Security, Fault Tolerance and Modeling of Grid Workflows in BPEL4Grid*

Ernst Juhnke, Tim Dörnemann, Roland Schwarzkopf, Bernd Freisleben
Dept. of Mathematics and Computer Science, University of Marburg
{ejuhnke, doernemt, rschwarzkopf, freisleb}@informatik.uni-marburg.de

Abstract: BPEL is the de facto standard for business process modeling in today's enterprises and is a promising candidate for the integration of business and scientific applications that run in Grid or Cloud environments. In this paper, selected components of *BPEL4Grid*, a scientific workflow system for developing service-oriented Grid applications based on BPEL, are presented. The focus of the paper is on security aspects of workflow composition and fault tolerance mechanisms for long-running workflows. Furthermore, two workflow modeling tools targeted at users with different levels of knowledge about BPEL are briefly described.

1 Introduction

The Business Process Execution Language for Web Services (BPEL4WS or WS-BPEL [ACD$^+$03]) is the de facto standard for web service composition in business applications. It enables the construction of complex web services composed of other web services that act as the basic activities in the newly constructed service. Access to a *process* is exposed by the execution engine through a web service interface (Web Services Description Language, WSDL), allowing the process to be accessed by web service clients or to be used as a basic activity in other processes.

This paper presents an overview of some of the key components of BPEL4Grid (see Figure 1), a scientific workflow system for developing service-oriented Grid applications, built on BPEL and related standards (SOAP, WSDL, Eclipse). In particular, security aspects of workflow composition and fault tolerance mechanisms for long-running workflows are discussed. Furthermore, two workflow modeling tools that we have developed are briefly described: (1) DAVO (Domain-Adaptable Visual Orchestrator which is the foundation for ViGO (Visual Grid Orchestrator), a graphical modeling tool that supports the BPEL standard and our Grid extensions. (2) SimpleBPEL, a tool that allows domain experts with little or no knowledge about BPEL to compose workflows from predefined, domain-specific sub-workflows ("snippets").

*This work is financially supported by the BMBF (D-Grid Initiative)

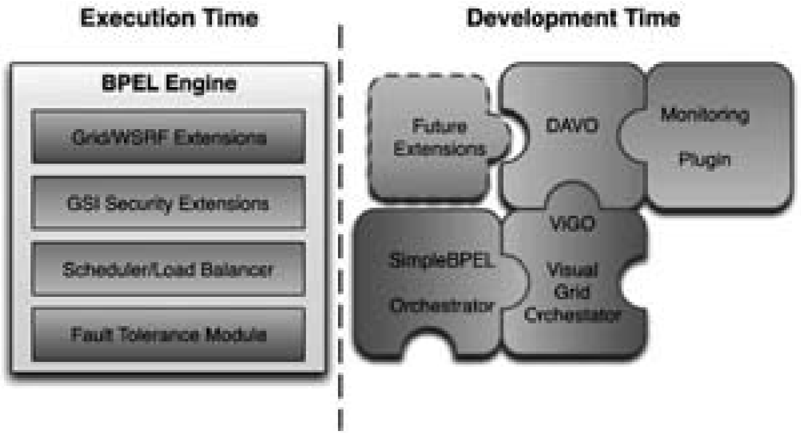

Figure 1: Components of BPEL4Grid – only components highlighted in blue will be described

2 Grid Workflow Security

While BPEL works well for traditional web services, it has a number of drawbacks with respect to the more complex world of WSRF-based Grid computing, especially where security is concerned. The BPEL security concept is not equipped to deal with complex multi-protocol Grid environments and does not integrate with the Grid Security Infrastructure (GSI). While BPEL is mainly focused on anonymous HTTPS-based TLS security or manual role-based authentication encoded in SOAP headers, Grid computing has a mandatory user-centric security approach using X.509 certificates, which far exceeds the scope and capability of the BPEL security model.

