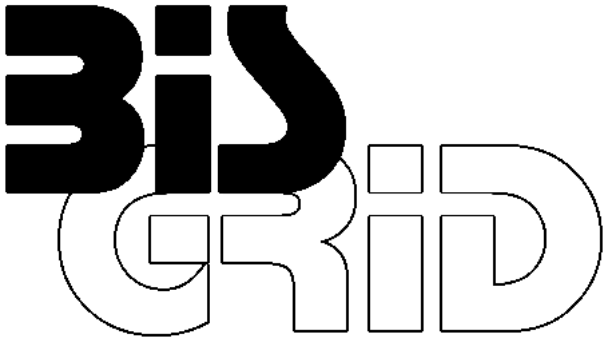


Betriebliche Informationssysteme:
Grid-basierte Integration und Orchestrierung

Deliverable 2.1

Work Package 2

Catalogue of WS-BPEL Design Patterns

August 31st, 2008

GEFÖRDERT VOM

Promotional Reference: 01IG07005

Authors:

André Höing
Technische Universität Berlin
Faculty of Information Technologies
Complex and Distributed IT Systems

Stefan Gudenkauf, **Guido Scherp**
OFFIS Institute for Information Technology
R&D-Division Energy

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.

Date:
2008/05/15

This document represents the Deliverable 2.1 “Catalogue of WS-BPEL Design Patterns” of work package 2 in the project BIS-Grid[1], a BMBF-funded project of the German D-Grid[2] initiative. It describes design patterns that are used to create Grid Service orchestrations to be deployed in the *BIS-Grid engine*. Please note that the document reflects ongoing work and thus may be replaced by updated document versions.

[1] http://www.bisgrid.de
[2] http://www.d-grid.de

Contents

1 Introduction

This document represents Deliverable 2.1 "Catalogue of WS-BPEL Design Patterns" of work package 2 in the project BIS-Grid[3], a BMBF-funded project of the German D-Grid[4] initiative. In general, this document describes design patterns that are used in order to create Grid Service orchestrations to be deployed in the *BIS-Grid engine*. The development of this engine, built on the basis of the Grid middleware UNICORE 6 and the WS-BPEL engine ActiveBPEL, is also part of the project. Please note that Deliverable 2 is closely related to Deliverable 3.1, the specification of the BIS-Grid engine.

In the following, we describe in Section 1.1 the motivation and objectives of BIS-Grid. In Section 2, we introduce concepts, standards, and technologies that underlie the WS-BPEL design patterns. The prerequisites for the WS-BPEL patterns are presented in Section 3, and the WS-BPEL design patterns are presented in the sections 4 and 5. In Section 6 we conclude this catalogue.

1.1 Motivation

Enterprises typically run various information systems for special purposes. Information on enterprise resources, customers, and products have to be stored and accessed. Thereby, information is held redundantly in these systems, making their maintenance a complex and expensive task. Enterprise Application Integration (EAI) [CHKT05] has become a well-established way to integrate such heterogeneous business information systems, often accomplished via service orchestration in service-oriented architectures (SOA): business processes considered as workflows are mapped to the technical system level, relieving maintenance. Web Services provide means to enable service orchestration as well as to hide the underlying infrastructure.

Usually, enterprises are structured in departments which often have their own information technology (IT) resources such as servers and storage systems. As enterprises grow, their departments tend to evolve into "silos" containing large amounts of IT resources. Establishing Enterprise Grids [MT] is one approach to bridge department boundaries in order to balance workload and to reduce resource demand of otherwise isolated departments. Summarising, EAI and Enterprise Grids have become well-established means to address challenges of business information systems integration and resource sharing. Nevertheless, these challenges are addressed individually and only enterprise-wide. Business information systems integration and resource sharing are furthermore not well-integrated.

Grid technologies such as the Grid middlewares UNICORE 6[5] and Globus Toolkit 4[6] are based on the Web Service Resource Framework (WSRF) [Ban06], a standard that allows stateless Web Services to become stateful. Stateful WSRF-based Web Services are also called Grid Services and provide the basis to build SOAs using Grid technologies.

[3] http://www.bisgrid.de
[4] http://www.d-grid.de
[5] http://www.unicore.eu
[6] http://www.globus.org/toolkit/

Thus, Grid technologies and EAI have much in common since both technologies focus on integration problems within an heterogeneous environment - Grid technologies on resource level and EAI on application level.

In BIS-Grid we intend to realise a service layer in the application domain of business information systems. The goal is to enable Grid technologies to be used for the integration of decentralised business information systems, bridging the gap between information system integration and resource sharing, and allowing EAI to dynamically traverse organisation-specific boundaries under consideration of enterprise security. This goes beyond the idea of Enterprise Grids because Enterprise Grids only take parts of Grid technologies and concepts from the Grid domain, and adapt and employ them within an enterprise.

The project especially addresses small and medium enterprises (SME). BIS-Grid enables these enterprises to design and run workflows realised as Grid Service orchestrations to develop and provide dynamic solutions for information systems integration, resource sharing, and utilisation without having to run own large and expensive computing and storage systems. On the technical side of the project, a WS-BPEL-based workflow engine, called BIS-Grid engine or BIS-Grid middleware, shall be developed that is capable of integrating Grid Services. Thereby, developing appropriate WS-BPEL design patterns to orchestrate Grid Services without extending or manipulating the language is also part of the project.

2 Fundamentals

In this section we introduce concepts, standards, and technologies that underlie the WS-BPEL design patterns. These are in particular the *Web Service Business Process Execution Language* (WS-BPEL), the *WS-Addressing* recommendation, the *Web Service Resource Framework* (WSRF), and the *BIS-Grid Engine*.

2.1 Web Service Business Process Execution Language

The *Web Service Business Process Execution Language* (WS-BPEL [OAS07]) is an XML-based OASIS standard that was originally introduced as BPEL4WS (Business Process Execution Language for Web Services) [BPE06] in 2002 by IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems, which are all commercial facilities. WS-BPEL describes business workflows whose activities are implemented as external Web Services. Functions provided by the business workflow itself are also offered via a Web Service interface. WS-BPEL is a well-established technology for service orchestration, and utilises existing standards such as XML Schema[W3C], WS-Addressing[Web] and XPATH[W3C07]. The WS-BPEL standard offers comprehensive elements to model the control flow (sequences, loops, parallel flows, etc.) of a business workflow as well as for data manipulation (e.g. using XPATH as a query language). More complex features like fault and error handling are considered, too. Together with the concept of compensation (roll back failed activities) transactional behaviour is realisable in the presence of Web Service invocations.

The model of a business workflow in WS-BPEL only uses abstract parts (port types and operations) of a Web Service's WSDL[7] interface to describe the external message communication. Binding these abstract interfaces to concrete service locations, also called service endpoints, is obliged to the executing WS-BPEL engine. In general, binding can be distinguished between static and dynamic binding in which service endpoints are expressed in the WS-Addressing notation. Figure 1 presents an overview on the communication elements of WS-BPEL. `partnerLinkType` elements are a WSDL extension that define a communicational relationship between two partners (Web Services). Therefore, a `partnerLinkType` element contains all WSDL port types participating in the communication in which one role is assigned to each WSDL port type. In the process a `partnerLink` element is used as an instance of one `partnerLinkType`. A `partnerLink` element assigns one (synchronous communication) or two (asynchronous communication) roles (WSDL port types) of the corresponding `partnerLinkType` to either the process itself (attribute `myRole`) and/or to the external partner (attribute `partnerRole`). The attributes `myRole` and `partnerRole` can have only one role in each case.

We have selected WS-BPEL to model business workflows capable of orchestrating Grid Services because it is a de-facto standard widely adopted in industry and is already used in some Grid projects. Knowledge with the adaption of the ActiveBPEL to fulfil Grid-specific requirements has already been collected (e.g. see [DOE07], [EBC+06], [ESCR07],

[7] Web Services Description Language

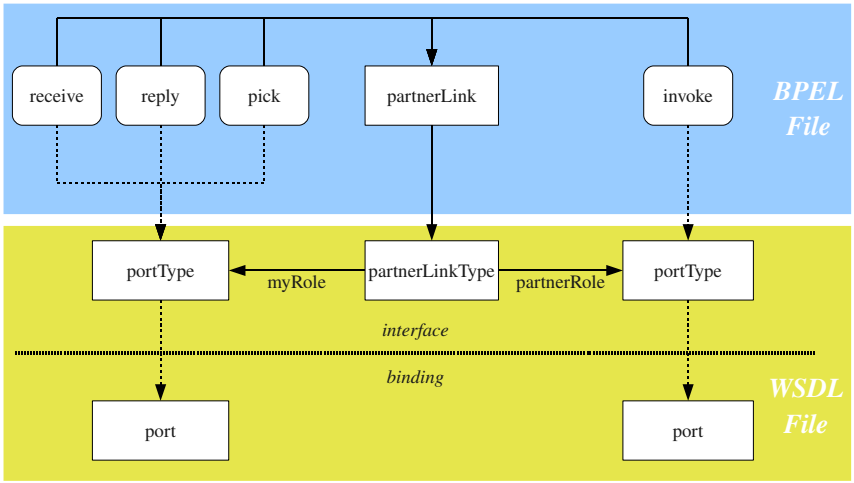


Figure 1: Communication Elements in WS-BPEL

[Ley06]). Furthermore, several engines are availabe to execute WS-BPEL-based business workflows. In BIS-Grid we decided to use ActiveBPEL (see Section 2.4).

2.2 WS-Addressing

Web Services Addressing (WS-Addressing)[Web] is a W3C recommendation for a transport-neutral way to address Web Services and exchange messages. This is achieved by using two constructs *endpoint references* and *message information headers* to normalise information which can be processed independently of transport or application. The WS-Addressing recommendation consists of a set of three documents:

- Web Services Addressing 1.0 - Core
- Web Services Addressing 1.0 - SOAP Binding
- Web Services Addressing 1.0 - Metadata

The WS-Addressing working group was closed in September 2007 with the publication of the "Metadata" document. In early 2006 the "Core" and "SOAP Binding" documents where published, the initial input was delivered by IBM, BEA System, Microsoft, SAP and SUN in 2004 [OHM+04].

WS-Addressing defines the way how message headers direct SOAP-messages to entities that can be referenced, and describes mechanisms to redirect replies or faults to a specific location. Furthermore it provides an XML format for exchanging endpoint references and end-to-end endpoint identification. These are the key-features to provide security on the message layer since messages can be transported on a broad spectrum of transport protocols, e.g. SMTP, FTP, TCP, without decoding and re-encoding the

message. Messages can also be partly encoded or signed, which allows intermediate Web Services to interpret public (not encoded) parts of the message or parts of the message which are specially encoded for only this Web Service. Therefore, a Web Service can be implemented which is able to route SOAP-messages and to load balance them to different endpoints by adding its own header information.

2.3 Web Service Resource Framework

The *Web Services Resource Framework* (WSRF) [Ban06] is an XML-based OASIS standard that emerged from the Grid context and has been developed by IBM, Hewlett-Packard (HP), the Globus Alliance[8] and others. The aim of WSRF is to provide compatibility of Grid computing capabilities described in the Open Grid Service Infrastructure (OGSI [TCF+03]) to Web Service standards. In this way, WSRF can be regarded as a refactoring of the OGSI. WSRF extends stateless Web Services, so that stateful interactions with so-called Web Service resources are facilitated in a standard manner. More concrete, WSRF is a set of five specifications concerning the association of Web Services and resources, each describing a certain aspect of stateful Web Services, also known as Grid Services. The WSRF standard consists of the following specifications:

- Web Service Resource (WS-Resource) [GKM+06] defines a WS-Resource. A WS-Resource describes the relationship between a Web Service and a resource in the WS-Resource Framework, the pattern by which resources are accessed through Web Services, and the means by which WS-Resources are referenced.

- Web Service Resource Properties (WS-ResourceProperties) [GT06] defines the means to declare resource properties as part of a Web Service description, defines the message exchanges for querying and updating resource property values, defines a standard means by which requesters can use the WS-Notification [wsn] specification to receive notification messages related to changes in resource property values, and discusses security issues concerning resource properties.

- Web Service Resource Lifetime (WS-ResourceLifetime) [SB06] defines message exchanges to standardise the means by which a WS-Resource may be destroyed (immediate destruction or scheduled destruction), and WS-ResourceProperties that may be used to inspect and monitor the lifetime of a WS-Resource.

- Web Services Service Group (WS-ServiceGroup) [MSB06] defines a means by which Web Services and WS-Resources can be aggregated or grouped together for a domain specific purpose. WS-ServiceGroup membership rules, membership constraints and classifications are expressed using WS-ResourceProperties. The specification also defines a `ServiceGroup` and a `ServiceGroupRegistration` interface to be composed with other application domain specific interfaces.

[8] http://www.globus.org

- Web Service Base Faults (WS-BaseFaults) [LM06] defines an XML Schema type for base faults, and rules how this fault type is used and extended by Web Services.

The WSRF standard is well-established in the Grid context and was adopted by the Grid middlewares Globus Toolkit 4 and UNICORE 6. The middlewares use WSRF Grid Services for basic functionalities such as file transfer and resource allocation that require stateful interaction.

