Interoperability of the BIS-Grid Workflow Engine with Globus Toolkit 4

Guido Scherp [1] and Wilhelm Hasselbring [1]

[1]Software Engineering Group, University of Kiel, 24098 Kiel, Germany

ABSTRACT

In the D-Grid project BIS-Grid we developed the BIS-Grid Workflow Engine in order to utilize a common WS-BPEL workflow engine for scientific workflow execution in WSRF-based Grid infrastructures. The BIS-Grid Workflow Engine itself is built on the Grid middleware UNICORE 6 to benefit from its security mechanisms and to automatically gain interoperability with UNICORE 6-based Grid infrastructures. As beside UNICORE 6 the Grid middleware Globus Toolkit 4 is prevalent in the D-Grid infrastructure, we decided to examine the interoperability of the BIS-Grid Workflow Engine with Globus Toolkit 4.

The interoperability with Globus Toolkit 4 concerns basically the support of its Grid Security Infrastructure (GSI) by UNICORE 6. In this paper we describe to what extend GSI is already supported by UNICORE 6 and what is missing. We furthermore give some details about our implementation for the BIS-Grid Workflow Engine to allow Globus Toolkit 4 service invocations within a WS-BPEL workflow execution.

Contact: gusch@informatik.uni-kiel.de

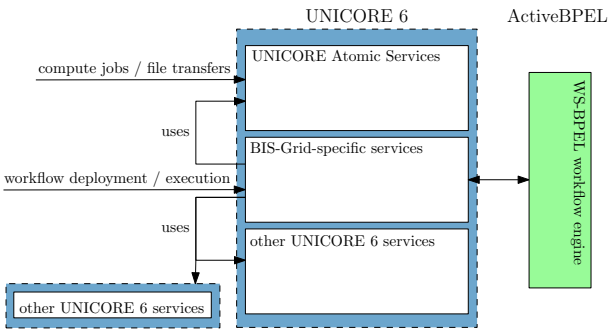

Fig. 1. General Architecture of the BIS-Grid Workflow Engine

Fig. 2. Aspects regarding GT4 interoperability by the BIS-Grid Workflow Engine

1 INTRODUCTION

In BIS-Grid[1], a project within the German Grid initiative D-Grid, we developed the so called BIS-Grid Workflow Engine[2], which utilizes the Web Services Business Process Execution Language (WS-BPEL)[12], a widely accepted standard for service orchestrations, to execute scientific workflows in WSRF[11]-based Grid infrastructures. Mainly due to a lack of supported security mechanisms, common WS-BPEL workflow engines are usually not interoperable with Grid middleware and Grid clients. Thus, we implemented a Grid wrapper for arbitrary WS-BPEL workflow engines through specific BIS-Grid-specific services for the Grid middleware UNICORE 6, see Figure 1. Based on this architecture we were able to use solely standard WS-BPEL elements for workflow execution that in principle allows a free choice of any WS-BPEL compliant workflow engine. Furthermore, the BIS-Grid-specific services in UNICORE 6 that wrap a WS-BPEL workflow engine could be implemented by solely using UNICORE 6 built-in mechanisms.

The BIS-Grid-specific services are mainly implemented as stateful WSRF services in which a state is represented by a WSRF instance. For this paper it is sufficient to know that for each workflow execution one WSRF instance is located in UNICORE 6 and one corresponding WS-BPEL workflow instance in the WS-BPEL workflow engine. It is technically assured that incoming and outgoing messages within a workflow execution are automatically mapped to and handled by the correct WSRF instances and corresponding WS-BPEL workflow instances. For more details about the BIS-Grid Workflow Engine architecture and the handling of instances please refer to [7, 8].

