tool composes the application workflow. It also uses workflow reduction algorithms and searches for the suitable solutions internally in the workflow in order to provide useful control patterns like loops and alternative or parallel branching. For more detailed description of the tool please refer to [11].

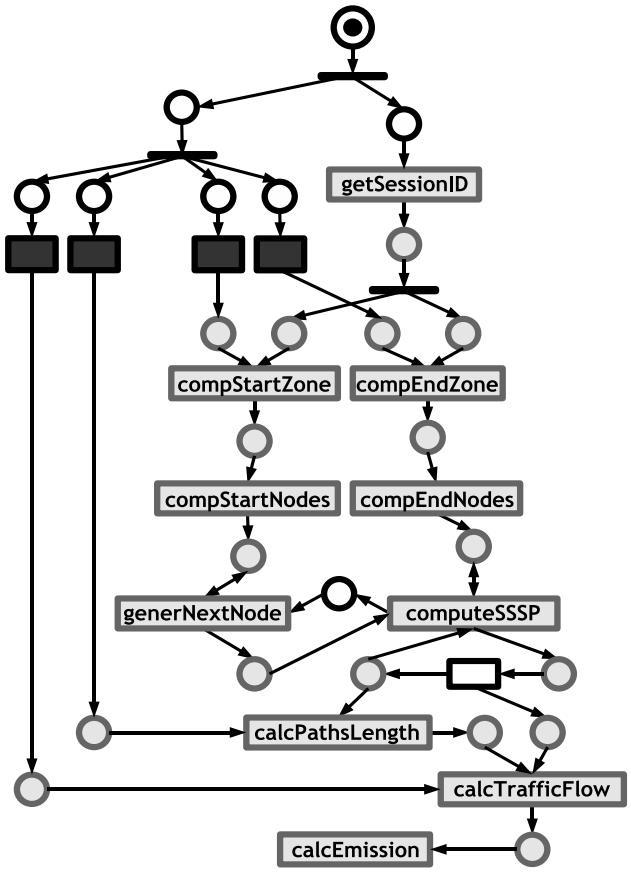

Figure 4: The application workflow after full composition.

The Fig. 4 shows the example of city traffic pollution analysis workflow after the full composition phase. It involves several Grid service operations some of which form the core simulation. There are also parallel executions and a loop involved as the application iterates in order to analyze the possible traffic scenarios. The final pollution emission calculator uses provided scenario with pollution-related data to result with the prediction of the pollution emission in a given case. The workflow describes the experiment on an abstract level and needs proper refinement and execution mechanism in order to be successfully completed. The overall time needed for construction of a workflow of this level of complexity is around 3-4 seconds. To compose larger workflows (not pictured here of complexity reasons) that count around 35-40 operations and several loops takes around 10 seconds what we claim is rather acceptable result.

2.3 Solution Refinement and Dynamic Execution

The result of the *abstract solution composition* described in the last section is an abstract workflow that needs to be mapped onto available resources in order to be executed. Therefore, our approach involves several subsequent workflow refinement steps, which are displayed in Fig. 5.

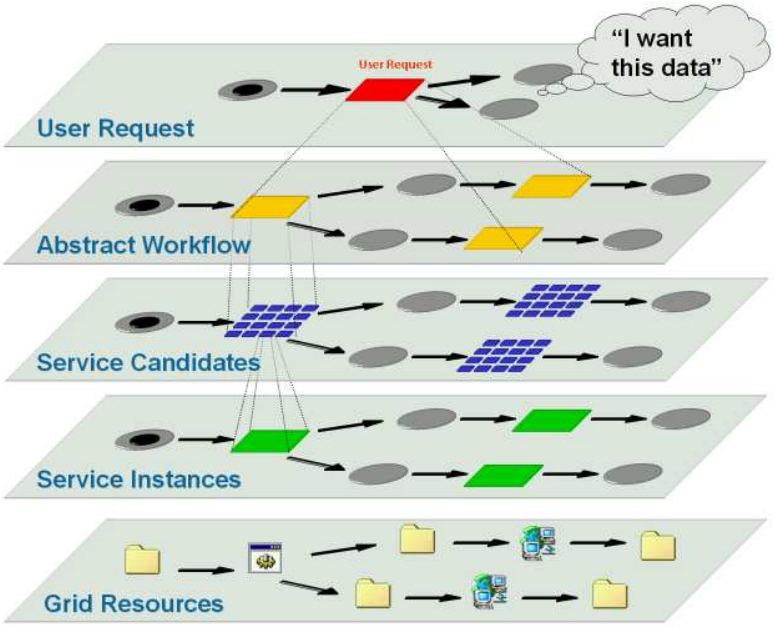

Figure 5: The workflow refinement process.

