GES 2007
German e-Science

Available online at
http://www.ges2007.de

Grid Workflow Modelling Using Grid-Specific BPEL Extensions

T. Dörnemann, T. Friese, S. Herdt, E. Juhnke and B. Freisleben

Department of Mathematics and Computer Science, University of Marburg,
Hans-Meerwein Straße, D-35032 Marburg, Germany
email: {doernemt, friese, seherdt, ejuhnke,
freisleb}@informatik.uni-marburg.de
phone: (+49 6421) 28 21 521, *fax:* (+49 6421) 28 573

Abstract

This paper discusses problems of Grid service composition using BPEL4WS. In particular, difficulties concerning the invocation of WSRF-based services are elucidated. A solution to this problem is presented by extending the BPEL specification, and an implementation based on the ActiveBPEL workflow enactment engine is described.

1 Introduction

Service-oriented Grid computing has gained tremendous interest in academic as well as in business environments. Many of the applications, especially from academic environments, have been designed as monolithic solutions that are hard to adapt, even to slight changes in the application requirements. Required adaptations must be implemented by programmers specialized both in Grid middleware and applications. The paradigm shift to service-orientation in Grid middleware opens the possibility to use a far more flexible software development approach, namely to compose applications from standard components, promising easier development and modification of Grid applications. The Business Process Execution Language for Web Services (BPEL4WS or BPEL) [1] has gained a lot of attention and broad adoption for composing component based business applications. The focus of the BPEL language is to enable the composition of basic web services into more complex applications. Its popularity in the business application domain makes BPEL very promising for process creation in the Grid domain, since many process execution, management and creation tools are expected to be developed in the future or are currently under development.

Modern Grid middleware environments like the Globus Toolkit 4 (GT4) [2], Unicore/GS [3] and gLite [4] are built on the Web Service Resource Framework (WSRF) [5] standard which extends web services. This allows the creation of so-called stateful web services which can store the state of operations and other properties without breaking the compatibility to standard web services.

In the Grid environment, however, BPEL has a major drawback: the current specification (version 1.1) is not capable of dealing with WSRF-compliant services [6, 7] transparently to the workflow designer. The designer has to manually model the creation of resources, copying identifiers and so on. Therefore, we present an extension to

the BPEL language which allows the interaction with both stateless and stateful services and their resources in an easy to use fashion. This faciliates the seamless integration of Grid applications in business applications and vice versa. Our implementation based on ActiveBPEL Engine is briefly described. Furthermore, our Eclipse-based collaborative workflow modelling tool [8] has been extended to reflect the changes to the BPEL vocabulary.

The paper is organized as follows. Section 2 briefly introduces the features of BPEL. The proposed extensions are presented in Section 3. Section 4 describes implementation issues, and section 5 discusses related work. Section 6 concludes the paper and outlines areas for future work.

2 Business Process Execution Language

The Business Process Execution Language for Web Services (BPEL4WS) has emerged from the earlier proposed XLANG [9] and Web Service Flow Language (WSFL) [10]. It enables the construction of complex web services composed of other web services that act as the basic activities in the process model of the newly constructed service. BPEL offers a conceptual distinction between abstract processes that describe the external view on the process model and executable processes that describe the workflow of the compound service and can be executed by a process execution engine in order to provide the functionality of the compound service to a client. Access to the process is exposed by the execution engine through a web service interface, allowing those processes to be accessed by web service clients or to act as basic activities in other process specifications.

BPEL features several basic activities which allow for interaction with the services being arranged in the workflow. This activities cover `invoke`, `receive` and `reply`. Furthermore, it is possible to wait for some time (`wait`), terminate the execution of the workflow instance (`terminate` activity), copy data from one message to another (`assign`), announce errors (`throw`), or just to do noting (`empty` activity).

To allow the composition of complex operations, a variety of structured activities exists. `Sequence` offers the ability to define ordered sequences of steps, `flow` executes a collection of steps in parallel whereas the execution order is given by links between the activities. The `switch` activity allows branching, `pick` allows to execute one of several alternative paths and loops can be defined using the `while` activity. Furthermore, BPEL includes the feature of scoping activities and specifying fault handlers and compensation handlers for scopes. Fault handlers get executed when exceptions occur, for instance, through the execution of the mentioned throw actitivy. Compensation handlers are activated when faults occur or when compensation activities that force compensation of a scope are executed.

