A Management Framework for WS-BPEL

Tammo van Lessen, Frank Leymann, Ralph Mietzner, Jörg Nitzsche,
and Daniel Schleicher
Institute of Architecture of Application Systems,
University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
{tammo.van.lessen, frank.leymann, ralph.mietzner, joerg.nitzsche, daniel.schleicher}
@iaas.uni-stuttgart.de
`http://www.iaas.uni-stuttgart.de/`

Abstract

WS-BPEL is the standard to define executable business processes in a Web service world. Numerous commercial and open source BPEL engines exist on the market today that allow the execution of process models defined in BPEL. However, these execution engines only provide access to process model and process instance data in terms of proprietary APIs. In this paper we present an approach that models BPEL process models and process instances as resources and thus provides a uniform access scheme for process model and process instance data. This is crucial because access to process model and process instance data is needed in different scenarios that are of key relevance in enterprises today. These scenarios include compliance checking, repair of faulted business processes as well as real-time monitoring of business processes. The lack of a uniform access scheme to process model and process instance data hampers the exchangeability of BPEL engines and therefore results in a potential vendor lock-in.

1. Introduction

WS-BPEL provides a standardized way to describe and execute business processes by orchestrating Web services. BPEL processes are used today in various scenarios ranging from credit application processes in banking that involve human interaction, to fully automated processes in backend systems that orchestrate fine granular Web services into higher level Web services. Management of process models and instances is needed for example for process repair, real-time monitoring and compliance checking. For the purpose of process repair an administrator needs for instance to examine processes and to change variable values. Most of the BPEL engines available today support management of both, process models as well as process instances. However, there is no standardized way in which process engines expose their internal process instance and process model data so that it can be used by external management tools and frameworks.

In this paper we propose a uniform way to allow engine external tools to access process instance and process model data. We present an approach that allows exposing process models and instances as standardized resources which in fact prevents users from a vendor lock-in. The resources can be accessed and modified by external tools in a standardized manner. Exposing process model and process instance data as resources provides further advantages that are not captured by the management capabilities of today's BPEL engines. A resource oriented management API allows applications to subscribe to property changes of the standardized resources representing process models and instances instead of subscribing to a set of vendor specific predefined events as it is the case with the current BPEL engines. Thus, our approach enables implementing the aforementioned scenarios once and exchange the underlying BPEL engine without the need of adapting applications using the management API. As a result, BPEL engines become interchangeable and implementations that are build on top of the management API of a BPEL engine can be ported easily from one engine to the other.

The contribution of the management framework proposed in this paper is to provide a uniform means to retrieve and modify data from process instances that are already running. Information about such process instances needs to be retrieved or modified in case of

978-0-7695-3399-5/08 $25.00 © 2008 IEEE
DOI 10.1109/ECOWS.2008.25

unforseen occurrences. Such occurrences could for example be the malfunction of a Web service that is utilized by a process. In this case it could be necessary to put that process from a faulty state back to a state where it can be resumed. It is not our intention to describe a framework for debugging processes, however, this general approach can be used as a basis for such a framework.

The remainder of the paper is structured as follows. Section 2 provides the required background information about BPEL and resources in general. Related work is presented in Section 3. The idea of representing BPEL process models and instances as resources is motivated and elaborated in Section 4. In Section 5 the use of the unified interface for BPEL processes is illustrated. Section 6 presents suitable technologies for implementing the management framework and Section 7 presents a prototypical implementation. Section 8 concludes the paper and presents directions for future work.

2. Background

BPEL [3] is the standard for specifying business processes in Web services (WS) environments and has gained broad acceptance in industry and research. It enables both, the composition of WSs [25] and rendering the composition itself as WSs providing a recursive aggregation model for WSs. BPEL provides several so called *structured activities* that facilitate describing the control flow between the *interaction activities*, i.e. operations of WS. Data flow in BPEL is implicit; data is stored in shared variables that are referenced and accessed by interaction activities and manipulation activities. BPEL processes are intended to support robust applications. Thus, transactionality and fault handling are an integral part of BPEL. They are defined by means of (i) scopes (units of work with compensation-based recovery semantics), (ii) compensation handlers (define how to compensate already completed activities in a custom manner) and (iii) fault handlers (define how to proceed when faults occur). BPEL is designed to be extensible. It enables defining custom activity types using the `extensionActivity` mechanism as well as custom data manipulation primitives via an `extensionAssignOperation`. Additionally, extension elements and attributes can be defined for all BPEL constructs. Well-known extensions to BPEL support human interactions (BPEL4People [1]) and use of sub-processes (BPEL-SPE [18]).