The decision to built on UNICORE 6 enables the BIS-Grid Workflow Engine to execute scientific workflows in UNICORE 6-based Grid infrastructures. As the Grid middleware Globus Toolkit 4 (GT4) is also a prevalent middleware in the D-Grid infrastructure, we decided to extend the BIS-Grid Workflow Engine respectively UNICORE 6 by the interoperability with GT4. Both Grid middleware provide (WSRF) services built on similar SOAP stacks and service containers, but they differ in the applied security mechanisms. Thus, establishing the interoperability with GT4 comes with the need that UNICORE 6 supports its security mechanism called Grid Security Infrastructure (GSI). Therefore, two interoperability aspects have to be considered, see Figure 2. First, the invocation of services deployed in UNICORE 6 by a GT4 client. Second, the invocation of GT4 services within a workflow execution. We decided in BIS-Grid to focus on the second aspect mainly due to its major relevance compared to the first aspect and its less effort required for implementation.

[1] http://www.bisgrid.de

[2] http://bis-grid.sourceforge.net/

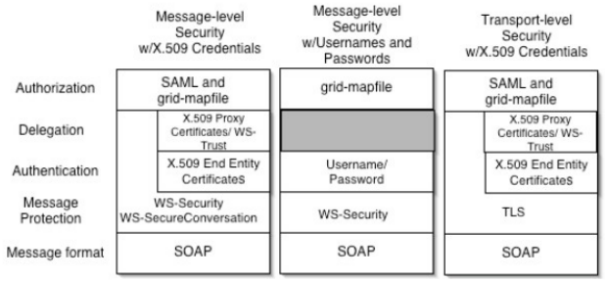

Fig. 3. Overview of GSI (from [14])

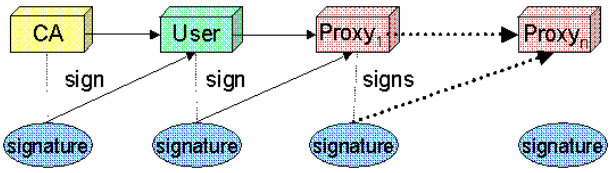

Fig. 4. Chain of Proxy Certificates (from http://www.globus.org/security/overview.html)

In Section 2 we give a brief introduction into GSI. Details about GSI support in UNICORE 6 and our reasons to focus on GT4 service invocations within a workflow execution are discussed in Section 3. Implementation details to enable GT4 service invocations in the BIS-Grid Workflow Engine are presented in Section 4.1. After discussing related work in Section 5 we conclude the paper with an outlook to future work in Section 6. Additional details about GT4 interoperability in the BIS-Grid Workflow Engine can also be found in [9].

2 GRID SECURITY INFRASTRUCTURE

Security in Globus Toolkit 4 is based on the so called Grid Security Infrastructure (GSI) which itself utilizes many existing security standards, see Figure 3. It provides three kinds of message protection, based on WS-Security, WS-SecureConversation and Transport Layer Security (TLS)[3], in which different mechanisms for authentication, authorization and delegation are supported, Delegation utilizes so called X.509 proxy certificates. Technically, a proxy certificate is a self-signed certificate which is initially derived from a personal X.509 end entity (user) certificate (EEC) issued by a trusted certificate authority (CA). Such a (first) proxy certificate allows the creation of an arbitrary number of succeeding proxy certificates, see Figure 4.

A proxy certificate is used as an user's authentication credential that can be delegated to a GT4 resource allowing it to act on behalf of an user using its identity. Proxy certificates have the advantage that the private key of a user certificate is always kept locally, but proxy certificates lacks of a revocation mechanism. Thus, a proxy certificate has usually a short lifetime to minimize misuse in case of its interception. Finally, to allow a basic management of delegated credentials GT4 provides a delegation service. For additional information about GSI, please refer to [14].

[3] Also known as Secure Sockets Layer (SSL)

3 GSI SUPPORT IN UNICORE 6

As already mentioned, GT4 interoperability in the BIS-Grid Workflow Engine implies to support GSI by UNICORE 6. Thereby, the following interoperability aspects are relevant.

1. `GT4 client`: A GT4 client invokes a service of the BIS-Grid Workflow Engine respectively UNICORE 6.
2. `GT4 service invocation`: The BIS-Grid Workflow Engine respectively UNICORE 6 invokes a GT4 service (within a workflow execution).