All entities orchestrated in a workflow are seen as so-called "partners" in BPEL. Partners offer their functionality via their WSDL [11] port type description. The syntactical element `partnerLink` contains two attributes apart from the partner link type (which refers to the port type): `myRole` and `partnerRole` to specify which roles are played by the composition and the partner. During runtime, partners are mapped to actual service instances by the workflow-enactment engine.

3 Extensions to the Business Process Execution Language

Very common in WSRF-based frameworks is the use of factory patterns to instantiate resources. A factory is a web service exposing an operation (`createResource`) to create resources. Invoking this operation creates a new resource, generates an unique ID to identify the resource in later service calls and associates the resource with a web service. Thus, to invoke a stateful web service, the invoking client needs to know the ID of the resource(s) to be used. Since BPEL 1.1 was designed to operate on non-stateful web services, there is no standard way to store the unique identifier returned by the factory service and automatically use it in invoke operations on the service the resource was assigned to. Consequently, the identifier needs to be manually copied to the `ReferenceProperties` element of the SOAP [12] message [13] which adds additional complexity to the process definition and requires detailed knowledge of the specification of WSRF.

A solution to this problem using standard BPEL activities has been presented by Zager [14] who proposed to store the ID retrieved by the factory call, manually extract it using BPEL assign operations and copy it to the `referenceProperty` element of the WS-Addressing field to be used in the invocation of the service the resource is assigned to. This solution is not very intuitive, requires a lot of additional code writing and necessiates changes to the WSDL description of the WSRF service.

3.1 GridInvoke

Our proposed solution is based on introducing a new activity to the BPEL standard called `gridInvoke` (GI). It is derived from the `invoke` activity and transparently handles the invocation of state-aware WSRF services. This means that this new element of the language allows the invocation of state-aware services and the manipulation and querying of their resources. As described above, the resources to be assigned to the state-aware service must be created prior to the invocation. Therefore, we introduce the activities `gridCreateResourceInvoke` (GCRI) and `gridDestroyResourceInvoke` (GDRI) which handle the creation and destruction of WS-resources. The syntax of the constructs is described in listing 1, lines 10–15.

```
1  <partnerLinkSets>
2      <partnerLinkSet name="plsName">
3        <resourceLink name="rlnName">
4          <factory name="factoryName" partnerLink="factoryPL" />
5          <resource name="resourceName" partnerLink="resourcePL" />
6        </resourceLink>
7      </partnerLinkSet>
8    </partnerLinkSets>

10   <gridCreateResourceInvoke resourceLink="rlnName"
11     partnerLinkSet="plsName" />
12   <gridInvoke resourceLink="rlnName" partnerLinkSet="plsName"
13     operation="opName" inputVariable="inVar" outputVariable="outVar" />
14   <gridDestroyResourceInvoke resourceLink="rlsName"
15     partnerLinkSet="plsName"/>
```

Listing 1: Grid-specific extensions for the invocation of stateful WS

These activities need only to be invoked once before and after using the service. The required information, such as the the partner link of the factory service and the returned endpoint reference (EPR) pointing to the service the resource has been assigned to, are stored in so called Partner Link Sets (`partnerLinkSet`). This is done automatically and transparently to the BPEL designer at run-time of the process by the BPEL engine. Lines 3-6 of listing 1 define a resource link which consists of partner links of the factory and instance service to be used. By using the resource link in the activities in lines 10-15, the BPEL engine automatically creates resources (GCRI) by invoking the `createResource` operation of the factory port type, uses the correct resources in `gridInvoke` and destroys (GDRI) the resource upon request (line 14-15). As listing 1 shows, apart from once creating a partner link set, only two lines (one atomic activity without the need to copy data using `assign`) of BPEL code are required to interact with stateful, WSRF compatible, web services.

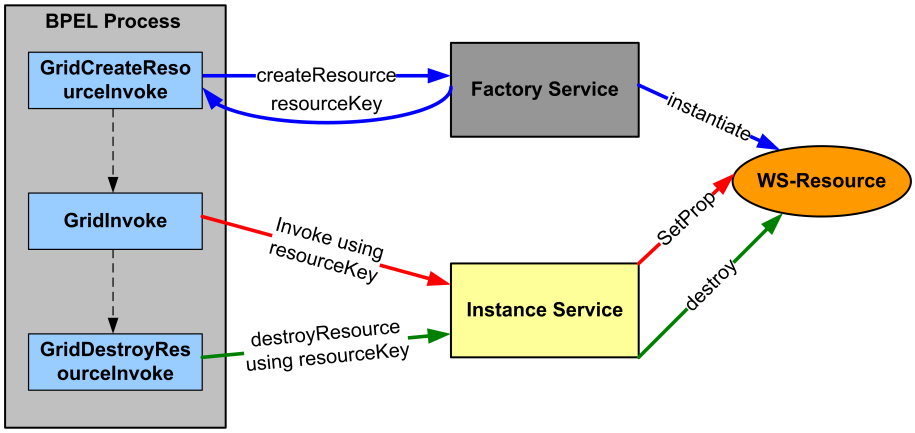

Figure 1: Execution chain of gridCreateResourceInvoke, gridInvoke and gridDestroyResouceInvoke

