Extending BPEL^{light} for Expressing Multi-Partner Message Exchange Patterns

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

Message exchange patterns provide means to define the message flow of a service and how these messages are related in an abstract and reusable manner. They are an integral part of WSDL 2.0 and allow defining operations that have a message exchange beyond request-response. They reduce the impedance mismatch between imperative programming and message orientation while emphasizing the message orientated nature of Web Services. Whereas BPEL defines a flow between Web Service operations, BPEL^{light} is an appropriate candidate to define the flow within operations since it abstracts from WSDL. In this paper we extended BPEL^{light} to facilitate capturing complex multi-lateral message exchanges. We refine the partner model and relax the definition of a conversation to enable modelling conversations that involve different partner types. We also extend the language with a first-class construct that enables storing, querying and thus distinguishing addressing information related to multiple partner instances. This way we enable modelling message exchange patterns that capture business logic in a reusable manner on an abstract level like for instance a request for bid scenario or a business transaction for purchase.

1. Introduction

Service-oriented architecture (SOA) [5] is the latest trend in integrating heterogeneous systems. It identifies agnostic, self-contained services that encapsulate high-level business concepts [13]. These business concepts often require long-running multi-step (conversational) interaction between partners. The Web Service technology (WS-*) [18] is the most prominent approach to implementing an SOA and has gained broad acceptance in industry and academia. Besides SOAP [10], WSDL 2.0 [7, 8] is the core specification of the WS-* standard stack. It provides means to express which messages a service is able to send and receive and their relation by grouping them into operations. The operations are typed using a URI that identifies a message exchange pattern (MEP) which is defined in natural language following a predefined template. However there are several disadvantages with respect to the W3C[1] proposed template: First, its expressivity is limited; MEPs involving several instances of a node for example can only be specified when the existing schema is extended [16]. Second, it is not precise enough; it is for instance not specified how a receiver of an optional message can find out whether the message will actually arrive or not. Third, the plain text representation is not machine-readable; thus Web service framework implementers have to interpret and implement each MEP separately.

The Business Process Execution Language (BPEL) [1] provides a comprehensive set of primitives to define a flow *between* Web Service operations. As it describes the message exchange from a services point of view like WSDL and since it has mind-share it is an eligible candidate for formalizing MEPs. In [17] BPEL^{light}, a BPEL dialect that abstracts from interface definitions, is used to specify the flow *within* operations. However, the presented MEPs are restricted to bi-lateral message exchanges and do not take multi-party interactions into account.

In this paper we extended this approach of defining MEPs to facilitate capturing complex multi-lateral message exchanges. We refine the partner model of BPEL^{light} and relax the definition of a conversation to enable modelling conversations that involve different partner types such as the requester's point of view of a *Request-with-Referral* [3, Pattern 11]. We also extend the language with a first-class construct that enables storing, querying and thus distinguishing addressing information related to multiple partner instances and thus allows for modelling e.g. the inititiators point of view of a *Request-for-Bid* [3, Pattern 7].

[1] http://w3.org

The remainder of the paper is structured as follows. WSDL and the MEPs in particular are presented in section 2. Section 3 gives an introduction into BPEL and BPEL^{light}. In section 4 it is shown how BPEL^{light} can be used to specify MEPs. The need for complex multi-partner MEPs is motivated in section 5. Extensions to BPEL^{light} that facilitate expressing multi-partner MEPs are presented in the following two sections: Section 6 deals with MEPs that involve multiple partner types and section 7 with multiple instances of a partner type. Section 8 concludes the paper and gives directions for future work.

2. WSDL Operation Types / Message Exchange Patterns

The Web Service Description Language is used in a Web Service world to describe a service in terms of messages the service is able to send and receive. It enables describing the *message*s themselves, how they are related, their *binding* to a transport protocol and the endpoint (*port*) where the *service* is waiting for incoming messages. The relation of messages is defined by grouping them into *operation*s. Operations are in turn grouped into so called *portType*s.

WSDL 1.1 provides a fixed set of four operation types:

1. *request-response* – the service first receives a request and then sends a response or a fault,
2. *one-way* – the service only receives a message
3. *solicit-response* – the service sends a request and receives a response or a fault
4. *notification* – the service sends a message.

