# Composing Services on the Grid Using BPEL4SWS

Jörg Nitzsche (corresponding author)

Tammo van Lessen

Dimka Karastoyanova

Frank Leymann

Institute of Architecture of Application Systems

University of Stuttgart

Universitaetsstrasse 38, 70569 Stuttgart, Germany

Tel.: +49(0)711 7816 -486/-487/-476/-470

Fax: +49(0)711 7816 -472

{joerg.nitzsche,tammo.van.lessen,dimka.karastoyanova,

frank.leymann}@iaas.uni-stuttgart.de

http://www.iaas.uni-stuttgart.de

20th June 2008

**Abstract**

Service composition on the Grid is a challenging task as documented in existing research work. Even though there are initial attempts to use the Business Process Execution Language (BPEL) to compose services on the Grid, still there is a significant lack of flexibility and reusability needed in scientific applications. In this paper we present BPEL for Semantic Web Services (BPEL4SWS) - a language that facilitates the orchestration of Grid Services exposed as traditional Web Services or Semantic Web Services using a process-based approach. It is based on the idea of WSDL-less BPEL and incorporates semantic descriptions of process activity implementations which increases the flexibility of business workflows as well as scientific workflows. Following an approach that uses a set of composable standards and specifications, BPEL4SWS is independent of any Semantic Web Service framework and therefore can also utilize any kind of Semantic Grid services. The advantages of BPEL4SWS are: (1) compliance with standards, (2) independence on service technologies, (3) applicability for both business applications as well as scientific workflows that use Grid services, (4) improved flexibility of processes.

**Keywords**: Grid, BPEL, BPEL4SWS, Grid Services, Web Services, Semantic Web Services, scientific workflows

# 1 Introduction

Web Service (WS) [38] technology is one implementation supporting the construction of applications according to the style of a service oriented architecture (SOA) [5, 21]. The primary advantage of Web Services is to aid

application integration. Composing WSs is currently enabled mainly by a process-based approach [24] embodied by the de facto standard BPEL (Business Process Execution Language) [1]. The service oriented approach has been successfully applied to Grid [12] environments by exposing Grid services as Web Services. Even though the requirements of scientific applications utilizing the grid differ from those of business applications [34] it has been shown by various research results that using the service abstraction is a very suitable approach when composing Grid services in complex scientific workflows [10, 33, 6, 35, 9]. The benefits of the process-oriented approach towards service composition have been fully utilized, however the WS-centric nature of BPEL has shown some deficiencies when addressing the requirements of Grid service compositions.

BPEL uses WSDL [7] descriptions to identify partner services, i.e. services are identified by port types and operations in the process models. As a result only services that implement a concrete interface can be used as implementations for a task in a process, which presents a major deficiency of BPEL.

One approach to weaken the focus of WSs on interfaces has evolved from the Semantic Web - the Semantic Web Service (SWS) technology. The most prominent SWS frameworks are the Web Ontology Language for Services (OWL-S) [28] and the Web Service Modeling Ontology (WSMO) [22]. The SWS technology introduces an additional level of abstraction and can be considered as an integration layer on top of Web Services. Instead of a syntactic description of a WS a declarative description of the service functionality is given. Similarly, the Grid computing infrastructures are

currently being extended with approaches borrowed from the Semantic Web and even Web 2.0 [25], which form a set of technologies that are meant to enable the Semantic Grid [32, 25]. Nevertheless, the major focus is still on enhancing the available Grid resources with semantic information (approaches to associating this information, languages for describing the meaning of Grid resources, registries for Semantic Grid Services, etc.) Still, not much attention is being paid to enabling composition of Grid resources and services in a more flexible manner. So far, scientific computations on the Grid are hard-coded sequences of resource utilisation and computational task submissions, according to a predefined algorithm, which frequently changes due to developments in the respective scientific area. IT specialists support scientists in this endeavour with the help of existing enterprise-strength BPEL modelling tools and run time environments, however the process is tedious and time-consuming due to the differences in knowledge and expertise on both sides.

In this paper we advocate the use of BPEL for composition of Grid services for the purposes of, for example, scientific workflows and life science applications. We argue though that BPEL will bring enormous benefits with respect to improved reusability and flexibility only if it enables composition of not only Grid services exposed as WSs. To improve the usability of Grid service compositions and the flexibility of building and executing them – characteristics demanded by Grid applications [10, 34] – we draw upon techniques developed for the Semantic Web and Semantic Web Service technology. We therefore present an extension to the BPEL language, that combines such techniques, which in combination address the above requirements. Chal-

lenges ensuing from the need to deal with large amounts of data in scientific applications are out of the scope of this paper.

The necessary background information about BPEL is provided in section 2. In section 3 we present BPEL4SWS, a language which enables describing activity implementations in a machine processable manner using Semantic Web technologies, as an alternative to specifying WS interfaces, i.e. it enables the use of Semantic Web Services – in addition to traditional Web Services. The BPEL4SWS framework exhibits and maintains the composability characteristics of the WS technology. BPEL4SWS processes can compose semantic WSs and conventional WSs intermixed within a single process; our assumption here is that the service abstraction is valid on the Grid and that any functionality can be considered as a service. To enable this, BPEL4SWS provides a grounding mechanism that contributes to maintaining BPEL4SWS processes compliant to standard WS-based communication. In our presentation we focus on the main aspects of BPEL4SWS and explain how they make up the functionality the language provides. These aspects are: (i) a WSDL-less interaction model ($\text{BPEL}^{\text{light}}$) for describing the process logic, (ii) the annotation of SWS descriptions (such as WSMO and OWL-S) to support semantic discovery, (iii) the usage of WS-* technology for the invocation of services and (iv) the usage of SAWSDL [11] to provide a seamless mapping between the XML representation of data and its ontological representation. The interplay of the different aspects is then illustrated in section 4 by means of four basic scenarios. We show the architecture of a BPEL4SWS engine in section 5 and make an overview of a prototypical implementation in section 6. Finally we identify related work in the fields of Semantic Grid and BPEL

extensibility (section 7) and state our findings in section 8.

# 2 BPEL

BPEL is the de facto standard for specifying business processes in a WS world. It enables both, the composition of WSs [38] and rendering the composition itself as WSs. Thus, BPEL provides a recursive aggregation model for WSs. The composition of WSs can be specified as a flow between WS operations. Therefore BPEL provides several so called *structured activities* that prescribe the control flow between the *interaction activities* that model interactions with other WSs. BPEL does not support explicit data flow; instead, data is stored in shared variables that are referenced and accessed by interaction activities and data manipulation activities (e.g. `<assign>` activity). The control flow between activities can be structured either in a block-based manner by nesting structured activities like `<sequence>` (for sequential control flow), `<flow>` (for parallel control flow) and `<if>` (for conditional branches in the control flow) activities, or graph-based by defining `<links>` (i.e. directed edges) between activities in a `<flow>` activity; both styles can be used intermixed.

In order to enable communication that is compliant to the Basic Profile [3] of the WS-Interoperability Organization (WS-I)[1], i.e. without using WSDL operations of type *notification* and *solicit-response*, BPEL introduces the concept of a partner link type which is defined as an extension to WSDL. A partner link type defines two roles in terms of port types, namely a port

[1] http://www.ws-i.org/

type the process offers to a partner and a port type the process requires from the corresponding partner, and binds them together. The operations of type *notification* and *solicit-response* of a role are expressed as operations of type *one-way* and *request-response*, the other role has to provide.