To support GSI, we have further extended our previously published [DFH+07] Grid-related BPEL extension *gridInvoke* by security-related settings [DSF08]. The extended *gridInvoke* activity allows us to define, for example, the security method to be used, choose whether to use encryption or signing of messages, and the delegation level of proxy certificates. The syntax of the extension is described in Listing 1.

```
<gridInvoke ...>
<security
 method="GSITransport | GSISecureMessage | GSISecureConversation"
 level="privacy | integrity"
 authz="none | self | host | anyString"?
 peer-credentials="filename"? anonymous="true | false"?
 delegation="none | full | limited"? />?
</gridInvoke>
```

Listing 1: Syntax of the security settings for invocation

Since the runtime of a process might be unknown and therefore longer than the proxy certificate's lifetime, automatic renewal of proxy's is offered by the BPEL4Grid engine. The workflow engine monitors both the runtime of the process and the proxy's lifetime. If the proxy's lifetime is about to expire, the engine will renew the certificate if desired by the user of the workflow. The BPEL engine contacts the given MyProxy server and retrieves a proxy certificate with the given lifetime from the server. Figure 2 illustrates this sequence. Since the whole conversation is secured using HTTPS (from client to BPEL engine) and pure TLS (from BPEL engine to MyProxy), it can be considered as secure. Further details, a discussion of the implementation and a performance evaluation can be found in a previous paper [DSF08].

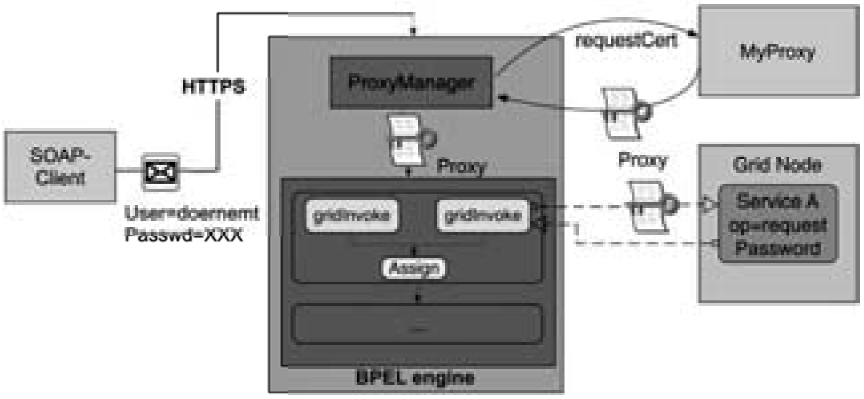

Figure 2: Integration of BPEL and Grid Security Infrastructure

3 Fault Tolerance of Grid Workflows

When long-running or computationally-intensive workflows are to be executed, fault handling is very important, since the failure of a single component might lead to an abandonment of the entire workflow. Many faults can be corrected by either simply retrying the failed operation or by substituting the failed component by an equivalent one.

While composition languages like WS-BPEL offer fault handling mechanisms, *infrastructural failures* like network timeouts and server outages should not be handled using the language mechanisms, since this would clutter the composition logic with non-functional aspects. Consequently, we identify classes of faults that can be handled automatically and define a policy language to configure automatic recovery behavior without the need for adding explicit fault handling mechanisms to the BPEL process. Furthermore, the approach provides automatic Cloud-based redundancy of services to allow substitution of defective services.

The developed solution [JDF09] adds a policy-based fault handling mechanism to BPEL without making any changes to the language standard. Using policies, it allows to enable and disable both retry and substitute actions. To reduce the number of required policies, a

Fault Classifier (FC) categorizes faults into groups. Instead of defining a policy for every type of fault, policies can be specified for each group of faults instead. Furthermore, the policies permit to set parameters such as the maximum number of retries, what kind of resources (dedicated hosts, Cloud resources, etc.) may be used for substitution and so on.

The Message Monitor (see Figure 3) monitors the response messages from invoked services and checks whether they contain fault messages or not. In the first case, the corresponding message is passed to the Fault Classifier (step 2). The classifier classifies the fault using rules stored in the Fault Database; the result is then passed (step 3) to the Policy Processor that applies all configured policies. The result contains zero or more recovery strategies with priorities (Retry is to be performed before Substitution, for instance).