A *resource* is in its abstract sense anything that has identity. A resource can be an electronic document, an image, a service or a collection of other resources. According to [5] not all resources are necessarily network retrievable; e.g., human beings, corporations, and bound books in a library can also be considered resources. Resources typically represent a specific set of data. This data can be rendered as documents (e.g. XML, HTML, JSON, Atom,...) or can be made accessible via so called resource properties which are basically specific views on the resource content. Resources are commonly used in Web architectures, i.e. in RESTful applications or in WS-*. In WS-* environments, resources themselves are rendered as WS-Resources [11] whereas their properties are rendered according to the WS-ResourceProperties specification [12]. Both specifications are part of the WS-ResourceFramework (WSRF)[1].

3. Related Work

Enterprise environments typically involve a huge number of components, data silos, queues, connections to legacy systems, etc. in a complex interplay. Being able to guarantee a certain degree of reliability, availability and other qualities of service, it is very important to have the ability to monitor each component and its connections to components it depends on. There exist manifold solutions for managing and observing enterprise components, such as SNMP [6] or the Java Management Extensions (JMX) [16]. The latter allows exposing Java beans as manageable resources, which have a unified but domain specific interface. Manageable resources encompass three concepts: attributes are exposed as key-value pairs, rendering the status of the resource in terms of well-typed values and the needed meta-data to enable others to understand the meaning of the value. While attributes mainly serve for monitoring purposes, operations can be used to actively influence the state of a resource, e.g. to shutdown a component. The third feature allows monitoring/management applications to subscribe to monitoring events. These can be used to push alerts to observers.

The rise of BPEL especially in the field of SOA (service-oriented architecture) leads to a shift of business logic: While traditional systems have hard-coded business logic with hard-coded dependencies, in SOA the concept of loose coupling of self-contained services is thoroughly implemented by inverting the dependencies (Inversion of Control [10]). In Web Services environments such services are orchestrated to more complex services using BPEL, i.e. the business logic is now manifested in the BPEL process model. This makes it even more important to expose BPEL processes as manageable resources and to fit them into global management environments.

In its reference model, the Workflow Management Coalition (WfMC) has standardized a set of in-

[1]`http://www.oasis-open.org/committees/wsrf/`

terfaces that Workflow Management Systems should implement. Interface 5 defines the requirements for a service-oriented Auditing and Management interface and describes methods to access audit data of process instances, activity instances, work items and allows to start/stop/configure process instances. It has been substantiated in [26] where C/C++ interfaces were defined. Later it has been redefined using XML and ASAP [21] in WfXML 2.0 [22]. More recently, a draft version of WfXML-R [23] has been published which adapts WfXML to expose its interface as a RESTful Web service. Although the latter approach also renders process models and instances as resources, the structure of resources is not following the structure of the process model and it currently does not allow manipulating attributes of process instances and activities.

While some vendors followed the recommendations of the WfMC with regard to interface 5, most vendors have developed their own management interface as well as their own audit log format.

Our approach aims at providing a simple but powerful mechanism to render business processes, in particular BPEL process models, as manageable resources. While it can be implemented directly on top of the internal data model of a process engine, it can also be mapped to existing management APIs. Apache ODE[2] or IBM WebSphere Process Server (WPS)[3] for instance expose comprehensive service-oriented management APIs which can be wrapped by a management facade and are that way accessible as manageable resources. Depending on the feature-richness of the originating API it might be the case that not all features can be successfully mapped. For instance it is impossible to realize blocking (i.e. transactional) handlers for management events with WPS while this is possible with Apache ODE and some BPEL engines may not allow modifying variable values remotely.

4. Mapping BPEL to Resource Definitions

In order to allow process model and process instance data to be accessed from outside the engine in a standardized fashion the engine has to expose its internal data structure related to the BPEL processes it manages. In the following subsections we show how individual artifacts of BPEL process models and instances are mapped to resources that represent the internal data model of the engine. Exposing the internal data model as resources has various advantages. First of all the generic resource representation abstracts from the implementation details that are used inside the engine. This allows engine manufacturers to keep their internal data models and wrap them with the resource model thus keeping the optimized internal model as well as offering a standardized resource representation. In case an engine does not support the standardized resource representation, such a representation can be layered on top of the existing proprietary API. As a result, only the information that is provided via the API can be represented as resources, but still this information can be accessed in a standardized manner.

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

[3]`http://www-306.ibm.com/software/integration/wps/`

4.1. Mapping of Process Models to Resources

Every process model that is deployed into an engine is exposed as a separate resource. The resource representing the process model contains nested resources that are derived from the BPEL XML document describing the process model according to the following general rules:

1. Every XML element with a cardinality bigger than one is mapped to a resource, i.e. all XML elements that can occur multiple times within their parent element are mapped to resources.

2. Every element that can contain elements with a cardinality bigger than one, which are not necessarily directly nested, are mapped to resources.