Figure 1 shows an example of such a partner link type. It defines a channel (*salesPLT*) between two abstract business partners (roles) called *buyer* and *seller* through which the partners exchange messages; these roles are defined as port types, in the example *buyerPT* and *sellerPT*. In cases of only one partner invokes the other partner the corresponding partner link type contains a single role. To establish a contract (i.e. an agreement between two partners which message channel to use), BPEL's partner links reference a partner link type and specify which role is taken by the process itself (`myRole`) and which role is taken by the partner (`partnerRole`).

The interaction activities [38] (`<receive>`, `<reply>`, `<invoke>` and `<pick>`) and the event handlers are used to define the actual message exchange corresponding to a partner link, i.e. data transmitted and style of communication (blocking vs. non-blocking). For that purpose, interaction activities reference a partner link and a WSDL operation. Receiving activities (i.e. `<receive>` and `<pick>`) as well as the event handler implement an operation of type one-way of the process' port type. In combination with the `<reply>` activity receiving activites can also implement a request-response operation of the process's port type. The `<invoke>` activity references an operation of the partner's port type, and implements either a request-response or a one-way operation.

The use of request-response operations for communication in BPEL is

illustrated in Figure 2 and Figure 3. The former depicts an invocation of the process realized by a <receive>-<reply> pair. Both activities reference an operation the process offers. The corresponding partner link only specifies myRole. The latter shows how a blocking invocation of a service can be modelled: The <invoke> activity of the process uses an operation provided by the partner service. In this case only the partnerRole of the partner link is specified. In the remainder of the paper this style of communication is also referred to as synchronous communication.

Figure 4 illustrates the use of a <receive> and an <invoke> activity to model the invocation of a process using two one-way operations, one provided by the partner service and one by the process as call back. The partner link used within this example references the partner link type given in Figure 1 and defines myRole="seller" and partnerRole="buyer". An example of an non-blocking invocation of a partner using two one-way operations is shown in Figure 5. In this example the partner link type presented in Figure 1 is also used but the partner link defines myRole="buyer" and partnerRole="seller". This modelling style is used in case of long-running processes/services because the WS-I basic profile also regulates that an HTTP binding must be used. A request-response operation with HTTP binding implemented by a <receive>-<reply> activity pair or an <invoke> would simply result in a time out. In the remainder of the paper this style of communication is also referred to as asynchronous communication.

The <pick> activity and the <eventHandler> play a special role with respect to the WSDL dependency since they do not depend on WSDL itself but encapsulate elements which references a WSDL operation, the

<onMessage> element and <onEvent> element respectively.

# 3 BPEL4SWS

As shown in the previous section BPEL makes use only of WSs to enable service composition. Partner interfaces are described using WSDL; they are hard-coded within the process logic. As a result, only services that implement the WSDL interface used in the BPEL definition can be used as activity implementations; services that provide the same functionality but implement other interfaces cannot be used. This hampers integration of functionally equal services.

Semantic Web Services describe services not only in terms of an interface but rather describe their functionality and capability using rich conceptual frameworks based on ontologies, i.e. in a machine processable manner, which for instance enables mediation between the description of the goal a consumer has and the functionality a service provides. For that reason Semantic Web Services increase the level of integration and can be considered an integration layer on top of Web Services where services are discovered based on their functionality and not based on their signature.

To enable the usage of Semantic Web Services technology within Grid processes there is a need for a process language that does not specify partner services using their WSDL description, but rather allows using higher level semantic descriptions. BPEL$^{light}$ [29] decouples process logic and interface definition (but still is applicable in a WS-* environment) and therefore makes for a good candidate as a basis for a process execution language for Semantic

Web Services.

Indeed, BPEL4SWS uses BPEL$^{light}$ as basis and allows to attach SWS descriptions to BPEL$^{light}$ so that SWS frameworks like OWL-S and WSMO and corresponding implementations can be used to discover and select SWS that implement the functionality required for an activity. In addition both, the SWS description and the process itself are partly grounded to WSDL to facilitate WS-* based communication (see section 3.3). Current SWS frameworks use ontologies as data model to facilitate semantic discovery. For that reason, SAWSDL is used to enable a seamless mapping of data between its XML representation and its ontological representation.

## 3.1 BPEL$^{light}$

BPEL$^{light}$ extends BPEL 2.0 via additional elements in a separate namespace[2] which act as a replacement for the WSDL-based interaction model. It provides a WSDL independent interaction model and (re-)introduces the concept of a partner.

BPEL$^{light}$ defines a mechanism to describe the communication between partners without any dependency on WSDL. Therefore it introduces an element, the `<conversation>` element, that facilitates grouping of interaction activities/messages and thus enables defining a complex message exchange between partners. Definitions of `<conversation>`s are encapsulated in a `<conversations>` element which is defined as a child of the `<process>` element.

BPEL 1.1 has the notion of a partner that comprises multiple partner

[2] `xmlns:bl=http://iaas.uni-stuttgart.de/2007/BPELlight`

links. However, the <partner> element has been removed in BPEL 2.0 because grouping partner links is considered a deployment issue and the partner is not evaluated during runtime but rather is only used for documentation purpose. BPEL$^{light}$ reintroduces the notion of a partner. Like a conversation, the partner element groups several messages, which may belong to different conversations. A partner in BPEL$^{light}$ enables expressing that multiple messages have to be exchanged with one and the same business entity. Additionally, BPEL$^{light}$ enables naming the partner and thus identifying a concrete organisation. Within a process multiple partners can be specified. BPEL$^{light}$ includes an extension of the <assign> activity that enables copying a partner identification into the <partner> element. Therefore the <to> specification is extended with a partner attribute that defines to which partner definition the concrete partner instance information (business entity) is copied. This is similar to copying an endpoint reference to a partner link in conventional BPEL.

In addition, WSDL independent interaction activities are needed. Due to the fact that the partnerLink and operation attribute of interaction activities and event handlers defined in BPEL are mandatory, these activities have a WSDL dependency. Thus, BPEL$^{light}$ introduces new interaction activities without the WSDL dependency using the <extensionActivity> mechanism. An <interactionActivity> can be configured such that it behaves like any of the basic interaction activities BPEL defines (<receive>, <reply> and <invoke>). Additionally, BPEL$^{light}$ defines WSDL independent <pick> and <eventHandler> elements. A set of interaction activities, that form a message exchange pattern, are grouped using the

<conversation> element. This way BPEL$^{light}$ enables modelling arbitrary message exchange patterns or service interaction patterns [4].

## 3.2 Attachment of SWS Descriptions

According to the composable approach we take for BPEL4SWS we do not only allow encode the semantic descriptions within the BPEL4SWS process model. Also WS-PolicyAttachment [2] can be used to add semantic annotations. In general, annotations can be attached anywhere, i.e. on the activity level as well as on the conversation level. We advocate attaching the SWS descriptions to conversations. The meaning of the semantic annotations on the conversation level is described in the following sections.

In general a BPEL$^{light}$ conversation can be multilateral, however, since all existing SWS frameworks only deal with bilateral message exchanges we consider a conversation bilateral in the remainder of the paper. In BPEL4SWS there are two different categories of conversations, *providing* and *consuming*. A *providing* conversation is a conversation with a partner via which the partner uses a service the process provides, a *consuming* conversation is a conversation with a partner, where the process uses a service provided by a partner.

### 3.2.1 Using OWL-S Attachments

OWL-S (Web Ontology Language for Services) [28] was the first approach towards describing services semantically. It uses ontologies as data model and describes a service in terms of Service Profile, Service Model and Service Grounding. The capabilities of a service in terms of *inputs*, *outputs*,

*preconditions* and *effects* are described in the Service Profile. The service model describes in which order messages have to be exchanged to consume the service's functionality and the Service Grounding defines which WSDL operations of a concrete service have to be used to exchange these messages.

