Employing WS-BPEL Design Patterns for Grid Service Orchestration using a Standard WS-BPEL Engine and a Grid Middleware*

A. Brinkmann[4], S. Gudenkauf[1], W. Hasselbring[1,2], A. Hoeing[3], O. Kao[3], H. Karl[4], H. Nitsche[4], G. Scherp[1]

[1] OFFIS Institute for Information Technology, R&D Division Energy
email: `[stefan.gudenkauf, guido.scherp]@offis.de`
[2] Univ. of Kiel, Dept. Computer Science, Software Engineering Group
email: `wha@informatik.uni-kiel.de`
[3] TU Berlin, Faculty IV - Electrical Engineering and Computer Science, Dept. of Telecommunication Systems, Complex and Distributed IT Systems
email: `[andre.hoeing, odej.kao]@tu-berlin.de`
[4] Univ. of Paderborn, Paderborn Center for Parallel Computing
email: `[brinkman, holger.karl, hn]@uni-paderborn.de`

Abstract

The BIS-Grid project extends the Grid middleware UNICORE 6 to orchestrate Web Services as well as stateful Grid Services by using the WS-BPEL standard. These extensions form the so-called BIS-Grid engine consisting of UNICORE 6 services and an arbitrary WS-BPEL engine. The design decision not to modify the WS-BPEL engine requires to address issues on Grid Service utilization and issues that result from the two-component architecture of the BIS-Grid engine directly in the process description. We address these issues by the application of WS-BPEL design patterns. In this paper we present our current work on WS-BPEL design patterns, and discuss their benefits and consequences in contrast to alternative solutions.

1 Introduction

A challenge for small and medium enterprises is the integration of heterogeneous information systems, which often lack budget and qualified personnel to operate dedicated Enterprise Application Integration (EAI) solutions. The Web Service Resource Framework (WSRF) is a standard that extends Web Services to be stateful by the introduction of *resource properties*. These Grid Services provide the technical foundation to realize EAI with Grid technology. In BIS-Grid, a BMBF-funded project in the context of the German D-Grid initiative (www.d-grid.de), we develop a framework that supports EAI and Grid utilization, and examine different application scenarios ranging from inhouse application to *Integration as a Service* (IaaS) provided by trusted Grid providers. In this context, we develop the *BIS-Grid engine* that orchestrates both Web and Grid Services.

*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.

The BIS-Grid engine is based upon service extensions to the UNICORE 6 Grid middleware (www.unicore.eu), WS-BPEL as the industry de-facto standard language for service orchestration, and an arbitrary WS-BPEL workflow engine.

One considerable challenge is to provide a flexible solution which is based on well-adopted standards without proprietary modification of the WS-BPEL engine to ensure exchangeability and compatibility. The design decision not to modify a WS-BPEL engine requires to address issues on Grid Service utilization directly in the process description. To address these issues, we have identified so-called WS-BPEL design patterns. Our approach is illustrated in this paper by the example of the BIS-Grid engine, using ActiveBPEL (www.activevos.com) as WS-BPEL engine and UNICORE 6 as Grid Service container. The paper is organized as follows. A brief overview on the architecture of the BIS-Grid engine is given in Section 2. Section 3 presents design patterns for Grid utilization, and Section 4 discusses patterns that address architecture-specific issues. In Section 5 we illustrate related work and discuss its benefits and drawbacks in comparison with our solution. Section 6 presents a conclusion and planned future work.

2 Architecture

The BIS-Grid engine consists of two components: A UNICORE 6 Grid middleware and an arbitrary WS-BPEL engine in the backend, only accessible via UNICORE 6 (cp. [6]). The UNICORE layer bridges the technical gap between a Grid environment and WS-BPEL. Each message to be sent between the WS-BPEL engine and external clients/services must pass this layer. In the UNICORE layer, for example, messages are extended by credentials such as SAML assertions, which are of increasing importance for security assertions in Grids but currently not supported in WS-BPEL.

