An Execution Engine for Semantic Business Processes

Tammo van Lessen[1], Jörg Nitzsche[1], Marin Dimitrov[2], Mihail Konstantinov[2], Dimka Karastoyanova[1], Luchesar Cekov[2], and Frank Leymann[1]

[1] Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstrasse 38, 70569 Stuttgart, Germany
{tammo.van.lessen,joerg.nitzsche,dimka.karastoyanova,
frank.leymann}@iaas.uni-stuttgart.de
[2] Ontotext Lab. / Sirma Group
135 Tsarigradsko Shose Blvd., Office Express IT Center, Sofia 1784, Bulgaria
{firstname.lastname}@ontotext.com

Abstract. In this paper we present the architecture and design of an extended BPEL engine that implements the operational semantics of BPEL4SWS. BPEL4SWS is an extension of the BPEL language with support for Semantic Web Service concepts like mediation and semantic descriptions of activity implementations. We describe the basic communication scenarios of processes with services and the interaction between the engine components involved in the execution of BPEL4SWS processes. The presented prototype is based on the open source BPEL engine Apache ODE, features improved configurability and facilitates the definition of additional BPEL extensions with minimal development effort.

1 Introduction

The Web Services Business Process Execution Language (BPEL) [1] is the de facto standard for the orchestration of Web Services. However, two major shortcomings of BPEL can be identified, namely (i) hard-coding of service interfaces, i.e. actual activity implementation types [2], and (ii) lack of semantics of used data types. Interfaces of partner services used within a BPEL process and the interface of the process itself are hard-coded within the process logic via WSDL [3] interfaces. Therefore, services providing equivalent functionality but through different WSDL interfaces cannot be used. In BPEL messages are described in XML and have no formal semantics. As a result, automated or semi-automated matching of and translation (mediation) between different XML schemata used by different business partners is not possible. Instead, handcrafted XPath expressions or other transformation approaches (like XSLT) are used in order to provide message manipulation on the syntactical level.

These shortcoming are eliminated by BPEL4SWS [4], which is an extension of BPEL. It uses ontologies as a data model, supports descriptions of activity

E. Di Nitto and M. Ripeanu (Eds.): ICSOC 2007 Workshops, LNCS 4907, pp. 200–211, 2009.

implementations independent of WSDL interfaces using Semantic Web Services and enables data manipulation, i.e. mediation, on an ontological level.

For conventional BPEL there is already a huge amount of tool support available. This holds for both modelling tools and execution environments. Since BPEL4SWS is an extension of BPEL, existing BPEL engines can be extended to support and make use of the new features introduced by BPEL4SWS. In this paper we show how the existing open source Apache ODE[1] BPEL engine can be extended in a non-intrusive manner to a BPEL4SWS compliant execution engine. The extended engine implements the basic scenarios for communication between processes and services as provided for by BPEL4SWS.

The paper is structured as follows. Section 2 gives a short introduction of BPEL4SWS. The communication scenarios it supports are described in section 3. The architecture and design of the extended ODE engine are presented in section 4 and the actual implementation is described in section 5. Section 6 concludes the paper and gives directions for future work.

2 BPEL for Semantic Web Services

BPEL4SWS [4] attempts to overcome the aforementioned deficiencies of BPEL by (i) allowing semantic descriptions of activity implementations (instead of referring to syntactic WSDL interfaces), (ii) using ontologies as an underlying data model and employing the concept of ontology mediation.

The attributes of BPEL interaction activities that refer to WSDL operations and to partner links, which are in turn dependent on WSDL interfaces, are mandatory. Hence there is a need for a new interaction model that is independent of WSDL. This interaction model is provided by BPEL^{light} [2]. It introduces a new interaction activity type (`interactionActivity`) using the BPEL `extensionActivity` mechanism. The `interactionActivity` is independent of WSDL and can be configured to behave like the different basic interaction activities in BPEL. Additionally, a `conversation` element is introduced which allows grouping together a set of activities that in combination are able to achieve a functional goal on behalf of the process or enable the process to provide functionality to other partners.