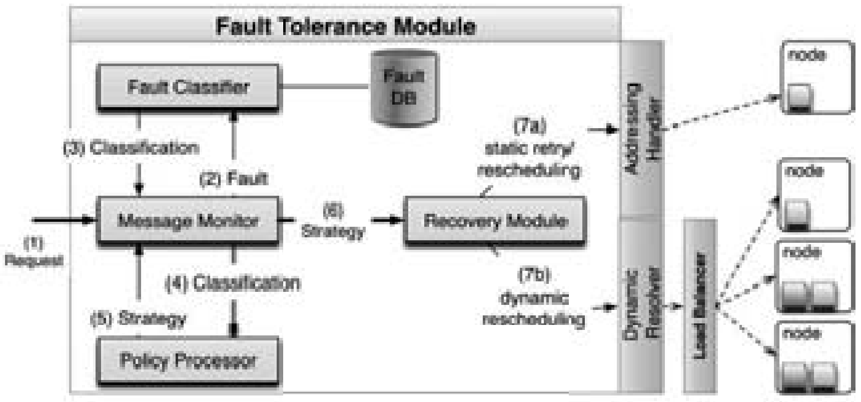

Figure 3: Sub-components of the Fault Tolerance Module

If the Retry strategy is applied, the previously invoked service is invoked again (step 7a). Otherwise, the Dynamic Resolver [DJF09] is executed to perform a dynamic scheduling on the resources declared in the Substitution strategy (step 7b). These resources may either be machines from a pool of available resources or virtual machines that are provisioned on-demand from a Cloud infrastructure like Amazon's EC2 [ec2] or Nimbus [nim]. Details concerning Cloud-based redundancy and the architecture of the on-demand provisioning framework can be found in [JDF09, DJF09].

4 Grid Workflow Modeling

4.1 Domain-Adaptable Visual Orchestrator (DAVO)

DAVO is a domain-adaptable, graphical BPEL editor [DMS$^+$09]. The key benefits that distinguish DAVO from other editors are the *adaptable data model* and the *user interface*, which permit customization to specific domain needs.

In DAVO, the activities are represented using objects of the class hierarchy shown in Figure

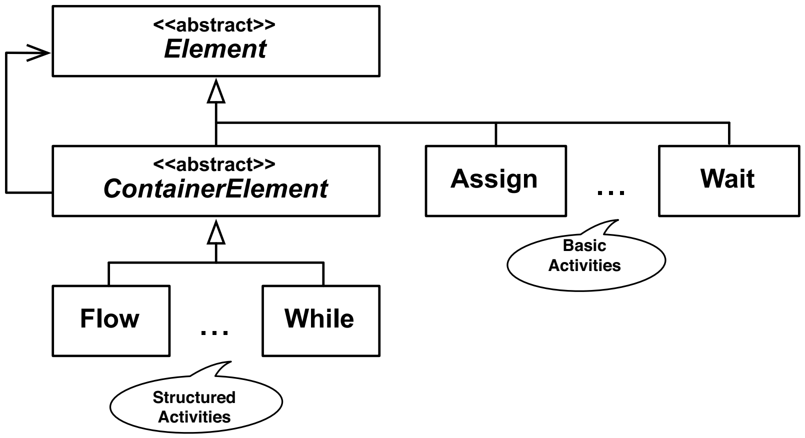

Figure 4: A simplified `Element` class hierarchy.

4. `Element` is the parent class of all activities, whereas `ContainerElement` is the parent class for all structured activities. The actual class hierarchy is more complex than the one displayed here. It contains additional abstractions for activities referring to other web services and sequential structured activities, which are omitted for simplicity since they are not relevant for the data model.

While it is sufficient for a standard BPEL workflow editor to use a simple data model to store the process' information, i.e. information for elements and attributes, this approach is not feasible with regard to extensibility. For the use of BPEL within and the adaptation of DAVO to specific domains, it is necessary to associate additional information with an activity (e.g. Grid Security-related settings). DAVO uses named properties to associate arbitrary information with activities. Besides name and value, these properties also contain an `IValidator` that can be used to check validity when `setValue()` is called. Additionally, the properties itself have various meta-properties, such as:

- `persistent` determines if the property value is stored together with the DAVO data model.
- `readOnly` and `visible`, which are used (together with various other meta-properties) to control the automatic creation of property views, as described in [DMS$^+$09].