OWL-S describes the execution of a Web service as a collection of remote procedure calls. We argue that this is only correct for a small percentage of the cases in Grid processes [30] since typically processes are long-running and require asynchronous communication. OWL-S describes a self-contained service and has no notion of two partners (requester and provider) that provide means to invoke each other (compare partner link). It is designed to ground to all four kinds of WSDL (1.1) operations: one-way, request-response, notification and solicit-response. However, due to the WS-I Basic Profile [3] the WSDL operations of type notification and solicit-response must not be used. The lack of a partner model is the major deficiency of OWL-S since WS-* based and WS-I compliant asynchronous communication, i.e. compliant to the basic profile, is not considered.

Nevertheless, OWL-S can be used in the context of BPEL4SWS, but only for the cases where short-running services are invoked using synchronous communication: an OWL-S service is attached to a conversation and the OWL-S service model describes the sequence of BPEL4SWS interaction activities associated with the conversation. An OWL-S description attached to a 'providing' conversation is grounded to the WSDL interface that describes the process, an OWL-S description attached to 'consuming' conversations is not grounded, because the WSDL interface implemented by the partner service is assumed to be unknown during design time. This way dynamic

service discovery independent of WSDL port types is enabled.

### 3.2.2 Using WSMO Attachments

Compared to OWL-S, WSMO (Web Service Modeling Ontology) is the more promising approach because its conceptual model enables standards (WS-*) based asynchronous communication. WSMO distinguishes between the description of a service (WSMO Web Service) and the description of requirements a client has on a service (WSMO Goal). Both descriptions are based on ontologies and contain a functional description that semantically describes what a service provides or a client requests in terms of *preconditions*, *assumptions*, *postconditions* and *effects* and an interface description, the so called WSMO choreography [31]. Thus WSMO enables expressing what kind of functionality a service provides, which message exchange is needed to consume its functionality as well as what a client aims to achieve and which message exchange the client will have. This message exchange can be grounded to WSDL operations (of type request-response and one-way) on every side of an interaction in such a way that both synchronous and asynchronous communication is enabled in a standards (WS-*) based and interoperable manner (i.e. WS-I compliant).

Listing 1 shows an example of a WSMO Web Service description [20]. It describes a part of the Amazon Web Service[3] which takes a search request for an item as input and provides a search result as output.

In order to use the WSMO framework for BPEL4SWS, WSMO goals are

[3] http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl

attached to 'consuming' conversations and Semantic Web Service descriptions are attached to 'providing' conversations. The choreography of goal and Web service describe the sequence of the BPEL4SWS interaction activities associated with the conversations they are attached to. The choreography presented in Listing 1 for instance corresponds to a BPEL process with a receive and subsequent reply activity like the one presented in figure 2. The "in" mode describes the receive activity, the "out" mode the reply activity and the transition rule describes the flow between these two messages. The same SWS description can be used to describe the conversation of the BPEL4SWS process presented in section 4.1. This way the process can be discovered by WSMO implementations using its WSMO Web Services description. Activity implementations of BPEL4SWS can be discovered by submitting the attached WSMO goal to a WSMO implementation. Similarly to OWL-S, the discovery of both, the process itself and its activity implementations is independent of WSDL port types.

To support the goal based communication features BPEL4SWS provides, a WSMO implementation like WSMX [15] which is part of the Semantic Service Bus (SSB) presented in [19] has to provide several operations [30]. To establish a conversation with a partner service that is able to fulfil a WSMO Goal, the `registerCommunication(goal):context` operation can be used. The returned context identifies the created conversation in the bus. Sending messages via an already established conversation to a partner service hosted by the SSB is enabled by two different operations: `invokeWebService(context, data):data` for blocking communication and `invokeWebService(context, data)` for non-blocking com-

munication.

The partner element in BPEL4SWS can be used to constrain that a partner has to satisfy multiple goals. Whenever a partner has to be discovered, a list of goals is sent to the entry point `findPartner(list of goals):partner` [30] of a WSMO implementation. And whenever a conversation starts that belongs to a specific partner, the entry point `registerCommunication(partner, goal):context` is invoked. The communication between the process and the service(s) is then conducted using the afore mentioned entry points.

In general, implementations of other SWS frameworks that provide similar communication features and that can be integrated easily into the SSB are also appropriate for supporting the service discovery task required when executing BPEL4SWS processes. However, since WSMO currently is the most suitable SWS framework for use in Grid processes due to its support for asynchronous communication we focus on using the WSMO framework for discovery of Semantic Web Services.

## 3.3 Grounding to WSDL

BPEL4SWS uses (semantic) Web Services as activity implementations and is exposed as Web Services as well as Semantic Web Services which are in turn grounded to WSs (compare Listing 1). Thus, the BPEL$^{\text{light}}$ interaction model also has to be partly grounded to WSDL. To preserve the decoupling of process logic and activity implementation definition this is done within an artefact called *grounding file* and not within the BPEL$^{\text{light}}$ description.

The grounding for the *consuming* conversation that are annotated with

a WSMO goal only specifies which WSDL operations are provided by the receiving activities. This is illustrated in Listing 2. This way, an engine implementation can resolve an incoming message to a certain activity within the process model. An `<invoke>`-like `<interactionActivity>` simply sends (and receives) a message to (and from) an SSB.

For *consuming* conversations without goal attachment and for *providing* conversations in general the grounding is more complex (see Listing 3). In this case the conversation has to be grounded to a partner link type which is required to support WSDL based asynchronous communication. Therefore, it has to be specified which role of the partner link type the partner service and the process itself take. In addition to the grounding of the conversation to the partner link type, all interaction activities, including the invoking ones have to be grounded to WSDL operations.

Using this *full* grounding for *consuming* conversations means that traditional Web Services are used. In this case a WSMO goal must not be attached to the conversation. Whenever a sending activity is performed, the engine implementation looks up the grounding file for the operation it has to invoke and whenever a message is received it can be dispatched to an activity in the process model using the information given in the grounding file.

By specifying also the partner role for a *providing* conversation, a process that is exposed as a Semantic Web Service can also be consumed like a traditional WS. The grounding file specifies how incoming messages are resolved to activities and which operations have to be used by sending activities.

In case the process is discovered using its SWS description and asyn-

chronous communication is used, the process does not call back (invoke) the partner directly using the operation specified in the grounding file but rather sends (and receives) a message to (and from) an SSB.

Enabling exposing a process as both, WSDL service and SWS is of utmost importance because if the process would only be exposed as a SWS, and invocation via traditional WS technology would not be supported, most of the clients (not supporting Semantic Web Service technology) would not be able to use its functionality. In this case, building a semantic Grid process and therefore a Semantic Web Service would not increase but rather reduce the number of clients, which is a knockout criterion.

## 3.4 Dualism of Data Representation Using SAWSDL

The type system used in BPEL($^{\text{light}}$) is XML Schema, i.e. BPEL is based on XML data processing. However, existing Semantic Web Service frameworks are using ontologies as data model and their implementations (e.g. WSMX) require data in ontological form to facilitate semantic service discovery.

Hence, when using BPEL in combination with SWS there is a need to transform between the ontological and XML representation of data. SAWSDL [11] provides semantically annotated data types as means to describe the so called lifting and lowering of data which is basically a transformation of the representation of data. It introduces the concepts of *modelReference*, *liftingSchemaMapping* and *loweringSchemaMapping*. The *modelReference* identifies the concept to which the XML data can be lifted, and the *liftingSchemaMapping* defines how this transformation from the XML representation to the ontological representation can be done. The *lower-*

*ingSchemaMapping* can be used to lower the data again from ontological to its XML representation.

In Listing 4 a (simplified) SAWSDL description of a service is given. A WSMO description for this particular service has been depicted in Listing 1.