BPEL4SWS builds on top of the interaction model provided by BPEL^{light} and can utilize SWS frameworks like OWL-S [5] and WSMO [6] to semantically describe what a conversation is meant to achieve. Due to its advantages over OWL-S (see [7]) the implementation we present in this paper focuses on supporting WSMO.

WSMO distinguishes between Goals and Web Services. A WSMO Web Service describes the functional and non-functional properties of a Web service in a machine-processible manner (i.e. using ontologies and abstract state machines). A WSMO Goal describes in a similar way the requirements a client has on a particular Web Service. Having this distinction allows to use Goals as query to

[1] `http://ode.apache.org/`

discover matching Web services. Applied to BPEL4SWS this means that depending on whether the process requires or provides functionality, either a WSMO Goal or a WSMO Web Service must be assigned to the affected conversation.

While lacking flexibility on an abstract level, WSDL provides excellent low level support. All kinds of protocols and all kinds of encodings can be defined and used for the purpose of communication. For that reason, Semantic Web Service frameworks (including WSMO) mainly use WSDL groundings for enabling communication and benefit from its flexibility and existing infrastructure. BPEL4SWS also defines a WSDL grounding. There are two different kinds of grounding according to the different use cases: (i) partial grounding of the interaction model when using and exposing Semantic Web Services, i.e. grounding of the receiving activities only and (ii) full grounding when exposing the functionality of a process as conventional Web Services for backward compatibility.

XML data is communicated "over the wire" when BPEL4SWS processes are invoked. Semantic Annotations for WSDL and XML Schema (SAWSDL) [8] is an approach that enables lifting XML data to an ontological representation and vice versa, i.e. it facilitates making data accessible to ontological reasoning. Thus, conditions in the process that contain ontologically described data can be evaluated using reasoning and semantic mediation can be applied (via an `<extensionAssignOperation>` called `<mediate>`). The semantic representation of data is also required to enable semantic discovery and invocation of Semantic Web Services using an SWS middleware (e.g. WSMX [9]).

3 Service Interaction Scenarios

In this section we present several execution scenarios of BPEL4SWS processes with partner services that have to be supported by an execution engine implementation. The execution scenarios include conventional invocation and invocation using a semantic middleware considering both asynchronous and synchronous invocation.

To support the goal based communication features BPEL4SWS provides, a semantic aware middleware like WSMX has to provide several operations [7]. 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 middleware. Sending messages via an already established conversation to a partner service hosted by the semantic middleware is enabled by two different operations: `invokeWebService(context, data):data` for synchronous communication and `invokeWebService(context, data)` for asynchronous communication. These operations are used by the BPEL engine presented in this paper to communicate with a WSMO-enabled middleware.

3.1 Synchronous Invocation of a BPEL4SWS Process

In case a process is exposed via a request-response WSDL operation it can be invoked synchronously, i.e. a client that invokes the process blocks and waits until

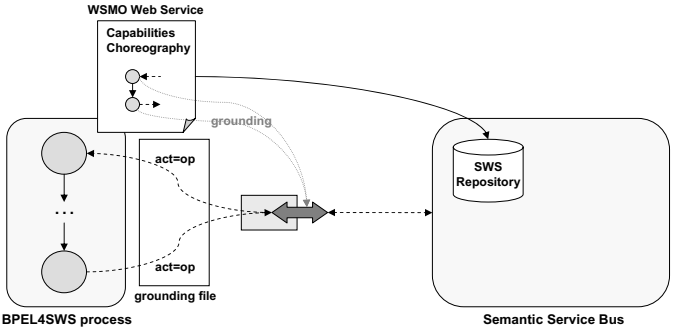

Fig. 1. Semantic Synchronous Invocation of a BPEL4SWS process

the return result is sent back. Therefore a receiving activity and a subsequent sending activity of the BPEL^{light} process logic are grounded to this particular WSDL operation. This happens in a grounding file, which contains deployment specific information.