All elements that are not mapped to resources are represented as resource properties.

In the following we will show exemplarily how these rules result in a resource based representation of a BPEL process model. For each deployed process model a resource is created identified by the QName of the process. This resource has per definition two child resources, one containing the process definition and the other containing its instantiations. Figure 1 shows the high-level mapping of process models and instances to resources. The resource representing the `process` itself is a resource according to rule 2 as it contains for instance the `variables` element which in turn contains the `variable` element with an unbounded cardinality. Like the already mentioned `variables` element all other nested elements of a process that contain elements with an unbounded cardinality like the `partnerLinks` or `correlationSets` are represented as nested resources of the process.

In general the name of a resource is the last location path element of the XPath identifying the represented element in the BPEL XML document. The name of a resource property is the name of the represented attribute or element as declared in the XML document. Therefore each resource can be uniquely identified via

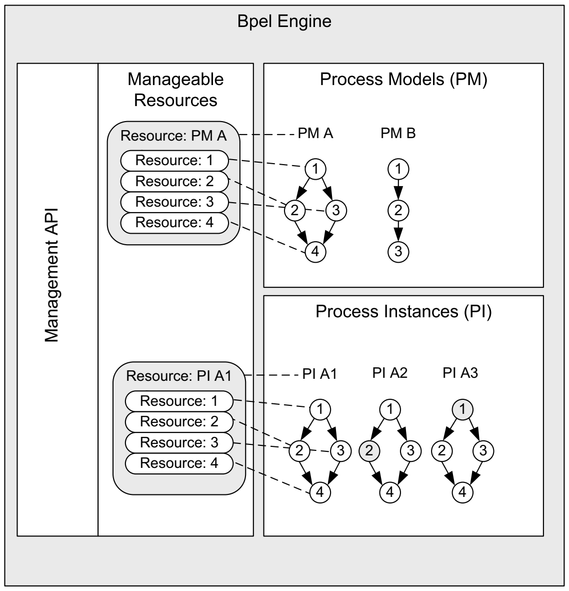

Figure 1. Mapping of Process Models, Instances and Activities to Resources

the XPath according to the underlying BPEL document, prefixed by the QName of the process model and either `/definition` (in case of a process model resource) or `/instances/${instanceId}` (in case of an instance resource).

Handlers such as `faultHandlers` and `eventHandlers` are represented as nested resources of the resource representing a `process` or `scope` activity they are defined in. This is because they are nested in elements (either the `<process>` element or the `<scope>` element) that are represented by a resource and they contain elements that can appear multiple times (rule 2). The root activity of a process which is usually a `flow` or `sequence` is mapped to a resource according to rule 2. They contain standard elements such as `source`s and `target`s which have an unbounded cardinality.

When mapping the `flow` activity to resources there are some special characteristics that need to be considered. In addition to the activities nested inside the `flow` activity, the `links` declared in the `flow` are represented as nested resources of the flow activity resource. These resources also contain information about transition conditions that are defined when using them as sources of an activity. Activities that are sources and targets of links contain a property list which refers to link resources. These link resources are defined as nested resources of the flow activity the links are declared in. The same principle applies to other BPEL constructs that are referenced by other constructs. For example `partnerLink`s that are referenced by interaction activities are referenced via their resource identifier form the resource representing the activity.

All other activities can be mapped following the rules defined above. In an `assign` activity for instance each copy operation is represented as a nested resource of the assign activity's resource.

Special consideration also has to be taken for BPELs *extensibility*. New types of activities can be introduced by means of the `extensionActivity`. Attributes and nested elements of new activity types are rendered as resources or resource properties according to the rules defined above. The same principle applies to `extensionAssignOperations` and any arbitrary BPEL extension attributes and elements.

4.2. Mapping of Process Instances to Resources

The mapping of process instances to resources is similar to the mapping of process models to resources. In general each instance of a process model is exposed as a separate resource. Similar to the BPEL elements inside a process model the BPEL elements inside the process instance are exposed as nested resources of the process instance. The mapping rules for resources and resource properties for the process instance resource representations are the same as for the process model resource representations. This ensures that process instance resources can be accessed following the same access scheme as process model resources. An activity nested in a scope in the process model is for example represented as a nested resource of that scope in both, the process model resource representation as well as the resource representation of a particular process instance.

For the ease of understanding each resource that represents a part of a process instance is also referred to as "instance resource". Resources representing a process instance or BPEL elements belonging to a particular process instance, have additional resource properties and nested resources besides the resources that represent (parts of) a BPEL process model. These additions are concerned with the runtime state of the resources. Each instance resource has an additional resource property, the `state` property. It denotes the current state of the resource. Since different BPEL elements can adopt different states these different instance resource types can have different state values. The possible states of a BPEL activity for example differ from those of a BPEL link. Table 1 lists the different resource types that have a state property and lists the possible values of the state property of the respective resource types.