The operation types correspond to the basic scenarios in enterprise application integration (EAI) [12] which are: (i) receiving a message triggers a response message (WSDL operation type request-response from the perspective of the service and the dual operation solicit-response from a requester's point of view) and (ii) receiving a message does not trigger a response (WSDL operation type one-way from the perspective of the service and the dual operation notification from a requester's point of view). However, as the WSDL specification is terse, the operation types solicit-response and notification were interpreted differently by some vendors. Notification for instance was implemented point-to-point by one group of vendors and one-to-many by another group of vendors. Therefore, the Basic Profile [2] of the Web Services-Interoperability (WS-I) Organization[2] defines that these operation types must not be used in order to achieve interoperability. Consequently, only two operation 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. As a consequence, in general it is not possible to describe the messages a service is able to send and receive only using the WSDL description of the service. Instead also WSDL descriptions of partner services have to be referred to like it is done in the WSDL extension partner link type. The result is a tighter coupling which contradicts the SOA paradigm of loose coupling.

```
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 [6]

To enable describing operations more precisely, WSDL 2.0, which became a W3C recommendation in 2007, introduces an extensible mechanism to identify operation types by means of message exchange patterns (MEP). This way all required types of operations can be defined, for instance one for sending a message point-to-point and one for one-to-many. A MEP defines the operation type of a WSDL operation by following a predefined template (see Listing 1) to describe the order in which messages that belong to the same operation are exchanged. 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 and faults used during the message exchange. In principle an MEP can define an arbitrary message exchange of a service with (a) partner service(s).

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

Eight MEPs are defined by official W3C documents [6, 7, 14]:

1. *In-Only* – The service receives a message.
2. *Robust In-Only* – The service receives a message and in case of a fault it returns a fault message
3. *In-Out* – The service receives a message and returns either a response message or a fault message
4. *In-Optional-Out* – The service receives a message and optionally returns a response message or a fault message
5. *Out-Only* – The service sends a message
6. *Robust-Out-Only* – The service sends a message and in case of a fault at the partner service it receives a fault message
7. *Out-In* – The service sends a message and receives either a response message or in case of a fault at the partner service it receives a fault message
8. *Out-Optional-In* – The service sends a message and optionally receives either a response message or in case of a fault at the partner service it receives a fault message

All of these patterns listed above describe a bilateral message exchange between the service and a partner service. The descriptions of the patterns identify the partner service using a node identifier (see Listing 1). This way the ambiguity of WSDL 1.1 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 operation type *notification*.

However, the template for describing MEPs proposed by the W3C is not precise enough to actually define a contract between the service and its partner service(s): patterns with optional receiving messages for instance are underspecified because it is not defined how services that implement the MEP should behave. They could for instance wait for a certain period until the message or the fault arrives. However, the exact behavior is not specified.

3. BPEL and BPEL^{light}

In the field of executable business processes the Business Process Execution Language (WS-BPEL or BPEL) is the de facto standard for modelling Web service compositions and has gained board acceptance in industry and research. BPEL is tightly coupled with

```
<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 [8] as its communication model is based on WSDL's concept of portTypes and operations.

To enable communication that is compliant to the Basic Profile [2] of the WS-Interoperability Organization, i.e. without using WSDL operations of type *notification* and *solicit-response*, BPEL introduces the concept of a partner link type which is defined as an extension to WSDL. A partner link type defines two roles in terms of port types and binds them together. The operations of type *notification* and *solicit-response* of a role are expressed as operations of type *one-way* and *request-response*, the other role has to provide. This definition of an abstract communication channel is used inside a BPEL process by a so called partner link. A partner link references a partner link type and defines which role is taken by the partner service and which role is taken by the process itself. Thus, it defines a concrete contract between a process and a partner service in terms of the operations each side has to provide.

According to the different operation types of WSDL provided by the two roles of the partner link type, BPEL defines several *basic interaction activities*. The `<receive>` activity implements either a one-way operation or, in conjunction with a `<reply>` activity, a request-response operation. The more complex `<pick>` activity may implement potentially multiple one-way operations or in combination with the `<reply>` activity multiple request-response operations (each incoming message represented by an `<onMessage>`-element). A `<pick>` activity may also specify a timer to define timing constraints like timeouts, e.g. to express how long the activity should wait for incoming messages. Like `<pick>` and `<receive>` activities, `<eventHandler>`s also implement one-way operations or, in combination with a corresponding `<reply>` activity, request-response operations. They facilitate receiving messages at any time as long as the scope they are defined for is active. The `<invoke>` activity is capable of invoking other Web Services by *using* operations of type one-way and request-response these partner services provide.