The implementation of this extension will be briefly described in section 4.

3.2 Eclipse-Based BPEL Designer Application

To make the development of Grid-enabled workflows as convenient as possible, we have developed an Eclipse-based BPEL designer application [8]. It provides the ability to adapt to the needs of different groups of developers, allowing Grid middleware experts to inspect and manipulate fine details of a Grid process (high-fidelity editing) while hiding complicated details from application domain experts (low-fidelity editing). To fill the gap between high- and low-fidelity editing, a collection of wizards assigns values to the hidden properties in the model elements, based on certain patterns and heuristics defined for the overall system. Furthermore, the application allows real-time collaboration between users by sharing the process model over network connections.

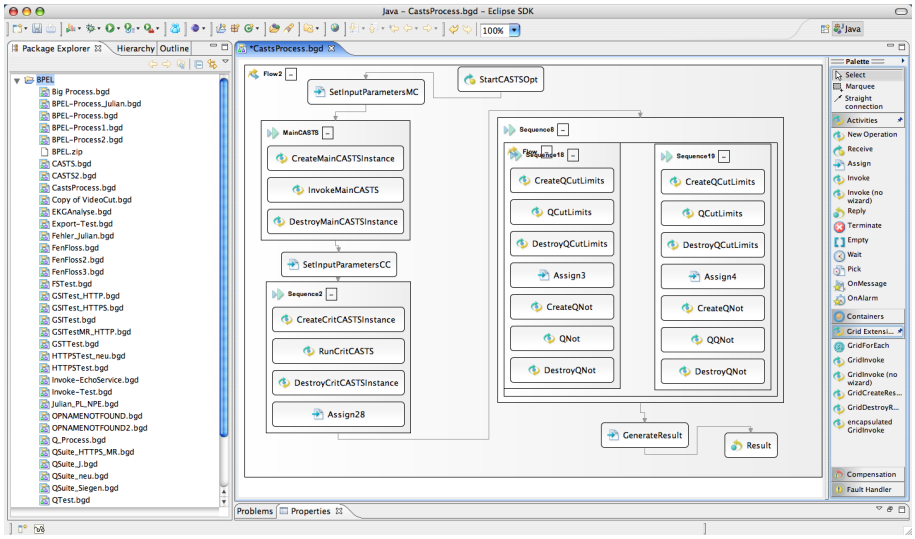

Figure 2: The grid-enabled BPEL designer displaying a simulation/optimization process

An integrated text-chat allows discussing the development process. Most notably, the BPEL designer application features an easy to use wizard for the creation of the activities explained above. After importing the services WSDL description (factory and instance), the required data is automatically generated by the wizard.

4 Implementation

The implementation of our extensions to the BPEL standard is based on the BPEL engine developed by Active Endpoints [15], because the engine is quite robust and the source code is available (GPL). Figure 3 gives an overview of the logical components of the ActiveBPEL engine. Of special interest for our work are the *Process Creation and Management* and the *Process* component itself (highlighted in the figure). The most important extension is the construct of PartnerLinkSets which encapsulates the handling of WSRF resources. A SOAP handler component has been developed which automatically inserts the Resource Key and other information needed to identify the resources into the SOAP Header of service calls. It is plugged into Apache Axis using the standard mechanism (client config). Besides implementing classes for handling and storing properties of GridInvoke, GridCreateResourceInvoke and GridDestroyResourceInvoke, the ActiveBPEL management GUI (web based) has been extended to reflect our changes to BPEL.

