WSDL 2.0 Message Exchange Patterns: Limitations and Opportunities[*]

Jörg Nitzsche, Tammo van Lessen, and Frank Leymann
Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
{joerg.nitzsche | tammo.van.lessen | frank.leymann}@iaas.uni-stuttgart.de
http://www.iaas.uni-stuttgart.de/

Abstract

The Web Service Description Language (WSDL) provides means to describe functional aspects of a service in a Service Oriented Architecture (SOA) based on Web Service technology. In contrast to its predecessor (WSDL 1.1), WSDL 2.0 does not define a fixed set of operation types but provides for a generic mechanism to define an operation by means of message exchange patterns (MEPs). In this paper we compare the expressivity of MEPs in general with other work and formalisms in the field of service interaction. Furthermore, we identify new MEPs and extend the template used to define MEPs to allow expressing more complex patterns. We give a refined definition of MEPs based on a detailed discussion and discuss how WSDL and the MEPs in particular can be combined with the choreography approach.

1. Introduction

The Web Service Description Language (WSDL) [6, 17] is part of the WS-* standard stack which comprises a set of composable specifications and standards used to define Web Services (WSs) [21], the most popular implementation of a service-oriented architecture (SOA) [19]. SOA is an architectural style where software components that provide a piece of functionality are encapsulated as agnostic services. These services communicate with each other via messages they exchange. There are several approaches to describe the message exchange among services. Choreographies [8] describe the communication of services (i.e. different participants) from a global point of view, i.e. a choreography description captures the overall interaction including the complete message exchange of every single participant of the choreography. Orchestrations [8] however, describe the message exchange of a single participant only. WSDL describes the functional aspects of a service and is also focused on a single participant. It defines the messages a service is able to send and receive and how these messages are related by grouping them into operations. Compared to orchestrations and choreographies, the relation between messages in WSDL is not described by specifying their potential ordering using a formal specification, but their grouping and classification as input, output or fault. In WSDL 1.1 [6] there are four different operation types. The operation types could be mixed up inadvertently with procedure calls, having a return value or not. Recently, WSDL 2.0 [17] has been standardized. It provides for a generic mechanism to describe operation types by means of message exchange patterns (MEPs). Thus, it removes the pretended dependency of service descriptions to imperative programming and fosters a message orientated way of thinking which is postulated by SOA instead of thinking in terms of (remote) procedure calls. In this paper we examine the existing predefined MEPs given in the adjuncts to the WSDL 2.0 specification and the W3C Note "Additional MEPs" [3] and discuss the expressivity of the W3C proposed approach to describe MEPs. We extend this approach to enable expressing more complex MEPs and discuss how WSDL and the MEPs in particular can be combined with the choreography approach. Based on this discussion we give a refined definition of MEPs.

The remainder of the paper is structured as follows. Related work in the field of service interaction is presented in section 2. Section 3 presents WSDL and explains MEPs in detail. A comprehensive sample scenario is given in section 4. Section 5 identifies two new MEPs in the sample scenario and shows how the W3C proposed approach can be extended to express these patterns. A detailed discussion about the needed expressivity of WSDL 2.0 MEPs in section 6 is followed by a definition

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

of MEPs in section 7. Section 8 concludes the paper and gives directions for future work.

2. Related Work in Service Interaction

A choreography language enables capturing the interaction between services from a global point of view. Such a language typically provides means to describe either interaction models or interconnection models [8]. Interaction models describe the order of the messages that are exchanged directly. Corresponding specifications facilitating interaction models are e.g. the Web Services Choreography Definition Language (WS-CDL) [14] and Let's Dance [10]. Interconnection models do not order the messages directly but connect orchestrations with each other. Thus, they provide indirect ordering of the messages via the control flow of the connected orchestrations. Languages that provide support for modeling interconnection models are BPMN [16] and the Business Process Execution Language for Choreographies (BPEL4Chor) [15]. WS-CDL and BPEL4Chor are tailor-made for WS-* and WSDL 1.1 in particular, BPMN and Let's Dance are high level description languages independent of any technology.

Orchestration languages describe the interaction of services from the point of view of a single participant similar to WSDL. In contrast to choreography languages that are only used for *modeling* interactions, orchestration languages like the Business Process Execution Language (BPEL) [1] can be used to create *executable* models that can be enacted in an orchestration engine. During the enactment of a process model the orchestration engine then actually realizes the modeled interaction.