The composition of Web Services can be specified as a flow between BPEL's interaction activities, and thus as a flow between WSDL operations. For the definition of such flows, BPEL provides several so called *structured activities* that help modelling the control flow between *interaction activities* that are described in terms of Web Service operations. The control flow between activities can either be structured in a block-based manner by nesting structured activities like `<sequence>` (for sequential control flow), `<flow>` (for parallel control flow), `<if>` (for conditional branches in the control flow) and `<while>` (for loops) activities, or graph-based by defining conditional or unconditional `<links>` (i.e. directed edges) between activities within a `<flow>` activity; both styles can be used intermixed. Data flow is implicitly defined via globally shared variables. Data can be copied from one variable to another using the `<assign>` activity.

BPEL uses WSDL operations and port types to define activity implementations. However, the actual access and adressing information which is defined in the WSDL port is not included in the BPEL process model. This information has to be provided either during deployment of the process model or even during runtime of a particular process instance. In the former case both, the endpoint reference (EPR) of the process itself as well as the EPRs of the partner services are specified via so called deployment descriptors. In the latter case, incoming messages initialize the partner role of a partner link which is then associated with the EPR of the partner service that has sent the message. During runtime, EPRs are stored in the partner link by means of a service reference container (`service-ref`) and can be both read and written by `<assign>` activities. This way it is possible to iterate over a set of EPRs and invoke different instances of a certain partner.

In [15] BPEL^{light} has been presented. It is an extension of BPEL 2.0 and removes BPEL's inherent dependency on WSDL. This enables describing *both*, the flow between *and* within Web Service operations. Therefore BPEL^{light} is obviously an eligible candidate for specifying MEPs in a machine-understandable fashion.

For the sake of the independence of any interface definition language, BPEL^{light} introduces a new WSDL-less interaction model using BPEL 2.0's extension activity mechanism. Therefore it defines the following new constructs:

- the `<interactionActivity>`, that replaces all basic interaction activities (`<receive>`, `<reply>` and `<invoke>`),
- a WSDL-less `<pick>` activity,
- a WSDL-less `<eventHandler>`, and
- a `<conversation>` element that enables grouping several of the afore-mentioned interaction activities.

Employing these concepts in combination with BPEL's control flow primitives facilitates modelling complex interaction with partner services independent of WSDL.

4. Using BPEL^{light} for defining WSDL 2.0 MEPs

The expressivity of the natural-language based template for defining MEPs is limited to sequential ordering of messages and the template is not machine-readable. In [17] an approach is presented that eliminates these drawbacks by using BPEL^{light} to define MEPs.

In contrast to process models, MEPs do not define data types and are generically defined as they are aimed to be reusable [16]. This requires from BPEL^{light} the possibility to define abstract process models to enable defining only a flow of abstract messages which need to be concretely bound to specific message types when they are used to define a WSDL operation. Hence, a new *Abstract Process Profile for Message Exchange Patterns* was defined. It restricts the common base in the following manner[3] [17]:

- Omission shortcuts (i.e. omitted elements) MUST NOT be used in the MEP profile with one exception: Timing definitions, i.e. `<for>`, `<until>`, and `<repeatEvery>`, MAY be omitted in `<onAlarm>` and `<wait>` elements. In this case, deadlines and durations MUST be defined by a newly introduced timing expression element (see Listing 3). The type `repeatEvery` is only applicable to `<onAlarm>` elements. This new element is necessary to have the possibility to express timing constraints without pinpointing whether they are durations, deadlines or repetitions.

[3]The upper case keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [4]

```
<mep:timingExpression
        expressionLanguage="anyURI"?
        type="for|until|repeatEvery"
        expression="expr"/>
```

Listing 3. MEP extension: timing expression

- Explicit opaque tokens, i.e. opaque activity, opaque attributes, opaque expression, and opaque from-spec, MUST NOT occur in MEP models except for variable references, types and time constraints. These opaque tokens denote the points of variability which have to be substituted later in order to come to a concrete meaningful form of this MEP process model.