After the composition of the abstract workflow, the next step is to map the abstract nodes onto matching services. This is done iteratively during the processing of the workflow. Each time the workflow engine reaches a transition related to an abstract (non-executable) operation, it calls a special workflow refinement service. This service refines the workflow description by searching for matching service candidates, which fulfill the requirements defined by the profile of the abstract nodes [8]. The decision of whether a service matches the requirements is done by rules that depend on several properties, such as functionality (e.g., service produces certain class of output data or side effect), performance (e.g., operation should complete within $1h$), or reliability (e.g., only services which have been operational during the last $72h$ should be taken into account). If it is possible to find matching service candidates, the refinement service attaches a list of the corresponding interface descriptions URLs (e.g. wsdl URLs) to the abstract transition.

The next step of the workflow refinement process consists of the selection of one service instance out of the list of available service candidates. In order to optimize this selection, the system uses a HEFT scheduling algorithm [19], which takes into account the recorded as well as the current monitoring information about the services and the Grid infrastructure. Each time the workflow engine reaches an abstract (non-executable) transition that contains list of service candidates, it invokes a scheduling service that selects one of the services in the list. In long-running workflows it makes sense to refine only the next workflow transitions that are ready to be invoked. This kind of scheduling is also referred to as *deferred planning* [7]. In short-running workflows or in cases where it is necessary to have some kind of co-allocation it is also possible to select service instances for all workflow nodes at once. In the case of stateful services, such as WSRF, several subsequent workflow operations often need to be invoked on the same service instance (e.g, $initialize \rightarrow simulate \rightarrow getData$). This is considered during the refinement process by annotating the workflow with *instance group labels*, which are taken into account during the scheduling process.

Each time the workflow engine detects an activated transition related to a concrete (executable) operation, it uses the corresponding lower-level Grid middleware in order to allocate the resources and to invoke the operation. The prototype implementation currently supports pure Web Service operations (using Axis), WSRF Grid Service operations, WS-GRAM program executions, and Reliable File Transfers (RFT) (using Globus Toolkit 4). Further Grid middleware can be supported by simply providing a specific Java class, which extends the abstract *Activity* class within the

workflow engine or by providing a Web Service interface.

The Petri Net formalism used in the prototype implementation makes it relatively easy to implement the workflow interpreter, as the workflow description just contains four different types of main elements: transitions, places, arcs and tokens. The workflow engine cycles through the graph and searches for activated transitions that are ready to fire, and dynamically delegates the workflow refinement to external services or the user. The workflow logic is inherently expressed by the structure of the Petri Net, so the workflow engine does not need to explicitly take care about specific workflow constructs, such as loops, choices, and the sequential or parallel execution of tasks. All workflow abstraction levels are described by the same workflow description language; so it is possible that one single workflow contains abstract as well as concrete workflow parts at the same time. This enables the workflow orchestration system to be highly dynamic and to react on changes in an unreliable Grid environment.

As the current state of the workflow is stored within the tokens of the Petri Net (called *marking*), it is possible to save the state by just writing the workflow description to a file or to an XML database. In our prototype implementation, we use an eXist XML database in order to store workflow checkpoints which can later be restored or migrated to another workflow engine. During the workflow processing the workflow engine annotates the workflow document with further runtime properties, such as performance information and fault messages, which are later automatically analyzed to gather knowledge for the future orchestration of similar workflows.

3 Comparison with other Scientific Workflow Systems

There are currently several scientific workflow systems that are established in the Grid community. While one may compare them with regard to many various criteria we would like to concentrate on the level of dynamism of the systems and their support for different layers of abstraction, as those are qualities that are particularly addressed in our approach. The working environments offered by Triana [4] and Taverna [15] systems bind workflow description documents directly with the realizations of their building elements. As the direct reference model offers the possibility of faster execution it lacks the needed level of flexibility in order to overcome possible changes in the pool of available resources. The information reuse in these systems is reduced in favor of the more transparent execution process what makes them fast application prototyping platforms.

In the Kepler [1] system there is an intermediate level of a single workflow element description. An actor is described in an abstract way through the ports that surround it and may be further implemented as a number of different realizations (including remote execution suitable for the Grid). However, as the decision on how to concrete an actor is taken at design time, only a single element of processing is a subject of possible reuse. The execution mechanism does not possess a dynamic refinement/scheduling ability.

Another group of systems (Askalon [9], Pegasus [6]) introduce, apart from their own workflow notation, a special language to describe the workflow in a more abstract way. When a user supplies this kind of description it is automatically translated into a concrete workflow realization containing all the execution details needed to carry the processing on. The advantage of this approach is that one may easily reuse the abstract workflows and thus the tool is flexible enough to work in a frequently changing environment. However, the fact that the whole translation process takes place before the execution results in a static scheduling and refinement approach – the decisions regarding the entire processing are taken before it starts. This may lead to runtime faults especially with long-running applications that are frequent in the scientific community.

A similar approach provides Karajan [18], a parallel scripting language, which is able to express abstract workflows, which then are bound onto the specific services or protocols during runtime. The scripting language itself, however, is rather complex, and there are no semantic methods that assist the users in orchestrating the workflows.