The SOAP Service Description Language (SSDL) [20] describes a message exchange from the point of view of a single service (like WSDL). SSDL abstracts from the notion of an operation and only describes message exchanges. It provides several plug-ins that define the message exchange either based on the WSDL MEPs, rules, or using process algebra. It implies the use of SOAP [13] and WS-Addressing [5].

In [2] a collection of patterns of service interaction is presented which allows benchmarking expressibility of modeling languages for service interaction, such as choreography and orchestration languages. The patterns reach from simple interactions like sending a message or receiving a message to complex interactions that involve multiple messages and different parties.

```
This pattern consists of [number] message[s, in order] as follows:

[enumeration, specifying, for each message]
A[n optional] message:
    1. indicated by an Interface Message Reference component whose
       message label is "[label]" and direction is "[direction]"
    2. [received from|sent to] ["some" if first mention] node
       [node identifier]

This pattern uses the rule [fault ruleset reference].

An Interface Operation using this message exchange pattern has a
message exchange pattern property with the value "[pattern IRI]".
```

Listing 1. WSDL 2.0 MEP template [18]

3. WSDL Operation Types / Message Exchange Patterns

In a WS world the Web Service Description Language is used to describe the functional aspects of a service, i.e. the messages the service is able to send and receive. This message exchange is described from the service's point of view.

WSDL 1.1 has been published as a note by the World Wide Web consortium (W3C)[1] in 2001. It enables describing both the messages themselves and the semantics of the messages, i.e. how the messages are related. The relation is defined by grouping the messages into operations which are again grouped into so called port types. WSDL 1.1 defines a fixed set of operation types: (i) *request-response* (the service first receives a request and then sends a response or a fault), (ii) *one-way* (the service only receives a message), (iii) *solicit-response* (the service sends a request and receives a response or a fault) and (iv) *notification* (the service sends a message). The reason why WSDL 1.1 specifies exactly these four operation types is that Web Service technology has been developed for the purpose of Enterprise Application Integration (EAI) [9]. Basic scenarios in application integration are (i) receiving a message triggering a response message (WSDL operation type request-response or solicit-response, respectively) and (ii) receiving a message not triggering a response (WSDL operation type one-way or notification, respectively). For these scenarios the WSDL operation types do not specify whether they are blocking or non-blocking. However, a common misperception is that the MEPs request-response and solicit-response are always blocking, i.e. that they are different incarnations of a remote procedure call (RPC). But, an RPC is just a special case of the more general MEP request-response/solicit-response. Since the WSDL specification is terse, two of the operation types were interpreted differently by some vendors: solicit-response and notification. Notification for instance was implemented point-to-point by one group of vendors and

[1]http://www.w3.org/

one-to-many by another group of vendors. Thus, the Basic Profile [12] of the WS-Interoperability Organization[2] defines that these operation types must not be used. Consequently, only two operations types from WSDL 1.1 are used in practice, one where a message is received only and another where receiving a message triggers a response.

WSDL 2.0 introduces a generic mechanism to describe operation types, called message exchange patterns (MEP). A MEP defines the operation type of a WSDL operation by describing the order in which messages that belong to the same operation are exchanged following a predefined template (see Listing 1). In the template, the bracketed items indicate a replacement operation. The *received from* and *sent to* are always from the point of view of the service, and participating nodes other than the service are implicitly identified as the originators of or destinations for messages in the exchange. A MEP in an operation is identified via the attribute *pattern* (see Listing 2). In contrast to WSDL 1.1, an operation can have multiple inputs, multiple outputs, multiple incoming faults and multiple outgoing faults that define the data types used during the message exchange. In principle a MEP can define an arbitrary message exchange of a service with (a) partner service(s) which fosters the change from a procedural way of thinking to a message oriented way. Thus, MEPs are an important step towards a loosely coupled SOA. Eight MEPs are defined by official W3C documents [3, 17, 18].

- In-Only – The service receives a message.
- Robust In-Only – The service receives a message and in case of a fault it returns a fault message
- In-Out – The service receives a message and returns a response message
- In-Optional-Out – The service receives a message and optionally returns a response message
- Out-Only – The service sends a message
- Robust-Out-Only – The service sends a message and in case of a fault at the partner service it receives a fault message
- Out-In – The service sends a message and receives a response message
- Out-Optional-In – The service sends a message and optionally receives a response message

All of these patterns describe a bilateral message exchange between the service and a partner service from the service's point of view. The descriptions of the patterns identify the partner service using a node identifier (see Listing 1). This way the ambiguity of operation types is resolved. The MEP out-only for instance defines a point-to-point interaction and thus provides the specification for exactly one of the interpretations of the

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

```
<operation name="xs:NCName"
           pattern="xs:anyURI"?
           style="list of xs:anyURI"? >
  <documentation />*
  [<input/>|<output/>|<infault/>|<outfault/>]*
</operation>
```

Listing 2. Definition schema for WSDL 2.0 operations

WSDL 1.1 operation type notification.