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 a semantic middleware (see Figure 1). Therefore a WSMO WS has to be modelled that describes the process' interface and capability semantically and grounds to the WSDL operation the process exposes. In addition, semantic annotations and lifting & lowering rules can be defined using the SAWSDL.

After the process has been discovered, the ontological instance data has to be lowered to its XML Schema representation. This is done by the middleware 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 semantic middleware lifts the returned XML data to an ontological level, i.e. creates ontological instances using the liftingSchemaMapping defined in the SAWSDL of the process.

3.2 Asynchronous Invocation of a BPEL4SWS Process

In case of asynchronous invocation a client that invokes the process via a one-way WSDL operation is not blocked until the return result is sent back. Instead it provides an endpoint where the process can call back via a WSDL one-way operation. The grounding file of the process defines that a receiving activity is grounded to

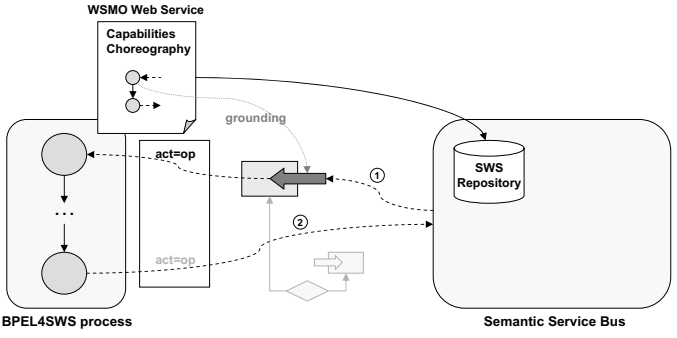

Fig. 2. Semantic Asynchronous Invocation of a BPEL4SWS process

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 previous scenarios, 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 semantic middleware as a Semantic Web Service by specifying a WSMO WS that describes the process interface and capability semantically 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 semantic middleware.

In case of semantic asynchronous invocation of the process as shown in Figure 2, the semantic middleware 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 middleware and this particular process instance. Via this header information, the process engine detects that it has been invoked semantically. 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 2) but rather sends the message to the semantic middleware using the entry point `invokeWebService(Context,Data)`.

3.3 Synchronous Invocation of Services

When a service is to be invoked, the conversation and the invoking activity are either grounded to the WSDL interface the service provides or they are semantically described using a WSMO Goal.

When the process engine executes an activity that first sends and then receives a message and there is no semantic attachment at the conversation, the grounding file is used to figure out which operation should be invoked. In case there is a semantic attachment at the conversation (linked with a synchronous invocation activity), there is no grounding defined (see Figure 3). Instead, at the beginning of the conversation, a goal is submitted to the semantic middleware. The middleware performs semantic discovery and initializes the communication between the discovered service and the process by creating context information and sending it back to the process engine.

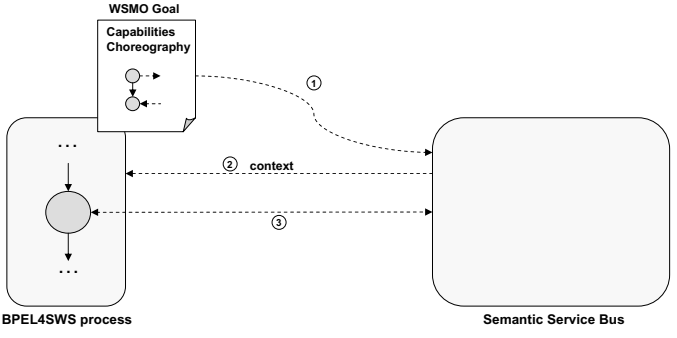

Fig. 3. Synchronous Invocation of a WSMO Web Service

3.4 Asynchronous Invocation of Services

The asynchronous invocation of a conventional service is similar to the conventional asynchronous invocation of a process. 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 synchronous communication mode asynchronous semantic invocation starts with submitting a goal associated with a conversation to the semantic middleware. However, in contrast to the synchronous invocation, the incoming message is grounded to a WSDL operation the process provides as call-back (see Figure 4). The sending activity is executed by sending ontological lifted data to the semantic middleware using the entry point `invokeWebService-(context, data)`. In a later step, the semantic middleware 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 the middleware 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.