2.4 BIS-Grid Engine

In the BIS-Grid project we focus on realising EAI using Grid technologies. One major objective is to proof that Grid technologies are feasible for information systems integration. Small and medium enterprises shall be enabled to integrate heterogeneous business information systems and to use external Grid resources and services with affordable effort. To do so, we develop a WS-BPEL-based workflow engine, the *BIS-Grid engine*, that is capable to integrate Grid Services. This engine is based upon service extensions to the UNICORE 6 Grid middleware, and upon using an arbitrary WS-BPEL workflow engine and standard WS-BPEL to orchestrate Grid Services. Also, it propagates service orchestrations as Grid Services.

Several reasons led us to the decision to use UNICORE 6. The main reason is that UNICORE 6 is a pioneer in adopting Grid standards, since the support of standards is essential for us, especially regarding security. As the WS-BPEL workflow engine to be used we have selected ActiveBPEL[9] since it exhaustively supports the WS-BPEL standard, and is well accepted in the business domain as well as in the Grid domain[10]. We refrain from extending well adopted standards and technologies as far as possible to increase sustainability. Instead, we use service extensions to UNICORE 6 to encapsulate the WS-BPEL engine by wrapping the message exchange of the WS-BPEL engine with (external) Grid Services and clients. These service extensions are basically a *Workflow Management Service* and several *Workflow Services*.

The Workflow Management Service and the Workflow Services are realised as Grid Services within UNICORE 6's service container, the UNICORE/X component. For each workflow deployed with the Workflow Management Service one Workflow Service will be created using a hot deployment mechanism without restarting UNICORE/X, whereas the new Workflow Service is named by the deployed workflow's name. These services manage and access the ActiveBPEL workflow engine that is located behind UNICORE/X. Direct communication with ActiveBPEL is not allowed. As a standard WS-BPEL workflow engine, it typically orchestrates stateless Web Services, and supports only basic security mechanisms, e.g. username-based and password-based authentication. Therefore, WSRF service state and advanced security concepts must be provided by the Workflow Management Service and the Workflow Services in the UNICORE/X service container.

[9] http://www.activevos.com/community-open-source.php

[10] In the Grid domain, ActiveBPEL is used by the UK Open Middleware Initiative (OMII), in the US project caGrid and in the German D-Grid project In-Grid [DOE07]

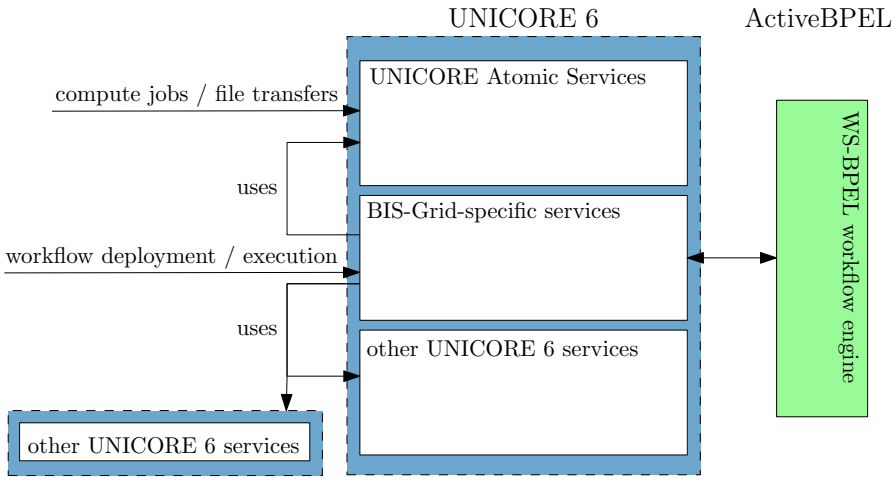
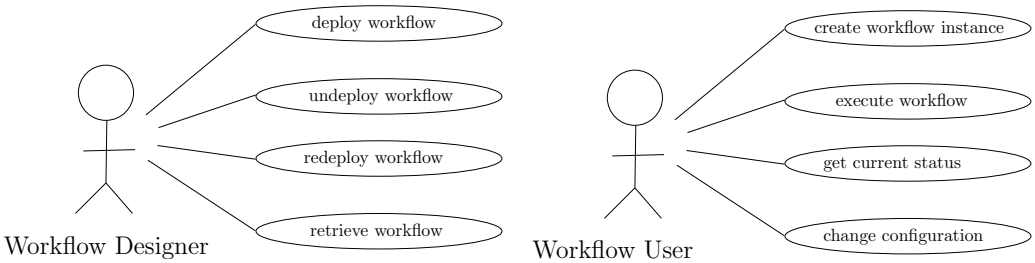


Figure 2: Overview on the Integration of the BIS-Grid Services into the UNICORE 6 Architecture

Figure 2 presents an overview on the architecture of the BIS-Grid engine. The Workflow Management Service and the Workflow Services are placed beside the so-called *UNICORE Atomic Services* and beside potential other Grid Services within the UNICORE 6 service container (UNICORE/X). The Atomic Services provide basic functionalities to support high performance Grid computing, and the other Grid Services may, for example, offer functions such as providing access to business information systems. The ActiveBPEL workflow engine is located behind and concealed by UNICORE 6. An important design decision was to neither extend the WS-BPEL standard nor to modify the ActiveBPEL workflow engine to enable WS-BPEL-based Grid Service orchestration. The reason is that Grid-specific extensions would conclude in a proprietary solution which presumably will not be interoperable with future modifications regarding the WS-BPEL standard as well as the workflow engine. Furthermore, we intend to enable the use of any WS-BPEL-compliant workflow engine without having to modify it. Leaving WS-BPEL as well as the ActiveBPEL workflow engine untouched ensures sustainability and flexibility, and future developments can be adopted easily. In addition, by placing the WS-BPEL workflow engine outside UNICORE 6 it can be deployed separately from a frondend node to support load balancing. For further details on load balancing the BIS-Grid workflow engine we refer to [GHH+08]. At last, use cases of the Workflow Management Service and the Workflow Service are shown in Figure 3. For more information about the BIS-Grid engine please refer to Deliverable 3.1 (Specification) [HHG+07] and Deliverable 3.2 (Documentation) [HGS08].



(a) Workflow Management Service (b) Workflow Service

Figure 3: Use Cases of the BIS-Grid Service Extensions to UNICORE 6

3 Prerequisites

Before discussing and introducing the particular BPEL patterns, we introduce the prerequisites and conventions we assume and apply within the patterns. Note that "..." represents a placeholder for additional valid code elements. Furthermore, we assume that one external Grid Service exists to be used within a WS-BPEL process.

3.1 Execution and Testing Environment

Regarding the development of WS-BPEL design patterns for Grid Service utilisation, the following environments are regarded: the development environment, the testing environment and the execution environment. The development environment and the (patterns) testing environment are described in this document. The execution environment is regarded in Deliverables 4.2 and 4.6, the realisation documentation of the application scenarios within BIS-Grid.

Development Environment 2x Windows XP SP2, ActiveBPEL Designer v4.0, Netbeans IDE 6.1

Testing Environment 1x Windows XP SP2, Apache Tomcat 5.5, ActiveBPEL engine v.4.0., UNICORE 6.1.0; 1x Debian Linux (sid), Apache Tomcat 5.5, ActiveBPEL engine v.4.0, UNICORE 6.0.1

3.2 Namespaces

We assume the following namespaces to be available for the WS-BPEL design patterns in the following sections. Beside the standard namespace for executable WS-BPEL processes we use additional namespaces for different WSDL definitions, which are introduced with their namespace prefixes in the following: `clt` references to elements of the WSDL definition for the Web Service used by workflow clients, `gsf` references to elements of the WSDL definition for the Factory Service of the external Grid Service, `gsr` references to elements of the external Grid Service WSDL definition, and `plt` references to a WSDL definition including parter link types (see Section 3.4). These namespaces MUST be available to any WS-BPEL design pattern described in this document. Also, each pattern MAY define additional namespaces if necessary. The definition of the initial namespaces is located in the (root)-element `process` of the WS-BPEL process and is shown in Listing 1.

Listing 1: Namespaces that must be available to the WS-BPEL Design Patterns

```
<process
    name="GridWorkflow"
    targetNamespace="http://bisgrid.d-grid.de/gridworkflow"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/
        executable"
```

```
      xmlns:clt="http://bisgrid.d-grid.de/wsdl/client"
      xmlns:gsf="http://bisgrid.d-grid.de/wsdl/gridservicefactory"
      xmlns:gsr="http://bisgrid.d-grid.de/wsdl/gridservice"
      xmlns:plt="http://bisgrid.d-grid.de/wsdl/partnerlinktype"
      ... >
    ...
</process>
```

3.3 Imports

Each namespace of the namespace prefixes `clt`, `gsf`, `gsr`, and `plt` (see Section 3.2) is used for an according WSDL definition. These WSDL definitions MUST be available to the WS-BPEL process. Thus, we assume the following import elements to be available for the WS-BPEL design patterns in the following sections. Also, each pattern MAY define additional imports if necessary. The imports are shown in Listing 2.

Listing 2: Import Section used in the WS-BPEL Design Patterns

```
<import  importType="http://schemas.xmlsoap.org/wsdl/"
         location="<URI>/client.wsdl"
         namespace="http://bisgrid.d-grid.de/wsdl/client"/>
<import  importType="http://schemas.xmlsoap.org/wsdl/"
         location="<URI>/gridservicefactory.wsdl"
         namespace="http://bisgrid.d-grid.de/wsdl/gridservice"/>
<import  importType="http://schemas.xmlsoap.org/wsdl/"
         location="<URI>/gridservicefactory.wsdl"
         namespace="http://bisgrid.d-grid.de/wsdl/
            gridservicefactory"/>
<import  importType="http://schemas.xmlsoap.org/wsdl/"
         location="<URI>/partnerlinktypes.wsdl"
         namespace="http://bisgrid.d-grid.de/wsdl/partnerlinktype
            "/>
...
```

3.4 Partner Link Types

Partner link types are used to describe the conversational relationship between two Web Services (see Section 2). Therefore, the roles played by each Web Service in the conversation and the according WSDL port types are specified. These port types define the operations that can be invoked at the partner. As our example scenarios currently are based on a synchronous communication only one Web Service is involved per partner link. Thus, we assume the following three partner link types to be available for the WS-BPEL design patterns: `ClientPLT` for the Web Service used by workflow clients, `GridServiceFactoryPLT` for the factory service of the external Grid Service, and `GridServicePLT` for the external Grid Service itself. These partner links types MAY be used

by any WS-BPEL design pattern described in this document. Also, each pattern MAY define additional partner links types if necessary. The partner link types are shown in Listing 3. Note that the partner link type definitions may be distributed across different WSDL files of the services that participate on the workflow.

Listing 3: Partner Link Types used in the WS-BPEL Design Patterns

```
<plnk:partnerLinkType xmlns:plnk="http://docs.oasis-open.org/
    wsbpel/2.0/plnktype" name="ClientPLT">
    <plnk:role name="ClientRole" portType="clt:ClientPT"/>
</plnk:partnerLinkType>

<plnk:partnerLinkType xmlns:plnk="http://docs.oasis-open.org/
    wsbpel/2.0/plnktype" name="GridServiceFactoryPLT">
    <plnk:role name="GridServiceRole" portType="gsr:GridServicePT
        "/>
</plnk:partnerLinkType>

<plnk:partnerLinkType xmlns:plnk="http://docs.oasis-open.org/
    wsbpel/2.0/plnktype" name="GridServicePLT">
    <plnk:role name="GridServiceFactoryRole" portType="gsf:
        GridServiceFactoryPT"/>
</plnk:partnerLinkType>
```

3.5 Partner links

Within a WS-BPEL process description partner links define the services that are orchestrated by the process (see Section 2). Each partner link is an instance of a partner link type and the WSDL port types (to be implemented) referenced by the roles of this partner link type are assigned to either an external Web Service (`partnerRole`) or the WS-BPEL process itself (`myRole`). We assume the following partner links to be available for the WS-BPEL patterns: `ClientPL` representing the Web Service used by workflow clients, `GridServiceFactoryPL` representing the factory service of the external Grid Service, and `GridServicePL` representing the Grid Service itself. These partner links MAY be used by any WS-BPEL pattern described in this document. Also, each pattern MAY define additional partner links if necessary. The definition of the partner links is shown in Listing 4.

Listing 4: Partner Links used in the WS-BPEL Design Patterns

```
<partnerLinks>
    <partnerLink name="ClientPL"
                 partnerLinkType="ClientPLT"
                 myRole="clt:ClientRole"/>
    <partnerLink name="GridServiceFactoryPL"
                 partnerLinkType="GridServiceFactoryPLT"
```

```
                partnerRole="gsf:GridServiceFactoryPTRole"/>
    <partnerLink name="GridServicePL"
                 partnerLinkType="GridServicePLT"
                 partnerRole="gsr:GridServicePTRole"/>
    ...
</partnerLinks>
```

3.6 Variables

One use of variables in WS-BPEL is to save and access messages which are exchanged during the process execution. Thus, variables are defined for the request and responses messages of each Web Service interface used in the WS-BPEL process. These Web Services are the Web Service used by workflow clients, the factory service of the Grid Service, and the Grid Service itself. These variables MAY be used by any WS-BPEL pattern described in this document. Also, each patterns MAY define additional variables if necessary. The definition of the variables is shown in Listing 5.