Instance Resource	Notion of the state resource property and possible values
process instance	specifies the current state of the process instance. One of {`instantiated`, `running`, `suspended`, `terminated`, `faulted`, `complete`}
activities	Specifies the current state of the activity. One of {`initial`, `inactive`, `ready`, `dead path`, `executing`, `waiting`, `terminated`, `faulted`, `complete`}
loops	Specifies the current state of a looping structured activity (e.g. while, repeatUntil, forEach) One of {`initial`, `inactive`, `ready`, `dead path`, `executing`, `waiting`, `terminated`, `faulted`, `complete`, `iteration_complete`, `check_condition` }
scopes	Specifies the current state of the scope. One of {`initial`, `inactive`, `ready`, `dead path`, `executing`, `event_handling`, `waiting`, `termination_handler`, `terminated`, `complete`, `fault_handling`, `compensation_executing`, `compensated` }
links	Specifies the current state of a link. One of {`undetermined`, `ready`, `evaluated`, `true`, `false`}

Table 1. Notion and possible values [17] for the *state* resource property of process instance resources

Additionally, we define a state resource property for other instance resources such as `variables`, `partnerLinks` and `correlationSets`, that may be `uninitialized` or `initialized`.

4.2.1. Handling of repeatable constructs. Special consideration has to be taken when representing repeatable activities, i.e. structured activities that execute a set of activities multiple times either in parallel or sequential. Such structured activities include `while`, `repeatUntil`, `forEach` and `eventHandler`. For each iteration of the repeatable activity a new resource representing the directly nested activity is added as a sub-resource of the resource representing the repeatable activity. As a result it is possible to access individual iterations of the repeatable activity. Figure 2 shows this for the example of a `while` activity. For each iteration of the directly nested activity of the `while` activity a new resource is created. This resource is annotated with the number of the iteration and contains all the nested activities and their values during the iteration of the structured activity it represents. The number of the iteration is only added for the sequential loops and can be used to determine the sequence of the individual iterations. A monitoring tool for instance is then able to display the data for the separate iterations of the loop (i.e. different variable values if the variable is declared inside the loop) by simply accessing the different resources that represent the iterations of the loop.

Fault handlers, termination handlers and compensation handlers are treated as normal nested activities of a scope.

4.3. Events and Notifications

During the execution of process models BPEL engines typically produce a continuous stream of events that indicate various status changes, like for instance the creation of new process instances, variable value changes, etc. Monitoring applications can highly benefit of being subscribed to such an event stream. That way it is possible to observe the engine regarding its healthiness or even to measure business relevant key performance indicators (KPIs). However, propagating such an unfiltered stream of events to monitoring applications may lead to severe performance issues. In our approach, observers can subscribe to manageable resources, i.e. to single parts of a process model or even to a single process instance. When registering for certain events on a process model resource, the events are also propagated for each instance of this resource. Events are only produced for those status changes a subscriber is registered to; that way the event stream is filtered on a fine-grained level and consequently the overhead caused by producing events can be minimized. In case a subscriber is registered to a resource, each change of a resource property raises an event. Beside the status of activities, changes of variable values and partnerlink EPRs are in particular of interest, as a possible property

