Using UNICORE and WS-BPEL for Scientific Workflow Execution in Grid Environments

G. Scherp[1], A. Höing[2], S. Gudenkauf[1], W. Hasselbring[1], O. Kao[2]

[1] OFFIS Institute for Information Technology, R&D-Division Energy, Escherweg 2, 26121 Oldenburg, Germany
email: [stefan.gudenkauf, guido.scherp]@offis.de, wha@informatik.uni-kiel.de

[2] Technische Universität Berlin, Faculty IV - Electrical Engineering and Computer Science, Dept. of Telecommunication Systems, Complex and Distributed IT Systems, Einsteinufer 17, 10587 Berlin, Germany
email: [andre.hoeing, odej.kao]@tu-berlin.de

Abstract. Within the BIS-Grid project[3], a BMBF-funded project in the context of the German D-Grid initiative, we developed the BIS-Grid workflow engine that is based upon service extensions to UNICORE 6 to use an arbitrary WS-BPEL workflow engine and standard WS-BPEL to orchestrate stateful, WSRF-based Grid services. Although aimed at proving the feasibility of applying Grid technologies for business information systems integration, we illustrate that this engine is also well-suited for scientific workflow execution, making standard WS-BPEL-based tooling accessible for scientific workflows.
In this paper, we describe using the BIS-Grid engine for the execution of scientific workflows. This includes a differentiation of scientific and business workflows in general and an analysis of the suitability of the BIS-Grid infrastructure to execute scientific workflows. We propose reusable WS-BPEL patterns for typical scientific workflow activities whereas job submission is focused. Finally, we prospect our future work.

1 Motivation

Modern Grid middlewares such as UNICORE 6[4] are based on the Web Service Resource Framework (WSRF) [5], a standard that extends classical, stateless Web services to be stateful. Like Web services, WSRF-based Web services, also called Grid services, can be orchestrated to form complex workflows that itself are provided as services by utilizing the Web Service Business Process Execution Language (WS-BPEL). Although originally developed for service orchestration in the business domain, WS-BPEL gained much attention from scientific communities to be adopted for the design and execution of scientific workflows.

[3] This work is supported by the German Federal Ministry of Education and Research (BMBF) under grant No. 01IG07005 as part of the D-Grid initiative.

[4] http://www.unicore.eu

[5] http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf

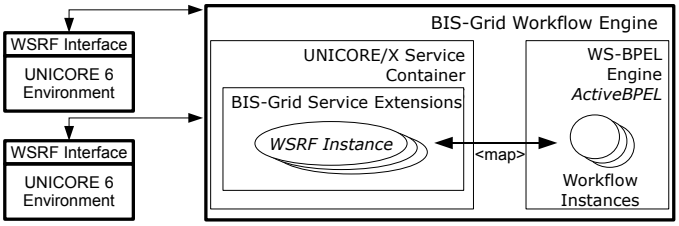

Fig. 1. Overview of architecture of the BIS-Grid workflow engine.

Within the BIS-Grid[6] project, a BMBF-funded project in the context of the German D-Grid initiative, we developed the BIS-Grid workflow engine that is based upon service extensions to UNICORE 6 to use an arbitrary WS-BPEL workflow engine and standard WS-BPEL[7] to orchestrate Grid services. These service extensions act as a WSRF proxy to the functionalities of the original WS-BPEL engine and to the deployed WS-BPEL workflows itself, cp. Fig. 1, providing a Workflow Management Service for workflow deployment and a generic Workflow Service for workflow execution and monitoring. For a more in-depth view on the architecture of this engine, see [10]. Although originally aimed at proving the feasibility of applying Grid technologies for the integration of business information systems, this engine is also suited for scientific workflow execution, making standard WS-BPEL-based tooling accessible for scientific workflow execution. To hide workflow complexity from the scientific user, we propose reusable WS-BEL patterns for typical scientific workflow activities such as job submission and data transfers instead of specific language extensions.

The paper is organized as follows. Related work is discussed in Sec. 2, followed by a short overview of the differences between scientific and business workflows in Sec. 3. Section 4 discusses the principal requirements for scientific workflow execution and how they are addressed by BIS-Grid. The use of WS-BPEL for scientific workflows including our WS-BPEL pattern is shown in Sec. 5 by the example of job submission. Section 6 provides an outlook on our future work, and Sec. 7 provides a conclusion.