The extension mechanism is illustrated in Figure 5. The core of each DAVO extension is an implementation of the `IModelExtender` (IME). A specific IME implementation knows the `ElementExtension` for each `Element`. After the `ElementFactory` has created the `Element` (1), it passes it to the `ElementExtender` (2), which then asks the IME for extensions for the given `Element` (3). After the IME has created the extension for the given `Element` (4), it is added to the `Element` (5). The `ElementExtension` can modify the `Element` in many ways. For example, it can add new properties or hide existing ones.

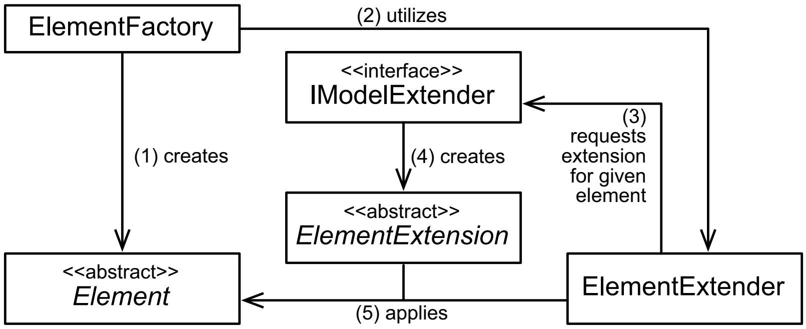

Figure 5: The `Element` extension mechanism.

New activities can be added by simply inheriting from an arbitrary class from the hierarchy and registering the new activity using one of DAVO's extensions points. As consequence, the new activity is added to DAVO's graphical user interface, as described in [DMS+09].

The described adaptability and extensibility features were used to implement the data model and visual representation of the aforementioned Grid-specific extensions in ViGO. Besides that, ViGO offers assistants to model the invocation of Grid services, allows us to graphically define security requirements and includes a WSRF-capable WSDL parser to import services into a workflow.

4.2 Simplified Modeling with SimpleBPEL

Recently, ViGO has been extended to allow parts of workflows ("snippets") to be saved in libraries. The newly developed SimpleBPEL Orchestrator allows us to import those libraries. Workflow developers may use the existing snippets as black boxes in their workflows. The tool checks whether snippets the user wants to combine in a workflow actually fit together. This is done by validating whether the output data of the first component fits to the input data of the succeeding component. If so, the tool automatically generates necessary BPEL code (such as *assign* operations) without the need for any action of the developer. Figure 6 illustrates a simple example of a workflow developed using SimpleBPEL and its actual representation in ViGO.

5 Related Work

Due to space restrictions, it is not possible to fully cover related work in all areas. We therefore only briefly present some exemplary related work.

Amnuaykanjanasin and Nupairoj [AN05] present a BPEL-based approach for orchestrating OGSI-based Globus Toolkit 3 Grid services using proxy services. Proxy services are

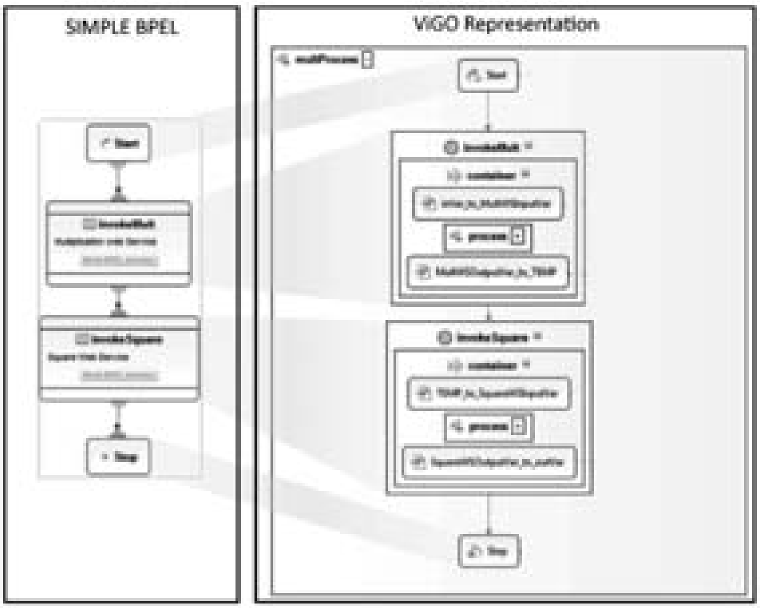

Figure 6: Workflow modeling with SimpleBPEL vs. modeling with ViGO

facade services that hide the Grid service's complexity and are invoked by the workflow engine instead of the original service. The call is then delegated to the Grid service. The approach supports security mechanisms of Globus Toolkit 3 security based on WS-Security. 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. However, lifetime management of proxy certificates is not addressed at all.