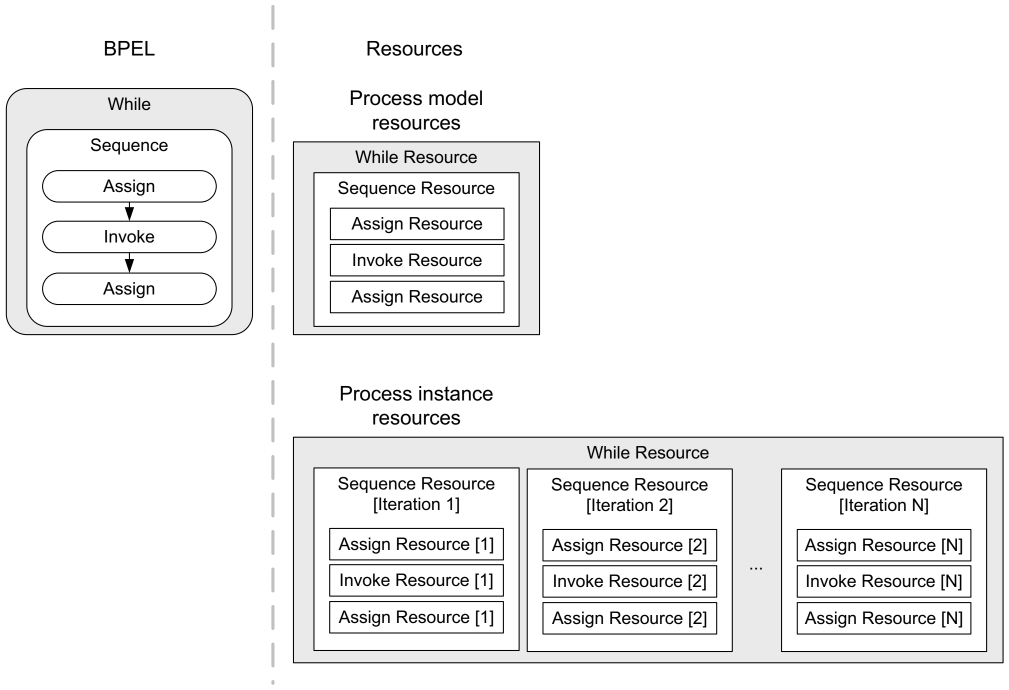

Figure 2. Mapping of BPEL loops to process model and process instance resources

change may be overridden by an external subscriber and replaced by another value; this enables an AOP-like programming model for BPEL processes. In such scenarios it is important that event notifications are propagated in a transactional and blocking fashion to ensure that the event-triggering resource has not been manipulated meanwhile.

4.4. Audit Trail Information and Resources

Most implementations of BPEL engines distinguish data of running instances of processes which is persisted in a runtime database and data about finished instances of processes which is persisted in an audit trail. Finished process instances are instances that are no longer running, for example instances that have successfully completed, have been terminated, faulted or compensated. In order to facilitate access to the audit trail, we propose to use the same access scheme for the audit trail data that we use for the running instance data.

Thus, our approach is also beneficial for e.g. assurance frameworks which analyze the audit trail information to detect whether regulations and laws were taken into account accurately during execution.

In particular that means that for each finished instance of a process model the information can be accessed via a resource representation that corresponds to the resource representation of the running instances. This approach also allows mixing the process execution data and audit trail data in the resource representations. Therefore each instance resource contains a nested read-only "audit trail data" resource that represents the entries in the audit trail that are available for this resource. These entries include resource property changes such as state changes, value changes, the date and time of the changes and other information kept in the audit trail (cf. [24]). Changes performed from the outside of the engine on the resources are also logged in the audit trail and are therefore included in the audit trail data resource.

5. Scenario

We illustrate the application of the resource-based management framework for BPEL processes based on a "Hello World" BPEL process as depicted in the right part of Figure 3: The process performs three basic activities that are nested inside a sequence activity. The first activity receives a message from a partner, the second one writes a string to the variable `answer`, and the third activity sends a message back to the partner. The arrow denotes the state of execution, i.e. the engine has not yet executed the assign activity. Figure 3 shows only the resource representing the variable `answer` in full detail, whereas all other resources are not shown completely in order to keep the figure simple. The figure shows the resource representation of the variable in a

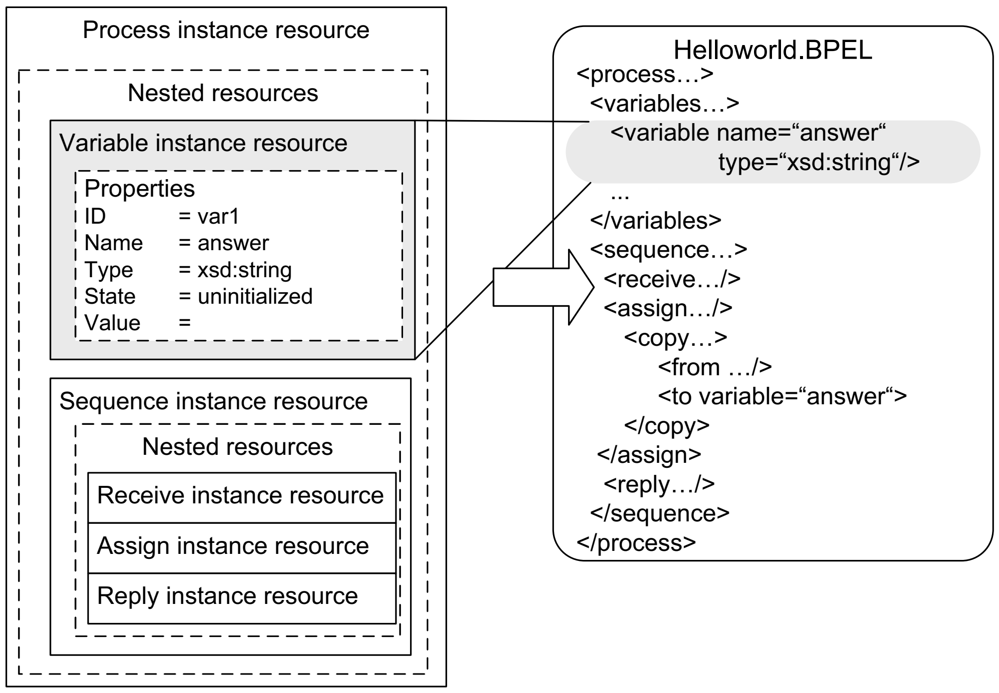

Figure 3. Hello World process and resource representation

state of the process instance where the variable has not been initialized yet. This is indicated by the `state` property of the variable instance resource that is set to `uninitialized`. In case a management tool would perform a get operation on the variable instance resource identified by the id "var1", it would get the resource as shown in Figure 3.

After the assign activity that writes to the `answer` variable has been executed the `state` property of the `answer` variable is set to `initialized`. This state is shown in Section 4. Additionally the `value` property of the `answer` variable is set to the value that was assigned to the variable ("HelloWorld" in this case). In case a monitoring tool would now perform a get operation on the resource "var1", it would get the resource as shown in Figure 4. In addition to the "raw" resource data the tool could now perform a get operation on the nested audit trail data and could retrieve all audit data that has been logged for the resource "var1". In case of the variable the audit data can be used to determine which values have been assigned to the variable during the runtime of the process instance. This can be very important for example for debugging or process mining. In case an administrator needs to perform changes in the process instance, the corresponding management tool can perform put operations on the resource to update it. For example a management tool could allow an administrator to change the value of the variable from "HelloWorld" to "HelloWorldModified" by performing a put operation on the resource "var1" that contains the new value. Changing variable values and other data in a process instance is an important task that needs to be performed by administrators to repair faulted process instances. Given a well-designed process model, process instances mostly fault because of non-foreseeable occurrences. In these situations the approach proposed in this paper is very important as it allows to arbitrarily changing any property in a process instance to repair the process instance. The question left open in this example so far is: How does the management tool know when the state or the value of the variable has changed without constantly polling the resource? Since polling would require a lot of get operations on resources and therefore would put an unnecessary load on the engine, a different mechanism must be employed. In order to be notified about property changes of the variable, the monitoring tool can subscribe to a property of a resource (in this case the "var1" resource). The engine then sends an event to the subscribers if the property changes. The subscribers can then access the changed resource and other resources in order to retrieve the necessary data they require upon such a resource change.

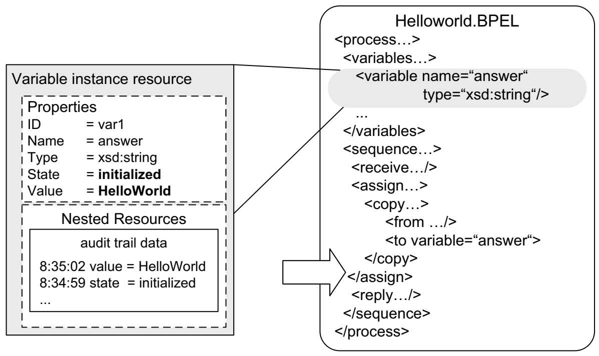

Figure 4. Hello World process after assign completed and resource representation

6. Suitable Technologies for the Management Infrastructure

In contrast to a traditional service-oriented management API (providing operations like `getActivityStatus(aid)` or `getVariableValue(varname)`), resource-orientation allows to browse process models and their instances and thus enables a reflective interaction model with a BPEL engine supporting a resource oriented access scheme. Instead of first explicitly asking for an ID of a deployed process model and querying its running instances by passing this ID, our resource-oriented approach makes such data available in a more convenient manner: Since the resource tree is modelled following the BPEL meta-model, the location of requested data can be derived according to the BPEL process definition. Modelling and using manageable resources for process management forms some requirements on the technical level which can be tackled by the numerous management infrastructures. There are numerous management solutions available of which we present the three most relevant including a discussion on their suitability for implementing our approach.

6.1. JMX

The Java Management Extensions (JMX) [16] adds management facilities to Java's runtime library. JMX can be used to e.g. manage JEE application servers or CPU/memory load of java programs locally as well as remote via RMI, HTTP or Web services. The management interface, a so called MBean is basically an ordinary Java interface defining all management operations to be exposed. Getters and setters are automatically recognized and exposed as attributes (readable and writable), all other methods are exposed as operations to allow invoking arbitrary management code. JMX also supports publish/subscribe-like notifications. Once modelled, MBeans are registered to an MBean server using a specific name which identifies that MBean in the server's context. These names can be organized hierarchically. That way our approach to resource-oriented BPEL management can be realized using JMX. Resources are rendered as MBeans, resource properties are attributes in the JMX terminology and events are pushed to client applications via JMX's notification mechanism. Nested sub resources can be realized with an appropriate tree-based naming scheme.