- To define generic MEPs data types MUST NOT be directly referenced by variables in MEP models. Instead, an opaque placeholder can be embedded which is to be replaced later. If message passing within the MEP process is not essential, `inputVariable` or `outputVariable` respectively can be marked opaque, the type is then automatically derived from the referencing WSDL messages. That way variable declarations can be omitted.

- Faults MUST be explicitly unmasked using BPEL^{light}'s `faultName` attribute in receiving activities (i.e. `bl:interactionActivity`, `bl:onMessage` within a `bl:pick`)

Using BPEL^{light}'s abstract profile for MEPs, an MEP can be defined as follows: Each MEP requires a seperate `<process>` definition. Within this definition an arbitrary flow between BPEL^{light}'s interaction activities (`bl:interactionActivity`, `bl:pick` and `bl:eventHandler`) can be defined using BPELs control flow primitives. Additionally a single conversation is defined to group these activities to an MEP.

Such a BPEL^{light} MEP can be used in WSDL 2.0 in the following manner. Referencing an MEP formalised in BPEL^{light} is analogue to referencing WSDL 2.0's textual MEPs. In BPEL the tuple `(targetNamespace, name)` serves as unique identifier of a process model which can be transformed to an IRI by concatenating the target namespace and the process name as an IRI with a fragment identifier. For instance the the IRI `http://.../mep-in-bpel#in-out` references the tuple `(http://.../mep-in-bpel, in-out)`. This IRI is used to identify the pattern by means of the `pattern` attribute in an operation.

To map input and infault, output and outfault messages to interaction activities in the MEP process model, the `messageLabel` attribute points to a receiving or sending activity respectively which holds the same message label in its `name` attribute.

For configuring timing expressions like for instance timeouts an extension element has been introduced which is to be placed next to message references in the `<operation>` element. To be as generic as possible, the new element `<mep:configure>` allows replacing opaque values of both attributes and elements by simply using XPath [9] to identify the appropriate node in the MEP model.

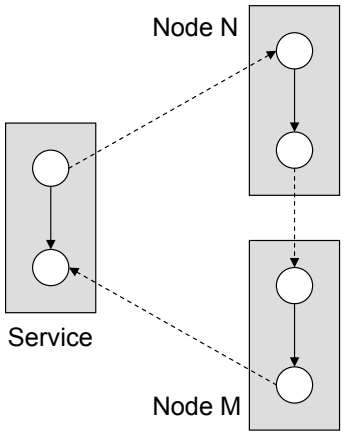

Figure 1. Scenario "Request-with-Referral".

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"

5. Multi-partner/Instance MEPs

The MEPs that are presented in the scope of the WSDL 2.0 specification and that have been formalized using BPEL^{light} in [17] only cover bi-lateral message exchanges between a service and another node. However, high level business concepts that are encapsulated by services often require conversational interaction with several partner nodes, i.e. these services take part in a complex choreography.

Modelling multi-partner MEPs is possible using the template because it enables identifying partner nodes explicitly. However, 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

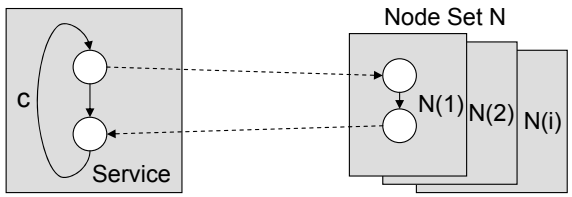

Figure 2. Scenario "Request-for-Bid".

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 5. MEP "Request-for-Bid"

of the same node type or they could be nodes of different types. In [16] extensions to the textual MEP template were introduced to provide an unambiguous interpretation.

Request-with-Referral [3] which is presented in figure 1 is a scenario that involves different node types. In this scenario a requester sends a request to a provider which delegates the request to another provider. The requester then receives the result from a provider that is different from the one the request was sent to. Listing 4 provides the template based description of the MEP *Request-with-Referral* that describes the requester's point of view. It can be observed, that the 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.