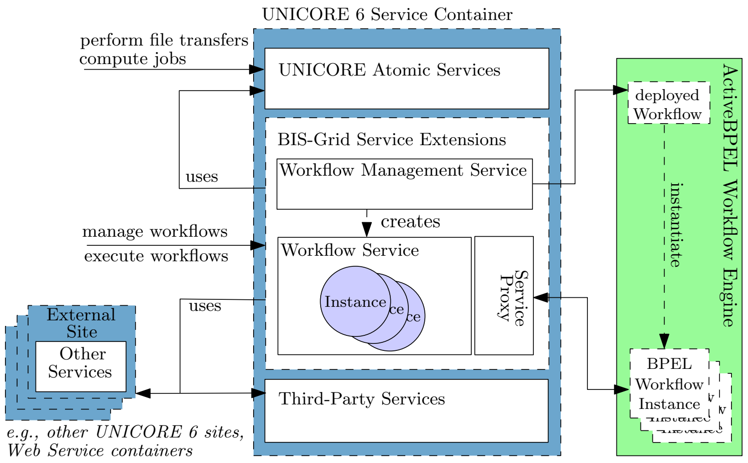

Fig. 1: Overview on the BIS-Grid engine architecture

The UNICORE layer is based upon service extensions to the service container of UNICORE 6, the UNICORE/X component (see Figure 1). They mainly consist of a *Workflow Management Service* and a generic *Workflow Service*. The Workflow Management Service provides capabilities such as process deployment, redeployment, and undeployment. During deployment the Workflow Management Service creates a new specialized version of the generic Workflow Service. Amongst other operations, the Workflow Service provides the complete Web Service interface of the original WS-BPEL process. Each process deployed in the WS-BPEL engine has a corresponding Workflow Service instance. A *Proxy Service* intercepts messages that are sent from process instances running in the WS-BPEL engine and dispatches them to the correct Workflow Service instance.

3 Grid Service Utilization Patterns

The Web Service Resource Framework represents the basis for modern Grid middlewares such as Globus Toolkit 4 and UNICORE 6. Since WSRF-based Grid Services are stateful, their invocation is more complex than the invocation of standard Web Services. The design pattern *Grid Service Invoke* consists of three primitive patterns which describe how each phase of a Grid Service invocation can be modeled with WS-BPEL (cp. Figures 2, 3). First of all, a Grid Service instance must be created (Grid Service Instance Create), then it is used once or several times (Grid Service Instance Use), and afterwards it must be destroyed (Grid Service Instance Destroy). The general pattern is described in Table 1. The development of the Grid Service Invoke pattern is based on observations of the SOAP message exchange between a Grid Service and a corresponding client as well as on a similar solution described by Ezenwoye et al. [5].

Motivation	In Grid environments, Grid middlewares are based on the Web Service Resource Framework (WSRF). Thus, services provided in Grids are WSRF services, which are stateful and require instantiation. The invocation of such a Grid Service is complex and requires the execution of certain activities in order.
Intention	Define a workflow that invokes a Grid Service by using its factory for instantiation, using the instance, and destroying the instance.
Behavior	See Figure 2
Participants	The invoker of the Grid Service, the Grid Service factory, the Grid Service instance.
Consequences	(1) Complex Grid Service invocation is encapsulated in a workflow. This reduces invocation errors and redundant implementation by reuse. (2) The Grid Service invocation workflow can be provided as a service itself. This simplifies the protocol of higher-level workflows and abstracts from invocation details. When the workflow language is hierarchical, the Grid Service invocation may be described with the same language as the high-level workflows.