2 Related Work

The Chemomentum project already provides workflow extensions for UNICORE 6, consisting of two UNICORE 6 service containers. The first represents a workflow engine that processes workflows on a logical level, the second represents a service orchestrator that transforms so-called Work Assignments into jobs, given in the Job Submission Description Language (JSDL) [1]. Both, this UNICORE 6 workflow system and the BIS-Grid engine, are implemented as service extensions to

[6] http://www.bisgrid.de
[7] I.e., we did not modify nor extend the WS-BPEL language.

the UNICORE 6 service container. However, the UNICORE 6 workflow system does not support the integration of a WS-BPEL workflow engine.

Akram et al. [2] identify requirements for scientific workflows – namely modularity, exception handling, mechanisms compensation/recovery, adaptivity and flexibility, and workflow management – by the example of a protein crystallography workflow. They also describe how the BPEL language addresses these requirements, and the shortcomings of BPEL for scientific workflows. Most prominently, these are the limited adaptivity regarding workflow modifications at runtime, the lack of support for user interactions by the BPEL specification, and the need to wrap non-portable engine-specific workflow management capabilities using appropriate standards in order to use them in a portable manner.

Regarding the use of BPEL for Grid service orchestration, Leymann proposes BPEL4WS[8] as foundation since it already fulfills many requirements of the WSRF standard [12]. The appropriateness of BPEL is also examined and confirmed in [5], [6], [7], [8], and [14]. These works mainly focus on scientific workflows and, except for Ezenwoye et al. [8], rely on extending or adapting BPEL, thus creating dialects.

The execution of jobs with WS-BPEL is also discussed in [15] in which a two-stage approach is proposed. In the first stage a base flow is modeled to define job execution, supplemented by a JSDL job description and a fault-handling policy based on WS-Policy[9]. This base flow is expanded automatically in the second stage by additional WS-BPEL fault-handling activities corresponding to the respective fault-handling policy. The execution of the workflow is based on two further non-WS-BPEL services, a job proxy to encapsulate job execution and to receive notification messages from a scheduling system, and a fault-handling service to apply extended fault-handling strategies such as workflow instance migration. The approach was implemented and tested on IBM software.

In [16], Zhao et al. present a visual tool that abstracts a typical sequence of BPEL activities for scientific computing to a new single activity. This sequence comprises steps like *submitTask* or *getTaskStatus* and looks slightly similar to the job submission workflow we present in Sec. 5. However, Zhao et al. focus on visual complexity in the workflow editor. Before workflow deployment, the proprietary code is translated to standard WS-BPEL.

3 Scientific vs. Business Workflows

A comparison between scientific workflows and business workflows is the topic of several publications – directly or indirectly, as, for example, in [2] and [3]. Thus, we will not present a complete comparison but focus on the principal differences of scientific and business workflows, see Tab. 1.

[8] BPEL4WS 1.1 is the predecessor of WS-BPEL 2.0.
[9] http://www.w3.org/Submission/WS-Policy/

Table 1. Scientific vs. business workflows.

Scientific workflows	***Business workflows***
Data-driven Control flow is implicit; an activity starts when the required input data is available.	*Control-driven* Data flow is implicit and data is manipulated when the corresponding activity is executed in the control flow.
User-centric Rights are often associated to persons (scientists) directly. The workflow designer is often also the workflow executor (does not necessarily hold for e-Science).	*Role-centric* Rights are associated to roles that are associated to persons. Elaborate role models for workflow participants/stakeholders.
Voluminous data handling Data handling often requires third-party transfers (data transfers between two remote servers that are initialized by a local client).	*Information handling* Workflow data is usually small, regarded as *information* and is stored in process variables during workflow execution.
Experiment implementation Monitoring is of great importance, especially for intermediary results of a workflow. Workflows tend to evolve quickly as knowledge on the domain/workflow is collected (cp. [3]).	*Service provisioning* Workflows must be guaranteed to complete and provide results as advertised to and contractually agreed with customers [3].

4 BIS-Grid Engine for Scientific Workflows

As depicted in Sec. 2, WS-BPEL becomes more and more important for the scientific community regarding the execution of scientific workflows in Grid environments. Modern Grid middlewares such as UNICORE 6 and Globus Toolkit 4 provide their functionalities – for example, data transfer and job submission – as (WSRF-based) Grid services. Scientific workflows that build on these Grid middlewares must be described as an ordered invocation of such Grid services. WS-BPEL, as the de facto standard for Web service orchestration, comes naturally in mind for Grid service orchestration, although originally being designed for business workflows.