An example for a scenario that involves different instances of the same node type is a *Request-for-Bid* that is composed of a *one-to-many-send* [3, Pattern 5] and a *one-from-many-receive* [3, Pattern 6], which is presented in Figure 2. The template based description of the MEP that describes the scenario from the requester's point of view is presented in Listing 5. It can be observed that for enabling distinguishing between different instances of a node type the template has been extended. In particular, 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 were introduced.

6. Multipartner MEPs in BPEL^{light}

In [15] a `<conversation>` has been defined as a bi-lateral message exchange between the process defined in BPEL^{light} and a partner service. Multiple conversations can then be grouped using a `<partner>` element to express that multiple conversations have to take place with exactly one partner. In section 4 it has been shown how BPEL^{light} and in particular the `<conversation>` can be used to model bi-lateral MEPs. To enable expressing MEPs that involve multiple partner types, the notion of a partner has to be relaxed to be more fine granular. Instead of grouping whole conversations, single `<interactionActivity>`s, i.e. messages, can be assigned a partner using the `partner` attribute. Note that partner definitions are mutually exclusive.

Listing 6 shows how the *Request-with-Referral* pattern can be expressed using the BPEL^{light} abstract profile for MEPs. The MEP process distinguishes two partner nodes, the *contacted provider* (N) and the *responding provider* (M). Both partners are referenced in the corresponding interaction activities. This way the referral behavior is reflected.

7. Multi-instances MEPs in BPEL^{light}

When a multi-lateral message exchange interacts with multiple instances of a partner type like for instance in a *request-for-bid* scenario, the actual endpoint references (EPRs) of these instances need to be accessible in a convenient manner in the MEP definition. Therefore the data stored behind the `<partner>` element is extended to additionally store EPR information, i.e. a partner is not only grouping messages that are to be exchanged with a certain partner but is also maintaining references to actual partner instances. In fact, the partner element keeps both the EPR it is currently associated with and a set of EPRs belonging to partners it may interact with in future or has been interacted with already. This information can be used in XPath expressions for browsing, selecting and iterating EPRs and EPR sets.

Three usage scenarios can be distinguished that can be used intermixed:

- **Single Instance** – A partner is associated with exactly one EPR. This EPR is explicitly assigned to a partner using a deployment descriptor.
- **Multi Instance (static)** – A partner is associated with exactly one EPR, singled out of a set of EPRs.

```
<bpel:process
   xmlns:bpel="http://.../wsbpel/2.0/process/
       abstract"
   xmlns:bl="http://.../bpel-light"
   suppressJoinFailure="yes"
   abstractProcessProfile="http://.../bpel-light/
       abstract/mep/2008/"
   targetNamespace="http://../mep-in-bpel"
   name="request-wih-referral">
   <bl:conversations>
      <bl:conversation
         name="request-wih-referral"/>
   </bl:conversations>
   <bl:partners>
      <bl:partner name="contacted-provider"/>
      <bl:partner name="responding-provider"/>
   </bl:partners>
   <bpel:flow>
      <bpel:links>
         <bpel:link name="L1"/>
      </bpel:links>
      <bl:interactionActivity
         name="Out"
         inputVariable="##opaque"
         partner="contacted-provider"
         conversation="request-with-referral">
         <bpel:sources>
            <bpel:source linkName="L1"/>
         </bpel:sources>
      </bl:interactionActivity>
      <bl:pick>
         <bpel:targets>
            <bpel:target linkName="L1"/>
         </bpel:targets>
         <bl:onMessage
            name="In"
            outputVariable="##opaque"
            partner="responding-provider"
            conversation="request-with-referral">
            <bpel:empty/>
         </bl:onMessage>
         <bl:onMessage
            name="InFault"
            faultName="##opaque"
            outputVariable="##opaque"
            partner="responding-provider"
            conversation="request-with-referral">
            <bpel:empty/>
         </bl:onMessage>
      </bl:pick>
   </bpel:flow>
</bpel:process>
```

Listing 6. "Request-with-Referral" MEP in abstract BPEL^{light}

This set is statically configured by a deployment descriptor.

- **Multi Instance (dynamic)** – A partner is associated with exactly one EPR, singled out of a dynamic set of EPRs. This set is populated during runtime either manually, by service discovery mechanisms or by incoming messages from different yet unknown partner instances.