# 4 Service Interaction Scenarios

In this section we show how an execution engine implementation uses the various artefacts described in the previous sections to implement four basic interaction scenarios. The interaction scenarios include invocation using traditional WS technology and invocation using an SSB considering both short-running and long-running interaction.

## 4.1 Invoking Short-running BPEL4SWS Processes

A process that first receives a message from and later returns a message to a partner service and is supposed to be short-running can be exposed as a request-response operation. In this case a receiving activity and a subsequent sending activity of the BPEL$^{\text{light}}$ process logic are grounded to this particular WSDL operation which is specified in a grounding file, that contains deployment specific information. The process logic as well as the grounding specification are presented in Figure 6. The WSDL specification has been presented in Listing 4

When a client invokes the WSDL operation, the call is resolved to a receiving activity in the process model by the process engine using the information given in the grounding file. Later, when the corresponding

sending activity is executed, the return value is assigned to the WSDL operation using the grounding file and the WSDL call is completed.

Additionally, the process can be made available as a Semantic Web Service at an SSB (see Figure 7). Therefore a WSMO WS has to be modelled that describes the process' interface and capability and grounds to the WSDL operation the process is exposed as (see and Listing 1).

In addition, semantic annotations and lifting & lowering rules have to be defined using SAWSDL (see Listing 4). The WSMO Web Service description has then to be registered by submitting it to the SSB which can be considered the deployment of the SWS.

After the process has been discovered by matching a goal submitted by a requester with its WSMO WS description which is also known as goal-based discovery , the ontological instance data has to be lowered to its XML Schema representation. This is done by the SSB via the loweringSchemaMapping defined using SAWSDL. In the next step, the service binding and location given in the WSMO WS grounding is used together with the XML data to invoke the BPEL4SWS process. The process engine processes the request like described above. After the WSDL call is completed, the SSB lifts the returned XML data to an ontological level, i.e. creates ontological instances using the liftingSchemaMapping defined in the SAWSDL of the process.

## 4.2 Invoking Short-running Partner Services

In case a message has to be sent to and received from a service that is supposed to be short-running, this can be implemented by a single `<invoke>`-like interactionActivity. The conversation and the invoking activity are either

grounded to the WSDL interface the service provides or they are described using an attached WSMO Goal.

When the process engine executes an activity that first sends and then receives a message and there is no goal attachment at the conversation, the grounding file is used to figure out which operation should be invoked. In case there is a goal attachment at the conversation (linked with a synchronous invocation activity), there is no grounding defined (see Figure 8). Instead, at the beginning of the conversation, a goal is submitted to the SSB. The SSB performs goal-based discovery and initializes the communication between the discovered service and the process by creating context information and sending it back to the process engine for correlation. This context is used to invoke the discovered service via the SSB using the entry point `invokeWebService(context, data):data`. Note that the data therefore has to be available at the ontological level.

## 4.3 Invoking Long-running BPEL4SWS Processes

A process that first receives and later returns a message to a partner service and is supposed to be long-running cannot be exposed as a request-response operation. In this case a receiving and following sending activity have to be mapped to two one-way operations because a request-response operation using a WS-I compliant HTTP binding would produce a time out. In this scenario a client invokes the process via a one-way WSDL operation and provides an endpoint (described via another WSDL one-way operation) where the process can call back. The grounding file of the process defines that a receiving activity is grounded to the one-way operation the process provides

and a subsequent sending activity is grounded to the one-way operation the client is supposed to provide.

When a client invokes the WSDL operation of the process it also submits information about the concrete endpoint and binding for the call-back. Like in the first scenario, the grounding file is used to dispatch the invocation to a certain activity in the process model. When processing the sending activity the process engine evaluates again the grounding file and uses the appropriate WSDL operation in conjunction with the endpoint information to call the client back.

Again, the process can also be made available at a WSMO implementation as a Semantic Web Service by specifying a WSMO WS that describes the process interface and capability on a higher level of abstraction in a machine processable manner and grounds the incoming message to the WSDL operation the process provides. The outgoing message however is not grounded to a specific operation because the WSDL operation of the partner service is considered unknown prior to execution; the call-back endpoint is provided by the SSB.

In case the process not invoked using traditional WS technology only, but first is discovered via its WSMO WS desription as shown in Figure 9, the SSB invokes the process using the grounding information given in the WSMO Web service description of the process and the lowered instance data. Additionally, it submits context information in the message header that identifies the communication between the SSB and this particular process instance. Via this header information, the process engine detects that it has been invoked by the SSB. When the process navigator reaches the corresponding sending

activity it does not use the WSDL operation specified in the grounding file (the light gray parts of Figure 9) but rather sends the message to the SSB using the entry point `invokeWebService(Context, Data)`.

## 4.4 Invoking Long-running Partner Services

In case a message has to be sent to and received from a service that implements a task which is supposed to be long-running, this can neither be grounded to a request-response operation the partner provides nor be described with a Goal that causes the usage of the entry point `invokeWebService(context, data):data`.

When using traditional WS communication this has to be grounded to two one-way operations one provided by the service and one provided by the process. This is similar to the invocation of a long-running process using traditional WS technology. When the process engine executes the sending activity the WSDL operation that is to be invoked is resolved using the grounding file. Later, when the invoked service calls back, the receiving activity (associated with the call back operation) is also discovered using the grounding file.

Similarly to the invocation of short-running services using goal-based discovery the invocation of long-running services starts with submitting a goal associated with a conversation to the WSMO implementation. However, in contrast to the invocation of short-running services, the incoming message is grounded to a WSDL operation the process provides as call-back (see Figure 10). The sending activity is executed by sending ontological lifted data to the SSB using the entry point `invokeWebService(context,`

`data)`. In a later step, the SSB uses the grounding information in the goal to call the process back via the provided WSDL one-way operation. This call back is done using XML data generated by lowering the ontological data according to the lowering rules described using SAWSDL. It is dispatched to the corresponding activity using the grounding file of the process.

# 5 Architecture of a BPEL4SWS Engine

The architecture of the BPEL4SWS engine is similar to well-known workflow engines – the main components are described in the following (see Figure 11).

## 5.1 Architecture Components

To manage and configure the engine one uses the *administration module*. The same component exposes operations for deployment and undeployment of processes.

The *deployment component* is responsible for deploying the artefacts needed to execute a BPEL4SWS process. These are a BPEL4SWS description, corresponding WSDL files and a deployment descriptor. The process model is validated, compiled and stored in the build-time database of the engine.

Process models and process instance data are stored in two logically separate repositories - the *build-time database* and the *Runtime database*. The build-time database stores the compiled process model representation while the Runtime database handles runtime data of all process instances being executed. Each process instance contains a reference to its corresponding process model in the build-time database.

The communication between the engine and external services and clients is handled by the *integration layer*. In particular, it is responsible for receiving external messages, dispatching them to the execution components and sending results back to the clients or partner services.

The integration layer implements the abstract endpoint implementation which is the bridge between the engine and a service bus. The service bus is the service middleware responsible for exposing different kinds of services (e.g. Grid Services exposed as WSs or Semantic Web Services) in a unified access scheme [19].

The *process navigator* uses the process models stored in the build-time database to execute their instances by navigating over the process models. The state of each process instance is stored in the runtime database. The navigator delegates the service interactions and actual message exchange to the integration layer and hence to the service bus.

For transforming XML data into ontology instances and vice versa the engine employs the *Lifting & Lowering component*. Semantically annotated XML variables can thus always be made available in terms of their ontological representation, whenever needed.

## 5.2 Component Interaction Scenarios

The interplay among the components of the architecture can be demonstrated in terms of the following scenarios: (i) process deployment and (ii) process execution.