Per se, WS-BPEL has some shortcomings that constrains its usability for scientific workflows. Originally, the language has been designed to orchestrate stateless Web services. Since data transfer and job execution, for example, have state, the WSRF standard was developed to enable stateful Web services (Grid services). Consequently, Grid service invocations are much more complex than standard Web services – a WSRF service instance has to be created, is used, and finally must be destroyed. To address this, we developed appropriate WS-BPEL patterns for Grid service invocations for the Grid middlewares UNICORE 6 and Globus Toolkit 4 [4, 11]. Further shortcomings do not originate from the WS-BPEL language directly but from the workflow execution environments and workflow design tools. Available WS-BPEL workflow engines, open source or commercial, are not fully interoperable with the security features of existing Grid middlewares. Such features are, for example, the support of SAML assertions [13] to present additional signed security tokens (as roles), or the support of the Grid Security Infrastructure (GSI) of Globus Toolkit 4 infrastructures that rely on proxy certificates. Furthermore, available WS-BPEL design tools are usually not suitable for scientific users because the applied workflow model is close to the technical WS-BPEL language. Considering this, scientific users require a workflow model fitting to their domain. For example, scientific users may need

to run several computations on selected data in a specific order by dropping boxes (computations) onto a workbench and drawing lines (data flow) between them. Prospects on these issues are presented in Sec. 6.

Table 2 presents an overview of the principal requirements we identified for scientific workflows, and presents which of them are addressed by the WS-BPEL language and tooling, or by the BIS-Grid engine. As shown, RQ-1 is met by WS-BPEL itself. Regarding monitoring (RQ-2), the BIS-Grid engine supports mechanisms that base upon the propagation of the monitoring capabilities of the internal WS-BPEL engine by the UNICORE 6 layer, see Fig. 1, [10], and [11]. Upon these, advanced capabilities such as pull- or push/notification-based monitoring of workflow activities can be implemented. Design-time error handling and compensation (RQ-3 and RQ-4) are met by WS-BPEL, while run-time error handling and compensation are matters of the workflow execution environment. While these are important issues to be addressed in operational execution environments, they are not focused in BIS-Grid and regarded as underlying the actual workflow execution engine. Regarding RQ-5, BIS-Grid relies on the the Netbeans IDE[10] and it's BPMN-oriented visual DSL (Domain-specific language). Together with appropriate WS-BPEL patterns, this represents an abstraction from the technical workflow implementation that is comfortable for general purpose workflow design both for business workflows as well as scientific workflows, and provides a basis for further domain-specific abstraction above the WS-BPEL pattern layer. RQ-6, voluminous data transfers, is addressed in the following sections in terms of *file staging*.

5 WS-BPEL Job Submission Pattern for Scientific Workflows

Scientific workflows often are realized as (batch) jobs that can be defined as non-interactive computational tasks that are intended to be executed on high-performance computing (HPC) systems. Grid middlewares typically support the submission of such jobs to an HPC system by utilizing the local batch system, but modern Grid middlewares also provide means for exposing their functionalities as Grid services, thus enabling service orchestration. Grid services such as job submission services support further standards like the Job Submission Description Language (JSDL) or OGSA Basic Execution Services (OGSA-BES)[11]. Using job submission services results in almost generic sequences of Grid and/or Web service invocations[12] that can be encapsulated in a generic and configurable WS-BPEL (sub-)workflow. Our BIS-Grid engine allows to execute such a workflow in Grid environments whereas the workflow itself is provided as a Grid service. This facilitates reuse in higher-level service orchestrations. In [4] we identified WS-BPEL patterns for orchestrating Grid services using standard

[10] http://www.netbeans.org/features/soa/index.html
[11] http://www.ogf.org/documents/GFD.108.pdf
[12] For most jobs, these sequences are almost identical, cp. [16].

Table 2. Principal requirements of scientific workflows.