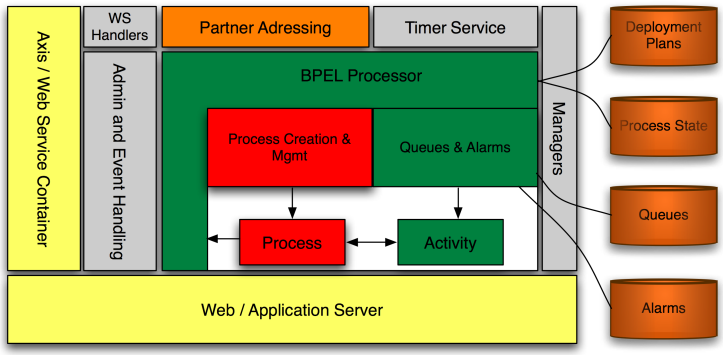

Figure 3: Logical components of the ActiveBPEL engine

4.1 Partner Link Sets

Partner link sets may contain multiple resource links, each corresponding to a WSRF resource. Resource links consist of a factory service and a resource itself. Both, factory and resource, handle the concrete partner link which points to the services to be invoked. The information where the resource is located is retrieved from the factory service and automatically copied to the resource link by our implementation. The resource key, which is important for Grid middleware correlation, is delivered by the factory service and – together with the endpoint information of the instantiated WSRF resource – stored in the resource element of the resource link.

4.2 Invoking a WSRF Resource

The `gridCreateResourceInvoke` activity identifies the corresponding partner link set before determining the resource link from it and constructing an invoke object. The created invoke object is added to the execution queue of the engine. As soon as the invoke is dequeued, any resource link information contained in that invoke is identified and the concrete endpoint is set in the Axis call. As soon as the response arrives, it is parsed and the resource key as well as the endpoint address are stored as a resource in the resource link. This information is handled by the partner link set data structure.

When `gridInvoke` is called, it performs a lookup for the partner link set and identifies the resource link corresponding to it. The engine's natural strategy to resolve a partner link is to look them up by their names. In order to use this mechanism, we decided to introduce unique identifiers for resources which are stored in the invoke object. Hence, the engine can resolve the resource link during the creation of an Axis call object. Subsequently, the correct endpoint information is saved within the call and the information about the resource key is put into the `MessageContext` (see SOAP Handler).

`GridDestroyResourceInvoke` is used when a WSRF resource is not required

anymore and therefore its lifetime should end. It constructs an invoke object in the same way gridInvoke does (but with the intention of destroying the resource). After a response arrives, the resource is removed from the partnerLinkSet.

4.3 SOAP Handler

The `SOAPHandler` is integrated into the handler chain of Apache Axis. It inspects the message context for given resource key information. If some information is found, the resource key is added to the SOAP header of the message, so that the Grid middleware can correlate the call with the correct WSRF resource. If no information is found, the call remains unchanged. In any case, the handler chain continues processing it.

4.4 Management GUI

As mentioned above, the management GUI is a web-based frontend. It now has extended functionalities to be able to display our extension in the process graph view.

5 Related Work

Several papers study the applicability of BPEL in service-oriented Grid environments. For example, Leymann [13] extensively illustrates the advantages of using workflow systems with focus on Grid environments. It is argued that some extensions to the BPEL standard may be needed to fully integrate BPEL workflows in Grid environments. The author states that especially monitoring capabilities are missing and that a separate standardization effort is required. Therefore, he concludes that Grid specific extensions of BPEL should be specified instead of defining new Grid-specific standards.

Slomiski [16] discusses benefits and challenges of using BPEL in Grid environments. The author compares both Open Grid Services Infrastructure (OGSI) [17] and WSRF [5] based Grid middlewares and concludes, that WSRF is much easier to use with BPEL than OGSI since WSRF defines extensions to WS technology instead of redefining it. Furthermore, questions such as supporting large data transfers, long running workflows and monitoring are briefly discussed. However, the paper does not address the particular question of invoking stateful services from BPEL.

Chao et al. [7] propose an architecture to enable Grid service composition based on OGSI and BPEL4WS. To hide complexity, their approach wraps Grid service clients as web services called Proxy Web Services. These Proxy Services are orchestrated in workflows using standard BPEL. All operations performed on the Proxy Services will be delegated to the actual Grid service. The approach seams feasible for OGSI which is, as already mentioned, much harder to use with BPEL than WSRF. However, it adds complexity to the Grid environment by creating a Proxy Service for every single Grid service. For this reason, the solution is not feasible for WSRF-based Grids.

Amnuaykanjanasin and Nupairoj [19] present a similar approach for the orchestration of OGSI-based Grid services using Proxy Services. The main difference to the work mentioned above is that security mechanisms (Globus Toolkit 3 security based on WS-Security [20]) and notifications are supported. Furthermore, the authors present

a tool for automatically creating Proxy Services. Despite the fact that the complexity of Grid environments is increased by this approach, the solution is interesting since it allows the usage of security and notification features.