6.2. REST

Representational State Transfer (REST) [8] is a popular architectural style for resource-oriented distributed systems like the World Wide Web. REST utilizes the four most popular HTTP [9] verbs, `GET` for retrieving, `PUT` for updating, `POST` for creating and `DELETE` for deleting a resource. Resources are uniquely identified using URIs [5] and are interlinked using hyperlinks to make the resource model browsable for humans as well as machines.

By using a smart URI design it is possible to distinguish between resources and collections (i.e. resources containing only hyperlinks to resources). That way it is possible to express sub-resources as required by our approach. Notifications are not directly supported by RESTful architectures. However there exist extensions (e.g. GENA [7], RNA [15]) that enable publish/subscribe behaviour either via asynchronous communication channels or via polling. To fully support our requirements polling for new notification events is not sufficient as it does not allow transactional/blocking event processing.

6.3. WSRF

The Web Service Resource Framework (WSRF) facilitates an approach to accessing Resources in a Web services environment. WSRF is a foundation of standards for defining, accessing, and managing stateful Web services (i.e. resources) defined using WS-Resources [11]. Such a resource defines resource properties which can be read, modified and queried by using WS-ResourceProperties [12]. Also of relevance is WS-Notification [20] which defines how to push information to subscribed services. Thus it meets all requirements our approach imposes on a suitable implementation framework.

Other approaches towards implementing our proposal based on WS-* technologies are (i) using Web Services Distributed Management (WSDM) which defines techniques for the management *of* Web services (MOWS) [13] *using* Web services (MUWS) [14] or (ii) creating a custom solution based on WS-Transfer [2] in combination with WS-Notification.

7. Implementation

As a proof of concept we have built a prototype that implements our approach towards a Management Framework for BPEL. It is based on the Apache ODE BPEL engine [4]. ODE is an Open Source BPEL engine, available under the Apache Software Licence 2.0. Furthermore we used the WS-ResourceFramework, mentioned in section 6, to build the Web service that is responsible for resource management. This interface could also easily be implemented with JMX [16] or REST [8]. For issuing events the WS-Notification framework [20] is used.

We extended the Apache ODE BPEL engine with a new Web service that provides the interface for the Management Framework for BPEL. The corresponding WSDL contains all WSRF functionality plus some WS-Notification functionality for subscribing to an event. Operations derived from the WSRF specification are for example `GetResourcePropertyDocument` or `SetResourceProperties`. Operations to manage running processes like `suspend` to pause a running process or `terminate` to terminate a running process are already provided by the standard Web services of ODE.

To run BPEL processes in ODE the XML document that represents a process has to be compiled into the internal object model for processes of ODE. This is done automatically with all BPEL documents that are placed in the *processes* subfolder of an ODE installation. In our prototype we map this object model of every instance of a deployed process to WS-Resources like depicted in section 4. These WS-Resources are just an representation of the internal object model of the engine and they are accessible through the Web service we added to ODE. Every query to this Web service is routed to the corresponding object of the internal object model of the desired process and the desired operation is executed. To address resources of a running process XPath expressions are used. The XPath expression must be built based upon the XML representation of the process containing the desired resource. The XPath expression to address a resource must be extended on the left side by adding `/${procQName}/instances/${instanceId}`. A valid XPath expression could look like this: `/{urn:iaas}helloWorld/instances/233/process/sequence/receive`.

To subscribe to an event every WS-Resource has a `subscribe` operation. This operation was taken from the WS-Notification specification. To enable ODE to detect property changes on its internal object model for processes every object now extends the abstract class *BpelEvent* so that event listeners could be registered on these objects in order to determine property changes. If such property changes are detected the clients subscribed to these events are notified.

Another Web service was created by using the WSDL documents from the WS-Notification specification to provide an interface for issuing events. In our prototype the events that are already implemented in the ODE BPEL engine are supported. In future we want to completely implement the BPEL event model [17].

8. Conclusion and Outlook

In this paper we presented an approach towards representing engine internal process models and instances as resources in an engine-independent manner. The approach enables interoperable resource based management of WS-BPEL processes. We showed a mapping from BPEL process models and instances to resources and presented scenarios that benefit from our approach in a sense that it provides for a more convenient and precise manner to register for events. In contrast to subscribing to proprietary events of BPEL engines, the resource-based approach allows subscribing to resource property changes. The resource-based approach thus prevents the emission of a potentially huge set of engine-defined events that are not important for any further processing. We demonstrated the applicability of our approach by means of a proof-of-concept implementation based on the Apache ODE BPEL engine.