Identifying a partner service explicitly, enables defining a MEP that involves different partners, i.e. different nodes. But it is not defined what it means when several different partner nodes are involved in a message exchange. These nodes could be several implementations of the same type of service, i.e. different instances of the same node type or they could be nodes of different types. Additionally, the template is not precise enough to actually define a contract between two or more parties: patterns with optional receiving messages for instance are underspecified because it is not defined how services that implement the MEP behave. They could for instance wait for a certain period until the message or the fault arrives. However, the exact behavior is not specified.

4. Sample Scenario

The scenario presented in Figure 1 involves several node types as well as several instances of one node type. Different node types are (i) the requester which is also the initiator of the overall communication, (ii) a provider and (iii) a banking service. There exist several instances of the provider node type.

In the first step the requester carries out a request

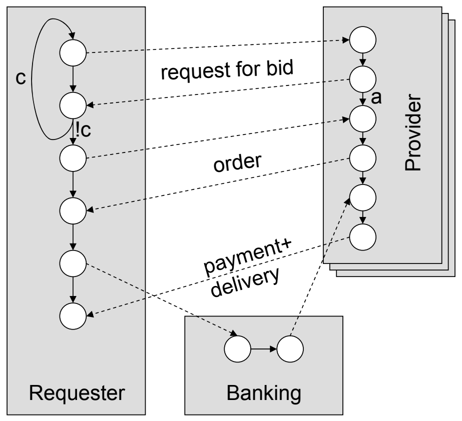

Figure 1. Sample scenario.

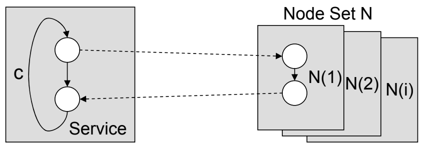

Figure 2. Scenario “request-for-bid”.

for bid. Therefore he sends out a message to several providers and waits for a certain time to receive response messages from the providers. Then he selects one of the providers and orders a product. He sends the product details to the provider which in turn sends a request for payment to the requester. Then the requester sends a message with the credit transfer details to a banking service. The banking service notifies the provider that the amount has been paid and the provider delivers the goods and sends a message to acknowledge that he has received the money.

5. Realisation

These kinds of complex scenarios involving several parties are typically modeled using the choreography approach. The state-of-the-art for execution is that all the information about the partner services is pushed to the participants, i.e. they know where to send a message and from which node to receive a message which is also expressed in their interfaces.

5.1. State-of-the-art (WSDL 1.1)

In WSDL 1.1 the expressivity of an operation is limited. The communication mode of an operation is always bilateral, i.e. there are always two nodes involved in the message exchange: the service itself and a partner node. Therefore, when capturing the knowledge that a service communicates with several nodes, a higher level language such as BPEL is needed. BPEL describes a flow between WSDL 1.1 operations. For communication purposes it defines so called partner links, one partner link for every node type. Information about different instances is hosted by the process instance via endpoint references stored in variables.

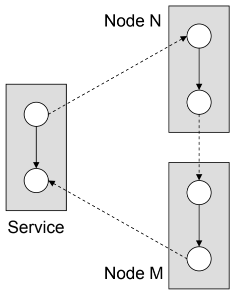

Figure 3. Scenario “request-with-referral”.

5.2. Using WSDL 2.0 MEPs

WSDL 2.0 enables distinguishing between different nodes. Thus, WSDL 2.0’s expressivity allows describing the service’s part of the choreography at the operation level. Taking the sample scenario above, we can find three different MEPs from the point of view of the requester of which the first and the third are so far not defined for WSDL 2.0: (i) a request-for-bid that is composed of a one-to-many-send [2] and a one-from-many-receive [2], which involves multiple instances of the provider role/type (see Figure 2); (ii) a send/receive [2] that involves one instance of the provider role/type; and (iii) a request-with-referral [2] that involves the banking service and the provider (see Figure 3).