The latter requires that the EPR of the partner instance is part of the incoming message, e.g. via WS-Addressing [11] headers. An example for this scenario is an auction-style interaction where an unknown set of bidders (i.e. partner instances of a particular partner type) can send in quotes. The EPR of each participant is stored in the EPR set within the `<partner>` element.

In case of multi-instance scenarios, MEP definitions mostly need access to the EPR set to address partners directly. In the auction scenario for instance the participant with the highest bid shall be notified, therefore the bid needs to be linked with the partner instance (i.e. its EPR). When the notification is to be sent, the EPR associated with the highest bid is copied to the `<partner>` element, i.e. the partner is now pointing to the highest bidder. A subsequently sent notification will be routed to the winner's endpoint.

The EPR information stored within the partner can be accessed in the same manner as variables can be used in BPEL. Partner elements hold an XML infoset exemplarily shown in figure 3 (left) and are accessible using the $-notation in XPath expressions. The `<current>` element contains the EPR currently associated with the partner while the `<list>` element contains a list of known EPRs. Each EPR has been automatically assigned an id that allows identifying and iterating EPRs using XPath. The infoset for expressing EPRs (`sref:service-ref`) has been adopted from the WS-BPEL 2.0 specification and allows the specification of EPRs independently of any addressing mechanisms [1, Section 6.3].

To associate a partner with a particular EPR, a BPEL `<assign>` can be used to copy an EPR from the EPR set to the partner EPR as shown in figure 3 (right).

In Listing 7 the abstract BPEL^{light} specification of the *Request-for-Bid* MEP is presented. The partner *bidder* is declared in the preamble of the process. Its EPR set is populated via a deployment descriptor. Since all bids should be sent out in parallel, a *parallel forEach* is used which executes its children activities concurrently. The parallel forEach iterates the EPR set defined for the *bidder* and sends out a request-for-bid to each endpoint adress. However, using the *current* EPR of *bidder* would result in an undeterministic behavior because each par-