During process *deployment* the process model is parsed, validated and transformed into an engine-internal representation, which is stored into the

build-time database. The WSDL interfaces of the process are used to expose it as a service, which is done by exposing new endpoints at the integration layer. During *process execution* four basic scenarios are of interest: receiving messages and sending messages on behalf of a process, evaluating conditions specified in terms of semantics and mediation.

Whenever the engine receives a message it either dispatches it to an existing process instance, or creates a new one. In both cases the messages arrive at the process endpoint at the integration layer. The correlation of a message to an existing or a new process instance is done by the integration layer. Once the message is consumed by the navigator, the corresponding interaction activity is executed.

Interaction activities can also send messages. Therefore during the execution of such an activity the integration layer receives a command for sending a message by the navigator. If the target is a WSDL Web Service, the message is serialized in XML. In case an SWS is invoked (via an SSB), the data representation is an instance of an ontological concept, generated by the Lifting & Lowering component. If the interaction is synchronous, the navigator suspends the process instance at the activity and resumes it once it receives a response – either XML data from a WSDL Web service or an instance of an ontological concept from an SWS. In contrast to this communication mode, whenever asynchronous communication is required the process instance is not blocked until the response is received.

# 6 Prototypical Implementation

After the comparison of existing BPEL engines, including Apache ODE, JBoss JBPM/BPEL[4] and ActiveBPEL[5], we have chosen to use the Apache ODE engine as a basis for our implementation. Our selection was based on various functional and non-functional criteria such as licensing, support for WS-BPEL 2.0, extensibility & integration options.

A number of extensions and modifications to Apache ODE were required to realise the architecture described in the section above and thus provide support for BPEL4SWS. Since the service bus provides the appropriate service abstraction it is up to the service bus implemention to enable the interaction with Grid services. This can be done in a way similar to the approach described in [19].

Apache ODE follows a lightweight and modular architecture but lacks support for extensibility, in particular for the elements `<extensionActivity>` and the `<extensionAssignOperation>`. We introduced a plug-in concept that allows plugging in the so-called *ExtensionBundles*. Bundles are linked to a specific extension namespace and consist of several operations referenced by `<extensionActivity>` and `<extensionAssignOperation>` elements in the BPEL process model and implement the concrete extension functionality. The bundles can be registered; namespaces are known to the process model through the `<extension>` element [36].

During deployment to Apache ODE the BPEL file is parsed into an in-memory representation (internal object model), several optimisations are

[4] http://www.jboss.org

[5] http://www.activebpel.org

performed and the process model is checked against a set of static analysis rules. We enhanced the parser and compiler of ODE to support the following elements: `<extensionActivity>`, `<extensionAssignOperation>`, `<extensions>` and `<extension>`.

The element `<conversation>` is simply treated as a BPEL extensibility element and is directly recognised and included by ODE into the internal object model.

Furthermore we have implemented the semantic counterpart of the invoke activity – `<interactionActivity>`. It performs a look up of the referenced `<conversation>` element, which in turn keeps a reference to a WSMO Goal. This goal is passed to the SSB which then discovers, selects and invokes a best-matching Semantic Web Service.

# 7 Related Work

Mandell and McIlraith [27] identified the shortcomings of BPEL with respect to the flexibility of service and partner discovery. They presented a proxy-based approach where service requests are delegated to a discovery service through a locally bound WSDL interface, i.e. they mix different levels of abstractions in the process model: services and infrastructure services. They use OWL-S to semantically describe the activity implementations of a BPEL process. Whether the language is extended has not been presented. Asynchronous and stateful communication between services is not discussed.

Meteor-S [37] also takes a proxy-based approach where all interactions are bound to virtual partners, hosted by a process configuration module.

The process configuration module delegates the service requests to concrete services either bound during deployment or during runtime. As the proxy is stateful, it enables creating an execution plan in case it is required to invoke several operations in order to achieve the specified goal. Asynchronous interaction between the process and the proxy, or stateful interaction via multiple synchronous invocations between the proxy and the process is not discussed. Like in the previous approach, it is not known whether the language is extended and how the semantic annotation of the interaction activities is done, i.e. whether a single `<invoke>` activity or a complete partner link is described semantically.

In contrast to the already mentioned approaches Karastoyanova et al. [18] present an extension of the language, namely an extension to the `<invoke>` activity. Their approach also uses OWL-S to describe activity implementations and only allows for synchronous invocation of OWL-S services.

Grid applications are often provided for use to scientists via scientific portals. Grid portals provide predefined sequences of computational steps for multiple scientific domains. Usually such portals are accessible via a Web interface and deal with security issues, Grid service execution and management based on input data from the scientists, monitoring of the complex computations on the Grid. The set of steps such applications must perform is often hard-coded and hence these applications exhibit rigidity and lack of reusability. Recently, together with the shift towards service orientation [13] multiple experiments have been conducted to assess the applicability of BPEL for enabling Grid service compositions. BPEL has been successfully applied in multiple Grid applications; examples include Grid workflows for

the automotive industry [6], chemical polymorph prediction applications [10], bioinformatics [8].

The authors of [16] focus on addressing the fact that BPEL in incapable of dealing with services compliant to the Web Service Resource Framework (WSRF) transparently to both the workflow designer and the execution environment. This work presents an extension to the BPEL language which allows the interaction with, as they state, both stateless and stateful services and their resources. In general, the extensions to the language enable the use of conventional WSs (which are considered by some authors to be exclusively stateless) and WSRF services (stateful). The approach in [16] aims at facilitating seamless integration of Grid applications in business applications and vice versa. The BPEL extensions are implemented in terms of an Eclipse-based workflow modelling tool and an extended BPEL engine implementation. Similar approach has been presented by [33], [9] and [35]; these research works focus either on modelling of such processes only, or on execution as well. In comparison, the approach presented in this paper is more generic, since it makes it possible to utilize any kind of business services and services on the Grid and not only WSDL and WSRF services.

In [23] we have shown how BPEL can be used even today in WSRF environments if generation of WSRF-specific fragments in corresponding BPEL processes is acceptable.

Significant research effort has been dedicated to providing the Grid with semantic information. The Semantic Grid is defined as an "extension of the current Grid in which information and services are given well-defined meaning through machine-processable descriptions which maximize the potential

for sharing and reuse" [32]. The available approaches and corresponding prototypes focus mostly on enabling discovery of Grid services based on semantic descriptions of services [14]. For example, the work presented in [26] demonstrates a semantic service discovery framework for Grid services, using DAML+OIL [17] (the predecessor of OWL) as ontology language. In [25] a layered architecture for the semantic Grid has been presented which draws upon service orientation in that each Grid service is exposed as a (Web) service. Each Grid service on the SGrid platform is a computational, an information or knowledge service and has a semantic description defined in a Semantic Service Ontology Repository. The authors present a CORBA implementation of the SGrid platform, that supports SOAP/HTTP service calls to WSs implemented in terms of Java, Fortran or C (the legacy applications). Our approach has a different focus. It concentrates on enabling composition of any kind of services - Web Services, Grid Services, legacy Grid Services - while focusing on their functional meaning rather than on interfaces described in terms of a particular service technology.

# 8 Conclusion

In this paper we presented BPEL4SWS – a flexible and comprehensive approach for composing Grid Services exposed as Web services or Semantic Web Services. It allows for the description of activity implementations semantically, i.e. using SWS concepts and thus enables application integration on a higher level of abstraction.