Listing 5: Variables Section used in the WS-BPEL Design Patterns

```
<variables>
    <variable name="ClientRequest" messageType="clt:
        WorkflowRequest"/>
    <variable name="ClientResponse" messageType="clt:
        WorkflowResponse"/>

    <variable name="GridServiceFactoryRequest" messageType="gsf:
        GridServiceFactoryRequest"/>
    <variable name="GridServiceFactoryResponse" messageType="gsf:
        gridServiceFactoryResponse"/>

    <variable name="GridServiceRequest" messageType="gsr:
        GridServiceRequest"/>
    <variable name="GridServiceResponse" messageType="gsr:
        GridServiceResponse"/>
    ...
</variables>
```

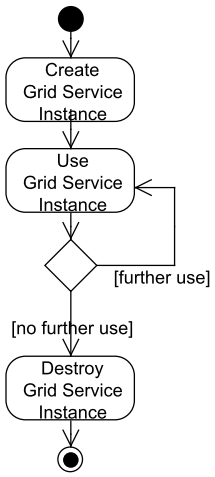


Figure 4: Grid Service Invocation

4 Grid Utilisation Patterns

As Grid Services are stateful services, their invocation, based on the WSRF standard, is quite different to standard Web Services. First of all a service instance is created, is then used once or several times, and afterwards it is destroyed. This is illustrated in Figure 4. By the following Grid utilisation patterns we describe how such a Grid Service invocation can be modelled in WS-BPEL. In general, for each middleware only one Grid utilisation pattern - called *Grid-Service-Invoke* - is necessary. The development of the patterns is based on observations of the SOAP message exchange between a Grid Service and a corresponding client as well as based on a similar pattern described by Ezenwoye et al. [ESCR07]. We focus on Grid Services in UNICORE 6, but with regard to Grid middleware interoperability and the fact that invocation pattern being very similar for UNICORE 6 and Globus Toolkit 4, we also present a working Grid-Service-Invoke pattern for Globus Toolkit 4 (Version 4.0.5).

In our test scenarios we used a simple Counter Service which starts with an initial integer value that can be incremented arbitrarily during its lifetime. Globus Toolkit 4 already offers such a service and a corresponding client by default. For UNICORE 6 we implemented an analogous Counter Service and a corresponding client. Since the whole *Grid-Service-Invoke* pattern is quite complex, we separate it into three sub-patterns:

- *Grid-Service-Instance-Create*
- *Grid-Service-Instance-Use*
- *Grid-Service-Instance-Destroy*

These three parts, including the corresponding SOAP messages, are described in the

following, working with UNICORE 6. Afterwards, we present the complete Grid-Service-Invoke pattern also for the Globus Toolkit 4 middleware and point out the differences. For the readability of the SOAP messages the following namespace declarations are assumed as to be given. We will omit them in the listings.

Listing 6: Namespaces that are assumed as given in the SOAP messages

```
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:add="http://www.w3.org/2005/08/addressing"
xmlns:fac="http://bisgrid.dgrid.de/services/counterservice/
    factorymessages"
xmlns:mes="http://bisgrid.dgrid.de/services/counterservice/
    messages"
xmlns:rl="http://docs.oasis-open.org/wsrf/rl-2"
xmlns:unic="http://www.unicore.eu/unicore6"
```

Furthermore, we use the following process structure for the Grid Service invocation patterns that extends the prerequisites in Section 3. The WS-BPEL elements used in the patterns are located within a `sequence`-element.

Listing 7: Process structure for Grid Service Invoke Pattern

```
<process ...
        xmlns:wsa="http://www.w3.org/2005/08/addressing">

    <!-- Imports, partner links etc. -->
    ...
    <variables>
        ...
        <variable name="GridServiceDestroyRequest" messageType="
            gsr:DestroyRequest"/>
        <variable name="GridServiceDestroyResponse" messageType="
            gsr:DestroyResponse"/>
        <variable name="DynamicEndpointReference" element="wsa:
            EndpointReference"/>
    </variables>

    <!-- Control Flow -->
    <sequence>
        <!-- Elements for the Grid-Service-Invoke pattern are
            located here -->
        ...
    </sequence>
</process>
```

4.1 Pattern Grid-Service-Instance-Create

Since a Grid Service factory is a standard Web Service, the creation of a Grid Service instance (Listing 8) is a single simple Web Service call using the `invoke`-Element. In the Counter Service example the request message (Listing 9) contains an initial counter value, and the response message (Listing 10) contains an endpoint reference to the newly created Grid Service instance. Please consider that the messages exchanged with the Counter Service are independent of WS-BPEL processes. We observed these messages to collect information about how a Grid Service invocation works, and used the information to model such an invocation in WS-BPEL to develop the pattern.

Listing 8: Pattern Grid-Service-Instance-Create

```
<!-- Call the Service Factory and receive the Endpoint
     Reference for the newly created service instance -->
<invoke inputVariable="GridServiceFactoryRequest" operation="
    create"
    outputVariable="GridServiceFactoryResponse"
    partnerLink="GridServiceFactoryPL"
    portType="gsf:GridServiceFactoryPT"/>
```

Listing 9: Pattern Grid-Service-Instance-Create - SOAP Request Message

```
<soap:Envelope>
   <soap:Body>

      <createCounterRequest xmlns="http://bisgrid.dgrid.de/
         services/counterservice/factorymessages">
         <startValue xmlns="http://bisgrid.dgrid.de/services/
            counterservice/factorymessages">10</startValue>
      </createCounterRequest>

   </soap:Body>
</soap:Envelope>
```

Listing 10: Pattern Grid-Service-Instance-Create - SOAP Response Message

```
<soap:Envelope>
   <soap:Body>
      <fac:createCounterResponse>

         <add:EndpointReference>
            <add:Address>
               http://pcoffis52.offis.uni-oldenburg.de
               :8080/DEMO-SITE/services/CounterService?
               res=3716af7b-055f-4bfd-85ab-15a233e1a9d3
```

```
                </add:Address>

                <add:Metadata>
                    <unic:ServerIdentity>
                        C=DE,ST=none,L=none,O=Unicore,
                        OU=none,CN=NJS Test Certificate,
                        1.2.840.113549.1.9.1=
                        #161c756e69636f72652
                        d737570706f7274406c697374732e
                        73662e6e6574
                    </unic:ServerIdentity>
                </add:Metadata>

            </add:EndpointReference>

        </fac:createCounterResponse>
    </soap:Body>
</soap:Envelope>
```

4.2 Pattern Grid-Service-Instance-Use

Based on the endpoint reference contained in the SOAP response message (Listing 10), after a Grid-Service-Instance-Create (Listing 9) another endpoint reference named `DynamicEndpointReference` is created, which is used to invoke the newly created service instance. In general, an endpoint reference defines the location where a service can be found and how it can be invoked (see Section 2). The corresponding URL to define the service location is placed in a so-called `wsa:Address` element. To invoke Grid Services besides this URL an additional identifier is needed to reference the correct Grid Service instance. In UNICORE 6 this is done by adding the identifier as an parameter to the URL, e.g. `<URL>?res=3716af7b-055f-4bfd-85ab-15a233e1a9d3` (Listing 10). All in all, the required `DynamicEndpointReference` to invoke UNICORE 6 Grid Services has the structure presented in Listing 11.

Listing 11: Pattern Grid-Service-Instance-Use - Endpoint Reference (UNICORE 6)

```
<wsa:EndpointReference>
    <wsa:Address/>
    <wsa:ReferenceProperties>
        <wsa:To/>
        <wsa:Action/>
    </wsa:ReferenceProperties>
</wsa:EndpointReference>
```

An endpoint reference's `wsa:Address` and `wsa:To` elements have to be filled with the content of the `wsa:Address` element from the SOAP response message previously presented. The endpoint reference element `wsa:Action` must contain the content of the

`soapAction` attribute that was defined for the target method in the binding of the Grid Service's WSDL interface (Listing 13).

The creation of the `DynamicEndpointReference` is done with appropriate `assign` operations. Afterwards, the new `DynamicEndpointReference` is copied to the Grid Service partner link `GridServicePL` and the Grid Service is invoked. The complete WS-BPEL code is shown in Listing 12. The request message for the Grid Service invocation contains all subelements of `wsa:ReferenceProperties` in the SOAP header. In the Counter Service request message (Listing 14), this header additionally contains a `wsa:MessageID` element which is further referenced in the response message (Listing 15) by a `wsa:RelatesTo` element. We omitted this message id in the WS-BPEL pattern since our tests worked without this element and WS-BPEL has no function to create UUIDs.

Listing 12: Pattern Grid-Service-Instance-Use (UNICORE 6)

```
<assign>
   <!-- Fill the input variable GridServiceRequest -->
   ...
   <!-- Initialise the variable DynamicEndpointReference -->
   <copy>
      <from>
         <literal>
            <wsa:EndpointReference xmlns:wsa="http://www.w3.
               org/2005/08/addressing" >
               <wsa:Address/>
               <wsa:ReferenceProperties>
                  <wsa:To/>
                  <wsa:Action/>
               </wsa:ReferenceProperties>
            </wsa:EndpointReference>
         </literal>
      </from>
      <to variable="DynamicEndpointReference"/>
   </copy>

   <!-- Set the service address in the variable
      DynamicEndpointReference -->
   <copy>
      <from part="response" variable="
         GridServiceFactoryResponse">
         <query>wsa:EndpointReference/wsa:Address</query>
      </from>
      <to variable="DynamicGridServiceEndpoint">
         <query>wsa:Address</query>
      </to>
   </copy>

```

```
        <!-- Set the destination in the variable
             DynamicEndpointReference -->
        <copy>
           <from part="response" variable="
              GridServiceFactoryResponse">
              <query>wsa:EndpointReference/wsa:Address</query>
           </from>
           <to variable="DynamicGridServiceEndpoint">
               <query>wsa:ReferenceProperties/wsa:To</query>
           </to>
        </copy>

        <!-- Set the action in the variable
             DynamicEndpointReference -->
        <copy>
           <from>
              <literal>
                 <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/
                    addressing">
                    ... <!-- insert soapAction attribute for
                        target method from WSDL-Interface -->
                 </wsa:Action>
              </literal>
           </from>
           <to variable="DynamicGridServiceEndpoint">
              <query>wsa:ReferenceProperties/wsa:Action</query>
           </to>
        </copy>

        <!-- Copy variable DynamicEndpointReference to the partner
             link GridServicePL -->
        <copy>
           <from variable="DynamicGridServiceEndpoint"/>
           <to partnerLink="GridServicePL"/>
        </copy>
     </assign>

     <!-- Invoke the Grid Service -->
     <invoke inputVariable="GridServiceRequest" operation="use"
      outputVariable="GridServiceResponse"
      partnerLink="GridServicePL"
      portType="gsr:GridServicePT"/>
```

Listing 13: Pattern Grid-Service-Instance-Use - Target Method WSDL Binding

```
<wsdl:definitions ... >
   ...
   <wsdl:binding  ... >
```

```
        ...
        <wsdl:operation name="request">
           <wsdlsoap:operation soapAction="http://bisgrid.dgrid.de
              /services/counterservice/service/add"/>
           <wsdl:input name="requestRequest">
              <wsdlsoap:body use="literal"/>
           </wsdl:input>
           <wsdl:output name="requestResponse">
              <wsdlsoap:body use="literal"/>
           </wsdl:output>
        </wsdl:operation>
        ...
    </wsdl:binding>
    ...
</wsdl:definitions>
```

Listing 14: Pattern Grid-Service-Instance-Use - SOAP Request Message

```
<soap:Envelope">
   <soap:Header">
      <wsa:To>http://pcoffis52.offis.uni-oldenburg.de:8080/DEMO-
         SITE/services/CounterService?res=3716af7b-055f-4bfd-85
         ab-15a233e1a9d3
      </wsa:To>
      <wsa:Action>http://bisgrid.dgrid.de/services/
         counterservice/service/add</wsa:Action>
      <wsa:MessageID>urn:uuid:7ED6FE10-36E1-CCEF-32AC-
         C9D8584A98EB</wsa:MessageID>
   </soap:Header>

   <soap:Body>
      <mes:addRequest>
         <mes:value>10</mes:value>
      </mes:addRequest>
   </soap:Body>
</soap:Envelope>
```

Listing 15: Pattern Grid-Service-Instance-Use - SOAP Response Message

```
<soap:Envelope>
   <soap:Header>
      <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</
         wsa:To>
      <wsa:Action>http://bisgrid.dgrid.de/services/
         counterservice/service/addAck</wsa:Action>
      <wsa:MessageID>urn:uuid:09A28AA6-F7DA-6E2A-9B76-1
         CDCC6C3987C</wsa:MessageID>
```

```
        <wsa:RelatesTo>urn:uuid:7ED6FE10-36E1-CCEF-32AC-
            C9D8584A98EB</wsa:RelatesTo>
    </soap:Header>
    <soap:Body xmlns:soap="http://schemas.xmlsoap.org/soap/
        envelope/">
        <mes:addResponse>
            <mes:currentValue>20</mes:currentValue>
        </mes:addResponse>
    </soap:Body>
</soap:Envelope>
```

4.3 Pattern Grid-Service-Instance-Destroy

Usually, after using a Grid Service instance the instance is destroyed. Using and destroying are WS-BPEL invocations to the same service instance with only the difference that the methods for using a Grid Service are defined by a workflow designer whereas the `Destroy` method is defined in the WS-ResourceLifetime specification of the WSRF (see 2) standard. Thus, an appropriate endpoint reference has to be created first. Comparing the SOAP header of the Counter Service request messages to use (Listing 14) and to destroy (Listing 17) a Grid Service instance, only the `wsa:Action` elements differ (apart from the message id). For the `Destroy` method this element must contain a specific content [11] which is the value of the `soapAction` attribute defined in the binding for the WS-ResourceLifetime WSDL interface[12]. Please note that binding information is implementation-specific and not part of the WSRF specification. Since we can reuse the previously created `DynamicEndpointReference` (see Section 4.2) only the `wsa:Action` element must be overwritten, and the destroy request message for the `Destroy` method contains an empty `Destroy` element.