Tab. 1: Pattern Grid Service Invoke

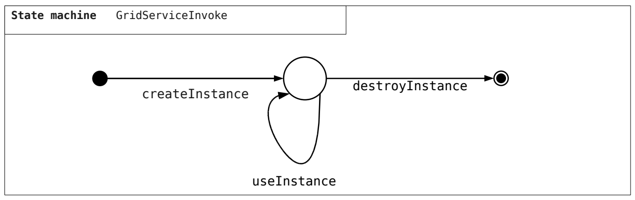

Fig. 2: Grid Service invocation

As an example, we implemented the Grid Service Invoke pattern in WS-BPEL for both the UNICORE 6 middleware and for Globus Toolkit 4 (v4.0.5). The example uses a simple counter service, which is started with an initial integer value that can be incremented arbitrarily during service lifetime. Globus Toolkit 4 provides such a service and a corresponding client by default. For UNICORE 6 we implemented an analogous service and client. The general procedure is that after the creation of a Grid Service instance via a corresponding factory service, information referencing the instance is returned. This information is then used to create new endpoint references for further dynamic invocations of the Grid Service instance, for example to invoke an operational method (increment counter) or to destroy the Grid Service instance. The complexity of Grid Service invocations in WS-BPEL is caused by the dynamic service invocation and especially by the construction of endpoint references with given WS-BPEL activities. Figure 3 illustrates the corresponding WS-BPEL process[1]. The example and the WS-BPEL code is presented in detail in [8].

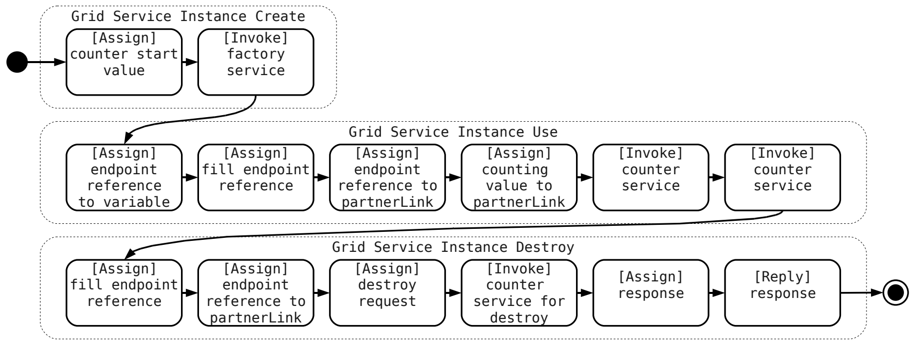
Grid Service Instance Create
[Assign] counter start value
[Invoke] factory service
Grid Service Instance Use
[Assign] endpoint reference to variable
[Assign] fill endpoint reference
[Assign] endpoint reference to partnerLink
[Assign] counting value to partnerLink
[Invoke] counter service
[Invoke] counter service
Grid Service Instance Destroy
[Assign] fill endpoint reference
[Assign] endpoint reference to partnerLink
[Assign] destroy request
[Invoke] counter service for destroy
[Assign] response
[Reply] response

Fig. 3: WS-BPEL WSRF counter service example

[1]Within Figure 3, the corresponding WS-BPEL activities are annotated in brackets. Please note that the counter service is invoked (a value is added) twice.

4 Architecture-Specific Patterns

The two-component architecture of the BIS-Grid engine requires to handle certain technical aspects in the process description. We address these aspects by applying architecture-specific patterns automatically to the WS-BPEL process description upon process deployment via XSL Transformations. Currently, we focus on two aspects, a mapping problem between UNICORE 6 and a WS-BPEL engine, and monitoring of running process instances. Each Workflow Service instance in UNICORE 6 is aligned to a process in the WS-BPEL engine. The service instance in UNICORE 6 is identified by a *resource ID*, and instances of a WS-BPEL process are identified by their *process IDs*. The Workflow Service instance serves as a proxy for the corresponding process. For example, it adds user credentials to external service invocations.

We applied two pattern to ensure that messages sent from a process instance will be mapped to the correct Workflow Service instance. The pattern *On Receive ID Retrieve* expects that each message sent to a process instance provides the resource ID of the corresponding Workflow Service instance. Within the process instance, this ID is stored in a specific WS-BPEL variable. The second pattern, *Pre-Invoke ID Assign*, ensures that the previously stored resource ID is appended to the message header of each message that is sent from the process instance by using an appropriate endpoint reference. Table 2 describes both patterns as one.