Tan and Turner [6] describe their experience on orchestrating WSRF-based Grid services (Globus Toolkit 4) using BPEL. They identify two main problems: (1) security mechanisms cannot be used due to technical problems like incompatible Axis [18] versions, and (2) it is not possible to easily address WS-resources. The author's solution to the addressing problem is to pass the endpoint reference identifying the created resource as an operation parameter to the Grid service. The service then has to identify the resources using the reference received in the SOAP call. In our opinion, this approach is not feasible since it requires handling code in every Grid service to be orchestrated. Hence, it is impossible to invoke existing standard services like WS-GRAM (Web Service Grid Resource Allocation and Management).

6 Conclusions

In this paper, we have presented an extension to the BPEL language which allows the interaction with both stateless and stateful services and their resources in an easy to use manner. An implementation based on the ActiveBPEL Engine has been described, and our Eclipse-based collaborative workflow modelling tool [8] has been extended to reflect the changes to the BPEL vocabulary.

A topic for further research is the seamless integration of security mechanisms (like WS-SecureConversation) as well as Virtual Organization Management into the BPEL engine and our workflow designing tool.

Furthermore, workflow execution tracing and logging combined with metadata extraction is a promissing subject. Basically, this approach should enable the user to query a database containing all afore executed workflows for workflow runs with specific characteristics. Thus, users get the ability to compare results of i.e. the same workflow with different input data which might be useful for the analysis of experimental series and other applications.

7 Acknowledgements

This work is financially supported by the German Federal Ministry of Education and Research (BMBF) (D-Grid Initiative, InGrid Project).

References

1. IBM (2003) BPEL4WS: Business Process Execution Language for Web Services, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
2. Globus Allicance, Globus Toolkit 4.0, http://www.globus.org/toolkit/
3. Unicore Forum, Unicore/GS, http://www.unicore.org/
4. EGEE Project, gLite, http://glite.web.cern.ch/glite/
5. OASIS, Web Service Resource Framework (WSRF) 1.2, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

6. K. L. L. Tan, K. J. Turner, Orchestrating Grid Services using BPEL and Globus Toolkit, 7th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting, Liverpool, June 2006.
7. K. Chao, M. Younas, N. Griffiths, I. Awan, R. Anane, C. Tsai, Analysis of Grid Service Composition with BPEL4WS, Proc. of 18th International Conference on Advanced Information Networking and Applications, 2004, IEEE Press, p. 284-289.
8. T. Friese, M. Smith, B. Freisleben, J. Reichwald, T. Barth, M. Grauer, Collaborative Grid Process Creation Support in an Engineering Domain, Proc. of the 13th International Conference on High Performance Computing, 2006, IEEE Press
9. Microsoft (2001): XLANG - Web Services for Business Process Design, http://xml.coverpages.org/XLANG-C-200106.html
10. IBM (2001): Web Services Flow Language, http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
11. World Wide Web Consortium (W3C), Web Service Definition Language (WSDL) 1.1, http://www.w3.org/TR/wsdl.html
12. World Wide Web Consortium (W3C), SOAP specification 1.2, http://www.w3.org/TR/soap/
13. F. Leymann, Choreography for the Grid: towards fitting BPEL to the resource framework, Concurrency and Computation: Practice and Experience, Volume 18, Issue 10, 2005, Wiley & Sons, p. 1201 - 1217
14. M. Zager, Business Process Orchestration with BPEL: BPEL supports time critical decision making, SOA/Web Services, http://webservices.sys-con.com/read/155631_1.htm
15. Active Endpoints, http://www.active-endpoints.com
16. A. Slomiski, On using BPEL extensibility to implement OGSI and WSRF Grid workflows, Concurrency and Computation: Practice and Experience, Volume 18, Issue 10, 2005, Wiley & Sons, p. 1229 - 1241
17. Global Grid Forum (GGF), Open Grid Services Architecture 1.0, http://www.ogf.org/documents/GFD.15.pdf
18. Apache Axis (version 1.21), http://ws.apache.org/axis/
19. Pichet Amnuaykanjanasin and Natawut Nupairoj, The BPEL Orchestrating Framework for Secured Grid Services, Proc. of the International Conference on Information Technology: Coding and Computing (ITCC'05), Volume I, 2005, IEEE Press, p. 348-253
20. OASIS, WS-Security specification 1.1, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss