Institute of Architecture of Application Systems

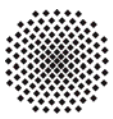
Towards Reference Passing in Web Service and Workflow-Based Applications

Matthias Wieland, Katharina Görlach, David Schumm, Frank Leymann

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{wieland, goerlach, schumm, leymann}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{WielandGSL09,
  author    = {Matthias Wieland and Katharina Görlach and David Schumm and
               Frank Leymann},
  title     = {Towards Reference Passing in Web Service and Workflow-Based
               Applications},
  booktitle = {Proceedings of the 13th IEEE International
               Enterprise Distributed Object Computing Conference, EDOC 2009,
               1-4 September 2009, Auckland, New Zealand},
  year      = {2009},
  pages     = {109-118},
  doi       = {10.1109/EDOC.2009.17},
  publisher = {IEEE Computer Society}
}
```

Universität Stuttgart
Germany

2009 IEEE International Enterprise Distributed Object Computing Conference

Towards Reference Passing in Web Service and Workflow-based Applications

Matthias Wieland, Katharina Görlach, David Schumm, Frank Leymann
Institute of Architecture of Application Systems, University of Stuttgart, Germany
Firstname.Lastname@iaas.uni-stuttgart.de

Abstract

In a Service-Oriented Architecture (SOA) based on Web Service technology the services typically communicate with each other by passing data values directly from one service to another. In the case the services are orchestrated by workflows the services receive their input values from the workflow engine and return their calculated results back to the engine by value. In this paper we show several use cases where such value passing behavior has drawbacks. To address this challenge we introduce the concept of pointers in SOA. Pointers allow services to pass their data by reference which is a fundamental advantage for Web Service communication. Furthermore we show an extension of BPEL that introduces reference variables as new type of data containers in workflows. In addition, for the management of pointers we present the Reference Resolution System which can be used in very flexible setups either as central or distributed system.

1. Introduction

Workflow technology provides methodologies and corresponding products to support modeling, execution, and management of business processes that have been carried out manually and through a diversity of non-integrated systems before [15]. Thus BPEL (Business Process Execution Language [3]) as a standard for workflow modeling and execution gained major influence in many enterprises and within the software industry. BPEL is building on the Web Service (WS) standards and provides a recursive aggregation model for WSs [7]. As BPEL is more and more used in different application domains, special requirements and new challenges arise. One important requirement that exists in many of such domains is a mechanism that allows using reference passing instead of the standard value passing behavior. In standard BPEL, data is stored in variables and is always passed by value to and from the WS to the workflow. As a solution we introduce a new type of BPEL variables, so-called reference variables, which can be used as pointers to data. Thereby, only pointers have to be passed between WSs and workflows and not the values of the variables anymore.

As motivation we present in the following three very distinct domains where this requirement occurs:

- Context-aware workflows [21]: Dynamic context data is changing very often because it is sensed with a high sampling rate. Therefore, in [21] a dynamic context variable was described as requirement of context-aware workflows. The BPEL reference variables introduced in this paper can be applied for exactly that dynamic evaluation of variables without explicitly updating them with an invoke activity in the workflow.
- Compliance [8]: Flexibly reacting to frequently changing requirements coming from laws and regulations, such as [16], becomes a necessary part of business process management. Using references to static values outside the business processes that are part of those requirements allows flexible adaption of business logic by changing the value once in the reference management system. The process model and all of its running instances do not have to be changed. The next time the value is used in any process it will be updated automatically because the system dereferences the actually valid value.
- Scientific workflows [6]: In scientific workflows large amounts of data are processed. By the usage of data references transmission of huge data sets to and from the workflow system can be reduced.

To share pointers between WSs and workflows they have to be managed externally to the workflow system. This allows building a global or company-wide variable concept which extends the BPEL variable concept that only supports process-wide accessible variables.

1541-7719/09 $25.00 © 2009 IEEE
DOI 10.1109/EDOC.2009.17

Based on the previously described domains we identified two general use cases for reference variables: Realizing *global dynamic variables* covering the domain of context-aware workflows and compliance (described in section 1.1) and *data flow between WSs* covering the domain of scientific workflows (described in section 1.2).

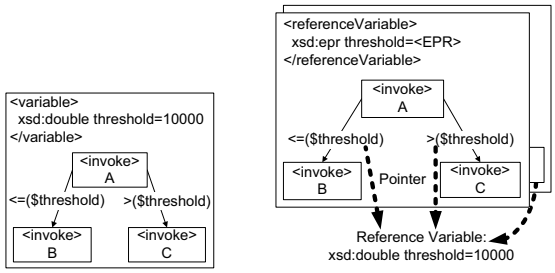
As defined in [17] a *pointer* is a programming language construct whose value *refers* directly to a *value* stored elsewhere. Thus a pointer is also called *reference*, and resolving the value to which a pointer refers to is called *dereferencing* the pointer, thus in this paper we use the term pointer and reference interchangeably. In programming languages pointers to data usually improve performance for repetitive operations such as traversing data [17].

For this paper that definition of a pointer is not precise enough. We use two types of pointers: *direct pointers* that fit the previously cited definition (e.g., a pointer on a hash value). Additionally we introduce *indirect pointers*. This type of pointer extends the existing definition as follows: An indirect pointer is a programming language construct whose value refers indirectly to a value stored elsewhere. Indirectly means the pointer points to an artefact (e.g. SQL query) that can be used to retrieve the value from a given system (e.g., SQL database).

In the REST architecture introduced in [11] references are a central part. A Uniform Resource Identifier (URI) [5] can be used for pointing to a resource. This paper uses concepts of REST to introduce references in SOA. We represent a reference in form of an *Endpoint Reference* (EPR) [20] (see Section 3.1). This EPR contains a set of references (i.e. URIs) that are combined in a semantic structure in the EPR: one URI for the reference resolution system and another URI for parameters used for the resolution of the reference (e.g., to address an SQL query). Our reference representation as EPR is designed self-contained. Therefore our EPRs can be used to transfer the complete state like in REST. Since there is the need for references in SOA we proposed this way of realization with existing Web Service standards.

The introduction of references to SOA has the following advantages:

- Reduced data transfer over the network: No data transmission to process engine forth and back between every Web Service call would be needed anymore.
- Improved performance of workflows: The process engine only has to handle the pointer, not the data itself. This leads to faster workflow execution. The Web Services themselves could handle the secure, traceable storage of the data.
- Improved workflow modeling: No technical actions like updating static values in process models are needed anymore if pointers are used. This results from the automatic dereferencing of the pointer on usage.



(a) Todays implementation of process variables.

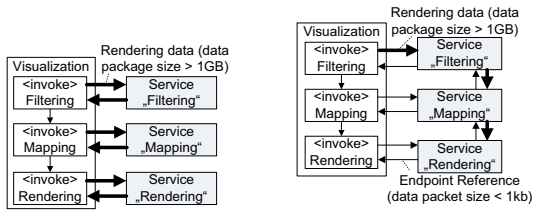
(b) Reference variable used in different workflows.

Figure 1. Example for a global company wide threshold variable.

1.1. Use Case 1: Global Dynamic Variable

The first generic use case for pointers in workflows is the usage of global variables that are valid and accessible beyond different workflows. This use case is illustrated in Figure 1. An example workflow is shown in (a) that uses the value of variable *threshold* for the evaluation of the transition condition between the activities A, B and A, C. In this example the value is stored in a standard BPEL `<variable>`. But let us assume there are a lot of different workflows using this *threshold* value, as in (b). In this case it is important to store the value outside the workflow, because otherwise if the value has to be changed, every workflow using it has to be remodeled and redeployed. This change might happen very often, e.g., because of a change in the dollar price, the federal funds rate or the company risk level. To keep this strategic values out of the control flow in (b) the pointers introduced in that paper are used. A transparent usage of pointers is realized in BPEL with the new construct `<referenceVariable>`. This allows an easy adaptation of running workflow instances without changing the underlying workflow model by adapting the value outside of the workflow in the Reference Resolution System (RRS) that stores the valid value of the *threshold* pointer. In this use case the pointers are dereferenced by the workflow engine itself, every time the *referenceVariable* is used in the control flow.

In the previous example the update frequency of the value is not very high. But there are scenarios where the dynamic change of the *referenceVariable* is very important. In the area of context-aware computing, context data is captured by sensors every second. A context-



(a) Transfer of huge data sets between workflow and services.

(b) Transfer of references between workflow and services.

Figure 2. Example for data transfer in scientific workflows.

aware workflow uses that sensed data as variable data, but the changes are so fast (<1sec) that it is not possible to send every change of all context objects to the workflow. By using pointers, this problem can be solved. The sensors update the data values directly in the system managing the values of the references and only if a context-aware workflow uses the pointer, the actual value is dereferenced and used.

1.2. Use Case 2: Data Flow Between Web Services

Workflows in the scientific area make use of a more data-centric approach compared to the business area. Huge amounts of data have to be processed in analysis, experiments and simulations. Compared to business workflows the data flow complexity is typically higher while the control flow complexity is typically less. This results in lots of transmissions of huge amounts of data which makes the usage of BPEL very costly for scientific workflows. The majority of scientific data is of no importance for the process logic, and thus can be ignored by the workflow management system. By using data references the transfer of huge amounts of data through the engine can be avoided, and no relevant information is lost.

The scientific workflow in figure 2 shows a typical visualization process in simulations. The visualization workflow receives a lot of data during simulation time and coordinates the execution of algorithms (filtering, mapping and rendering) on simulation data. When the simulation produces huge data sets they have to be passed to the *Filtering* service. Later the filtered data has to be sent back to the workflow in order to make it available to the *Mapping* service. Similar huge data sets have to be transferred while invoking the *Mapping* and *Rendering* service. Consequently the execution of the visualization workflow in figure 2 demands multiple transports of huge data sets through the engine. Usage of pointers in the visualization workflow shifts the responsibility for data transport to participating services and therefore scales down the amount of data that has to be transferred through the engine.

This use case addresses all applications where the transport of data is a cost factor for itself. This is the case in workflows where huge amounts of data are processed by various services, for example in simulation workflows. The transfer of processed data to the workflow engine and back to a subsequent service that takes care of further processing is not feasible for those applications in most cases. Yet, one exception is when this data is relevant for the further flow of control within the process. This circumstance is taken into account by introducing references in BPEL without replacing the common way to treat data, which is call by value and by allowing to access the value of the reference from within the workflow.

The data reference concept applies to all workflow systems, in this paper we demonstrate the approach based on BPEL, because it is an accepted standard for describing workflows in a WS environment.

The reminder of the paper is organized as follows: In section 2 an overview of related work to the contribution of this paper is presented. Building on that in section 3 the concept and schema structure of pointers as the main contribution of the paper is presented. Furthermore the architecture of the RRS as a system for the management of the introduced pointers and for storage and retrieval of the values is presented. In section 4, the following two necessary parts for realizing the contribution are presented: First we show how to realize a Web Service wrapper that allows making existing Web Services capable of passing values by reference without changing their implementation. Subsequently, we show how to realize a transparent integration of the pointer concept in BPEL. Section 5 concludes the paper and describes future steps and unsolved challenges in the presented pointers concept.

2. Related Work

BPEL data extensions and SQL integration: IBM's information integration for BPEL (II4BPEL) [14] allows simple and efficient access to relational database systems, using SQL, from within business processes. This approach of IBM allows dereferencing an SQL query to a result set containing the values. But this approach is tied to the usage of SQL databases for storing the data. Our approach allows storing the data of the pointers in any kind of system optimized for that kind of data. Other vendors provide comparable SQL support in BPEL which is described in detail in [19]. These solutions could also be used for SQL-based dereferenciation

of pointers. However, to use theses solutions, vendor-specific extended BPEL engines are needed. There also exist extensions to BPEL to cope with data intensive service applications like BPEL-DT [13]. In that work it is stated that Web Services and BPEL have weaknesses concerning the data aspects [13]. As a solution to that problem [13] introduces BPEL data transitions for modeling data flow. In addition our approach introduces pointers for data passing on the Web Service layer and BPEL reference variables for a transparent usage of reference-aware Web Services. In [1] a process engine extension is presented that allows using references to an internal data storage. However it is engine specific, not commonly applicable to Web Services and does not conform to the BPEL standard and, thus, is incompatible with other vendors.

Business Rules and Compliance: Business Rules provide a solution for increasing the flexibility of processes by separating decision algorithms (i.e., rules) from the process model. This approach is also applicable to separate compliance requirements from the process models, such as the one described in [18]. Due to the focus of business processes the applicability of business rules in the field of Web Services in general is limited, and typically business rules are bound to a certain implementation for the management and evaluation of rules.

Scientific workflows and BPEL: With BPEL as a de facto standard for business processes it is reasonable to use BPEL for the implementation of scientific workflows. In [2] exception handling, user interaction, recovery and rollback mechanisms were identified as advantages of BPEL in the scientific domain. Nevertheless there exist other requirements like adaption of workflows or more expressiveness for data handling that have to be satisfied. The approach presented here enables scientists to use data references during workflow modeling and avoids huge amounts of data that have to be transferred through the BPEL engine. Other possibilities for reducing the amount of data in scientific workflows for example are utilized in the Pegasus Workflow Management System [9]. It provides mechanisms for identifying existing data that do not need to be produced and the concept of garbage collection.

3. Architecture of the Reference Resolution System

Figure 3 shows the architecture of the Reference Resolution System (RRS). On the bottom of the figure the different possible clients of the system are shown:

- *Workflows* or *Web Services* that have initially only the rights to get a value of a reference.

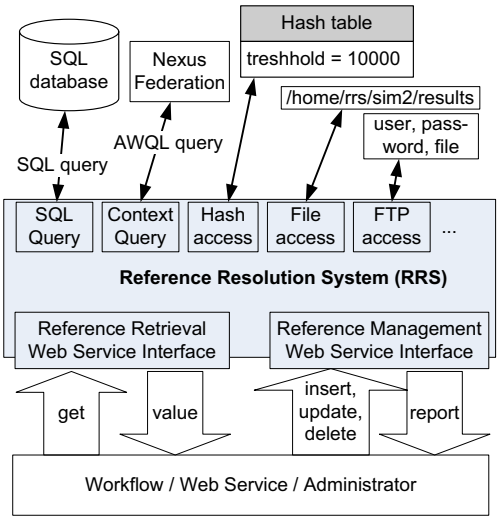


Figure 3. The Reference Resolution System (RRS) architecture.

- A human user, i. e. the *administrator*, who has the rights to insert, update, and delete references in the system. He also can permit other rights to a workflow or Web Service, e.g., a workflow "Set new risk level" could have the right to make an update on the threshold in figure 1.

For each of the different clients a Web Service interface is provided by the RRS. It is important to provide different interfaces as a basis for security and access authorization. This results from the assumption that most workflows and Web Services only read values of references and very seldom do writes. E. g. a context-aware workflow reads a dynamic context variables to access the actual context, but the variables are updated only by sensors behind the scenes and written directly to the data storage (Nexus Federation). Of course there exist domains where workflows and Web Services read and write values similarly. Workflows and Web Services in such domains use both interfaces in the RRS. The two different interfaces are:

- The first interface is the *Reference Retrieval Web Service* with the operation `get` and an endpoint reference (EPR) as input value. The concept of an EPR is introduced by the Web Services Addressing standard [20] and can be used without change for the representation of a pointer. The output of the operation is the value referenced by the given EPR.

- The second interface is the *Reference Management Web Service*, providing three operations: `insert` to create a new reference. The input is the value

of the reference, where to store it and how to retrieve the value later, e.g., an SQL insert statement, and an SQL query for retrieving the value from a database. The output is a report containing the newly generated related EPR. The `update` operation is used for changing a value stored for a reference, the input is the new value and how to store it; the output is a report stating if the update was successful. `Delete` removes a reference which means its value and all additional information is deleted out of the system. The input value is the EPR to be deleted, and the output is a report if the deletion was successful.

The main component is the *Reference Resolution System* itself. It connects both Web Service interfaces and holds a set of adapters that integrate very different *data sources*. Each adapter consists on the one hand of a *lookup service* for stored queries and information and on the other hand of a *execution service* for executing queries on the data source. Based on the reference an appropriate adapter is selected and used in order to resolve the reference to a value. On top of figure 3 the five most relevant adapters for the use cases are shown. However the system is extensible and adapters for any kind of data storage can be added:

- *SQL Query* for lookup of a stored SQL query and its execution on any JDBC compatible *SQL database*.
- *Context Query* for lookup and execution of an Augmented World Query (AWQL [12]) on the *Nexus Federation* [12] to gather context data as value of the reference.
- *Hash access* for direct lookup of the value in a *hash table*.
- *File access* for reading data from a *file system* and returning the data stream as value.
- *FTP access* for using *remote files* as value. To do so, the credentials username and password have to be provided.

RRS is designed to be used as a distributed system, hence different possible setups are supported: Either add a RRS to every Web Service, or provide a RRS for each data source, or connect one RRS to the workflow engine. Resulting from that a reference has to contain the endpoint of the RRS that is able to resolve the reference. Thus the aim is to store all information needed for dereferencing a pointer in the EPR that describes the reference. The structure of that EPR is described in the next section.

```
<wsa:EndpointReference>
    <wsa:Address>
        xs:anyURI
    </wsa:Address>
    <wsa:ReferenceProperties>
        <rrs:resolutionSystem>
            (xs:String | xs:anyURI |
            xs:QName)
        </rrs:resolutionSystem>
    </wsa:ReferenceProperties>
    <wsa:ReferenceParameters>
        (xs:anyURI |
        xs:any)
    </wsa:ReferenceParameters>
    <wsa:PortType>xs:QName</wsa:PortType>
    <wsa:ServiceName PortName="xs:NCName"?>
        xs:QName
    </wsa:ServiceName>
    <wsp:Policy>Policy</wsp:Policy>?
</wsa:EndpointReference>
```

Listing 1. EPR schema.

3.1. Representation of a Reference in an EPR

After describing the architecture we now present the most important part of the system, the reference itself, by describing how it is represented using the WS-Addressing concept of an EPR [20]. The aim of the reference design is to keep it as flexible and extensible as possible and to try to achieve that the EPR is completely self contained. That means that all information needed to resolve the reference shall be contained in the EPR. Listing 1 shows the schema of a EPR and how it is used for the representation of a reference. The style of the schema uses the BNF conventions: "?" denotes optionally (0 or 1), "*" (0 or more), "+" (1 or more) and "|" represents choice.

The different parts of the EPR are used for the following purposes: The `Address` part points to the endpoint of the RRS system that is managing the value of the reference. In `ReferenceProperties` the adapter within the RRS for resolving the reference is specified (`resolutionSystem`). All information the resolving adapter needs to work is specified in the `ReferenceParameters`. This information can be either provided directly like for example an SQL query, or the information can be referenced with a URI. In this case the information has to be provided on creation of the reference. Both solutions have advantages: The *direct specification* allows changing the query in the EPR while runtime. The *stored query* on the other hand

assures that everybody uses the same query and that nobody can inject unwanted queries. The `PortType` and `ServiceName` are technical parameters in order to allow dynamic binding of the RRS. `Policy` allows adding non-functional properties to the query execution, this part is optional. It could for example be used to specify the quality of context (QoC) that should be valid for the returned data (e.g., actuality, correctness, etc).

3.2. Main Design Issues and Benefits of the Architecture

Self-contained EPR: The EPR holds all data to resolve a pointer without any additional knowledge about the pointer context: the endpoint of the RRS in concern, the adapter for resolving the pointer and possibly the query itself. This allows using many different query-based data management systems. The traceability can be assured when RRS stores the history of all values over time. So by using the EPR and the workflow log with time stamps the workflow data can be requested later for audit.

Keep BPEL Code Clean: The main advantage is to keep the real data queries or data handling out of the workflow models. Whereby the workflow models get easier to understand for the workflow designer.

Integration of Existing Systems: The RRS allows the integration of existing data sources that have no Web Services interface into a SOA environment, because the RRS together with the EPR and the integration adapter serves as Web Service interface for these data sources. Furthermore the advantage is that the existing data source is improved by using RRS for the access because the data source now can be accessed by value and by reference. Furthermore the access of the data source is possible with one request in both cases and the data source can be integrated into workflows.

Easy-to-Use Interface: Because dereferenciation of pointers is the most used operation it has a simple interface for retrieving the values (Reference Retrieval Service). The management of the values is done using a more complex and secure interface (Reference Management Service).

3.3. Architecture usage examples

In this section the architecture usage is shown for the two use cases described in section 1. Figure 4 shows an example for use case 1 (global dynamic variable in

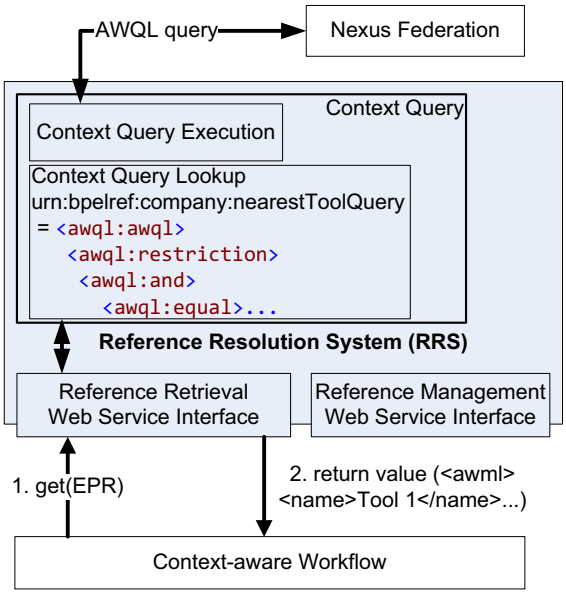


Figure 4. Use case 1: RRS provides dynamic variables for context-aware workflows.

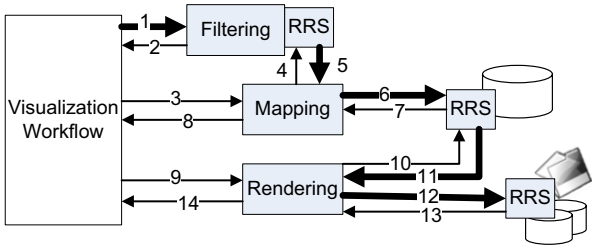


Figure 5. Use case 2: RRS for reference passing between WSs in scientific workflows.

section 1.1). Here a context-aware workflow uses a reference variable evaluating the nearest tool to a location. For the workflow the reference is transparently resolved on demand when the information is relevant for a decision. For retrieving that information a context query was previously created with the insert statement under the name: ”urn:bpelref:company:nearestTool”. Now the workflow uses the reference variable and the workflow engine resolves it to a value by using the `get` method (1.) with the EPR as input. Thereupon the RRS looks up the stored query for the given name and executes the query by using the specified Context Query adapter. The result set is returned to the workflow as return value (2.)

Figure 5 shows an example for use case 2 (see chapter 1.2). The scientific workflow passes references (thin arrows) instead of values and by that the engine is discharged of handling and passing all the high volume data results (thick arrows). First of all, the workflow sends

the first data value to a *Filtering* service (1). This service sends the filtered data to its own RRS. The RRS handles the storage of data and relates a reference to this data. This reference is sent back to the *Filtering* service that returns the reference to the workflow (2). In the next step, the visualization workflow invokes the *Mapping* service and sends data parameters by reference (3). The *Mapping* service sends the received reference to the RRS of the *Filtering* service which returns the related data (4 and 5). Now the *Mapping* service can process the data. Since services are responsible for data transfer in that way, huge amounts of data in scientific workflows do not need to be transported through the engine any longer.

As lots of scientific services the *Mapping* service uses an external database for the storage of results. In that case the RRS is deployed near to the data storage system. The *Mapping* service sends his results to the RRS of the favored database which in turn sends the data reference to the *Mapping* service (6 and 7).

Further execution of the visualization workflow in figure 5 invokes the *Rendering* service which uses different external RRS for data retrieval (10 and 11) and publication (12 and 13). The RRS that is used for data publication holds a file system and two data bases. For scientific workflows the choice of the appropriate storage system should be realized in the RRS depending on the data character and other criteria.

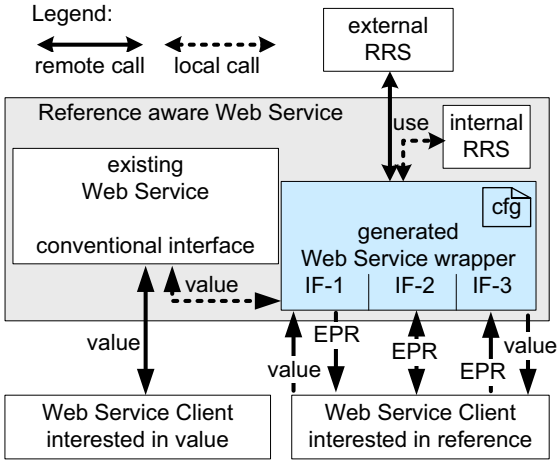


Figure 6. Realization of the WS wrapper for using RRS without changing the original WS.

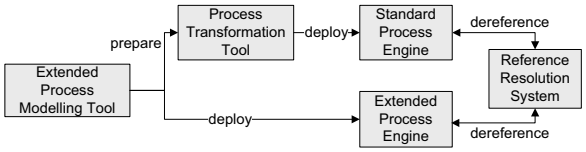


Figure 7. Alternative ways for realization of references in BPEL.

4. Realization Concepts

Two concepts for realizing the overall architecture are needed. The basis is the realization of reference-aware Web Services which is described in section 4.1. Here we show how to transform Web Services automatically to reference-aware Web Services. The aim of that is to allow clients freely to decide for each request whether they want to use value or reference passing. The second part is the realization in BPEL, which is presented in section 4.2. Here the aim is to show how reference-aware Web Services can be used transparently in BPEL via reference variables.

4.1. Realization of Reference-aware Web Services

Figure 6 shows how existing Web Services can be extended in order to get reference-aware ones; which means they can handle references in addition to values as input or output. The main idea is to generate a Web Service wrapper. Such a wrapper provides new interfaces that accept any combination of input and output of reference and value (Value in, EPR out; EPR in and out; EPR in, value out). Each EPR in the input or output of those wrapper interfaces is resolved using the specified RRS to a value. The resulting value is used to call the standard interface (value in, value out) of the wrapped service. The advantage is that value passing can be done locally and that the existing Web Service does not have to be modified at all.

The RRS can be installed locally on the same system or an external RRS can be used. This allows optimizing the overall system based on the data size and the surrounding conditions. Thus it is possible to ship the data to the functions that need the data prior to their execution.

4.2. Realization in BPEL Processes

We have identified two different approaches for implementing the reference variable extension for BPEL. We will discuss both of them in the following and highlight the advantages and shortcomings of each. As shown in figure 7 there are basically two different options for the realization, that do either employ model transformation (represented by the upper branch in the figure), or include an extension of the process engine (represented by the lower branch in the figure).

```
<referenceVariable name="refName" valueType="xsd:
    schema" referenceType="onInstantiation |
    fresh | periodic | external" period="
    duration"? external="partnerLink"?  />*
```

Listing 2. Code of a <referenceVariable>-schema.

Modeling References in BPEL: Both approaches have something in common, which is to extend the tool for modeling the processes, such as [10], in order to account for modeling in a reference-aware manner. The significant difference between the approaches is an additional model transformation step on the one hand, and an extension of the process engine on the other. The modeling of <referenceVariable>s requires introducing a new variable type. Therefore, we propose the schema shown in listing 2. This extension and its semantics are valid for both approaches and are thus discussed first.

The schema is an extension of the standard BPEL variable schema. The attribute *valueType* reflects the data type of the variable that is referenced. The reference variable itself implicitly has the type xsd:EPR in order to store the actual reference. Different optimization possibilities can be implemented based on the pattern of actuality needed for the access to reference values. This access type can be specified in the attribute referenceType. In this paper we propose four of them, many more are conceivable though. The different types are discussed when inspecting the transformation approach in detail.

As described in the first use case in section 1.1 it is often practical to use references to data within a workflow system. This allows flexibility and immediate reaction to change, this could either be required for all running processes or just for those that start at a certain point in time. These options represent the information, when and in which manner a reference should be resolved. A required dereferenciation is indicated in the process model, when a reference variable is accessed, for example within an <assign> activity. Such access is modeled with a special function call, for example val($referenceName) in order to dereference the value of the reference, but also allowing to access the reference, i.e., the EPR, itself.

```
<variable name="refName" type="xsd:schema"/>
<variable name="refNameEPR" type="xsd:EPR"/>
```

Listing 3. Code of generated variable declaration.

Engine Extension Approach: One opportunity for realizing our presented approach is shown in the lower branch in figure 7. Available open source BPEL engines such as [4] provide us with the possibility to extend the execution behavior. Reference handling can directly be implemented in the execution engine in order to create a high-performance solution. However, in doing so we would sacrifice standard-conformance and lose compatibility with other engine implementations. We have therefore decided to postpone this possibility of realization and to concentrate on the other, transformation-based approach that is discussed in the following.

```
<invoke name="refNameRefresh_1" partnerLink="RRS"
     operation="GET" inputVariable="refNameEPR"
    outputVariable="refName"/>
```

Listing 4. Code of dereferenciation activity.

Model Transformation Approach: Employing a model transformation step before the deployment of the process model allows us to implement reference handling virtually in processes *without requiring a modification of the execution engine*. The basic idea is to extend the BPEL language [3] only in the modeling tool, where we distinguish between common variables and references. The additional transformation step is generating standard-conform BPEL constructs for reference handling and injecting them into the original process model. For each reference that is declared in a process model, two variable declarations are generated as shown in listing 3.

The first declaration is used for holding the actual value, the second to store the pointer to it, which is represented by an EPR. Furthermore the RRS needs to be visible in the process model, which is achieved by generation of an according <partnerLink>. Finally listing 4 shows the way to actually dereference a pointer using a service invocation that calls the RRS which is the main part of the dereferenciation activity.

The transformation approach injects variable declaration and the dereferenciation activities into the workflow, in order to perform the refreshment of the value of the reference. As mentioned above the attribute *referenceType* allows the modeler to choose between the different options for refreshment of the values. Depending on the modeler's choice for this attribute, the dereferenciation activities need to be injected into the process at the appropriate positions as follows:

- *onInstantiation (default)*: At the instantiation of the process the value is retrieved from the RRS and the variable is set accordingly. This setting is

useful for defining business constants, that are set on process start and should be valid for the whole lifetime of the process instance. For each reference where onInstantiation is set, the RRS invocation activity shown in listing 4 is placed in a sequence that is executed on process instantiation.

- *fresh*: As fresh as possible - the value is retrieved each time the variable is being accessed. This setting is feasible for accessing frequently changing values that are external to the process, e.g., sensor data. In this setting the dereferenciation activity needs to be placed directly before each activity that accesses the value of the reference.

- *periodic*: A time value (e.g., 10 min) can be set in the attribute `period`. This attribute describes the maximum age a temporary reference value may have, that is stored locally. After this time the value is retrieved from the RRS and the temporary variable is refreshed. Therefore, an `<onAlarm>` element is placed in the global `<eventHandlers>` element during transformation. This construction accounts for periodic refresh, by repeatedly retrieving the value from the RRS.

- *external*: An external event in the field of Web Service orchestration is typically a message that is sent to the process instance (or a signal that is valid for all instances of one process model). By setting this value for the `referenceType`, an update can be triggered from the outside by sending an according message to the process engine. The service from which such a message is expected can be specified in the attribute `external`. The transformation step places an `<onEvent>` construct in the global `<eventHandlers>` element, to account for this setting.

This access type information for the refreshment of the values is needed in the workflow model with the following reason: In a SOA services should be loosely coupled. That means services should not be aware of the access type information which is relevant process logic information. Therefore, the access type information cannot be placed in the RRS service. Because if this would be done the RRS service ought to be aware of all access types of all workflows that are deployed. This would result in a tight coupling.

The main characteristic of this realization approach is that a modification of the execution engine is not required. This circumstance allows porting the presented functionality to basically any standard-conform execution engine and compatibility issues between the various existing engines are avoided. However a shortcoming of this realization is that the transformation step introduces new activities and variable definitions, compared to the original process model. This impacts monitoring and debugging instruments that will register the execution of activities that are not contained in the original process model.

5. Conclusion and Future Work

In this paper we introduced a method to allow data passing between Web Services based on reference passing instead of value passing. This forms the basis for implementing a new reference variable type in BPEL which has the following advantages for the workflow:

First the workflow can pass the results from one Web Service to another by using these pointers stored as EPR in a reference variable. Thus the workflow engine is disburdened of handling all data passing between the orchestrated Web Services which helps to reduce network traffic and processing time.

Second the new reference variables allow completely new data handling possibilities. Normally the variables in a workflow are set explicitly, in contrast the value of a reference variable is dereferenced on the time of usage (or how specified see section 4.2). That means a reference variable always provides the actual valid value without explicitly updating it in the control flow. This is very useful for context-aware workflows because in this area data is often sensed by sensors and is updated very often. Without using reference variables it is not easy to keep the values in the workflow always up-to-date. Furthermore the update activities pollute the workflow model.