RQ	*Requirement description*	*Compliance*
RQ-1	*Modularity and composability.* Although evolving in nature, scientific workflows often base upon recurring standard activities. Regarding different levels of abstraction, providing sub-workflows as standard activities of superior workflows facilitates reuse and maintenance. Separating workflows in different parts facilitates the scalability of workflow execution.	WS-BPEL is composable by nature.
RQ-2	*Monitoring.* Scientific workflows are often long-running; progress monitoring and the inspection of intermediary results is therefore an important issue.	The BIS-Grid engine supports basic monitoring of workflow state. WS-BPEL `EventHandlers` improve simple execution state monitoring.
RQ-3	*Error handling and fault tolerance.* The long-running nature of scientific workflows causes interruption due to errors to be regarded as highly undesirable. Mechanisms should ideally address design-time (error handling for forseen events) and run-time (fault tolerance).	WS-BPEL provides mechanisms for error handling at design time.
RQ-4	*Adaptability.* As the underlying resource infrastructure may change during workflow execution, workflow adaptability is regarded as desirable. Mechanisms should ideally address design-time (compensation for forseen events) and run-time.	WS-BPEL provides mechanisms for compensation at design-time.
RQ-5	*Domain-specificity.* Often, scientists are not only the users of scientific workflows but also their designers. This requires adequate domain-specific modeling while technical details should be concealed as far as possible.	BIS-Grid uses Netbeans IDE for workflow design, using it's BPMN-like visual DSL and appropriate WS-BPEL patterns [4, 11] to abstract from technical workflow implementation.
RQ-6	*Voluminous data transfers.* Scientific computations are often based on voluminous data. This data has to be transferred to the respective computing resources.	Generally, third party transfers can be modeled and executed with BIS-Grid. JSDL-compliant Grid middlewares provide file staging mechanisms as defined by JSDL.

WS-BPEL. Based on this pattern we developed a WS-BPEL pattern to encapsulate Grid service invocations for job submission to UNICORE 6. A description of this job submission pattern is presented in Tab. 3.

Job submission to UNICORE 6 consists of several phases that are described briefly in the following. Additionally, Fig. 2 illustrates the corresponding workflow. Please note that we omitted exception handling and compensation for the sake of clarity. The current signature of the job submission workflow is described in Tab. 4. At least, the endpoint to a UNICORE 6 target system and the JDSL Job description is mandatory as input. At the moment, the output is a job failed/succeeded message. This workflow itself can be used in high level workflows in which a resource broker is invoked to choose a target system with least load before submitting a job.

1. `Job Submission Receive`: A JSDL job description and configuration parameters are received and stored in process variables.
2. `Target System Service Instance Create`: A Target System Service instance is created via the default factory instance of the Target System Factory Service.
3. `Target System Service Instance Submit Job`: The JSDL job description is submitted to the Target System Service instance which creates a Job Management Service instance.

Table 3. Job Submission Pattern.

	Pattern description
Motivation	Scientific workflows often are designed as non-interactive computational tasks in the form of (batch) jobs. Regarding reuse, there is the need to integrate such jobs in workflows that are designed on a higher level of abstraction than jobs.
Intention	Define a workflow that executes a (batch) job on a UNICORE 6 installation by using the target system service and it's job management service to submit and start a job and its respective data, and to retrieve the job's outcome upon completion.
Behavior	See Figure 2.
Participants	The invoker of the job submission workflow, the Target System Factory Service, the Target System Service, and the Job Submission Service.
Consequences	(1) Job submission is encapsulated in a workflow using an appropriate workflow description language. This facilitates the reuse of existing jobs on a higher level of abstraction and reduces submission errors and redundant user-triggered submissions by reuse. (2) Workflow reuse also simplifies the protocol of high-level workflows and abstracts from submission details. When the workflow language allows hierarchical composition, as WS-BPEL, the job submission may be described with the same language as the high-level workflows.

4. `Job Management Service Instance Start Job`: The Job Management Service instance is used to start job execution.
5. `Job Management Service Instance Retrieve Result`: The status of the job execution is fetched periodically from the Job Management Service until the job is completed or failed, afterwards the job result is stored in a process variable.
6. `Job Management Service Instance Destroy`: The Job Management Service instance is destroyed.
7. `Target System Service Instance Destroy`: The Target System Service instance is destroyed.
8. `Job Result Reply`: The job result is returned.

Table 4. Signature of the job submission workflow.

Name	***Type***	***Description***
	Input parameters	
TargetSystem (mandatory)	wsa:EndpointReferenceType	The endpoint to the Target System Factory Service that should be used to create the Target System Service instance.
JSDL (mandatory)	jsdl:JobDefinition_Type	The job description according to the JSDL standard.
LifetimeIntervall (optional)	xsd:duration	Maximum runtime of the job execution. If omitted, a default lifetime is used.
WaitInterval (optional)	xsd:duration	Waiting time between two job execution status retrievals. If omitted, a default waiting time is used.
	Output parameters	
Result	xsd:string	The result (successful or failed) of the job execution.