4 Conclusions

This report presents the Grid scientific workflow system that is meant to assist the user to compose, refine and execute an application in a distributed environment. The most important features of the system we would like to stress are:

- *multiple abstractness levels* that help to express the application logic on sufficiently high level for the end user to comprehend it and for the workflow to be reusable,
- *dynamism* that allows the runtime refinement of the workflow elements that are invoked and the lazy scheduling strategies that use the most recent information available,

- *semantic driven* approach in workflow composition and refinement in order to keep the workflow description legible for the human expert and also to introduce certain level of automation,
- completeness provided with covering every step of workflow creation, enactment, storing and reuse within the same integrated environment.

As the presented system covers the stages of problem specification (Sect. 2.1), solution generation (Sect. 2.2) and dynamic execution (Sect. 2.3) it forms an important part of a semantic-aware problem solving environment for scientific applications. The approach was tested with two pilot applications, including a flood scenarios simulation and the traffic pollution emission calculations.

Acknowledgements

The work described in this paper is supported in part by the European Union through the IST-2002-004265 Network of Excellence CoreGRID and the IST-2002-511385 project K-WfGrid.

References

[1] S. Bowers, B. Ludaescher, A.H.H. Ngu, and T. Critchlow. Enabling scientific workflow reuse through structured composition of dataflow and control-flow. In *IEEE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow)*, 2006.

[2] M. Bubak, T. Gubala, M. Kapalka, M. Malawski, and K. Rycerz. Workflow composer and service registry for grid applications. *Future Generation Computer Systems*, 21(1):79–86, 2005.

[3] M. Cannataro and D. Talia. Semantics and knowledge grids: Building the next-generation grid. *IEEE Inteligent Systems*, 19(1):56–63, 2004.

[4] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang. Programming Scientific and Distributed Workflow with Triana Services. *Concurrency and Computation: Practice and Experience (Special Issue: Workflow in Grid Systems)*, 18(10):1021–1037, 2006.

[5] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. From open grid services infrastructure to ws-resource framework: Refactoring & evolution, 2004.

[6] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Metha, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing*, pages 25–39, 2003.

[7] E. Deelman, T. Kosar, C. Kesselman, and M. Livny. What makes workflows work in an opportunistic environment? *Concurrency and Computation: Practice and Experience*, 18(10):1187–1199, 2005.

[8] L. Dutka, Sz. Natanek, and J. Kitowski. Automatic application builder - tool for automatic service selection. In *5th Cracow Grid Workshop (CGW05) – Workshop Proc.*, page to appear. ACC Cyfronet AGH, 2006.

[9] T. Fahringer, S. Pllana, and A. Villazon. A-gwl: Abstract grid workflow language. In *Int. Conf. on Computational Science (ICCS)*, pages 42–49. Springer, 2004.

[10] C. Goble and D. de Roure. The grid: an application of the semantic web. *ACM SIGMOD Record – Special section on semantic web and data management*, 31(4):65–70, 2002.

[11] T. Gubala, D. Harezlak, M. Bubak, and M. Malawski. Semantic composition of scientific workflows based on the petri nets formalism. In *The 2nd IEEE International Conference on e-Science and Grid Computing*, page 12. IEEE Computer Society Press, 2006.

[12] A. Hoheisel and M. Alt. Petri nets. In Ian J. Taylor, Dennis Gannon, Ewa Deelman, and Matthew S. Shields, editors, *Workflows for eScience*. Springer, 2006.

[13] B. Kryza, R. Slota, M. Majewska, J. Pieczykolan, and J. Kitowski. Grid organizational memory – provision of a high-level grid abstraction layer supported by ontology alignment. *Future Generation Computer Systems*, 23(3):348–358, 2007.

[14] M. Laclavik, E. Gatial, Z. Balogh, O. Habala, G. Nguyen, and L. Hluchy. Experience management based on text notes (embet). In *Proc. of eChallenges 2005 Conf.*, 2005.

[15] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics Journal*, 17(20):3045–3054, 2004.

[16] M.P. Singh and M.N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. Wiley, 2005.

[17] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.

[18] G. von Laszewski, M. Hategan, and D. Kodeboyina. Java CoG Kit Workflow. In I. Taylor, D. Gannon, E. Deelman, and M. Shields, editors, *Workflows for eScience*. Springer, 2006.

[19] M. Wieczorek, R. Prodan, and T. Fahringer. Comparison of workflow scheduling strategies on the grid. In *6-th Int. Conf. on Parallel Processing and Applied Mathematics PPAM'2005*, volume 3911, pages 792–800. Springer-Verlag LNCS, 2006.

[20] S.C. Wong, V. Tan, W. Fang, S. Miles, and L. Moreau. Grimoires: Grid registry with metadata oriented interface. *IEEE Distributed Systems Online*, 6(10), 2005.

[21] J. Yu and R. Buyya. A taxonomy of scitific workflow systems for grid computing. *Special Issue on Scientific Workflows, SIGMOD Record*, 34(3), 2005.