Listing 16: Pattern Grid-Service-Instance-Destroy (UNICORE 6)

```
    <assign>
        <!-- Set the action in the variable
            DynamicEndpointReference -->
        <copy>
            <from>
                <literal>
                    <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/
                        addressing">
                        http://docs.oasis-open.org/wsrf/rlw-2/
                            ImmediateResourceTermination/
                            DestroyRequest
                    </wsa:Action>
                </literal>
            </from>
```

[11] http://docs.oasis-open.org/wsrf/rlw-2/ImmediateResourceTermination/DestroyRequest
[12] http://docs.oasis-open.org/wsrf/rlw-2.wsdl

```
            <to variable="DynamicGridServiceEndpoint">
               <query>wsa:ReferenceProperties/wsa:Action</query>
            </to>
         </copy>

         <!-- Copy variable DynamicEndpointReference to the partner
              link GridServicePL -->
         <copy>
            <from variable="DynamicGridServiceEndpoint"/>
            <to partnerLink="GridServicePL"/>
         </copy>

         <!-- Initialise the variable GridServiceDestroyRequest -->
         <copy>
            <from>
               <literal/>
            </from>
            <to part="Destroy" variable="GridServiceDestroyRequest"
               />
         </copy>
      </assign>

      <!-- Destroy the service instance -->
      <invoke inputVariable="GridServiceDestroyRequest" operation="
         Destroy"
       outputVariable="GridServiceDestroyResponse"
       partnerLink="GridServicePL"
       portType="gsr:GridServicePT"/>
```

Listing 17: Pattern Grid-Service-Instance-Destroy - SOAP Request Message

```
<soap:Envelope>
   <soap:Header>
      <wsa:To>http://pcoffis52.offis.uni-oldenburg.de:8080/DEMO-
         SITE/services/CounterService?res=3716af7b-055f-4bfd-85
         ab-15a233e1a9d3
      </wsa:To>
      <wsa:Action>http://docs.oasis-open.org/wsrf/rlw-2/
         ImmediateResourceTermination/DestroyRequest</wsa:
         Action>
      <wsa:MessageID>urn:uuid:FC2312A5-8D6C-B739-BB8E-9
         BE0C878B87F</wsa:MessageID>
   </soap:Header>

   <soap:Body>
      <rl:Destroy/>
   </soap:Body>
</soap:Envelope>
```

Listing 18: Pattern Grid-Service-Instance-Destroy - SOAP Response Message

```
<soap:Envelope>
   <soap:Header>
      <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</
         wsa:To>
      <wsa:Action>http://docs.oasis-open.org/wsrf/rlw-2/
         ImmediateResourceTermination/DestroyRequestAck</wsa:
         Action>
      <wsa:MessageID>urn:uuid:25665DA8-5CBC-2B9C-5516-6
         C972D0A8F3C</wsa:MessageID>
      <wsa:RelatesTo>urn:uuid:FC2312A5-8D6C-B739-BB8E-9
         BE0C878B87F</wsa:RelatesTo>
   </soap:Header>

   <soap:Body>
      <rl:DestroyResponse/>
   </soap:Body>
</soap:Envelope>
```

4.4 Pattern Grid-Service-Invoke for UNICORE 6

The Grid Service invocation pattern *Grid-Service-Invoke* is a sequence of the three patterns *Grid-Service-Instance-Create*, *Grid-Service-Instance-Use* and *Grid-Service-Instance-Destroy*. Consider that the pattern Grid-Service-Instance-Use can occur several times. Listing 19 shows the complete Grid-Service-Invoke pattern within the WS-BPEL process structure intended for the UNICORE 6 middleware.

Listing 19: Pattern Grid-Service-Invoke (UNICORE 6)

```
<process ...
        xmlns:wsa="http://www.w3.org/2005/08/addressing">

   <!-- Imports, partner links etc. -->
   ...
   <variables>
      ...
      <variable name="GridServiceDestroyRequest" messageType="
         gsr:DestroyRequest"/>
      <variable name="GridServiceDestroyResponse" messageType="
         gsr:DestroyResponse"/>
      <variable name="DynamicEndpointReference" element="wsa:
         EndpointReference"/>
   </variables>

   <!-- Control Flow -->
   <sequence
      <!-- ------------------------------ -->
```

```
<!-- Grid-Service-Instance-Create  -->
<!-- ---------------------------- -->
<!-- Call the Service Factory and receive the Endpoint
   Reference for the newly created service instance -->
<invoke inputVariable="GridServiceFactoryRequest"
   operation="create"
        outputVariable="GridServiceFactoryResponse"
        partnerLink="GridServiceFactoryPL"
        portType="gsf:GridServiceFactoryPT"/>

<!-- ---------------------------- -->
<!-- Grid-Service-Instance-Use     -->
<!-- ---------------------------- -->
<assign>
   <!-- Fill the input variable GridServiceRequest -->
   ...
   <!-- Initialise the variable DynamicEndpointReference
      -->
   <copy>
      <from>
         <literal>
            <wsa:EndpointReference xmlns:wsa="http://www.
               w3.org/2005/08/addressing" >
               <wsa:Address/>
               <wsa:ReferenceProperties>
                  <wsa:To/>
                  <wsa:Action/>
               </wsa:ReferenceProperties>
            </wsa:EndpointReference>
         </literal>
      </from>
      <to variable="DynamicEndpointReference"/>
   </copy>

   <!-- Set the service address in the variable
      DynamicEndpointReference -->
   <copy>
      <from part="response" variable="
         GridServiceFactoryResponse">
         <query>wsa:EndpointReference/wsa:Address</query>
      </from>
      <to variable="DynamicGridServiceEndpoint">
         <query>wsa:Address</query>
      </to>
   </copy>

   <!-- Set the destination in the variable
      DynamicEndpointReference -->
   <copy>
```

```
            <from part="response" variable="
                GridServiceFactoryResponse">
60              <query>wsa:EndpointReference/wsa:Address</query>
            </from>
            <to variable="DynamicGridServiceEndpoint">
                <query>wsa:ReferenceProperties/wsa:To</query>
            </to>
65      </copy>

        <!-- Set the action in the variable
            DynamicEndpointReference -->
        <copy>
            <from>
70              <literal>
                    <wsa:Action xmlns:wsa="http://www.w3.org
                        /2005/08/addressing">
                        ... <!-- insert soapAction attribute for
                            target method from WSDL-Interface -->
                    </wsa:Action>
                </literal>
75          </from>
            <to variable="DynamicGridServiceEndpoint">
                <query>wsa:ReferenceProperties/wsa:Action</query>
            </to>
        </copy>
80
        <!-- Copy variable DynamicEndpointReference to the
            partner link GridServicePL -->
        <copy>
            <from variable="DynamicGridServiceEndpoint"/>
            <to partnerLink="GridServicePL"/>
85      </copy>
    </assign>

    <!-- Invoke the Grid Service -->
    <invoke inputVariable="GridServiceRequest" operation="use"
90              outputVariable="GridServiceResponse"
                partnerLink="GridServicePL"
                portType="gsr:GridServicePT"/>

    <!-- ------------------------------ -->
95  <!-- Grid-Service-Instance-Destroy -->
    <!-- ------------------------------ -->
    <assign>
        <!-- Set the action in the variable
            DynamicEndpointReference -->
        <copy>
100         <from>
                <literal>
```

```
                    <wsa:Action xmlns:wsa="http://www.w3.org
                        /2005/08/addressing">
                        http://docs.oasis-open.org/wsrf/rlw-2/
                            ImmediateResourceTermination/
                            DestroyRequest
                    </wsa:Action>
105              </literal>
              </from>
              <to variable="DynamicGridServiceEndpoint">
                 <query>wsa:ReferenceProperties/wsa:Action</query>
              </to>
110        </copy>

           <!-- Copy variable DynamicEndpointReference to the
              partner link GridServicePL -->
           <copy>
              <from variable="DynamicGridServiceEndpoint"/>
115           <to partnerLink="GridServicePL"/>
           </copy>

           <!-- Initialise the variable GridServiceDestroyRequest
              -->
           <copy>
120           <from>
                 <literal/>
              </from>
              <to part="Destroy" variable="
                 GridServiceDestroyRequest"/>
           </copy>
125     </assign>

        <!-- Destroy the service instance -->
        <invoke inputVariable="GridServiceDestroyRequest"
           operation="Destroy"
                     outputVariable="GridServiceDestroyResponse"
130                  partnerLink="GridServicePL"
                     portType="gsr:GridServicePT"/>
     </sequence>
  </process>
```

-

4.5 Pattern Grid-Service-Invoke for Globus Toolkit 4

Listing 20 shows the *Grid-Service-Invoke* pattern for Globus Toolkit 4. It is very similar to the Grid-Service-Invoke pattern for UNICORE 6 but has the following differences:

- Globus Toolkit 4 and UNICORE 6 use different namespaces for WS-Addressing which affects the used namespaces in the patterns. The namespaces are *http://*

www.w3.org/2005/08/addressing for use with UNICORE 6, and *http://schemas.xmlsoap.org/ws/2004/03/addressing* for use with Globus Toolkit 4.

- In Globus Toolkit 4, the `DynamicEndpointReference` for Grid Service invocations may contain an additional `wsa:From` element in the `ResourceProperties`. This element itself contains a `wsa:Address` element with the `string`-type content *http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous*. Thus, an additional operation is needed to copy the correct `wsa:From` element to the endpoint reference. We used the term "MAY" because in our tests this element could be omitted. The reason is that the string content just indicates an "anonymous" endpoint in a synchronous communication. So it does not contains any relevant information to ensure that a response message reaches the correct recipient, and furthermore we considered only synchronous invocations. The Counter Service message we captured contains the wsa:From attribute.

- The identifier in the endpoint reference returned after Grid Service Instance creation is located as an arbitrary element in the `wsa:ResourceProperties`. Thus, an additional operation is needed to copy this identifier element (here `gsa:GridService ResourceKey`) to the `DynamicEndpointReference` used for Grid Service invocations.

- Grid Services in Globus Toolkit 4 use a different binding for the WS-ResourceLifetime WSDL interface as UNICORE 6. So, the correct `soapAction` attribute value in Globus Toolkit 4 is *http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime/Destroy*, and in UNICORE 6 the value is *http://docs.oasis-open.org/wsrf/rlw-2/ImmediateResourceTermination/DestroyRequest*. This affects the `copy` operation for the `wsa:Action` in the `DynamicEndpointReference` that is used for the `invoke` to destroy the Grid Service instance. Note that the different `soapAction` attributes also show that different versions of the WS-ResourceLifetime specification are used: Globus Toolkit 4 uses version 1.2 Draft, and UNICORE 6 uses version 2.

- The WS-ResourceLifetime specification (see Section 2.3) contains an XML Schema and a WSDL interface. UNICORE 6 uses the exact XML Schema, but a slightly different but compatible WSDL interface. Globus Toolkit 4 uses both the exact XML Schema and the exact WSDL interface. The difference concerning the Grid-Service-Invoke pattern for Globus Toolkit 4 is that the message part containing the `Destroy` element is called `DestroyRequest` in Globus Toolkit 4, and `Destroy` in UNICORE 6. Thus, only the `part` attribute in the `copy` operation to initialise the `GridServiceDestroyRequest` is affected. Note that usually the (WS-ResourceLifetime) XML Schema is imported in the Grid Service WSDL interface, whereas the (WS-ResourceLifetime) WSDL interface is included directly meaning the methods and messages are defined within the namespace of the Grid Service.