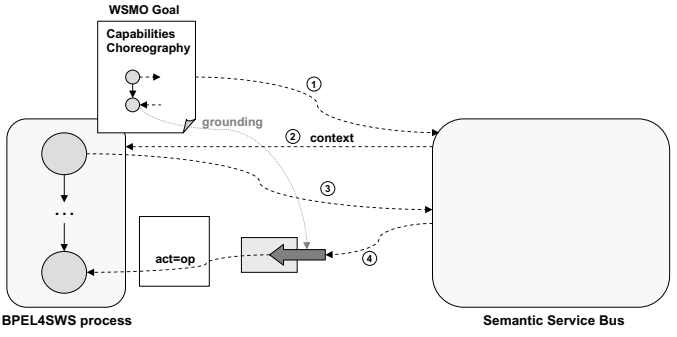

Fig. 4. Asynchronous Invocation of a WSMO Web Service

3.5 Partner-Based Semantic Web Service Discovery

The partner element in BPEL4SWS can be used to constrain that a partner has to satisfy multiple goals. This partner can either be an abstract partner that is discovered during runtime or a concrete partner. The concrete partner can be configured during design time or can be assigned to the partner element during runtime. Whenever a partner has to be discovered, a list of goals is sent to the semantic middleware using the entry point `findPartner(list of goals):partner` [7]. 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 as presented in the previous sections.

4 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 5).

4.1 Components of the Architecture

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 buildtime database of the engine.

Process models and process instance data are stored in two logically separate repositories - the *buildtime database* and the *runtime database*. The Buildtime 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 Buildtime database.

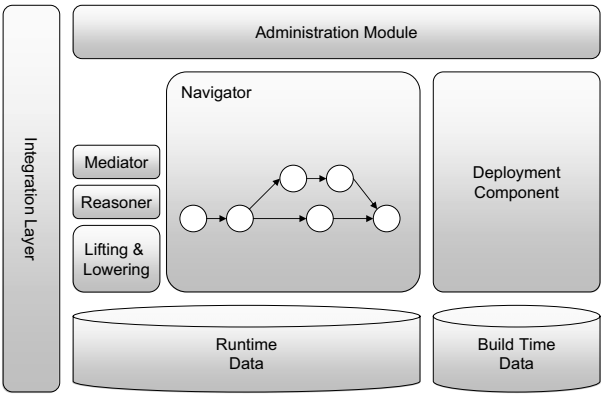

Fig. 5. Architecture of a BPEL4SWS Engine

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 *process navigator* uses the process models stored in the Buildtime database to execute their instances. It navigates over the process model for each of its instances. The navigator stores the state of each process instance in the Runtime database. Whenever a service interaction must be performed, this component delegates the interaction and the actual message exchange to the integration layer.

Transition conditions on control connectors and join conditions that are defined by logical expressions are evaluated by the *reasoner*. XPath expressions are directly processed by the process navigator.

The *mediation component* is responsible for handling the data mismatches when the process model makes use of different ontologies.

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.

4.2 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. This representation is then stored into the Buildtime database. The WSDL interfaces of the process are used to expose it as a service, which is done by exposing a new endpoint at the integration

layer. During *process execution* four basic scenarios are of interest: receiving messages and sending messages on behalf of a process, evaluating semantically defined conditions 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 a semantic middleware infrastructure like WSMX), 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 data – either XML 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.