By introducing a new element (*find_bind*) into the BPEL, Karastoyanova et al. [KHC+05] have presented their approach for runtime adaptability. The mechanism is able to find services, e.g. by querying a UDDI registry. Based on policies, it selects suitable services and binds them to process instances. In case a service call fails, a process instance repair is guaranteed by rebinding to another port. Selection criteria can be modified at runtime.

Di Penta et al. [DEV+06] present WS Binder that allows the (re-) binding of *partnerLink*s to services during the runtime of a process. For this aim, the authors use a proxy architecture. *PartnerLink*s are bound to proxy services instead of the original target services, meaning that the workflows need to be adapted to run in the environment. If a failure occurs at runtime, the proxy services are rebound to target services determined by the framework's discovery and selection component.

6 Conclusion

In this paper, selected components of *BPEL4Grid* were presented: security aspects of workflow composition, fault tolerance mechanisms for long-running workflows, and corresponding workflow modeling tools. Future work will be devoted to extending the functionality of BPEL4Grid with respect to workflow analysis, validation, and monitoring.

References

[ACD$^+$03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and Sanjiva Weerawarana. *Business Process Execution Language for Web Services Version 1.1*, 1.1 edition, May 2003.

[AN05] P. Amnuaykanjanasin and N. Nupairoj. The BPEL Orchestrating Framework for Secured Grid Services. In *ITCC (1)*, pages 348–353. IEEE, 2005.

[DEV$^+$06] M. Di Penta, R. Esposito, M. Villani, R. Codato, M. Colombo, and Elisabetta Di Nitto. WS Binder: a Framework to Enable Dynamic Binding of Composite Web Services. In *Proceedings of the 2006 Int. Workshop on Service-oriented Software Engineering*, pages 74–80. ACM, 2006.

[DFH$^+$07] T. Dörnemann, T. Friese, S. Herdt, E. Juhnke, and B. Freisleben. Grid Workflow Modelling Using Grid-Specific BPEL Extensions. In *Proceedings of German e-Science Conference (GES)*, pages 1–8, 2007.

[DJF09] T. Dörnemann, E. Juhnke, and B. Freisleben. On-Demand Resource Provisioning for BPEL Workflows using Amazon's Elastic Compute Cloud. In *Proceedings of the 9th IEEE Int. Symposium on Cluster Computing and the Grid*, pages 140–147. IEEE, 2009.

[DMS$^+$09] T. Dörnemann, M. Mathes, R. Schwarzkopf, E. Juhnke, and B. Freisleben. DAVO: A Domain-Adaptable, Visual BPEL4WS Orchestrator. In *Proceedings of the IEEE 23rd Int. Conference on Advanced Information Networking and Applications (AINA '09)*, pages 121–128. IEEE, 2009.

[DSF08] T. Dörnemann, M. Smith, and B. Freisleben. Composition and Execution of Secure Workflows in WSRF-Grids. In *Proceedings of the 8th IEEE Int. Symposium on Cluster Computing and the Grid*, pages 122–129. IEEE, 2008.

[ec2] Amazon Web Services LLC, Amazon Elastic Compute Cloud (EC2). `http://aws.amazon.com/ec2/`.

[JDF09] E. Juhnke, T. Dörnemann, and B. Freisleben. Fault-Tolerant BPEL Workflow Execution via Cloud-Aware Recovery Policies. In *Proceedings of 35th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 31–38. IEEE, 2009.

[KHC$^+$05] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. Buchmann. Extending BPEL for Run Time Adaptability. In *EDOC '05: Proceedings of the Ninth IEEE Int. EDOC Enterprise Computing Conference*, pages 15–26. IEEE, 2005.

[nim] Nimbus, Nimbus IaaS Cloud Computing Solution. `http://www.nimbusproject.org`.