The presented framework is composed of a set of specifications and is

by design independent of any specific service technology, i.e. not specific to e.g. Grid services exposed as Web services. It introduces an additional level of indirection and thus abstracts away details specific for concrete service technologies from process models. Moreover it renders service compositions independent of any Semantic Web Service technology providing even more benefits. The BPEL language is extended to facilitate specifying activity implementations via annotations; the activity implementations can be Grid services exposed as Web services or made available through any other service technology to come. Additionally, in contrast to other approaches the interfaces of the service middleware are not hard-wired in the process model. Interfacing the service middleware is not a concern of the process models that reflect the business logic and is considered to be part of the configuration of the execution environment. The configurability of the processes and the underlying runtime is utilized/applied to bring in the greatest benefit possible. Therefore, BPEL4SWS separates concerns of process modelling from middleware concerns much more appropriately/successfully than BPEL.

Another advantage BPEL4SWS provides is the support for asynchronous communication which is essential for Grid processes. Furthermore, it is designed to support representation of data in process models in terms of both conventional type systems and in terms of ontologies. This is enabled by an XML $\leftrightarrow$ Ontology dualism for process data representation (including the mapping between data and its semantic description) that makes it possible to exchange data with traditional services in enterprises and on the Grid, as well as to consume and produce data during the interaction with any Semantic Web Service infrastructure. Hence this approach shows improvements in

interoperability and therefore facilitates integration enormously. Supporting various attachment and service descriptions also raises more requirements on system and process designers. Thus appropriate tool support for modelling BPEL4SWS processes and semantic annotations is crucial and is in the scope of our future work.

## Acknowledgements

The work published in this article was partially funded by the SUPER project[6] under the EU 6th Framework Programme Information Society Technologies Objective (contract no. FP6-026850).

## References

[1] A. Alves et al. Web Services Business Process Execution Language version 2.0. Committee specification, OASIS, January 2007.

[2] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, A. Malhotra, H. Maruyama, et al. Web Services Policy Attachment (WS-PolicyAttachment). *W3C Member Submission*, April 2006.

[3] K. Ballinger, D. Ehnebuske, C. Ferris, M. Gudgin, C.K. Liu, M. Nottingham, and P. Yendluri. Basic Profile Version 1.1. *WS-I Specification*, 2004.

[6] `http://www.ip-super.org/`

[4] Alistair Barros, Marlon Dumas, and Arthur ter Hofstede. Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection. Technical Report FIT-TR-2005-02, Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia, March 2005.

[5] S. Burbeck. The Tao of e-business services. *IBM Corporation*, 2000.

[6] Kuo-Ming Chao, Muhammad Younasb, and Nathan Griffiths. BPEL4WS-based coordination of Grid Services in design. *Computers in Industry*, 57(8-9):778 – 786, 2006.

[7] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1, 2001.

[8] Remko de Knikker, Youjun Guo, Jin long Li, Albert KH Kwan, Kevin Y Yip, David W Cheung, and Kei-Hoi Cheung. A Web Services Choreography Scenario for Interoperating Bioinformatics Applications. *BMC Bioinformatics, 2004*, 2004.

[9] Tim Dörnemann, Thomas Friese, Sergej Herdt, Ernst Juhnke, and Bernd Freisleben. Grid Workflow Modelling Using Grid-Specific BPEL Extensions. In *German e-Science Conference*, May 2007. Baden-Baden, Germany.

[10] Wolfgang Emmerich, Ben Butchart, Liang Chen, Bruno Wassermann, and Sarah L. Price. Grid Service Orchestration Using the Business Process Execution Language (BPEL). *Journal of Grid Computing, 2006.*, 3:283 – 304, 2006.

[11] J. Farrell and H. Lausen. Semantic Annotations for WSDL and XML Schema. *W3C Recommendation*, August 2007.

[12] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.

[13] Gannon, Plale, Christie, Fang, Huang, Jensen, Kandaswamy, Marru, Pallickara, Shirasuna, Simmhan, Slominski, and Sun. Service Oriented Architectures for Science Gateways on Grid Systems. In *Service-Oriented Computing - ICSOC 2005*, 2005.

[14] Carole Goble and David De Roure. The Grid: An Application of the Semantic Web. *SIGMOD Record*, 31(4), 2002.

[15] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX - A Semantic Service-oriented Architecture. *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pages 321–328, 2005.

[16] Marburg Ad hoc Grid Environment Group. Grid Workflow Modelling Using Business Process Execution Language (BPEL).

[17] I. Horrocks. DAML+OIL: a Description Logic for the Semantic Web. *IEEE Data Engineering Bulletin*, 25:4–9, 2002.

[18] D. Karastoyanova, F. Leymann, J. Nitzsche, B. Wetzstein, and D. Wutke. Parameterized BPEL Processes: Concepts and Implementation. In *4th International Conference on Business Process Management (BPM)*, September 2006. Vienna, Austria.

[19] D. Karastoyanova, T. van Lessen, J. Nitzsche, B. Wetzstein, D. Wutke, and F. Leymann. Semantic Service Bus: Architecture and Implementation of a Next Generation Middleware. In *2nd International Workshop on Services Engineering (SEIW)*, April 2007. Istanbul, Turkey.

[20] J. Kopecky, D. Roman, and J. Scicluna. WSMO Use Case: Amazon E-commerce Service. 2005. *Final Draft*.

[21] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2004.

[22] H. Lausen, A. Polleres, and D. Roman. Web Service Modeling Ontology (WSMO). *W3C Member Submission*, 2005.

[23] F. Leymann. Choreography for the Grid: towards fitting BPEL to the Resource Framework: Research Articles. *Concurrency and Computation: Practice & Experience*, 18:1201–1217, 2006.

[24] F. Leymann and D. Roller. *Production workflow*. Prentice Hall, 2000.

[25] M. Li, P. van Santen, D. W. Walker, O. F. Rana, and M. A. Baker. SGrid: a Service-oriented Model for the Semantic Grid. *Future Generation Computer Systems*, 20(1):7–18, January 2004.

[26] Simone A. Ludwig and S. M. S. Reyhanib. The Grid: An Application of the Semantic Web. *Journal of Web Semantics, Special Issue on Web Semantics: Science, Services and Agents on the World Wide Web*, 4(4):1 – 13, 2006.

[27] D.J. Mandell and S.A. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. *Proceedings of the Second International Semantic Web Conference*, pages 227–241, 2003.

[28] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al. OWL-S: Semantic Markup for Web Services. W3C Member Submission. *World Wide Web Consortium*, 2004.

[29] J. Nitzsche, T. van Lessen, D. Karastoyanova, and F. Leymann. BPEL$^{\text{light}}$. In *5th International Conference on Business Process Management (BPM)*, September 2007. Brisbane, Australia.

[30] J. Nitzsche, T. van Lessen, D. Karastoyanova, and F. Leymann. WSMO/X in the Context of Business Processes: Improvement Recommendations. *International Journal of Web Information Systems, ISSN: 1744-0084*, 2007.

[31] D. Roman, J. Scicluna, and J. Nitzsche. D14 V 0.4: Ontology-based Choreography, 2007.

[32] semanticgrid.org. Semantic Grid Community Portal.

[33] Aleksander Slomiski. On using BPEL extensibility to implement OGSI and WSRF Grid workflows: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1229–1241, 2006.

[34] Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields. *Workflows for e-Science: Scientific Workflows for Grids*. Springer, 1 edition, December 2006.

[35] Kenneth J. Turner and Koon Leai Larry Tana. A Rigorous Approach to Orchestrating Grid Services. *Computer Networks, 2007*, 51:4421 – 4441, 2007.

[36] T. van Lessen, J. Nitzsche, M. Dimitrov, D. Karastoyanova, M. Konstantinov, and L. Cekov. An Execution Engine for BPEL4SWS. In *2nd Workshop on Business Oriented Aspects concerning Semantics and Methodologies in Service-oriented Computing (SeMSoc) in conjunction with ICSOC, To appear*, September 2007. Vienna, Austria.