The description of the MEP request-for-bid is pre-

This pattern consists of multiple messages, in order, as follows:

For each node i of a set of nodes N

A message:
– indicated by a Interface Message Reference component
  whose {message label} is *"Out"* and {direction} is *"out"*
– sent to node N(i)

An optional message:
– indicated by a Interface Message Reference component
  whose {message label} is *"In"* and {direction} is *"in"*
– received from node N(i)

This pattern uses the rule 2.2.1 Fault Replaces Message. An operation using this MEP has a {MEP} property with the value *"http://www.iaas.uni–stuttgart.de/2007/10/wsdl/rfb"*.

Listing 3. MEP "request-for-bid"

This pattern consists of exactly two messages, in order, as follows:

A message:
– indicated by a Interface Message Reference component
  whose {message label} is *"Out"* and {direction} is *"out"*
– sent to some node N

A message:
– indicated by a Interface Message Reference component
  whose {message label} is *"In"* and {direction} is *"in"*
– received from some node M where M <> N

This pattern uses the rule 2.2.1 Fault Replaces Message. An operation using this MEP has a {MEP} property with the value *"http://www.iaas.uni–stuttgart.de/2007/10/wsdl/rwr"*.

Listing 4. MEP "request-with-referral"

sented in Listing 3. To enable distinguishing between different instances of a node type and different node types we have to extend the pattern description template with the notion of *a set of nodes* (indicating that all nodes within this set are of the same type), an *index* to identify a node within the set of nodes, and a *for each* statement to enable iterating the set. Listing 4 shows the description of the MEP request-with-referral. It can be observed, that the WSDL MEP does not describe the whole request-with-referral scenario, but only those messages received or sent by the requester. It remains undefined how the two different nodes (N and M) communicate with each other, i.e. whether they communicate directly (like in Figure 3) or via other nodes.

6. Expressivity of MEPS

As shown, the MEPs in WSDL can in principle express the communication with different node types and multiple instances of one node type. However, MEPs are not expressive enough to define the overall interaction of a choreography like the one presented in Figure 1. It can only describe the part of the interaction that is visible from the point of view of a single service.

In the preceding section we presented how a choreography description can be implemented in a traditional way. The knowledge about the sequence and destination (end point references (EPRs) of partner services) of messages is pushed to the participants. This is also reflected in their WSDL descriptions, at least on an abstract level, by describing the sequence of messages and whether they have the same or different destinations.

However, there is a second option how the choreography may be realized. The concrete information about the participating services, i.e. their EPRs is hosted by a middleware component, the service bus [4, 7] that is aware of the whole choreography. During execution, services hand over messages to the service bus which in turn delivers the messages to the recipients as defined by the choreography. Still, the WSDL descriptions of the services may define the sequence of messages the service will send and receive, and whether messages have to be exchanged with one single node type or multiple node types; this indicates in which choreography the services can be involved in. However, this information is not needed during execution because the whole message exchange is managed by the service bus.

When using such a choreography-aware service bus it is also possible, that a participating service does not care if only a single partner type (i.e. node type) is involved into its message exchange or if a third party or even more parties are included. In case a service does not care how many different parties (i.e. different node types) are involved in its message exchange, the service would not define any nodes in its MEP. As a consequence, the service could be used to implement any role in a choreography where the given sequence of messages is needed, no matter where these messages are actually sent and from where the messages are received. This kind of service that is completely agnostic about its partner services nicely fits into the paradigm of a loosely coupled system. Note that this is only possible when using a bus that is choreography-aware. Such a service bus hosts the choreography definition used to determine the actual partner to which a message must be delivered.

7. Definition of MEPs

There are two distinct ways to use MEPs when describing the interaction of a service when using WSDL. The notion of an operation and of MEPs in particular can be reduced to one message, i.e. sending or receiving a message, which corresponds to the WSDL 1.1 operation types one-way and notification. The relations of these single messages can be defined using languages like BPEL or BPEL$^{\text{light}}$ [11]. By using WSDL 2.0's MEPs it is also possible to describe the complete interaction of a service including the message exchange with all partner services within one single operation. In this case, an operation may consist of thousands of messages and dozens of partners. However, in both cases no added value is achieved by the notion of an operation, because no reusable unit of a higher complexity is defined:

- Single message operations can be *reused* but *do not* define a unit of *higher complexity*; single messages exist without the notion of an operation.
- Message exchanges that specify the complete interaction of a service define units of *higher complexity*, but they are *too complex* to be *reusable*.

To achieve added value via the notion of an operation it should be both: a unit of higher complexity that deals with a certain kind of problem and it should also be reusable. As a consequence, a service can be involved in scenarios reaching from a very basic interaction between two parties to complex scenarios where multiple node types and multiple instances of a node are included.