Whenever the navigator needs to evaluate conditions that are expressed in terms of logical expressions, all data visible in the current scope is ontological lifted using the Lifting & Lowering component. Thereafter the ontological data is provided to the reasoner which evaluates the logical expression and returns the result to the navigator.

During the execution of the `mediate` activity the navigator delegates the mediation to the mediation component that first uses the Lifting & Lowering component in case the data is not available in a semantic form and then discovers and executes an appropriate mediator. After the mediation the result is returned to the navigator.

5 Implementation

In order to choose a BPEL execution engine that we can base our work on, we have evaluated several open source options, including Apache ODE, JBoss BPEL[2] and ActiveBPEL[3], according to various functional and non-functional criteria such as licensing, support for WS-BPEL 2.0, extensibility & integration options, community and industry adoption. Apache ODE has been selected as the option satisfying our requirements to the highest extent.

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.

[2] `http://www.jboss.org`
[3] `http://www.activebpel.org`

5.1 Apache ODE Extensibility

Apache ODE follows a lightweight and modular architecture but lacks support for extensibility so far, in particular for the elements `extensionActivity` and the `extensionAssignOperation`. For that reason, we introduced a plug-in concept to the engine that allows plugging in so called `ExtensionBundles`. Such a bundle is linked to a particular extension namespace and consists of several operations that are referenced by `extensionActivity` and `extensionAssignOperation` elements in the BPEL process model and implement the concrete extension functionality. The bundles can be registered in the engine via a configuration properties file; its namespace must be made known to the process by declaring it in an `<extension>` element.

5.2 Parser, Compiler, and Object Model

Deployment in Apache ODE happens in two steps. First, the BPEL file is parsed into an in-memory representation. Next, the compiler transforms the parsed process model into an optimized object model. The compiler executes several optimisations that simplify the implementation of the navigation component. Additionally the process model is checked against a set of static analysis rules. To implement BPEL4SWS, ODE needs to be able to parse and compile WS-BPEL 2.0 extensions. Therefore, we changed both its Parser and Compiler to support the elements `<extensions>`, `<extension>`, `<extensionAssignOperation>` and `<extensionActivity>`.

The elements needed to model a BPEL4SWS `<conversation>` do not require a special handling by the parser as all unknown elements are preserved in the object model and can be accessed from the extension implementations at any time.

5.3 Interaction Activities

In the first development integration, we extended the BPEL engine with a semantic counterpart of the invoke activity, i.e. an `interactionActivity` that first sends and subsequently receives a message. This activity implementation 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 Semantic Web Services execution environment (WSMX) which then discovers, selects and invokes a best-matching Semantic Web Service.

The next step will be the full implementation of the `interactionActivity` that enables a conversational (i.e. asynchronous) communication between the process and the provider of a WSMO Web Service once the asynchronous communication mode is supported by the WSMX infrastructure.

5.4 Semantic Assign – Data Mediation

In order to enable the engine to perform data mediation, i.e. transformation between instances of different ontological concepts, the engine relies on WSMX and

its mediation component. Data mediation is defined by a `<mediate>` element – a custom assign operation. It takes two parameters, the source and the target variable. The engine analyses their SAWSDL annotations to find out which ontological concepts are representing the variables' type. Then the engine delegates the mediation to WSMX which discovers an appropriate data mediator that is capable to transform from the source to the target ontology and invokes the actual transformation.

5.5 Monitoring and Event Logging

Business Process Monitoring [10] and in particular Business Activity Monitoring (BAM) [11] can strongly benefit from semantic annotated data. By means of appropriate ontologies, monitoring dashboards can group and visualize audit events, which semantically belong together.

In order to enable semantic process monitoring, the Apache ODE logging event infrastructure has been extended to publish process events in a WS-Notification [12] compliant manner. The events are serialised instances of the event ontology (EVO) [13] that is capable of capturing the information needed for semantic monitoring and mining.