Listing 20: Pattern Grid-Service-Invoke (Globus Toolkit 4)

```
<process ...
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/
           addressing">

    <!-- Imports, partner links etc. -->
    ...
    <variables>
       ...
       <variable name="GridServiceDestroyRequest" messageType="
          gsr:DestroyRequest"/>
       <variable name="GridServiceDestroyResponse" messageType="
          gsr:DestroyResponse"/>
       <variable name="DynamicEndpointReference" element="wsa:
          EndpointReference"/>
    </variables>

    <!-- Control Flow -->
    <sequence>
       <!-- ------------------------------ -->
       <!-- Grid-Service-Instance-Create   -->
       <!-- ------------------------------ -->
       <!-- Call the Service Factory and receive the Endpoint
          Reference for the newly created service instance -->
       <invoke inputVariable="GridServiceFactoryRequest"
          operation="create"
               outputVariable="GridServiceFactoryResponse"
               partnerLink="GridServiceFactoryPL"
               portType="gsf:GridServiceFactoryPT"/>

       <!-- ------------------------------ -->
       <!-- Grid-Service-Instance-Use      -->
       <!-- ------------------------------ -->
       <assign>
          <!-- Fill the input variable GridServiceRequest -->
          ...
          <!-- Initialise the variable DynamicEndpointReference
             -->
          <copy>
             <from>
                <literal>
                   <wsa:EndpointReference xmlns:wsa="http://
                      schemas.xmlsoap.org/ws/2004/03/addressing"
                       >
                      <wsa:Address/>
                      <wsa:ReferenceProperties>
                         <wsa:To/>
                         <wsa:Action/>
```

```
40                      <wsa:From/>
                        <gsr:GridServiceResourceKey/>
                    </wsa:ReferenceProperties>
                </wsa:EndpointReference>
            </literal>
45      </from>
        <to variable="DynamicEndpointReference"/>
    </copy>

    <!-- Set the service address in the variable
        DynamicEndpointReference -->
50  <copy>
        <from part="response" variable="
            GridServiceFactoryResponse">
            <query>wsa:EndpointReference/wsa:Address</query>
        </from>
        <to variable="DynamicGridServiceEndpoint">
55          <query>wsa:Address</query>
        </to>
    </copy>

    <!-- Set the destination in the variable
        DynamicEndpointReference -->
60  <copy>
        <from part="response" variable="
            GridServiceFactoryResponse">
            <query>wsa:EndpointReference/wsa:Address</query>
        </from>
        <to variable="DynamicGridServiceEndpoint">
65          <query>wsa:ReferenceProperties/wsa:To</query>
        </to>
    </copy>

    <!-- Set the action in the variable
        DynamicEndpointReference -->
70  <copy>
        <from>
            <literal>
                <wsa:Action xmlns:wsa="http://schemas.xmlsoap.
                    org/ws/2004/03/addressing">
                    ... <!-- insert soapAction attribute for
                        target method from WSDL-Interface -->
75              </wsa:Action>
            </literal>
        </from>
        <to variable="DynamicGridServiceEndpoint">
            <query>wsa:ReferenceProperties/wsa:Action</query>
80      </to>
    </copy>
```

```
            <!-- Set the source in the variable
                DynamicEndpointReference -->
            <copy>
85              <from>
                    <literal>
                        <wsa:From xmlns:wsa="http://schemas.xmlsoap.
                            org/ws/2004/03/addressing">
                            <wsa:Address>
                                http://schemas.xmlsoap.org/ws/2004/03/
                                    addressing/role/anonymous
90                          </wsa:Address>
                        </wsa:From>
                    </literal>
                </from>
                <to variable="DynamicGridServiceEndpoint">
95                  <query>wsa:ReferenceProperties/wsa:From</query>
                </to>
            </copy>

            <!-- Set the resource key in the variable
                DynamicEndpointReference -->
100         <copy>
                <from part="response" variable="
                    GridServiceFactoryResponse">
                    <query>wsa:EndpointReference/wsa:
                        ReferenceProperties/gs:
                        GridServiceResourceKey</query>
                </from>
                <to variable="DynamicGridServiceEndpoint">
105                 <query>wsa:ReferenceProperties/gs:
                        GridServiceResourceKey</query>
                </to>
            </copy>

            <!-- Copy variable DynamicEndpointReference to the
                partner link GridServicePL -->
110         <copy>
                <from variable="DynamicGridServiceEndpoint"/>
                <to partnerLink="GridServicePL"/>
            </copy>
        </assign>
115
        <!-- Invoke the Grid Service -->
        <invoke inputVariable="GridServiceRequest" operation="use"
                    outputVariable="GridServiceResponse"
                    partnerLink="GridServicePL"
120                 portType="gsr:GridServicePT"/>
```

```
        <!-- ------------------------------ -->
        <!-- Grid-Service-Instance-Destroy -->
        <!-- ------------------------------ -->
125     <assign>
           <!-- Set the action in the variable
              DynamicEndpointReference -->
           <copy>
              <from>
                 <literal>
130                 <wsa:Action xmlns:wsa="http://schemas.xmlsoap.
                       org/ws/2004/03/addressing">
                       http://docs.oasis-open.org/wsrf/2004/06/
                          wsrf-WS-ResourceLifetime/Destroy
                    </wsa:Action>
                 </literal>
              </from>
135           <to variable="DynamicGridServiceEndpoint">
                 <query>wsa:ReferenceProperties/wsa:Action</query>
              </to>
           </copy>

140        <!-- Copy variable DynamicEndpointReference to the
              partner link GridServicePL -->
           <copy>
              <from variable="DynamicGridServiceEndpoint"/>
              <to partnerLink="GridServicePL"/>
           </copy>
145
           <!-- Initialise the variable GridServiceDestroyRequest
              -->
           <copy>
              <from>
                 <literal/>
150           </from>
              <to part="DestroyRequest" variable="
                 GridServiceDestroyRequest"/>
           </copy>
        </assign>

155     <!-- Destroy the service instance -->
        <invoke inputVariable="GridServiceDestroyRequest"
           operation="Destroy"
                     outputVariable="GridServiceDestroyResponse"
                     partnerLink="GridServicePL"
                     portType="gsr:GridServicePT"/>
160  </sequence>
</process>
```

5 Implementation-specific Patterns

Besides aspects that meet requirements from the application domain, there are also aspects that are specific to the implementation. Such aspects meet technical requirements that can not be solved otherwise than in the WS-BPEL description, but are not part of the actual workflow. Therefore they ideally should be hidden from the workflow designer. This section presents patterns that represent such implementation-specific aspects.

5.1 UNICORE 6/ActiveBPEL Mapping Problem

Since the BIS-Grid engine utilises an existing WS-BPEL engine, it has to deal with two different instances of the same workflow. As described in Deliverable 3.1 [HSG+07], the specification of the BIS-Grid engine, there is a regular instance of a WS-BPEL workflow in the actual WS-BPEL engine which manages the workflow execution on the one hand (referred to as *workflow instance*), and the WSRF *Workflow Service instance* in the UNICORE 6 layer on the other hand. It is the UNICORE 6 layer's task to check the incoming and outgoing communication and prepare the appropriate messages for Grid utilisation. To do so it is necessary to assign a message from the WS-BPEL engine to the corresponding WSRF Workflow Service instance and vice versa. In ActiveBPEL, a workflow instance is identified by an internal process id. In UNICORE 6, a Workflow Service instance is identified by the service name and a resource ID which is also contained in its endpoint reference. Unfortunately, there is no identification information in the SOAP messages that are send between UNICORE's service container (the UNICORE/X component) and the WS-BPEL engine which enables to conclude the original workflow instance that caused to sent the message. Such information is necessary for both synchronous and asynchronous service invocations, e.g. since appropriate security credentials must be assignable to messages.

As described in Deliverable 3.1 [HSG+07], the WS-BPEL engine uses UNICORE 6 as a proxy to forward all messages. Therefore, all messages reach UNICORE 6 in the same way. Listing 21 shows a request message sent from ActiveBPEL to the Counter Service of Globus Toolkit 4 and caught by a simple proxy program. The message calls the `add`-method with parameter *5* of the Counter Service instance with ID *789238503*. Note that namespace URLs are omitted. As one can see, there is no ID or any other identification information in the message that allows to conclude the original WS-BPEL workflow instance that caused the request message. For synchronous service invocations, the ActiveBPEL engine established a new TCP connection for each request and expects the response message over this connection. In this way it can map incoming messages to the corresponding ActiveBPEL workflow instance. For asynchronous service calls incoming messages can be mapped to the correct service instance by using WS-BPEL correlation sets. However, the BIS-Grid services have no information about the original ActiveBPEL workflow instance that sent a message because of the absence of an ID.

Listing 21: Sample ActiveBPEL SOAP Invoke Message

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="..."
    xmlns:xsd="..." xmlns:xsi="...">
  <soapenv:Header>
    <ns2:CounterKey xmlns="..." xmlns:ns1="..."
        xmlns:soapenv="..." xmlns:wsa="..."
      xmlns:xsd="..." xmlns:xsi="..."
      soapenv:mustUnderstand="0"
      xmlns:ns2="...">
        789238503
      </ns2:CounterKey>
  </soapenv:Header>

  <soapenv:Body>
    <aetgt:add xmlns:aetgt="...">5</aetgt:add>
  </soapenv:Body>
</soapenv:Envelope>
```

Our solution to solve this problem is to modify the WS-BPEL workflow description so that the identifier used by UNICORE's service container UNICORE/X is known to corresponding WS-BPEL workflow instances and is used in external message communication. Other solutions are conceivable that rely on changing the ActiveBPEL engine's source code. Since this would restrict the possibility to update the engine to a newer version or to exchange it completely, we dismiss such approaches. To realise our solution, we stipulate that a WS-BPEL workflow expects the resource ID of a corresponding Workflow Service instance to be attached to an ingoing instance-creating message call received by the workflow. Second, we stipulate an `assign`-operation before each foreign service invocation within the WS-BPEL workflow description. This assign operation inserts the resource ID into the outgoing message whose XML schema was previously extended by an appropriate element to carry this ID. All in all, the following requirements must be met:

- RQ1: The corresponding XML schema for all ingoing *start*-messages[13] must be extended by an extra variable holding the UNICORE 6 Workflow Service instance resource id.
- RQ2: There must be a dedicated process variable to store the resource ID during the whole process execution.
- RQ3: There must be an `assign` activity after each instance-creating activity which copies the ID from the message into the process variable.
- RQ4: All XML schemas for outgoing messages, i.e. used by `reply` or `invoke` activities, must be extended by an extra variable to carry the id.

[13] I.e. messages which are addressed to `receive` or `pick` activities with the `createInstance` attribute set to *yes*.

- RQ5: There must be an `assign` activity that copies the ID from the process variable into the corresponding message variable before all activities that cause an outgoing message.

The realisation of these requirements can be implemented in two different ways regarding the BIS-Grid engine. First, a workflow designer tool can be created or extended to make use of WS-BPEL patterns that cover the requirements RQ1 to RQ5. In contrast to the patterns described in Section 4, the WS-BPEL patterns discussed in this section are solely implementation-specific and have to be used applying a fixed set of insertion rules (e.g. RQ3). Therefore, we think they should be hidden from the user, both to prevent modelling errors and to conceal technical complexity. This leads to the second possibility, namely to insert or extend parts of the WS-BPEL workflow description automatically by the BIS-Grid engine's Workflow Management Service upon workflow deployment. Regarding the opposite direction, ActiveBPEL offers the possibility to ask for the internal process ID in form of the `getProcessId()` method. If this ID is needed, e.g. to use the *BpelEngineAdmin* service for monitoring issues, an additional WS-BPEL pattern must be added which propagates the process ID to the Workflow Service, e.g. within the first outgoing message.

5.1.1 Pattern On-Receive-ID-Retrieve

The first requirement to solve the UNICORE 6/ActiveBPEL mapping problem is that all *start*-messages that call a `receive` activity in the WS-BPEL workflow must be extended by an extra variable holding the UNICORE 6 workflow service instance resource ID (RQ1). When the start message is sent, an instance of the corresponding WS-BPEL workflow must be created, and the UNICORE 6 Workflow Service instance resource ID must be stored in a dedicated process variable to store the resource ID during the whole process execution (RQ2). To do so, the workflow must be able to retrieve the resource ID from the incoming message, either from the message header or from the body.

[Kel07] describes a solution of retrieving information from the header of a SOAP request. This solution relies upon extending the WSDL of a WS-BPEL process by overwriting the binding of the Web Service representing the process to specify a custom header, and on extending the receive statement by an attribute representing the header of the message received. Listing 22 illustrates such a `receive` statement. The namespace `bpelx` represents the namespace of the added header attribute.

Listing 22: Receive Activity extended by a `headerVariable` Attribute

```
...
<receive name="receiveInput"
         partnerLink="..."
         portType="..."
         operation="..."
         variable="..."
         createInstance="yes"
         bpelx:headerVariable="..."/>
```

...

Unfortunately, this solution is not applicable for us since we intend not to modify nor to extend the WS-BPEL language but only to use existing standard language elements. Since we can not access the message header otherwise, we send the UNICORE 6 Workflow Service instance resource ID within the message body as part of the standard message content, although the resource ID is not part of the original information that is necessary for the workflow to execute properly. In Listing 23 we describe a pattern that meets the first requirement presented in Section 5.1 (RQ1) by retrieving the resource ID from a SOAP message body. The pattern simply consists of extending the start message by a special message part that represents the UNICORE 6 Workflow Service instance resource ID, and of inserting an additional variable into the WS-BPEL description that exclusively holds the resource ID as a value. In our case the variable is named *ucWorkflowServiceInstanceResourceID*. After each starting activity - a `receive` or a `pick` activity with `createInstance`-Attribute set to `yes` - an `assign` activity is performed that copies the resource ID from the message part of the start message to the *ucWorkflowServiceInstanceResourceID* variable. Listing 24 illustrates a message in a WSDL document that is intended to start a workflow and is extended by a special message part containing the UNICORE 6 Workflow Service instance resource ID. Note that in this example we use the exemplary string *ID-1234-5678-9100* as the workflow instance ID. Also, the prerequisites apply as described in Section 3.

Listing 23: Pattern On-Receive-ID-Retrieve - WS-BPEL-Document

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="On-Receive-ID-Retrieve"
    targetNamespace="..."
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/
       executable"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:clt="http://my.namespace.org/wsdl/client"
    ...>

    <!-- Imports, partner links etc. -->
    ...
    <variables>
       ...
       <variable name="PartnerOperationIn" messageType="clt:
          startWorkflowMessage"/>
       <variable name="ucWorkflowServiceInstanceResourceID" type=
          "xsd:string"/>
    </variables>

    <sequence>
       <receive name="..."
          createInstance="yes"
          partnerLink="..."
```

```
            operation="..."
            portType="..."
            variable="PartnerOperationIn"/>

        <assign name="...">
            <copy>
                <from>string($PartnerOperationIn.
                    ucWorkflowServiceInstanceResourceID)</from>
                <to variable="ucWorkflowServiceInstanceResourceID"/>
            </copy>
        </assign>
        ...
    </sequence>
</process>
```

Listing 24: Pattern On-Receive-ID-Retrieve - WSDL Excerpt

```
    ...
    <message name="startWorkflowMessage">
        <part name="aMessagePart" type="xsd:string"/>
        <part name="ucWorkflowServiceInstanceResourceID" type="xsd
            :string"/>
    </message>
    ...
```

5.1.2 Pattern Pre-Invoke-ID-Assign

The pattern in Listing 25 meets all requirements presented in Section 5.1 except for the first (RQ2 to RQ5). The pattern assigns an ID to a WS-BPEL `partnerLink` before the Grid Service invocation. The particular steps are to initialise a dynamic endpoint reference by copying a literal XML-skeleton representing the type `EndpointReference` to the variable `DynamicEndpointReference`, then copying the ID into the variable, and copying the variable to the `partnerLink` of the Grid Service to be invoked. Note that in this example, we use the exemplary string *ID-1234-5678-9100* as the ID. Also, the prerequisites apply as described in Section 3.

Listing 25: Pattern Pre-Invoke-ID-Assign - WS-BPEL-Document

```
    ...
    <!-- Assign ID to partnerLink before Grid Service invocation
        -->
    <assign name="AssignID">
        <!-- Initialise dynamic Endpoint Reference: -->
        <!-- Copy literal XML-skeleton of EndpointReference in the
            variable DynamicEndpointReference -->
        <copy>
            <from>
                <literal>
```

```
                <wsa:EndpointReference xmlns:s="...">
                   <wsa:Address>...</wsa:Address>
                   <wsa:ServiceName PortName="...">s:...</wsa:
                      ServiceName>
                   <wsa:ReferenceProperties>
                      <!-- Custom elements defined here will be
                         mapped to the SOAP Header -->
                      <!-- The following information is to be
                         adapted to your purpose -->
                      <my:mapping-information xmlns:my="...">
                         <my:ID> </my:ID>
                      </my:mapping-information>
                   </wsa:ReferenceProperties>
                </wsa:EndpointReference>
             </literal>
          </from>
          <to variable="DynamicEndpointReference"/>
       </copy>

       <!-- Copy ID into the variable -->
       <copy>
          <from>
             <literal>ID-1234-5678-9100</literal>
          </from>
          <to variable="DynamicEndpointReference">
             <query>/wsa:EndpointReference/wsa:
                ReferenceProperties/my:mapping-information/my:
                ID</query>
          </to>
       </copy>

       <!-- Copy the variable to the partnerLink -->
       <copy>
          <from variable="DynamicEndpointReference" />
          <to partnerLink="gridServicePL" />
       </copy>
    </assign>
    ...
```

Listing 26 shows the WSDL document of the Grid Service that shall be invoked. Amongst standard WSDL elements it contains an import of the WS-Addressing schema and the definition of a message representing the dynamic endpoint reference used in the pattern in Listing 25.

Listing 26: Pattern Pre-Invoke-ID-Assign - gridServicePartnerLink WSDL-Document

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="gridService" targetNamespace="..."
    xmlns="http://schemas.xmlsoap.org/wsdl/"
```

```
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/
         addressing"
      ...>

  <!-- Import WS-Addressing schema -->
  <wsdl:types>
     <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema">
        <xsd:import namespace="http://schemas.xmlsoap.org/ws
           /2003/03/addressing" schemaLocation="addressing.xsd
           "/>
     </xsd:schema>
  </wsdl:types>

  <!-- Define appropriate message representing an
     EndpointReference-->
  <message name="DynamicEndpointReference" type="wsa:
     EndpointReferenceType"/>

  <!-- messages, portTypes, bindings, services,
     partnerLinkTypes -->
  ...
</definitions>
```

Listings 27 illustrates the `partnerLink` for the Grid Service in the ActiveBPEL-specific project deployment descriptor (PDD) document. Note that the value of the `endpointReference` attribute is *dynamic* and the value of the `invokeHandler` is *default:Address*.

Listing 27: Pattern Pre-Invoke-ID-Assign - ActiveBPEL PDD Document

```
...
<partnerLink name="gridServicePL">
   <partnerRole endpointReference="dynamic" invokeHandler="
      default:Address"/>
</partnerLink>
...
```

5.1.3 Pattern Insertion Example

Figure 5 illustrates a simple workflow that represents a loan approval process. The example workflow is a sequence-based version of the tutorial workflow that comes with both the ActiveBPEL engine and the ActiveBPEL Designer (both v.5.0.2), meaning that the top-level process element is a `sequence` instead of a `flow` element and the control flow is represented without using `links`. This means that the control flow is described explicitly by `sequences` and decisions. The informal description of the workflow is as follows:

"*The loan approval process starts by receiving a customer request for a loan amount. If*

the request is for less than $10,000, the risk assessment Web service is invoked to assess whether the customer is a low or high risk. If the customer is low risk, the request is approved. Otherwise, the request is sent to the approval service for review. If the amount is for $10,000 or more, the request is always sent to the approver for review. The customer can receive feedback from the assessor or approver." [ABP08]

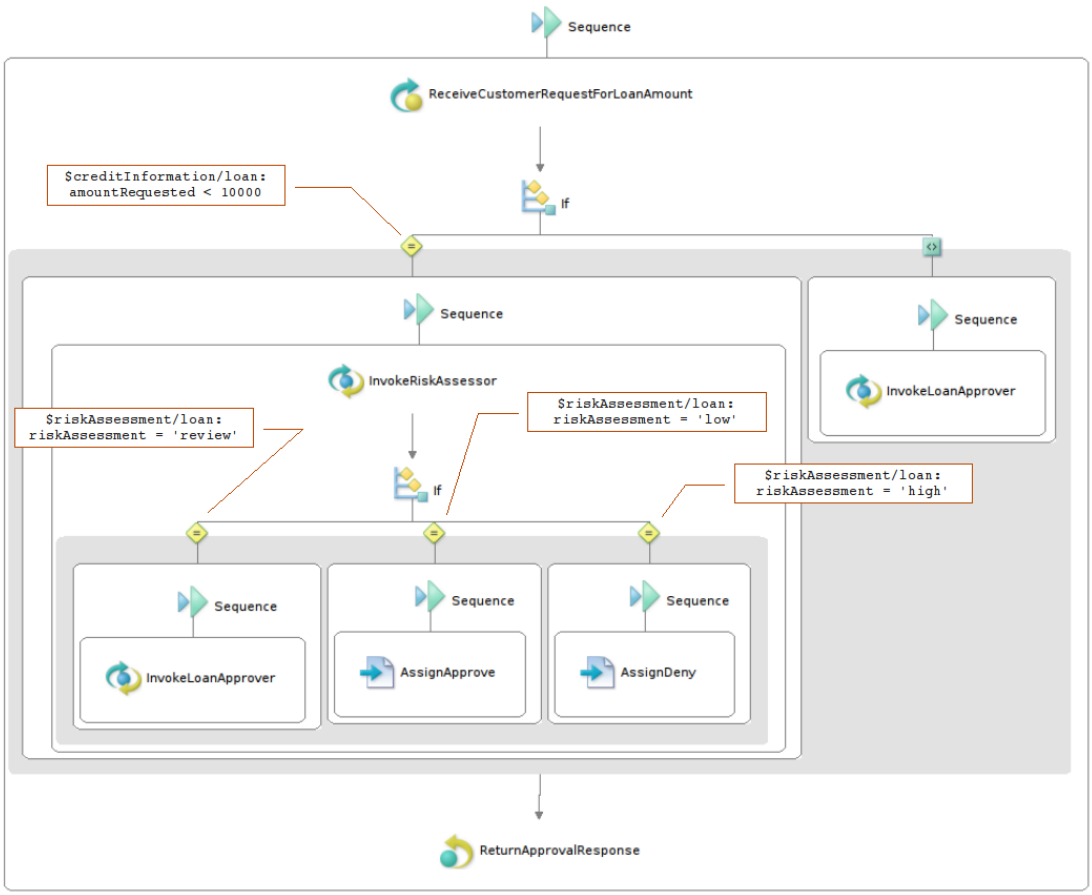


Figure 5: Loan Approval Process (ActiveBPEL Diagram)

Figure 6 shows the loan approval process after inserting the two Patterns On-Receive-ID-Retrieve and Pre-Invoke-ID-Assign. There is an `assign` activity after the instance-creating `receive` that stores the UNICORE 6 Workflow Service instance ID in a WS-BPEL-internal variable, and an `assign` activity before each `invoke` activity that prepares a dynamic endpoint reference construct, puts the stored Workflow Service instance ID into the construct, and appends the construct to the respective partner link.

Listing 28 presents an XML Schema transformation that describes the transformation of a WS-BPEL 2.0 description into an equivalent Grid-enabled WS-BPEL 2.0 description. Please note that the transformation only applies to sequence-based WS-BPEL since flow-

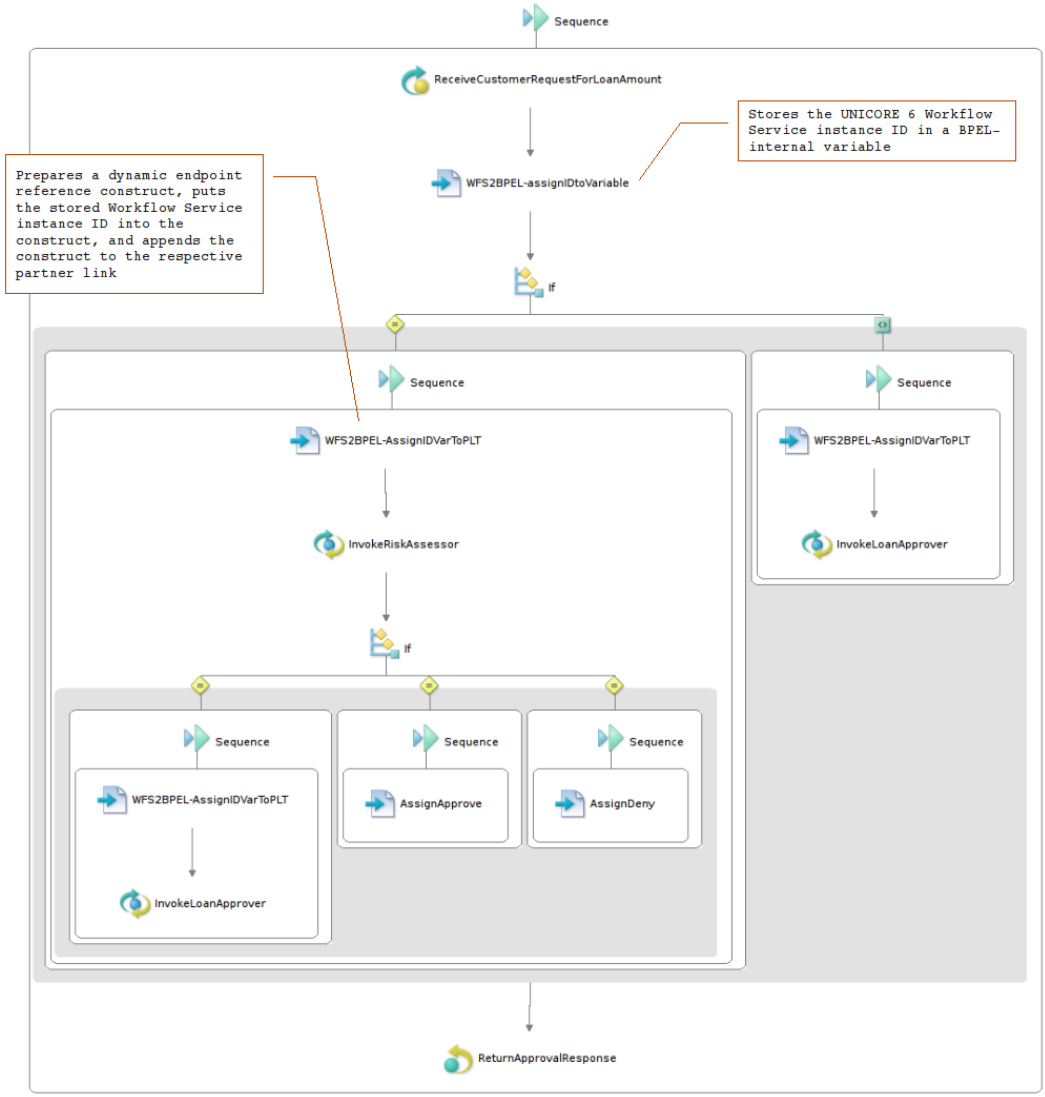


Figure 6: Grid-enabled Loan Approval Process (ActiveBPEL Diagram)

based WS-BPEL requires the link structure to be refactored when additional activities are added. The general behaviour of the transformation is simply to copy the original WS-BPEL description (1). Next, a `variables` element is inserted contains all necessary WS-BPEL `variables` if there are no `variables` in the original WS-BPEL description (2a), or the necessary WS-BPEL `variables` are inserted directly in the `variables` if the `variables` element is already existing in the original WS-BPEL description (2b). The necessary WS-BPEL `variables` are:

- *ucWorkflowServiceInstanceResourceID*. A variable that represents the ID of a Workflow Service instance that is expected to be part of a start message reaching an

instance of this WS-BPEL workflow description.

- *PartnerOperationIn.* A variable that represents the incoming start message of the workflow instance. This variable is only added if there is no such element (as specified by an instance-creating `receive` or `pick` activity's[14] `variable` attribute).

After each instance-creating activity an `assign` activity is performed that copies the resource ID from the message part of the start message to the *ucWorkflowServiceInstanceResourceID* variable (3). Also, for each instance-creating `receive` or `pick` activity the following must be done (4):

- IF the respective activity has an attribute `variable` that is not *null*, adapt the `from` part in the `assign` to be inserted after the activity corresponding to the `variable` attribute's value.
- ELSE add a `variable` attribute to the activity and a corresponding `variable` element to the `variables` of the WS-BPEL description.

Finally, the ID of the UNICORE Workflow Service instance has to be assigned to a WS-BPEL partnerLink before each Grid Service invocation. The particular step is to insert an `assign` activity before each `invoke` activity that initialises a dynamic endpoint reference by copying a literal XML-skeleton representing the type `EndpointReference` to the variable `DynamicEndpointReference`, then copies the ID into the variable, and finally copies the variable to the partnerLink of the Grid Service to be invoked.

Listing 28: WS-BPEL Grid Transformation - XSLT Document

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:transform
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
        executable"
    version="2.0">
  <xsl:output method = "xml" version="1.1"
      encoding ="UTF-8" indent = "no"/>

  <!-- 1. Copy the given BPEL file -->
  <xsl:template match="node()|@*">
    <xsl:copy>
      <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
  </xsl:template>

  <!-- 2a. Insert a 'variables' element with all necessary '
      variable' subelements if not available -->
```

[14] An instance-creating `receive` or `pick` is a WS-BPEL 2.0 `receive` or `pick` activity that has it's `createInstance` attribute set to "yes"

```
    <xsl:template match="bpel:process">
20    <xsl:copy>
        <xsl:copy-of select="@*"/>
        <xsl:if test="./not(bpel:variables)">
          <bpel:variables>
              <bpel:variable name="
                  ucWorkflowServiceInstanceResourceID" type ="xsd:
                  string"/>
25              <bpel:variable element="wsa:EndpointReference"
                    name="DynamicEndpointReference"/>
              <bpel:variable name="PartnerOperationIn" messageType
                  =""/>
          </bpel:variables>
        </xsl:if>
        <xsl:apply-templates select="node()"/>
30    </xsl:copy>
    </xsl:template>

    <!-- 2b. Insert a 'variable' element named "
        ucWorkflowServiceInstanceResourceID" that holds the
        resource ID as a value, and a 'variable' "
        PartnerOperationIn" that represents the incoming start
        message, if not specified by the instance-creating '
        receive' or 'pick' -->

35  <xsl:template match="bpel:variables">
      <xsl:copy>
        <xsl:apply-templates select="node()|@*"/>
        <bpel:variable name="ucWorkflowServiceInstanceResourceID"
            type="xsd:string"/>
        <bpel:variable name="PartnerOperationIn" messageType=""/>
40    </xsl:copy>
    </xsl:template>

    <!-- 3. After each instance-creating activity, insert an '
        assign' that copies the resource ID from the start message
         to the "ucWorkflowServiceInstanceResourceID" variable. 4.
         After each instance-creating activity, if the activity
        has an attribute 'variable', adapt the 'from' part in the
        assign. Else, add a 'variable' attribute to the activity
        and a corresponding 'variable' element to the 'variables'
        of the bpel description -->

45  <xsl:template match="bpel:receive|bpel:pick">
      <xsl:copy>
        <xsl:apply-templates select="node()|@*"/>
        <xsl:if test="./not(@variable)">
          <xsl:attribute name="variable">PartnerOperationIn</xsl:
```

```
            attribute>
      </xsl:if>
    </xsl:copy>

    <xsl:if test="./@createInstance='yes'">
      <bpel:assign name="WFS2BPEL-assignIDtoVariable" >
        <bpel:copy>
          <xsl:if test="./@variable">
            <bpel:from>string($<xsl:value-of select="./@variable
                " />.ucWorkflowServiceInstanceResourceID)</bpel:
                from>
            <bpel:to variable="
                ucWorkflowServiceInstanceResourceID"/>
          </xsl:if>
          <xsl:if test="./not(@variable)">
            <bpel:from>string($PartnerOperationIn.
                ucWorkflowServiceInstanceResourceID)</bpel:from>
            <bpel:to variable="
                ucWorkflowServiceInstanceResourceID"/>
          </xsl:if>
            </bpel:copy>
        </bpel:assign>
      </xsl:if>

  </xsl:template>

  <!-- 5. Insert an 'assign' activity before each 'invoke#
      activity that initialise a dynamic endpoint reference and
      copies it to the partner link of the Grid Service to be
      invoked. -->

  <xsl:template match="bpel:invoke">

    <bpel:assign name = "WFS2BPEL-AssignIDVarToPLT" >
      <bpel:copy>
        <bpel:from>
          <bpel:literal>
            <wsa:EndpointReference xmlns:wsa="http://www.w3.org
                /2005/08/addressing">
              <wsa:Address></wsa:Address>
              <wsa:ServiceName PortName=""></wsa:ServiceName>
              <wsa:ReferenceProperties>
                <mapping-information>
                  <ID></ID>
                </mapping-information>
              </wsa:ReferenceProperties>
            </wsa:EndpointReference>

          </bpel:literal>
```

```
          </bpel:from>
          <bpel:to variable="DynamicEndpointReference"/>
        </bpel:copy>
        <bpel:copy>
          <bpel:from variable="ucWorkflowServiceInstanceResourceID
              "/>
          <bpel:to variable="DynamicEndpointReference">
            <bpel:query>/wsa:EndpointReference/wsa:
                ReferenceProperties/my:mapping-information/my:ID</
                bpel:query>
          </bpel:to>
        </bpel:copy>
        <bpel:copy>
          <bpel:from variable="DynamicEndpointReference"/>
          <bpel:to>
            <!-- fill the 'partnerLink' attribute of the 'to'
                element with the value of the 'partnerLink'
                attribute of the original 'invoke' element-->
            <xsl:attribute name="partnerLink"><xsl:value-of select
                ="@partnerLink"/></xsl:attribute>
          </bpel:to>

        </bpel:copy>
      </bpel:assign>

      <xsl:copy>
        <xsl:apply-templates select="node()|@*"/>
      </xsl:copy>
    </xsl:template>

</xsl:transform>
```

After the transformation, the inserted endpoint reference construct must be completed by inserting information that is necessary to address the corresponding UNICORE/X Workflow Service instance. This information is the name of the service, the address of the service, the service's port name, and the namespace of the service. We refer to this information as a *UNICORE service endpoint reference*. Listing 29 shows an endpoint reference construct after insertion of an exemplary UNICORE service endpoint reference.

Listing 29: Endpoint Reference Example - Transformed WS-BPEL Document

```
...
<wsa:EndpointReference
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/
          addressing"
      xmlns:s="http://uc.service.example">
    <wsa:Address>http://demo.bisgrid.de:8000/service<</wsa:
        Address>
    <wsa:ServiceName PortName="ExampleServicePort">s:
        ExampleService</wsa:ServiceName>
```

```
        <wsa:ReferenceProperties>
           <my:mapping-information xmlns:my="http://my.infos.
              example">
              <my:ID />
           </my:mapping-information>
        </wsa:ReferenceProperties>
     </wsa:EndpointReference>
     ...
```

5.2 Pattern Bpel-Process-ID-Retrieval

With the WS-BPEL patterns discussed in Sections 5.1.1 and 5.1.2 the WS-BPEL workflow instance is able to address its corresponding Workflow Service instance in UNICORE when making an external Grid Service call. Vice versa, the BIS-Grid Workflow Service needs to know the *process ID*s of the corresponding WS-BPEL workflow instances in the ActiveBPEL engine[15]. Using this information, services running in the UNICORE/X service container can use ActiveBPEL engine services to retrieve monitoring and debugging information about the WS-BPEL workflow instances. Therefore, the availability of process IDs to the UNICORE services represents the foundation for higher level monitoring and debugging services.

This demands a WS-BPEL pattern that is partly workflow engine-specific. If the workflow engine is exchanged, this pattern must be modified according to the new WS-BPEL engine. Generally, we use the WS-BPEL `event handler` mechanism to provide a new operation to the workflow that returns the process ID. By using an event handler, such calls can be performed in parallel to the "actual" workflow processing and will not affect it. The first step is to modify the WSDL of the WS-BPEL workflow. That means, we insert two new messages, a new operation, and a second role description to the partner link type. Since this operation is a non-instance-creating operation we need a `correlation set` that enables the engine to map the message to the correct workflow instance. Here, we simply use UNICORE 6 Workflow Service instance ID that has been already send to the WS-BPEL workflow instance when it is created (cp. Section 5.1). The parts that must be added to the WSDL are shown in Listing 30.

Listing 30: Pattern Bpel-Process-ID-Retrieval - WSDL-Document

```
<wsdl:definitions name="...">
  ...
  <!-- Messages to request the process ID -->
  <wsdl:message name="getProcessIdResponse">
    <wsdl:part name="processId" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="getProcessIdRequest">
    <wsdl:part name="ucWorkflowServiceResourceId" type="xsd:
       string"/>
```

[15] A *process ID* identifies a certain WS-BPEL workflow instance in the ActiveBPEL engine.

```
    </wsdl:message>
    ...
    <!-- port type with 'getProcessId' operation -->
    <wsdl:portType name="processIdPT">
      <wsdl:operation name="getProcessId">
        <wsdl:input message="tns:getProcessIdRequest"/>
        <wsdl:output message="tns:getProcessIdResponse"/>
      </wsdl:operation>
    </wsdl:portType>

    <plnk2:partnerLinkType xmlns:plnk2="http://docs.oasis-open.org
       /wsbpel/2.0/plnktype" name="...">
      ...
      <!-- New role that offers the process ID port type -->
      <plnk2:role name="processIdProvider" portType="tns:
         processIdPT"/>
    </plnk2:partnerLinkType>

    <!-- properties for the correlation set -->
    <vprop:property xmlns:vprop="http://docs.oasis-open.org/wsbpel
       /2.0/varprop" name="CorrelateWorkflowId" type="xsd:string"
       />

    <!-- the 'ucWorkflowServiceResourceId' of the instance-
       creating message must match the message '
       ucWorkflowServiceResourceId' -->
    <vprop:propertyAlias xmlns:vprop="http://docs.oasis-open.org/
       wsbpel/2.0/varprop" messageType="tns:getProcessIdRequest"
       part="ucWorkflowServiceResourceId" propertyName="tns:
       CorrelateWorkflowId"/>
    <vprop:propertyAlias xmlns:vprop="http://docs.oasis-open.org/
       wsbpel/2.0/varprop" messageType="tns:..." part="
       ucWorkflowServiceResourceId" propertyName="tns:
       CorrelateWorkflowId"/>
</wsdl:definitions>
```

The second step is to insert code for process ID retrieval into the WS-BPEL description. Therefore, we insert a new *event handler* that gets the UNICORE 6 Workflow instance ID and returns the process ID of the corresponding WS-BPEL workflow instance. This event handler uses the newly introduced partner link, `workflowPL`. The inserted WS-BPEL code is shown in Listing 31. Figure 7 illustrates the event handler as an ActiveBPEL Designer WS-BPEL diagram.

Listing 31: Pattern Bpel-Process-ID-Retrieval - WS-BPEL-Document

```
<process>
  <!-- Partner link definitions -->
  <partnerLinks>
    ...
    <!-- partner link for workflow ID retrieval -->
```

```
    <partnerLink myRole="processIdProvider" name="workflowPL"
       partnerLinkType="wf:workflowPLT"/>
  </partnerLinks>

  <eventHandlers>
    ...
    <onEvent messageType="wf:getProcessIdRequest" operation="
       getProcessId" partnerLink="workflowPL" portType="wf:
       processIdPT" variable="getProcessIdRequest">
      <!-- the correlation set that must be true for this event
         -->
      <correlations>
        <correlation initiate="yes" set="
           getProcessIdCorrelationSet"/>
      </correlations>
      <scope>
        <variables>
          <variable messageType="wf:getProcessIdResponse" name="
             getProcessIdResponse"/>
        </variables>
        <flow>
          <links>
            <link name="sendAnswer"/>
          </links>
          <assign>
            <sources>
               <source linkName="sendAnswer"/>
            </sources>
            <copy>
              <from>abx:getProcessId()</from>
              <to part="processId" variable="
                 getProcessIdResponse"/>
            </copy>
          </assign>
          <reply operation="getProcessId" partnerLink="
             workflowPL" portType="wf:processIdPT" variable="
             getProcessIdResponse">
            <targets>
              <target linkName="sendAnswer"/>
            </targets>
          </reply>
        </flow>
      </scope>
    </onEvent>

  </eventHandlers>
  ...
</process>
```

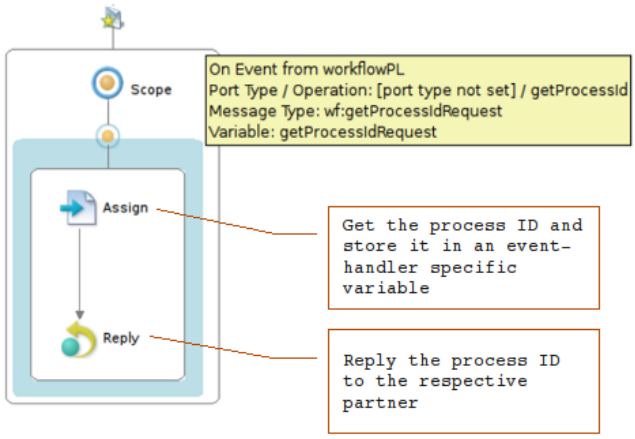


Figure 7: Pattern Bpel-Process-ID-Retrieval - Event Handler (ActiveBPEL Diagram)

In Listing 32 the XML Schema transformation is shown that inserts the pattern into a standard sequence-based WS-BPEL document. The general behaviour is that the whole WS-BPEL description is copied (1). Also, a new `partnerLink` is inserted that represents the caller (2a), and a new *event handler* is inserted that expects the UNICORE 6 Workflow Service instance ID and returns the ActiveBPEL workflow/process instance ID (2b). This event handler uses the previously introduced `partnerLink`.