This paper mainly focuses on how to represent engine internal process models and instances and how to access this data. In future work we will further describe how the manipulation of process model and process instance data influences the execution of WS-BPEL processes. This will involve a thorough examination of resource properties that can be changed at any time and properties that can only be changed under certain conditions during runtime of a process instance. This is of particular importance as for example the change of the status of a link has thorough influence on the execution of a whole process instance. Our future work also includes research on how the resource based management approach we presented for WS-BPEL engines can be extended for middleware supporting other WS-* standards (such as

WSDL, WS-Addressing, WS-Coordination etc.). Furthermore, we will investigate how Complex Event Processing (CEP) [19] can help to reduce and customize the events emitted by the BPEL engines and to define KPI-related rules directly in the management infrastructure.

Acknowledgements

The work published in this article was partially funded by the SUPER project[4] under the EU 6th Framework Programme Information Society Technologies Objective (contract no. FP6-026850) and the MASTER project[5] under the EU 7th Framework Programme Information Society Technologies Objective (contract no. FP7-216917).

References

[1] Agrawal, A. et al. WS-BPEL Extension for People (BPEL4People), Version 1.0, 2007.

[2] Alexander, J. et al. Web services transfer (ws-transfer). W3C Member Submission, W3C, Sept. 2006.

[3] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guízar, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu. Web Services Business Process Execution Language Version 2.0. Committee Specification, OASIS Web Services Business Process Execution Language (WSBPEL) TC, Jan. 2007.

[4] Apache ODE Team. Apache ODE BPEL Engine.

[5] T. Berners-Lee, R. T. Fielding, and L. Masinter. Uniform resource identifiers (uri): Generic syntax. Internet RFC 2396, August 1998.

[6] Case, J. and Fedor, M. and Schoffstall, M. and Davin, J. A Simple Network Management Protocol (SNMP), 1990.

[7] J. Cohen and S. Aggarwal. General event notification architecture base. Internet draft, Internet Engineering Task Force, July 1998. Expired.

[8] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, Irvine, California, 2000.

[9] R. T. Fielding, J. Gettys, J. C. Mogul, L. Masinter, P. J. Leach, H. Frystyk Nielsen, and T. Berners-Lee. Hypertext transfer protocol — http/1.1. Internet Draft, HTTP Working Group, March 1998.

[10] M. Fowler. Inversion of Control Containers and the Dependency Injection pattern, Jan. 2004.

[11] S. Graham, A. Karmarkar, J. Mischkinsky, I. Robinson, and I. Sedukhin. Web services resource 1.2 (ws-resource). *OASIS, January*, 2006.

[12] S. Graham and J. Treadwell. Web services resource properties 1.2 (ws-resourceproperties). *OASIS, April*, 2006.

[13] I. S. Heather Kreger, Kirk Wilson. Web Services Distributed Management: Management of Web Services (MOWS 1.1). OASIS Standard, OASIS Web Services Distributed Management TC, Oct. 2006.

[14] W. V. Heather Kreger, Vaughn Bullard. Web Services Distributed Management: Management Using Web Services (MUWS 1.1) Part 1. OASIS Standard, OASIS Web Services Distributed Management TC, Oct. 2006.

[15] S. Jacobs. Rna: Restful notification architecture. Draft, July 2003.

[16] JSR 3 Expert Group. Java management extensions (JMX). JSR 3, Java Community Process, 2000. Available at `http://www.jcp.org/en/jsr/detail?id=3`.

[17] D. Karastoyanova, R. Khalaf, R. Schroth, M. Paluszek, and F. Leymann. BPEL event model. "Technical Report Computer Science" 2006/10, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, University of Stuttgart, Institute of Architecture of Application Systems, November 2006.

[18] M. Kloppmann, D. Konig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. WS-BPEL Extension for Sub-processes – BPEL-SPE. *Joint white paper, IBM and SAP*, 2005.

[19] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, 2002.

[20] OASIS Web Services Notification (WSN) TC. Web services notification 1.3. *OASIS*, Oct. 2006.

[21] K. Swenson, J. Ricker, and M. Krishnan. Asynchronous Service Access Protocol (ASAP), May 2005.

[22] K. D. Swenson, S. Pradhan, and M. D. Gilger. WfXML 2.0: XML Based Protocol for Run-Time Integration of Process Engines. WfMC Draft, WfMC, Oct. 2004.

[23] K. D. Swenson, S. Pradhan, M. D. Gilger, M. Zukowski, and P. Cappelaere. WfXML 2.0: XML Based Protocol for Run-Time Integration of Process Engines. Draft 0.4, WfMC, Jan. 2008.

[24] W. van der Aalst, B. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267, 2003.

[25] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. 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.

[26] Workflow Management Coalition. Workflow Management Coalition Audit Data Specification. WfMC Technical Report, WfMC, Sept. 1998.

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

[5]`http://www.master-fp7.eu/`