Motivation	In the BIS-Grid architecture, messages sent from a process instance must be mapped to the corresponding Workflow Service instance.
Intention	Change the WS-BPEL process description and its WSDL interfaces to manage a unique ID that identifies a corresponding UNICORE 6 Workflow Service instance. Regarding UNICORE 6, the service name and the unique instance ID can be used for identification.
Structure	*(1) WSDL:* Identify the WSDL representing the interface of the WS-BPEL process and insert an additional message part for each incoming message to carry a UNICORE 6 service instance ID. *(2) WS-BPEL:* (a) Create a new process variable that shall store a UNICORE 6 service instance ID. (b) Insert an `assign` activity after each `receive` or `pick` activity that has an attribute `createInstance` set to *yes*. This `assign` copies a UNICORE 6 service instance ID that is appended to an incoming start message into the previously created process variable. (c) Create a new process variable that shall store a dynamic endpoint reference. (d) Before each `invoke` activity, insert an `assign` activity that initializes a dynamic endpoint reference by copying a literal XML-skeleton representing the type `EndpointReference` into the variable previously created, copying the service instance ID into the `ReferenceProperties` part of the variable, and copying the variable to the *partnerLink* of the Grid Service to be invoked. The BPEL code and the `EndpointReference` literal XML-skeleton is presented in [8].
Participants	The BIS-Grid Proxy Service, the WS-BPEL engine.

Behavior	The Workflow Service instance attaches its ID to each message addressed to the WS-BPEL engine. The targeted WS-BPEL process stores the ID and attaches it to each outgoing message — e. g. for external service invocation. The BIS-Grid Proxy Service then uses the ID to forward the message to the corresponding Workflow Service instance, where the ID is removed from the message header before it is redirected to the external service.
Consequences	(1) Changes of both the WSDL description and the process description are necessary. (2) Non-domain-specific code is inserted into the WS-BPEL process description (i. e. code not being part of the domain-specific service to be represented by the process).

Tab. 2: Pattern WSRF/WS-BPEL Instance Mapping

Another issue is monitoring of process instances via the corresponding Workflow Service instances. Since monitoring is not in the focus of WS-BPEL it is obliged to the WS-BPEL engine. ActiveBPEL, for example, provides an *AdminService* to retrieve information on process instances. This service requires a process ID for process instance identification. Also, higher monitoring services require means to identify process instances. We address this issue by extending the *Pre-Invoke ID Assign* to assign the specific WS-BPEL process ID to each outgoing message. The pattern is described in Table 3.

Motivation	Workflow management and monitoring require the identification of running WS-BPEL process instances.
Intention	Implement a mechanism that propagates the ID of WS-BPEL process instances to the Workflow Service instance.
Structure	Extend the pattern *Pre-Invoke ID Assign* by extending the `assign` activity that is inserted before each `invoke` activity in the respective WS-BPEL description (cp. Table 2): (1) Extend the `EndpointReference` XML-fragment to store a process ID. (2) Add a `copy` statement that uses the WS-BPEL engine's function to retrieve the process ID and that stores it in the newly created XML-fragment part.
Participants	The BIS-Grid Workflow Service, the WS-BPEL workflow engine.
Consequences	(1) Changes of the WSDL description are unnecessary because the SOAP header is used to propagate the process ID. (2) The Workflow Service is able to address the WS-BPEL process instance via its ID only after the first `invoke` activity has been executed. (3) Non-domain-specific code is inserted into the WS-BPEL process description.

Tab. 3: Pattern Process ID Retrieval

5 Related Work

Regarding the use of BPEL for Grid Service orchestration, Leymann proposes BPEL4WS[2] as foundation since it already fulfills many requirements of the

[2]BPEL4WS 1.1 is the predecessor of WS-BPEL 2.0.

WSRF standard [9]. The appropriateness of BPEL is also examined and confirmed in [4], [10], [1], and [7]. Dörnemann et al. present a solution based on extending the BPEL4WS specification [3]. Emmerich et al. evaluate reliability, performance, and scalability of the open source workflow engine ActiveBPEL in the context of complex scientific Grid workflows [4]. Ezenwoye et al. [5] show the utilization of standard BPEL for using WSRF resources, leaving out the handling of dynamic information such as resource identifiers to workflow engine implementation. As a preparatory work, we have described the requirements that apply to a Grid-enabled workflow system in [7]. UNICORE already provides workflow extensions, originating from the Chemomentum project. This system consists of two UNICORE service containers — a workflow engine, processing workflows on a logical level, and a service orchestrator that transforms so-called Work Assignment into jobs, given in Job Submission Description Language (JSDL). Both the UNICORE 6 workflow system and the BIS-Grid engine have in common that they are implemented as service extensions to the UNICORE service container. However, the UNICORE 6 workflow system does not support the integration of a WS-BPEL workflow engine well-adopted in industry.