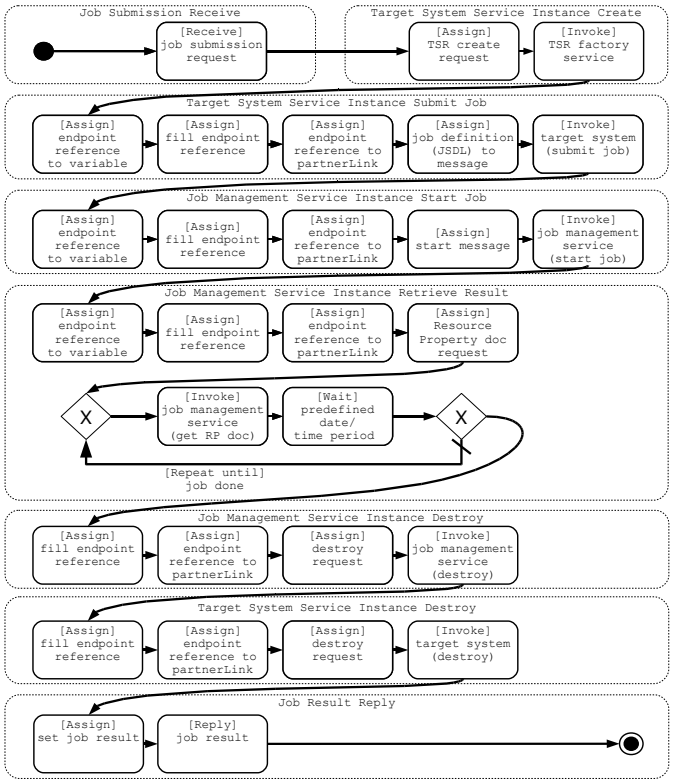

Fig. 2. Job submission process.

The JSDL standard includes the description of file staging (RQ-6) to initiate file transfers before (stage-in) and after (stage-out) the actual job execution. This mechanism can be used to transfer input and output data (often several gigabytes) for a single job execution. Hence, the corresponding job submission workflow does not include explicit file transfer activities. Note that the proposed pattern is not directly applicable to all kinds of scenarios. For example, a scientific user may want to run several jobs using the same input data that should not be transfered for each single job execution. A solution to this scenario is to provide a special file transfer workflow that can be directly integrated in the job submission workflow, or in higher-level workflows (cp. Sec. 6).

6 Prospects

As already stated in Sec. 5, file staging for compute jobs is implicitly possible (cp. RQ-6). The staging is defined in JSDL, placing the responsibility for pre- and post-job execution of data transfers on the execution environment. More complex scenarios, however, require more sophisticated functionalities. For example, this

is the case for explicit data staging as input for a bundle of jobs. Generally, file staging requires to transfer files from a source to a destination – supported by appropriate protocols such as GridFTP or UNICORE File Transfer Services. Typical file transfers are executed between a local user client and a remote destination, or between a remote source and a remote destination, the latter being referred to as *third-party transfers*. Since available workflow engines usually are not designed to pass and transfer voluminous data (RQ-6) directly, third-party transfers can be realized by workflows – using the workflow engine solely for transfer coordination. We propose to regard such transfer workflows as a single, configurable workflow activity specific to the scientific domain (cp. RQ-5).

We developed a WS-BPEL pattern to invoke Grid services in Globus Toolkit 4 (GT4) [4]. This pattern provides a basis to develop GT4 job submission and file transfer workflows analogous to those discussed in this paper. However, beside mere service orchestration it is necessary to address the respective security infrastructures. Since the BIS-Grid engine is based on UNICORE 6, the secure invocation of (external) UNICORE 6 Grid services in workflows is guaranteed. To enable the secure invocation of GT4 Grid services we plan to support the GT4 Grid Security Infrastructure (GSI) in the BIS-Grid engine. This will be evaluated in an appropriate application scenario which is currently being prepared. Another important issue for scientific workflows is scalability, which is already considered in the design of the BIS-Grid engine [9]. Nevertheless, scalability is currently not supported directly and thus regarded as future work.

7 Conclusion

In this paper, we described to use the BIS-Grid workflow engine, consisting of service extensions to UNICORE 6 and an arbitrary WS-BPEL engine, for the execution of scientific workflows. Thereby we focused on job submission as an important aspect of scientific workflows, and presented an appropriate WS-BPEL pattern for job submission with UNICORE 6. Previously, we discussed the principal differences of scientific and business workflows, and presented the principal requirements of scientific workflows. We also presented our future work focusing on advanced file staging mechanisms, on interoperability with Globus Toolkit 4 by supporting the Grid Security Infrastructure and by developing analogous WS-BPEL patterns specific to Globus Toolkit 4, and on regarding scalability.