Thus, the technical issues concerning the two interoperability aspect are:

- Support of WS-Security, WS-SecureConversation and TLS for message protection.
- Support of proxy certificates for delegation.
- Support of the GT4 delegation service for credential management respectively proxy certificate management.
- Support of SAML and grid-mapfile for authorization.

The UNICORE 6 security stack is based on standard X.509 certificates and TLS. Support of WS-SecureConversation and proxy certificates for message protection and delegation, for example, are not covered by UNICORE 6 security mechanisms. However, a UNICORE 6 client can generate a proxy certificate which is added to the header of a SOAP message. This proxy certificate is extracted from the SOAP message within UNICORE 6 and attached to the corresponding internal WSRF instance. It is obvious that central authentication and delegation mechanisms of GSI are not supported by UNICORE 6. And it needs no further technical details to conclude that supporting the first interoperability aspect (`GT4 client`) results in a major refactoring and extension of the UNICORE 6 security stack. Consequently, we skipped that aspect mainly due to limited project resources. But we can utilize the UNICORE 6 mechanism to create and store proxy certificates in the second interoperability (`GT4 service invocation`) aspect, which is described in the following.

Regarding the second interoperability aspect (`GT4 service invocation`) the BIS-Grid Workflow Engine respectively UNICORE 6 acts as a GT4 client. Generally, a GT4 client must support GSI message protection mechanisms as well as proxy certificates[4]. GT4 already offers client libraries to invoke a GT4 service. Thus, in our approach we built on those GT4 client libraries when a GT4 service has to be invoked within a workflow execution. The needed proxy certificate can be fetched from the internal WSRF instance of the workflow execution, which was previously attached by a UNICORE 6 client. As the invocation of GT4 services within the BIS-Grid Workflow Engine has a major relevance for the project and its implementation effort is significantly less than to support GT4 clients, we implemented this approach. Implementation details are described in the following Section.

[4] The support of the GT4 delegation service and grid-mapfile is located on the Grid middleware side. SAML assertions are omitted as they are generally not supported in the D-Grid infrastructure.

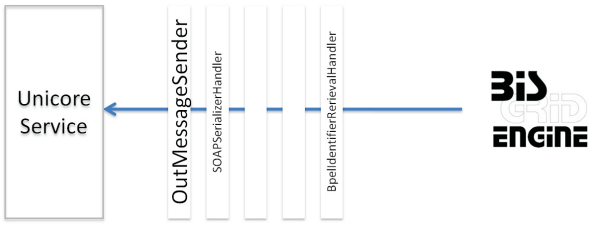

Fig. 5. Default UNICORE 6 handler chain of the BIS-Grid Workflow Engine (from [9]).

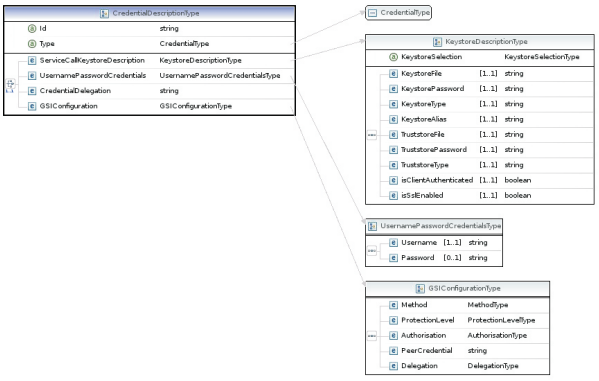

Fig. 6. Credential description in the BIS-Grid Deployment Descriptor (from [9]).

4 INVOCATION OF GLOBUS TOOLKIT 4 SERVICES WITHIN THE BIS-GRID WORKFLOW ENGINE

Incoming and outgoing messages are processed in UNICORE 6 through handler chains. Such a handler chain can be modified during runtime, which allows a flexible message handling for receiving and sending messages. The BIS-Grid Workflow Engine has a default outgoing handler chain to invoke external UNICORE 6 services[5], see Figure 5. The default handler chain is modified at runtime to invoke an external GT4 service, which is described in Section 4.1. This GT4 handler chain was further extended to support the GT4 delegation service, which is described in Section 4.2.

The configuration of external service invocations and its credentials is located in the so called BIS-Grid Deployment Descriptor. Figure 6 depicts the configuration of credentials. This information is used to distinguish between an external UNICORE 6 service and an external GT4 service invocation and to use the correct credentials. For more details about the BIS-Grid Deployment Descriptor and especially the configuration of external GT4 service invocations, please refer to [8, 9].

4.1 GT4 handler chain

The default handler chain of the BIS-Grid Workflow Engine ends with the handler `OutMessageHandler`, see Figure 5. This handler supports TLS and X.509 certificate based authentication and is responsible for the invocation of an external UNICORE 6 service. For a GT4 service invocation the `OutMessageHandler` is exchanged by the so called `AxisMessageHandler` at runtime. Its name indicates that the GT4 SOAP stack is based on Axis[6]. The `AxisMessageSender` uses the Java GT4 client libraries to configure and execute a GT4 service invocation. Furthermore, the GT4 client libraries itself uses a GSI specific Axis client handler chain, see Figure 7.

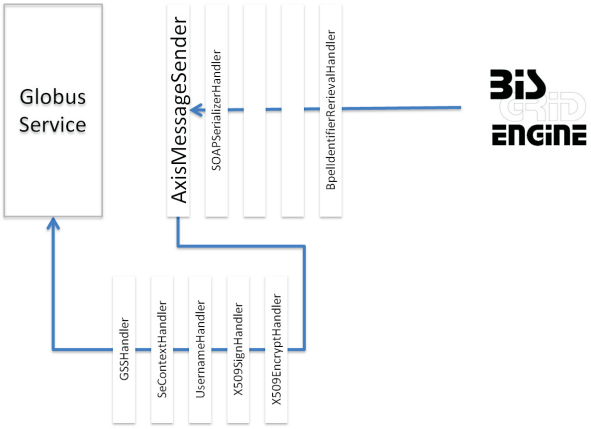

Fig. 7. GT4 handler chain (from [9]).

Even if UNICORE 6 and GT4 have different security concepts, both implementations are based on some standard Java libraries as `WSS4J` and `XMLSec`, but in different and incompatible version. This library version conflicts prevents running both handler chains in one Java virtual machine. Therefore, we implemented two solutions to avoid this conflict. In the first solution we implemented a Java classloader that replaces the default Java class loader in the GT4 handler chain starting with the `AxisMessageHandler`. In the second solution the `AxisMessageHandler` uses an RMI call to invoke the GT4 specific part of the GT4 handler chain running in its own Java virtual machine. The Java virtual machine for the corresponding RMI server is started and stopped by an additional section in the UNICORE 6 start-stop-script.

4.2 Support of Globus Toolkit 4 Delegation Service

In order to allow a more flexible delegation mechanism GT4 offers a delegation service for basic credential management. A credential is usually represented by a proxy certificate. Technically, the delegation service is implemented as stateful WSRF service. Each delegated proxy certificate is attached to one WSRF instance identifiable by an unique resource key. This resource key can be used, for example, to reference a credential respectively a proxy certificate used for data staging activities in a job submission. If the WSRF instance is destroyed the corresponding proxy certificate is deleted, too.

As first step we examined how the mechanisms works to delegate a proxy certificate as credential via the GT4 delegation service. For this reason we executed a job submission including file staging via the GT4 command line tool `globusrun-ws` with enabled debug mode and analyzed the exchanged SOAP messages. Furthermore,

[5] Standard web services are supported, too.

[6] http://axis.apache.org/axis/

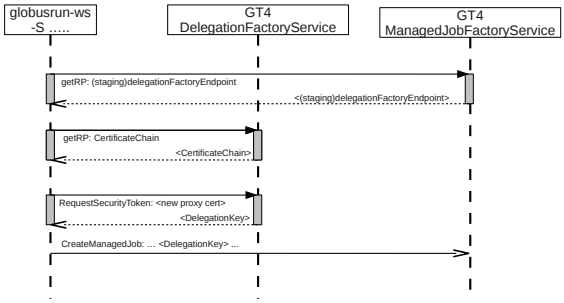

Fig. 8. Utilization of the GT4 delegation service in a job submission.

we inspected the source code of `globusrun-ws`. A proxy certificate derived from a user certificate was previously generated with the GT4 command line tool `grid-proxy-init`. Regarding the delegation service we identified the following mechanism, cp. Figure 8:

1. The WSRF resource property `delegationFactoryEndpoint` of the GT4 job submission service called `ManagedJobFactoryService` is fetched to get the endpoint for the associated GT4 delegation (factory) service called `DelegationFactoryService`.
2. The endpoint for the `DelegationFactoryService` is used to get the WSRF resource property `CertificateChain`. In our case the host certificate of the GT4 resource was returned.
3. The proxy certificate of the user is used to generate a new proxy certificate signed with the public key of the host certificate.
4. The new proxy certificate is delegated to the GT4 resource via the `DelegationFactoryService`. It returns the resource key `DelegationKey`, which identifies the corresponding WSRF instance. The `DelegationKey` is automatically added to the job description as credential reference, before the job is submitted via the `ManagedJobFactoryService`.
5. After the job execution is finished the delegated proxy certificate is deleted by destroying the corresponding WSRF instance.

Each WS-BPEL workflow engine is capable to create and handle the SOAP messages exchanged in this scenario. But to the best of our knowledge no existing WS-BPEL workflow engine supports the generation of proxy certificates as described above. Thus, we developed a solution to locate the proxy generation into the GT4 handler chain. We assume that a proxy certificate were previously attached to the corresponding WSRF instance of the workflow execution. Based on the WS-BPEL workflow perspective to following mechanism is applied:

1. The WS-BPEL workflow fetches the WSRF resource property `delegationFactoryEndpoint` of the `ManagedJobFactoryService`.
2. The WS-BPEL workflow fetches the WSRF resource property `CertificateChain` respectively the host certificate from the corresponding `DelegationFactoryService` by using dynamic binding based on the `delegationFactoryEndpoint`.
3. The WS-BPEL workflow invokes the `DelegationFactoryService` with the host certificate as credential.
4. The GT4 handler chain of the BIS-Grid Workflow Engine detects the invocation of a GT4 delegation service and generates a new proxy certificate based on the proxy certificate from the WSRF instance and signed with the public key of the host certificate in the SOAP message. Afterwards, the host certificate in the SOAP message is replaced by the new proxy certificate.
5. The GT4 handler chain uses the modified SOAP message for the invocation of the `DelegationFactoryService` to get the `DelegationKey` which is returned to the WS-BPEL workflow.
6. The WS-BPEL workflow adds the `DelegationKey` to the job description as credential reference and submits the job to the `ManagedJobFactoryService`.
7. The WS-BPEL workflow deletes the delegated proxy certificate by destroying the corresponding WSRF instance after the job execution is finished.

This approach allows a flexible management of delegated proxy certificates as credentials within a WS-BPEL workflow whereas the point of delegating or destroying a credential can be chosen by the workflow designer.

5 RELATED WORK

The general suitability for WS-BPEL to execute scientific workflows, especially in WSRF-based infrastructures, is shown in several publications [15, 4, 16, 5, 10, 13, 1, 2, 3]. Regarding the complexity to invoke stateful WSRF services some approaches as [15, 10, 2, 3] suggest extensions to WS-BPEL[7] or its predecessor BPEL4WS. Consequently, both a WS-BPEL/BPEL4WS workflow editor and a WS-BPEL/BPEL4WS workflow engine must be provided that supports those language extensions. A language extension for BPEL4WS, for example, is presented in [2, 3] that allows the invocation of stateful, WSRF-based GT4 services. To support these new language elements an existing workflow editor and workflow engine were extended. This approach hampers a migration to newer versions of the workflow editor or workflow engine. However, we reused the solution to utilize the GSI specific Axis client handler chains of the GT4 client libraries for GT4 service invocations.