The presented related work mainly focuses on scientific workflows. This means that business-specific aspects of workflow execution are not covered sufficiently. For example, business workflows require to regard roles of participants and access policies, and differentiate between control and data flow, while in scientific workflows, there often is no explicit role model, and data flow is focussed. Also, except for Ezenwoye et al. [5], the presented work relies on extending or adapting BPEL, thus creating dialects. The A-WARE project addresses business workflows in Grid environments and orchestrates UNICORE 6 services with the help of a standard BPEL engine and a service bus [2]. However, workflows are not provided as Grid Services and the use of a service bus prefers inhouse application in large enterprises before application in Grids.

6 Conclusion

In this paper, we presented WS-BPEL design patterns applied in the context of the BIS-Grid engine, a two-component workflow engine based upon the Grid middleware UNICORE 6 and an arbitrary WS-BPEL engine. The patterns identified address Grid Service utilization and issues specific to the architecture of the BIS-Grid engine (mapping the corresponding instances of a workflow to each other and identifying WS-BPEL process instances to enable monitoring and management services). The benefit of applying these patterns is that neither the industry de-facto standard WS-BPEL nor the WS-BPEL engine as part of the BIS-Grid engine must be modified, avoiding a proprietary WS-BPEL dialect and preserving the WS-BPEL engine's exchanegability. A consequence of this approach is that the issues mentioned must be addressed directly in the WS-BPEL process description, considerably extending the WS-BPEL code and, in the case of architecture-specific patterns, adding non-domain-specific code.

While we regard the benefits of our approach as crucial for the adoption of the BIS-Grid engine, we intend to alleviate the consequences in our future work. Regarding the complexity of WS-BPEL code for Grid Service utilization, we plan the extension of the Netbeans BPEL Designer for Grid Service orchestration; regarding non-domain-specific code in the process description, the BIS-Grid engine already provides capabilities to automatically apply architecture-specific patterns upon workflow deployment, to be completed in our future work, too.

References

1. 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.
2. L. Clementi, C. Cacciari, M. Melato, R. Menday, and B. Hagemeier. A Business-Oriented Grid Workflow Management System. In *Euro-Par 2007 Workshops: Parallel Processing HPPC 2007, UNICORE Summit 2007, and VHPC 2007*, pages 131–140, Rennes, 2007.
3. T. Dörnemann, T. Friese, S. Herdt, E. Juhnke, and B. Freisleben. Grid Workflow Modelling Using Grid-Specific BPEL Extensions. German e-Sience Conference 2007, May 2007.
4. 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)*, number 3, pages 283–304. Springer, 2006.
5. 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.
6. Stefan Gudenkauf, Wilhelm Hasselbring, Felix Heine, André Höing, Odej Kao, and Guido Scherp. BIS-Grid: Business Workflows for the Grid. In *The 7th Cracow Grid Workshop*. Academic Computer Center CYFRONET AGH, 2007.
7. Stefan Gudenkauf, Wilhelm Hasselbring, Felix Heine, André Höing, Odej Kao, and Guido Scherp. A Software Architecture for Grid Utilisation in Business Workflows. GITO-Verlag, Berlin, 2008.
8. André Höing, Guido Scherp, and Stefan Gudenkauf. BIS-Grid - Betriebliche Informationssysteme: Grid-basierte Integration und Orchestrierung - Deliverable 2.1 Work Package 2: Catalogue of WS-BPEL Design Patterns. Technical report, Technische Universität Berlin, Faculty of Information Technologies, Complex and Distributed IT Systems and OFFIS Institute for Information Technology, R&D Division Business Information Management, August 2008.
9. Frank Leymann. Choreography for the Grid: towards fitting BPEL to the resource framework: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1201–1217, 2006.
10. Aleksander Slomiski. On using BPEL extensibility to implement OGSI and WSRF Grid workflows: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1229–1241, 2006.