Future work is to research whether it is possible and useful to integrate business rules into the RRS, as rule engines usually need complex input data in order to evaluate the rule. Another subject of future research is if visibility and scoping concepts are needed for references. For example a reference could be visible on different layers: workflow instance level, workflow model level or a company wide level (e.g., motivated by privacy reasons). Furthermore security is an important issue, until now it is handled very restrictive, only reading is allowed for non-administrators in order to avoid conflicts in changing the values of references. This has to be refined whenever using our approach for scientific workflows.

Additionally, the RRS can realize more functionality like a dynamic resource choice in order to facilitate the usage of workflow technology in the scientific domain. At last the RRS can be part of a grid middleware since scientific workflows often are executed in a grid environment.

Another problem that has to be researched in future

work is the garbage collection of the references. There is no garbage collector available yet but it would be important to remove unused references in order to save storage space. Furthermore the aliasing problem that always occurs when using pointers has to be thought of, because it is more complex than in normal programming languages for following reasons: First, the usage of references is distributed between different workflows and RRS installations, not only direct references but also indirect ones are possible. Second, the data accessed via references have no simple data types but are complex big data sets different references can point on. Because of this reason the analysis of the references is difficult at runtime and it is an unsolved problem how to detect aliasing between different references.

Acknowledgments The work published in this article was partially funded by the DFG project Nexus (SFB627), the COMPAS project[1] under the EU 7th Framework Programme Information and Communication Technologies Objective (FP7-215175) and the DFG Cluster of Excellence Simulation Technology (EXC310).

References

[1] External Variables in Apache Ode. `http://ode.apache.org/external-variables.html`.

[2] A. Akram, D. Meredith, and R. Allan. Evaluation of BPEL to Scientific Workflows. In *CCGRID*, pages 269–274. IEEE Computer Society, 2006.

[3] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, C. K. Liu, R. Khalaf, D. Knig, 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 Project. Apache Orchestration Director Engine (ODE). `http://ode.apache.org`.

[5] T. Berners-Lee, R. T. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. Internet RFC 3986, 2005.

[6] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Characterization of Scientific Workflows. *3rd Workshop on Workflows in Support of Large-Scale Science (WORKS08)*, 2008.

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

[8] F. Daniel, F. Casati, V. D'Andrea, S. Strauch, D. Schumm, F. Leymann, E. Mulo, U. Zdun, S. Dustdar, S. Sebahi, F. de Marchi, and M.-S. Hacid. Business Compliance Governance in Service-Oriented Architectures. In *Proc. of AINA'09*. IEEE Press, 2009.

[9] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. Pegasus: Mapping Scientific Workflows onto the Grid. *Across Grids Conference*, 2004.

[10] Eclipse BPEL Project. Eclipse BPEL Designer. `http://www.eclipse.org/bpel/`.

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

[12] M. Großmann, M. Bauer, N. Hönle, U.-P. Käppeler, D. Nicklas, and T. Schwarz. Efficiently Managing Context Information for Large-Scale Scenarios. In *Proc. of the Third IEEE Intl. Conf. on Pervasive Computing and Communications*, 2005.

[13] D. Habich, S. Richly, S. Preissler, M. Grasselt, W. Lehner, and A. Maier. BPEL-DT - Data-aware Extension of BPEL to Support Data-Intensive Service Applications. In C. Pautasso and T. Gschwind, editors, *WEWST*, volume 313 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[14] IBM. Information Integration for BPEL on WebSphere Process Server. `http://www.alphaworks.ibm.com/tech/ii4bpel`, 2005.

[15] F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000.

[16] P. Sarbanes and M. Oxley. Sarbanes- Oxley Act of 2002 (SOX), 2002.

[17] P. J. Plauger and J. Brodie. *ANSI and Iso Standard C Programmer's Reference*. Microsoft Programming Series, 1992.

[18] F. Rosenberg and S. Dustdar. Business Rules Integration in BPEL A Service-Oriented Apporach, In: Proceedings of the 7th International IEEE Conference on E-Commerce Technology (CEC05). 2005.

[19] M. Vrhovnik, H. Schwarz, S. Radeschuetz, and B. Mitschang. An Overview of SQL Support in Workflow Products. In *Proc. of the 24th International Conference on Data Engineering (ICDE 2008), Cancún, México, April 7-12, 2008*, pages 1–8. IEEE, April 2008.

[20] W3C. Web Services Addressing 1.0 - Core, W3C Recommendation. `http://www.w3.org/TR/ws-addr-core/`, 2006.

[21] M. Wieland, O. Kopp, D. Nicklas, and F. Leymann. Towards Context-Aware Workflows. In B. Pernici and J. A. Gulla, editors, *CAiSE07 Proc. of the Workshops and Doctoral Consortium Vol.2*. Tapir Acasemic Press, 2007.

[All links were last followed on March 5, 2009.]

[1] `http://www.compas-ict.eu/`