[7] The standard provides an extension mechanism to define language extensions.

In [6] the utilization of conventional workflow technology as WS-BPEL for scientific simulations is discussed. A discussion about the interoperability with GSI is not included.

To the best of our knowledge, no comparable solution exists to invoke the GT4 delegation service within WS-BPEL.

6 CONCLUSION AND FUTURE WORK

In this paper we discussed the interoperability of the BIS-Grid Workflow Engine with the Grid middleware Globus Toolkit 4 (GT4) which results in a discussion about how to enable UNICORE 6 to support the Grid Security Infrastructure (GSI) used in GT4. For this reason we examined the use of GT4 clients and the invocation of GT4 services. Due to its major significance and less effort in realization we implemented a solution for the BIS-Grid Workflow Engine that allows the invocation GSI-secured GT4 services within WS-BPEL as well as the delegation of proxy certificates via the GT4 delegation service.

Currently, we use the recommended lifetime (about 11 days) when a new proxy certificate is generated for credential delegation. We intend to allow an arbitrary proxy certificate lifetime based on additional information added to the SOAP message beside the host certificate when the GT4 delegation service is invoked.

REFERENCES

[1]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.

[2]Tim Dörnemann, Thomas Friese, Sergej Herdt, Ernst Juhnke, and Bernd Freisleben. Grid Workflow Modelling Using Grid-Specific BPEL Extensions. 2007.

[3]Tim Dörnemann, Matthew Smith, and Bernd Freisleben. Composition and execution of secure workflows in wsrf-grids. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:122–129, 2008.

[4]Wolfgang Emmerich, Ben Butchart, Liang Chen, Bruno Wassermann, and Sarah Price. Grid Service Orchestration Using the Business Process Execution Language (BPEL). *Journal of Grid Computing*, 3(3-4):283–304, September 2005.

[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]Katharina Görlach, Mirko Sonntag, Dimka Karastoyanova, Frank Leymann, and Michael Reiter. *Conventional Workflow Technology for Scientific Simulation*, pages 1–31. Guide to e-Science. Springer-Verlag, März 2011.

[7]Stefan Gudenkauf, Felix Heine, Andre Höing, Jens Lischka, Holger Nitsche, and Guido Scherp. BIS-Grid Deliverable 3.1: Specification. Technical report, BIS-Grid, December 2007.

[8]Stefan Gudenkauf, Andre Höing, Dirk Meister, Holger Nitsche, and Guido Scherp. BIS-Grid Deliverable 3.4: Final Version of the WS-BPEL Engine. Technical report, BIS-Grid, May 2009.

[9]Stefan Gudenkauf, Andre Höing, and Guido Scherp. BIS-Grid Deliverable 3.5: GT4 interoperability. Technical report, BIS-Grid, April 2010.

[10]Frank Leymann. Choreography for the Grid: towards fitting BPEL to the resource framework: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1201–1217, 2006.

[11]OASIS. Web services resource framework (wsrf). http://www.oasis-open.org/committees/wsrf/.

[12]OASIS. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 04 2007.

[13]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.

[14]The Globus Security Team. Globus toolkit version 4 grid security infrastructure: A standards perspective. Technical report, 2005.

[15]Yong Wang, Chunming Hu, and Jinpeng Huai. A New Grid Workflow Description Language. In *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 257–260, Washington, DC, USA, 2005. IEEE Computer Society.

[16]Bruno Wassermann, Wolfgang Emmerich, Ben Butchart, Nick Cameron, Liang chen, and Jignesh Patel. *Sedna: A BPEL-Based Environment for Visual Scientific Workflow Modeling*, chapter 0, pages 427–448. Springer, 2006.