Listing 32: Process ID Retrieval Transformation - XSLT Document

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:transform
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
          executable"
      xmlns:wf="http://bisgrid.d-grid.de/workflow"
      version="2.0">
   <xsl:output method = "xml" version="1.1"
      encoding ="UTF-8" indent = "no"/>

   <!-- 1. Copy the given BPEL file -->
   <xsl:template match="node()|@*">
      <xsl:copy>
         <xsl:apply-templates select="node()|@*"/>
      </xsl:copy>
   </xsl:template>

   <!-- 2a. Insert a new 'partnerLink' that represents the
       caller -->

   <xsl:template match="bpel:partnerLinks">
```

```
        <xsl:copy>
            <xsl:apply-templates select="node()|@*"/>
                <bpel:partnerLink myRole="processIdProvider" name="
                    workflowPL"
                        partnerLinkType="wf:workflowPLT"/>
        </xsl:copy>
    </xsl:template>

    <!-- 2.2 Insert a new event handler that expects the UNICORE
        6 Workflow Service instance ID and returns the ActiveBPEL
         process instance ID. If there is no 'eventHandlers'
        element in the process, add it and the respective 'event
        handler'. -->
    <xsl:template match="bpel:process">
        <xsl:copy>
            <xsl:copy-of select="@*"/>
            <xsl:apply-templates select="node()"/>

            <xsl:if test="./not(bpel:eventHandlers)">
                <bpel:eventHandlers>
                    <bpel:onEvent messageType="wf:getProcessIdRequest
                        " operation="getProcessId"
                            partnerLink="workflowPL" portType="wf:
                                processIdPT"
                            variable="getProcessIdRequest">
                        <bpel:correlations>
                            <bpel:correlation initiate="yes" set="
                                getProcessIdCorrelationSet"/>
                        </bpel:correlations>
                        <bpel:scope>
                            <bpel:variables>
                                <bpel:variable messageType="wf:
                                    getProcessIdResponse"
                                        name="getProcessIdResponse"/>
                            </bpel:variables>

                            <bpel:flow>
                                <bpel:links>
                                    <bpel:link name="sendAnswer"/>
                                </bpel:links>

                                <bpel:reply operation="getProcessId"
                                    partnerLink="workflowPL"
                                        portType="wf:processIdPT" variable
                                            ="getProcessIdResponse">
                                    <bpel:targets>
                                        <bpel:target linkName="sendAnswer"
                                            />
                                    </bpel:targets>
```

```
                        </bpel:reply>

60                      <bpel:assign>
                           <bpel:sources>
                              <bpel:source linkName="sendAnswer"
                                 />
                           </bpel:sources>
                           <bpel:copy>
65                            <bpel:from>abx:getProcessId()</
                                 bpel:from>
                              <bpel:to part="processId" variable
                                 ="getProcessIdResponse"/>
                           </bpel:copy>
                        </bpel:assign>

70                   </bpel:scope>
                  </bpel:onEvent>
               </bpel:eventHandlers>
            </xsl:if>
         </xsl:copy>
75    </xsl:template>

      <!-- If there is an 'eventHandlers' element in the process,
         add the respective 'event handler' only -->
      <xsl:template match="bpel:eventHandlers">
         <xsl:copy>
80          <xsl:apply-templates select="node()|@*"/>

            <bpel:onEvent messageType="wf:getProcessIdRequest"
               operation="getProcessId"
                  partnerLink="workflowPL" portType="wf:processIdPT
                     "
                  variable="getProcessIdRequest">
85             <bpel:correlations>
                  <bpel:correlation initiate="yes" set="
                     getProcessIdCorrelationSet"/>
               </bpel:correlations>
               <bpel:scope>
                  <bpel:variables>
90                   <bpel:variable messageType="wf:
                        getProcessIdResponse" name="
                        getProcessIdResponse"/>
                  </bpel:variables>

                  <bpel:flow>
                     <bpel:links>
95                      <bpel:link name="sendAnswer"/>
                     </bpel:links>
```

```
                    <bpel:reply operation="getProcessId"
                        partnerLink="workflowPL"
                            portType="wf:processIdPT" variable="
                                getProcessIdResponse">
100                     <bpel:targets>
                            <bpel:target linkName="sendAnswer"/>
                        </bpel:targets>
                    </bpel:reply>

105                 <bpel:assign>
                        <bpel:sources>
                            <bpel:source linkName="sendAnswer"/>
                        </bpel:sources>
                        <bpel:copy>
110                         <bpel:from>abx:getProcessId()</bpel:
                                from>
                            <bpel:to part="processId" variable="
                                getProcessIdResponse"/>
                        </bpel:copy>
                    </bpel:assign>
                </bpel:flow>
115
            </bpel:scope>
        </bpel:onEvent>
    </xsl:copy>
</xsl:template>
120
</xsl:transform>
```

5.3 ActiveBPEL Deployment Descriptor Service Binding

There are two important possibilities to describe service bindings in the ActiveBPEL deployment descriptor (PDD). Either, the service to be bound is *static* or the service is *dynamic*. Regarding the services in the BIS-Grid engine, there are factory services that create corresponding *Workflow Management Service* and *Workflow Service* instances. Both are located within the UNICORE/X service container, being realised as standard Web Services or WSRF Grid Services.

Within the ActiveBPEL deployment descriptor, a factory service can be bound by describing its `partnerLink` as a static endpoint reference - provided that the service is realised as a standard Web Service. This is illustrated in the first example in Listing 33: The value of the `parnerRole`'s `endpointReference` of the factory service `partnerLink` is set to *static* and an `Address`, a `ServiceName`, and the appropriate `ReferenceProperties` element are specified. Corresponding to Listing 27, the second example in Listing 33 illustrates the dynamic service binding of a Grid Service: The `endpointReference` attribute of the `partnerRole` element is set *dynamic*, and the `invokeHandler` attribute's value is *Address*, indicating that the service location (that is, the endpoint reference)

is provided dynamically within the WS-BPEL process. Thereby, the location must be assigned dynamically within a `copy` operation of an `assign` activity.

Listing 33: ActiveBPEL PDD Document

```
<process ... xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/
    addressing" ...>
    ...
    <!-- Example 1: Static service binding -->
    <partnerLink name="gridServiceFactoryPL">
        <partnerRole endpointReference="static" invokeHandler="
            default:Address">
            <wsa:EndpointReference xmlns:s="http://bisgrid.d-grid.
                de/wsdl/factoryservice" >

                <wsa:Address>
                    http://my.example.com/FactoryService</wsa:
                        Address>

                <wsa:ServiceName PortName="SOAPPort">
                    s:FactoryService</wsa:ServiceName>

                <wsa:ReferenceProperties>
                    <wsa:To xmlns:wsa="http://schemas.xmlsoap.org/ws
                        /2004/03/addressing">
                        http://my.example.com/FactoryService
                    </wsa:To>
                </wsa:ReferenceProperties>

            </wsa:EndpointReference>
        </partnerRole>
    </partnerLink>

    ...
    <!-- Example 2: Dynamic service binding -->
    <partnerLink name="gridServicePL">
        <partnerRole endpointReference="dynamic" invokeHandler="
            default:Address"/>
    </partnerLink>
    ...
</process>
```

6 Conclusion

In this document we describe design patterns that are used to create BIS-Grid-compatible Grid workflows based on the WS-BPEL language. Such Grid workflows are essentially Grid Service orchestrations to be deployed in the BIS-Grid engine [HSG+07], consisting of the Grid middleware UNICORE 6, BIS-Grid-specific service extensions to UNICORE 6, and the WS-BPEL engine ActiveBPEL. The patterns differ in their application: Grid Service utilisation on the one hand and implementation-specific aspects, especially the integration of ActiveBPEL with the UNICORE 6 service extensions, on the other hand. Also, we introduced the fundamental standards and technologies that are required for understanding and applying the patterns, and we described the prerequisites that apply to the patterns presented in this document.

Since the development of the BIS-Grid engine is ongoing work at the point where this document is created, there may be updates of this document in near future. These updates may include patterns that cover the aspect human interaction, since human interaction is essential in the workflow scenarios regarded in the BIS-Grid project, and since human interaction will be regarded in future service extensions to the BIS-Grid engine. Nevertheless, when becoming necessary, additional implementation-specific patterns, grid utilisation patterns, and other patterns may also be included in future document iterations.

References

[ABP08] ActiveBPEL Designer UserŠs Guide. Technical report, Active Endpoints, Inc., March 2008. Version 5.0.1.

[Ban06] Tim Banks. Web Services Resource Framework (WSRF) - Primer v1.2. http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf, May 2006.

[BPE06] Business Process Execution Language for Web Services Version 1.1. `ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf`, 2006. Letzter Zugriff: 2006-11-15.

[CHKT05] S. Conrad, W. Hasselbring, A. Koschel, and R. Tritsch. *Enterprise Application Integration.* Elsevier, 2005.

[DOE07] Grid Workflow Modelling Using Grid-Specific BPEL Extensions. German e-Sience Conference 2007, May 2007.

[EBC+06] Wolfgang Emmerich, Ben Butchard, Liang Chen, Sarah L. Price, and Bruno Wassermann. Grid Service Orchestration Using the Business Process Execution Language (BPEL). (3):283–304, 2006.

[ESCR07] 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.

[GHH+08] Stefan Gudenkauf, Wilhelm Hasselbring, Felix Heine, André Höing, Guido Scherp, and Odej Kao. Bis-Grid: Business Workflows for the Grid. In Marian Bubak, Michal Turala, and Kazimierz Wiatr, editors, *CGW'07 Proceedings*, pages 86–94, Krakow, Poland, 2008. ACC CYFRONET AGH.

[GKM+06] Steve Graham, Anish Karmarkar, Jeff Mischinsky, Ian Robinson, and Igor Sedukhin. Web Services Resource 1.2 (WS-Resource). `http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf`, April 2006. OASIS Standard.

[GT06] Steve Graham and Jem Treadwell. Web Services Resource Properties 1.2 (WS-ResourceProperties). `http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf`, April 2006. OASIS Standard.

[HGS08] Andre Höing, Stefan Gudenkauf, and Guido Scherp. BIS-Grid Deliverable 3.2: Documentation - WS-BPEL Engine. Technical report, BIS-Grid, August 2008.

[HHG+07] Felix Heine, Andre Höing, Stefan Gudenkauf, Guido Scherp, Holger Nitsche, and Jens Lischka. BIS-Grid Deliverable 3.1: Specification. Technical report, BIS-Grid, December 2007.

[HSG+07] André Höing, Guido Scherp, Stefan Gudenkauf, Felix Heine, Holger Nitsche, and Jens Lischka. BIS-Grid - Betriebliche Informationssysteme: Grid-basierte Integration und Orchestrierung - Deliverable 3.1 Work Package 3: WS-BPEL Engine. 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 and Universität Paderborn, Paderborn Center for Parallel Computing, 2007.

[Kel07] Marc Kelderman. Using custom headers in BPEL. http://orasoa.blogspot.com/2007/09/using-custom-headers-in-bpel.html [last visited 2008/05/08], September 2007.

[Ley06] Frank Leymann. Choreography for the Grid: towards fitting BPEL to the resource framework: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1201–1217, 2006.

[LM06] Lily Liu and Sam Meder. Web Services Base Faults 1.2 (WS-BaseFaults). `http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf`, April 2006. OASIS Standard.

[MSB06] Tom Maguire, David Snelling, and Tim Banks. Web Services Service Group 1.2 (WS-ServiceGroup). `http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf`, April 2006. OASIS Standard.

[MT] Tapio Niemi Matti Heikkurinen Miika Tuisku, Juho Karppinen. Java Enterprise Grids using JBoss middleware. `http://gridblocks.hip.fi/opencms/export/sites/gridblocks/downloads/J2EE_JEMS_Grid.pdf`. last vistied 17.12.2007.

[OAS07] OASIS WSBPEL Technical Committee. Web Services Business Process Execution Language (WSBPEL) Primer. May 2007.

[OHM+04] D. Orchard, S. Holbrook, J. Marsh, M. Goodner, and E. Gutentag. WS-Addressing Submission Request to W3C. Technical report, Systems, IBM, Microsoft, SAP AG, and Sun Microsystems, Inc, August 2004.

[SB06] Latha Srinivasan and Tim Banks. Web Services Resource Lifetime 1.2 (WS-ResourceLifetime). `http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf`, April 2006. OASIS Standard.

[TCF+03] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselmann, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt. Open Grid Services Infrastructure (OGSI). 2003.

[W3C] W3C Working group. XML Schema 1.1. `http://www.w3.org/XML/Schema`.

[W3C07] W3C Working group. XML Path Language (XPath) 2.0. `http://www.w3.org/TR/xpath20/`, January 2007. W3C Recommendation.

[Web] Web Services Addressing Working Group. Web Services Addressing 1.0. `http://www.w3.org/2002/ws/addr/`. W3C Recommendation.

[wsn] OASIS Web Services Notification (WSN) TC. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn`.