6 Conclusion and Future Work

BPEL4SWS extends BPEL 2.0 with the ability to use semantic information for describing activity implementations using semantics and thus independent of their interface descriptions. In addition, data models used in processes are represented semantically using ontologies, which enable the use of process relevant data for reasoning. Mismatches on the data and process level can also be resolved using mediation on the ontological level. Unlike other existing approaches, e.g. METEOR-S [14], the BPEL4SWS processes contain no reference to any implementation infrastructure, but rather only use semantic descriptions to define requirements toward service functionality or capabilities of the process being a service.

In this paper an extended engine implementing BPEL4SWS has been presented. The architecture of the enhanced engine enables support for execution and monitoring. It features improved configurability, and it makes it easier to provide support for other language extensions using the extension bundles concept without major implementation effort.

As part of our future work we intend to enhance and improve the implementation of the `interactionActivity` in BPEL4SWS to support also asynchronous communication modes. This task depends on the ability of the WSMX infrastructure to support asynchronous communication with Semantic Web Services; such a feature is not yet available. A monitoring tool that supports monitoring is currently being developed. It is not only based on conventional data logs but also on semantic information and the ability to reason over it; BAM-like features will be in the focus of our future work with emphasis on the use of semantics for this purpose.

Acknowledgements

The work published in this article was partially funded by the SUPER project[4] under the EU 6^{th} Framework Programme Information Society Technologies Objective (contract no. FP6-026850).

References

1. Alves, A., et al.: Web Services Business Process Execution Language version 2.0. Committee specification, OASIS (2007)
2. Nitzsche, J., van Lessen, T., Karastoyanova, D., Leymann, F.: BPEL^{light}. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 214–229. Springer, Heidelberg (2007)
3. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1 (2001)
4. Nitzsche, J., van Lessen, T., Karastoyanova, D., Leymann, F.: BPEL for Semantic Web Services. In: Proceedings of the 3rd International Workshop on Agents and Web Services in Distributed Environments (AWeSome 2007) (2007)
5. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al.: OWL-S: Semantic markup for web services. W3C Member Submission. In: World Wide Web Consortium (2004)
6. Roman, D., Lausen, H., Keller, U., de Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Kifer, M., König-Ries, B., Kopecky, J., Lara, R., Oren, E., Polleres, A., Scicluna, J., Stollberg, M.: Web Service Modeling Ontology, v1.4. WSMO working draft, DERI (2007), http://www.wsmo.org/TR/d2/v1.4/
7. Nitzsche, J., van Lessen, T., Karastoyanova, D., Leymann, F.: WSMO/X in the Context of Business Processes: Improvement Recommendations. International Journal of Web Information Systems (2007), ISSN: 1744-0084
8. Farrell, J., Lausen, H.: Semantic annotations for WSDL and XML Schema. W3C working draft, W3C (2006), http://www.w3.org/TR/sawsdl/
9. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX – a semantic service-oriented architecture. In: Proceedings of the International Conference on Web Services (ICWS 2005), Orlando, USA (2005)
10. Zur Muehlen, M., Rosemann, M.: Workflow-based process monitoring and controlling-technical and organizational issues. In: Proceedings of the 33rd Annual Hawaii International Conference on System Science (HICSS-33), Los Alamitos, California (2000)
11. Hellinger, M., Fingerhut, S.: Business Activity Monitoring: EAI meets Data Warehousing. EAI Journal, 18–21 (July 2002)
12. Graham, S., Hull, D., Murray, B.: WS-BaseNotification. OASIS standard (2006)
13. Pedrinaci, C., Domingue, J.: Towards an ontology for process monitoring and mining. In: Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2007), Innsbruck, Austria (2007)
14. Verma, K., Gomadam, K., Sheth, A.P., Miller, J.A., Wu, Z.: The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. Technical report, University of Georgia, Athens (2005)

[4] http://www.ip-super.org/