[37] K. Verma, K. Gomadam, A.P. Sheth, J.A. Miller, and Z. Wu. The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. *LSDIS METEOR-S project*, pages 6–24.

[38] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D.F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2005.

## Jörg Nitzsche

Jörg Nitzsche is a research associate at the Institute of Architecture of Application Systems at the University of Stuttgart, Germany. He holds a Diploma in Computer Science. He participates in the EU funded SUPER project. He does research in the area of process description languages, semantics for business processes and frameworks and middleware for (Semantic) Web Services.

## Tammo van Lessen

Tammo van Lessen is a research associate at the Institute of Architecture of Application Systems at the University of Stuttgart, Germany. He holds a Diploma in Computer Science. He participates in the EU project SUPER. His research interests include middleware for SOA, Business Process Management and semantically enriched Web Service applications.

## Dimka Karastoyanova

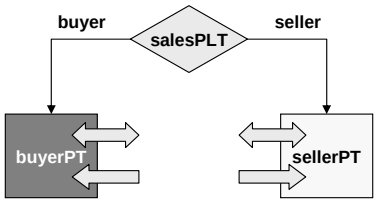
Dimka Karastoyanova is a research and teaching assistant at the Institute of Architecture of Application Systems at the University of Stuttgart, Germany. She holds a PhD degree in computer science. Her primary field of research is Web Services (WS) and the related standards and technologies. She is interested in techniques for promoting reuse and flexibility of WS compositions.

## Frank Leymann

Frank Leymann is a full professor of computer science and director of the Institute of Architecture of Application Systems at the University of Stuttgart, Germany. His research interests include service-oriented computing and middleware, workflow- and business process management, programming in the large, transaction processing, integration technology and architecture patterns. Before accepting his professor position he worked for two decades for IBM Software Group building database and middleware products. As an IBM Distinguished Engineer and elected member of the IBM Academy of Technology he contributed to the architecture and strategy of IBM's entire middleware stack as well as IBM's On Demand Computing strategy. From 2000 on, he worked as co-architect of the Web Service stack.

```
<wsdl:definitions
   targetNamespace=...
   xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
   ...
   <plnk:partnerLinkType name="salesPLT">
      <plnk:role name="buyer"
                 portType="buyerPT" />
      <plnk:role name="seller"
                 portType="sellerPT" />
   </plnk:partnerLinkType>
   ...
</wsdl:definitions>
```

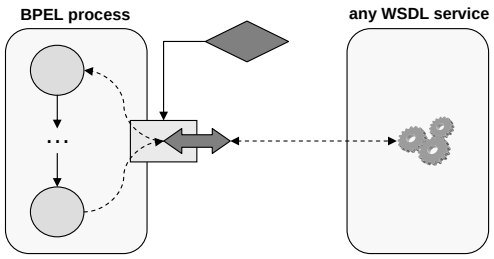
(a) code snippet




(b) scenario

Figure 1: The WSDL extension `<partnerLinkType>`

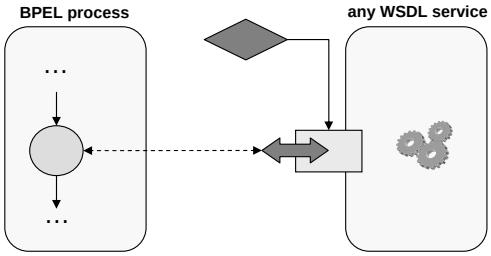
```
<flow>
   <links>
      <link name="receive-to-..." />
      <link name="...-to-reply" />...
   </links>...
   <receive name="receiveOrder"
            partnerLink="salesPL"
            operation="order"
            variable="item">
      <sources>
         <source linkName="receive-to-..." />...
      </sources>...
   </receive>
   <reply name="sendConfirmation"
          partnerLink="salesPL"
          operation="order"
          variable="confirmation">
      <targets>
         <target linkName="...-to-reply" />...
      </targets>...
   </reply>...
</flow>
```




(a) code snippet (b) scenario

Figure 2: Invocation of a short-running BPEL process

```
<invoke name="orderItem"
        partnerLink="salesPL"
        operation="order"
        inputVariable="item"
        outputVariable="confirmation">
   ...
</invoke>
```

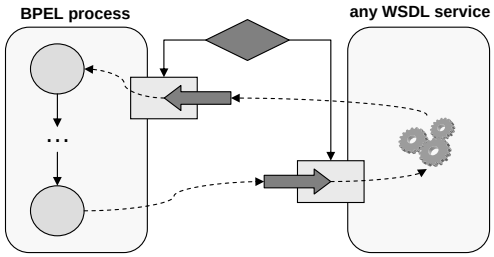



(a) code snippet (b) scenario

Figure 3: Blocking invocation of a WSDL service

```
<flow>
   <links>
      <link name="receive-to-..." />
      <link name="...-to-send" />...
   </links>...
   <receive name="receiveOrder"
            partnerLink="salesPL"
            operation="order"
            variable="confirmation">
      <targets>
         <target linkName="receive-to-..." />...
      </targets>...
   </receive>...
   <invoke name="sendConfirmation"
           partnerLink="salesPL"
           operation="getOrder"
           inputVariable="item">
      <sources>
         <source linkName="...-to-send" />...
      </sources>...
   </invoke>
</flow>
```

(a) code snippet

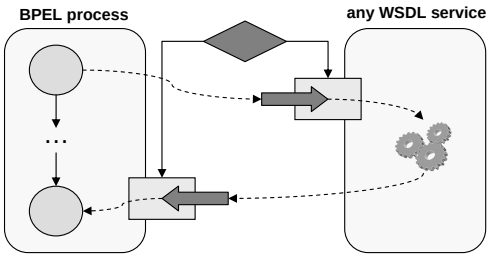



(b) scenario

Figure 4: Invocation of a long-running BPEL process

```
<flow>
   <links>
      <link name="send-to-..." />
      <link name="...-to-receive" />...
   </links>...
   <invoke name="orderItem"
           partnerLink="salesPL"
           operation="getOrder"
           inputVariable="item">
      <sources>
         <source linkName="send-to-..." />...
      </sources>...
   </invoke>...
   <receive name="receiveConfirmation"
            partnerLink="salesPL"
            operation="getConfirmation"
            variable="confirmation">
      <targets>
         <target linkName="...-to-receive" />...
      </targets>...
   </receive>
</flow>
```

(a) code snippet




(b) scenario

Figure 5: Non-blocking invocation of a WSDL service

```
namespace {_"http://www.wsmo.org/webServices/amazonWebService#",
           am _"http://www.wsmo.org/ontologies/amazon/amazonOntology#",
           bk _"http://www.wsmo.org/ontologies/amazon/bookOntology#"}

webService _"http://www.wsmo.org/webServices/amazonWebService"
  ...
  capability
    precondition
      ?request memberOf am#itemSearchRequest or
      ...
    postcondition
      (?request[author hasValue ?author] memberOf am#searchBooks implies
            exists ?container
        (?container memberOf am#itemContainer and
              forall ?item
          (?container[items hasValue ?item] memberOf am#itemContainer implies
                ?item[author hasValue ?author] memberOf bk#amazonBook
          )
        )
       ) and
       ...
    ...

  interface amazonWSInterface
    choreography
      stateSignature
        importsOntology _"http://www.wsmo.org/ontologies/amazon/amazonOntology#"

        in
          concept am#itemSearchRequest withGrounding
            _"http://webservices.amazon.com/AWSECommerceService/2007-02-22# wsdl.
                interfaceMessageReference(AWSECommerceServicePortType/ItemSearch/In)",
          ...
        out
          concept am#itemContainer withGrounding {
            _"http://webservices.amazon.com/AWSECommerceService/2007-02-22# wsdl.
                interfaceMessageReference(AWSECommerceServicePortType/ItemSearch/Out)",
          ...
      transitionRules

        if (?ItemSearchRequest memberOf am#itemSearchRequest) then
          add(_# memberOf am#itemContainer)
        endIf
        ...
```