References

1. Job Submission Description Language (JSDL) Specification, Version 1.0. http://www.gridforum.org/documents/GFD.56.pdf, November 2005.
2. Asif Akram, David Meredith, and Rob Allan. Evaluation of BPEL to Scientific Workflows. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pages 269–274, Washington, DC, USA, 2006. IEEE Computer Society.

3. Roger Barga and Dennis Gannon. Scientific versus Business Workflows. In *Workflows for e-Science*, pages 9–16. Springer London, 2007. ISBN 978-1-84628-519-6 (Print) 978-1-84628-757-2 (Online).
4. André Brinkmann, Stefan Gudenkauf, Wilhelm Hasselbring, André Höing, Odej Kao, Holger Karl, Holger Nitsche, and Guido Scherp. Employing WS-BPEL Design Patterns for Grid Service Orchestration using a Standard WS-BPEL Engine and a Grid Middleware. In Marian Bubak, Michal Turala, and Wiatr Kazimierz, editors, *CGW'08 Proceedings*, pages 103–110, Cracow, Poland, 2009. ACC CYFRONET AGH.
5. Kuo-Ming Chao, Muhammad Younas, Nathan Griffiths, Irfan Awan, Rachid Anane, and C-F Tsai. Analysis of Grid Service Composition with BPEL4WS. In *Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA'04)*, volume 01, page 284, Los Alamitos, CA, USA, 2004. IEEE Computer Society.
6. Tim Dörnemann, Thomas Friese, Sergej Herdt, Ernst Juhnke, and Bernd Freisleben. Grid Workflow Modelling Using Grid-Specific BPEL Extensions. 2007.
7. Wolfgang Emmerich, Ben Butchard, Liang Chen, Sarah L. Price, and Bruno Wassermann. Grid Service Orchestration Using the Business Process Execution Language (BPEL). In *Journal of Grid Computing (2006)*, pages 283–304. Springer, 2006.
8. Onyeka Ezenwoye, S. Masoud Sadjadi, Ariel Cary, and Michael Robinson. Orchestrating WSRF-based Grid Services. Technical report, School of Computing and Information Sciences, Florida International University, April 2007.
9. Stefan Gudenkauf, Wilhelm Hasselbring, Felix Heine, André Höing, Guido Scherp, and Odej Kao. Bis-Grid: Business Workflows for the Grid. In *CGW'07 Proceedings*, pages 86–94, Krakow, Poland, 2008. ACC CYFRONET AGH.
10. Stefan Gudenkauf, Wilhelm Hasselbring, André Höing, Guido Scherp, and Odej Kao. Workflow Service Extensions for UNICORE 6 - Utilising a Standard WS-BPEL Engine for Grid Service Orchestration. *Lecture Notes in Computer Science: Euro-Par 2008 Workshops - Parallel Processing*, 5415, 2009.
11. Stefan Gudenkauf, André Höing, and Guido Scherp. Catalogue of WS-BPEL Design Patterns. Technical report, 05 2008.
12. Frank Leymann. Choreography for the Grid: towards fitting BPEL to the resource framework: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1201–1217, 2006.
13. Nick Ragouzis, John Hughes, Rob Philpott, Eve Maler, Paul Madsen, and Tom Scavo. Security Assertion Markup Language (SAML) V2.0 Technical Overview. `http://www.oasis-open.org/committees/download.php/22553/sstc-saml-tech-overview-2%200-draft-13.pdf`, February 2007. Working Draft.
14. Aleksander Slomiski. On using BPEL extensibility to implement OGSI and WSRF Grid workflows: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1229–1241, 2006.
15. Wei Tan, Liana Fong, and Norman Bobroff. BPEL4Job: A Fault-Handling Design for Job Flow Management. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 27–42, Berlin, Heidelberg, 2007. Springer-Verlag.
16. Zhili Zhao, Ruisheng Zhang, Jiazao Lin, Ying Chen, Huajian Zhang, and Lian Li. An Improved Visual BPEL-Based Environment for Scientific Workflow. In *GCC '08: Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing*, pages 435–441, Washington, DC, USA, 2008. IEEE Computer Society.