Following these principles we have identified the MEPs request-for-bid, solicit-response, and request-with-referral in the example given in Figure 1 and not the MEP "place a request for bid, order something from a chosen provider and include a third party during payment". In the last section we showed that it is necessary to identify the partner nodes, different node types and different instances of a node type in a MEP, when implementing WS in a traditional way, i.e. without extensive middleware support. When implementing a scenario using a choreography-aware service bus it is possible to define the MEPs without defining different node types. Constraining the partner types of the MEP to a special choreography hampers reuse of services. However, this is only possible if the service itself does not have constraints on the partners he interacts with.

As a result, we come to the following definition: "A message exchange *pattern* (MEP) specifies in a *reusable* manner the ability of a service to receive and/or send messages. It describes the set of exchanged messages in terms of their order and multiplicity, i.e. whether a message is send to or received from a single node or whether a message is sent to or received from multiple instances of a node. Optionally, also different node types can be identified."

8. Conclusion

In this paper we discussed the nature of WSDL 2.0 MEPs in depth. We discussed the expressivity of the template given in the specification for describing MEPs and extended it to facilitate distinguishing different node types and instances of node types. We identified MEPs missing in WSDL 2.0 and presented a WSDL 2.0 style template description for request-for-bid and request-with-referral.

We compared the expressivity of MEPs in general with other work and formalisms in the field of service interaction. In particular, we discussed how the MEPs in WSDL operations can be combined with the choreography approach and introduced the idea of a choreography providing routing information that is hosted by a middleware, the service bus. This choreography-aware service bus leads to services that are completely agnostic about their partner services and fits nicely into the paradigm of a loosely coupled system. Such a service bus is part of our future work. Based on a detailed description we gave a refined definition of MEPs suited for both conventional use (i.e. partner information pushed to the interfaces) and using a choreography aware service bus.

Even though we extended the template for defining MEPs, it is still not precise enough to actually define a contract between two or more parties. Patterns with optional messages are under-defined because it does not define how the service behaves, i.e. if there is a certain period the service waits for the message or fault. For that reason and the current template being not well suited for describing more complex patterns a more expressive formalism is needed to define MEPs precisely and unambiguously. This is part of our future work.

References

[1] A. Alves et al. Web Services Business Process Execution Language Version 2.0. Committee specification, OASIS Web Services Business Process Execution Language (WSBPEL) TC, 2007.

[2] A. Barros et al. Service Interaction Patterns. In *3rd International Conference on Business Process Management (BPM)*, 2005.

[3] A. Lewis. Web Services Description Language (WSDL) Version 2.0: Additional MEPs. *W3C Note*, 2007.

[4] D. A. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[5] D. Box et al. Web Services Addressing (WS-Addressing). *W3C Member Submission*, 2004.

[6] E. Christensen et al. Web Services Description Language (WSDL) 1.1. *W3C Note*, 2001.

[7] F. Leymann. The (Service) Bus: Services Penetrate Everyday Life. In *3rd International Conference on Service Oriented Computing (ICSOC)*, 2005.

[8] G. Decker et al. An Introduction to Service Choreographies. In *it special issue on Service-Oriented Architectures*. Oldenbourg Wissenschaftsverlag, 2008. (to appear).

[9] G. Hohpe et al. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman, 2003.

[10] J. M. Zaha et al. Let's Dance: A Language for Service Behavior Modeling. In *14th International Conference on Cooperative Information Systems (CooPiS)*, 2006.

[11] J. Nitzsche et al. BPEL$^{\text{light}}$. In *5th International Conference on Business Process Management (BPM)*, 2007.

[12] K. Ballinger et al. Basic Profile Version 1.2. 2007.

[13] M. Gudgin et al. SOAP Version 1.2 Part 1: Messaging Framework. *W3C Recommendation*, 2007.

[14] N. Kavantzas et al. Web Services Choreography Description Language Version 1.0. *W3C Candidate Recommendation*, 2005.

[15] O. Kopp et al. BPEL4Chor: Extending BPEL for Modeling Choreographies. In *6th International Conference on Web Services (ICWS)*, 2007.

[16] Object Management Group. Business Process Modeling Notation (BPMN) Version 1.1. 2008.

[17] R. Chinnici et al. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. *W3C Recommendation*, 2007.

[18] R. Chinnici et al. Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts. *W3C Recommendation*, 2007.

[19] S. Burbeck. The Tao of e-Business Services. *IBM Corporation*, 2000.

[20] S. Parastatidis et al. SOAP Service Description Language (SSDL), 2005.

[21] S. Weerawarana et al. *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.