Listing 1: (Partial) WSMO Web service description of the Amazon Web Service.

```
<grounding process="QName">

   <activity name="NCName"
             portType="QName"
             operation="NCName"/>*

</grounding>
```

Listing 2: Partial grounding

```
<grounding process="QName">
   <conversation name="NCName"
             partnerLinkType="QName"
             myRole="NCName"
             partnerRole="NCName"/>*
   <activity name="NCName"
             operation="NCName"/>*
</grounding>
```

Listing 3: Full grounding

```
<wsdl ... targetnamespace="http://webservices.amazon.com/AWSECommerceService/2004-11-10" ...>
  <types>
    <xs:element
      name="ItemSearch"
      sawsdl:modelReference="am:itemSearchRequest"
      sawsdl:liftingSchemaMapping="http://...itemSearchRequestLifting.xslt"
      sawsdl:loweringSchemaMapping="http://...itemSearchRequestLowering.xslt">
      ...
    </xs:element>
    <xs:element
      name="ItemSearchResponse"
      sawsdl:modelReference="am:itemam#itemContainer"
      sawsdl:liftingSchemaMapping="http://...itemSearchResponseLifting.xslt"
      sawsdl:loweringSchemaMapping="http://...itemSearchResponseLowering.xslt">
      ...
    </xs:element>
    ...
  </types>
  <portType name="AWSECommerceServicePortType">
    <operation name="ItemSearch">
      <input element="tns:ItemSearch"/>
      <output element="tns:ItemSearchResponse"/>
    </operation>
    ...
  </portType>
</wsdl>
```

Listing 4: (Partial) SAWSDL description of the Amazon Web Service

```
<process
   targetnamespace="http://www.example.org/processes"
   name="amazon" ...>
  ...
  <bl:conversations>
     <bl:conversation
        name="salesConv"
        b4s:ws="http://www.wsmo.org/webServices/
                amazonWebService#"/>...
  </bl:conversations>...
  <flow>
     <links>
        <link name="receive-to-..." />
        <link name="...-to-send" />...
     </links>...
     <extensionActivity>
        <bl:interactionActivity name="receiveRequest"
                        conversation="salesConv"
                        inputVariable="item">
           <sources>
              <source linkName="receive-to-..."/>...
           </sources>...
        </bl:interactionActivity>
     </extensionActivity>
     <extensionActivity>
        <bl:interactionActivity name="sendResponse"
                        conversation="salesConv"
                        outputVariable="confirmation">
           <targets>
              <target linkName="...-to-send"/>...
           </targets>...
        </bl:interactionActivity>
     </extensionActivity>...
  </flow>
</process>
```

(a) Process logic

```
<grounding
   ...
   xmlns:am="http://webservices.amazon.com/
             AWSECommerceService/2004-11-10"
   process="http://www.example.org/processes#amazon"
   ...>
   <activity name="receiveRequest"
             portType="am:AWSECommerceServicePortType"
             operation="ItemSearch"/>*
   <activity name="sendResponse"
             portType="am:AWSECommerceServicePortType"
             operation="ItemSearch"/>
</grounding>
```

(b) Grounding specification

Figure 6: BPEL light and grounding

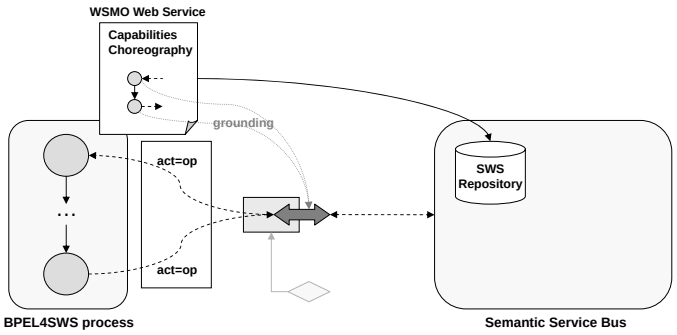



Figure 7: Discovery and invocation of a short-running BPEL4SWS process based on a WSMO WS description.

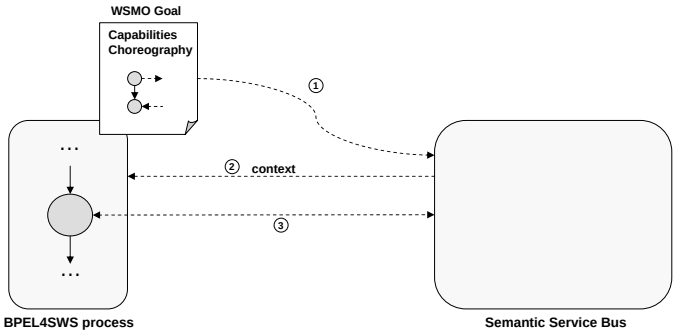



Figure 8: Discovery and invocation of a short-running activity implementation based on a WSMO Goal description.

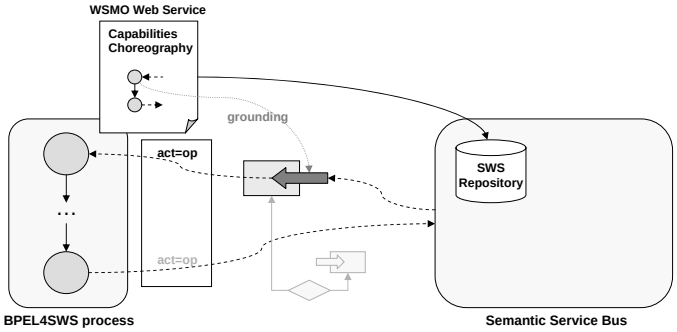



Figure 9: Discovery and invocation of a long-running BPEL4SWS process based on a WSMO WS description.

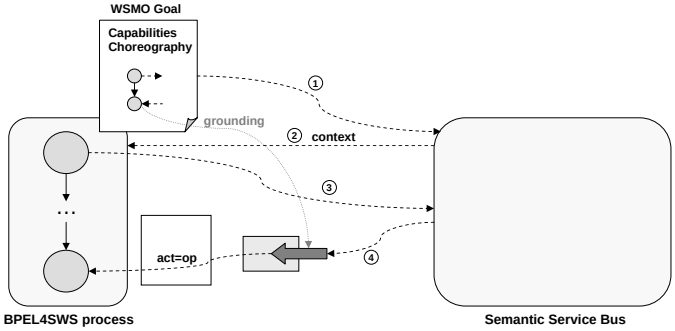



Figure 10: Discovery and invocation of a long-running activity implementation based on a WSMO Goal description.

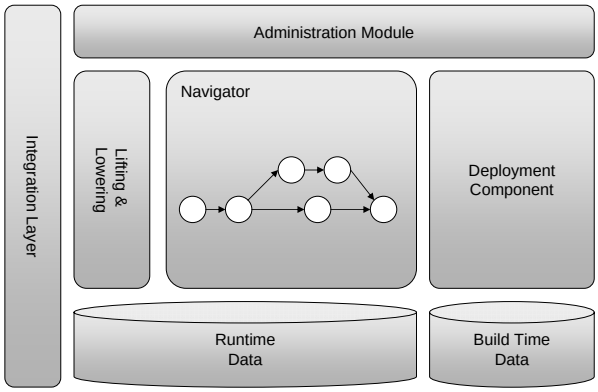



Figure 11: Architecture of a BPEL4SWS Engine.