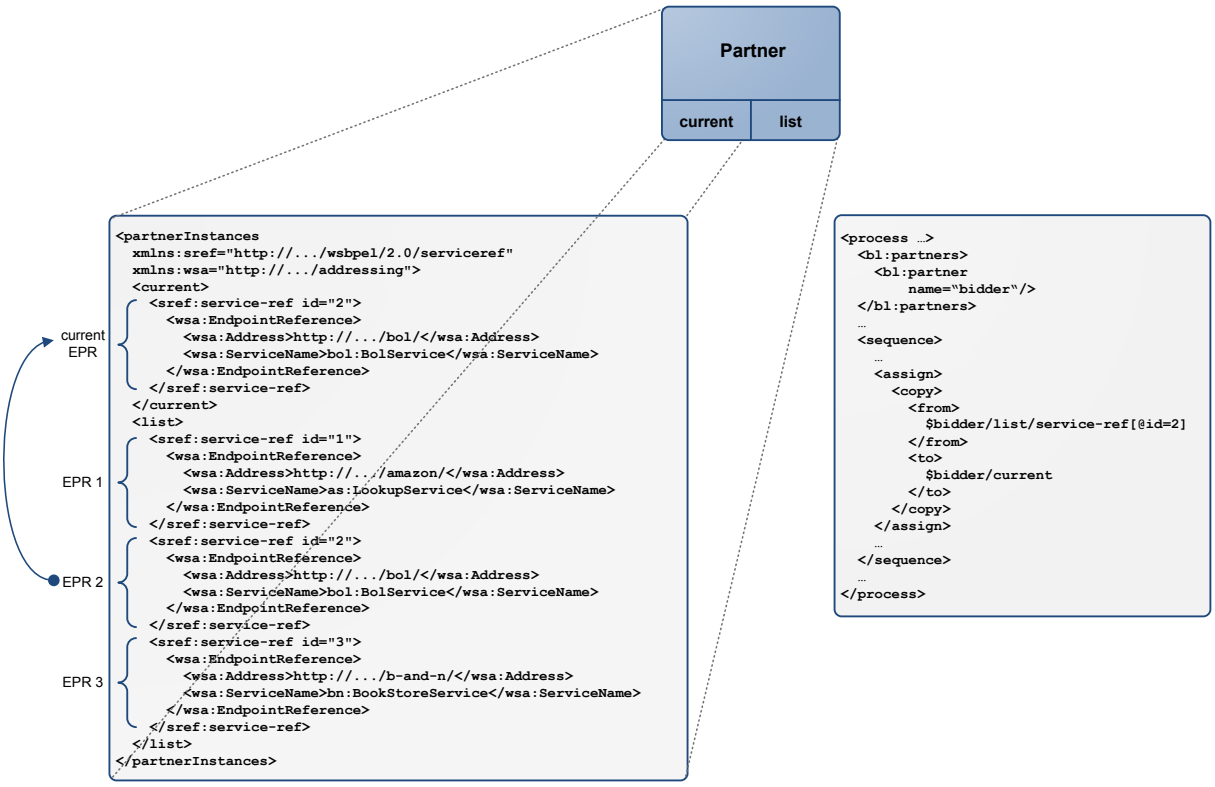

Figure 3. Copying partner EPRs

allel execution would override the EPR again and the MEP would not follow the intended behavior. Thus, a local copy of the partner declaration is defined (*isoB*). A particular partner instance is selected in each iteration, by copying an entry of the EPR set to the locally defined *isoP*. The entry that is copied is identified by the value of the `counter` variable. As an alternative to using local variables, also an isolated scope could be used to deal with these race conditions.

8. Conclusion

Message exchange patterns provide means to define the message flow of a service and how these messages are related in an abstract and reusable manner. They are an integral part of WSDL 2.0 and are one of the major improvements compared to its predecessor WSDL 1.1. They allow defining operations that have a message exchange beyond request-response and reduce the impedance mismatch between imperative programming and message orientation while emphasizing the message orientated nature of Web Services.

BPEL^{light} is an eligible candidate to define the message flow within operations since it abstracts from WSDL operations. In this paper we extended BPEL^{light} to facilitate capturing complex multi-lateral message exchanges. The partner model has been refined and allows to assign a partner to each single message. This is a significant improvement with respect to the expressivity of BPEL^{light} since conversations are no longer restricted to bi-lateral but may involve different partner types. In addition, BPEL^{light} has been enriched with a first-class construct that enables storing, querying and thus distinguishing multiple partner instances. Thus, the extensions made enable expressing multi-lateral message exchanges involving both, multiple partner types as well as multiple partner instances of the same type.

The definition of arbitrary complex MEPs, in particular multi-lateral MEPs in WSDL 2.0 has high impact on their binding to transport protocols. While in WSDL 1.1 the set of supported MEPs was fixed, the binding rules have been static as well. Currently, the same applies to the WSDL 2.0 standard as only the basic MEPs *In-Only*, *Robust In-Only* and *In-Out* are part of the standard. The rest of them are not (yet) standardized and are therefore not considered in binding specifications. The definition and discussion of protocol bindings for complex MEPs is part of our future work. Especially the binding of multi-partner/multi-instance MEPs is a challenge as there are manifold options to bind for in-

```
<bpel:process
   xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/abstract"
   xmlns:bl="http://iaas.uni-stuttgart.de/BPELlight"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   abstractProcessProfile="http://iaas.uni-stuttgart.de/BPELlight/abstract/mep/2008/"
   name="request-for-bid"
   suppressJoinFailure="yes"
   targetNamespace="http://www.iaas.uni-stuttgart.de/mep-in-bpel">
   <bpel:variables>
      <bpel:variable name="counter" type="xsd:long"/>
   </bpel:variables>
   <bl:partners>
      <bl:partner name="bidder"/>
   </bl:partners>
   <bl:conversations>
      <bl:conversation name="requestForBid"/>
   </bl:conversations>

   <bpel:forEach counterName="counter" parallel="yes">
      <bpel:startCounterValue>1</bpel:startCounterValue>
      <bpel:finalCounterValue>count($bidder/list/sref:service-ref)</bpel:finalCounterValue>
      <bpel:scope>
         <bl:partners>
           <bl:partner name="isoB"/>
         </bl:partners>
         <bpel:sequence>
            <bpel:assign>
               <bpel:copy>
                  <bpel:from>$bidder/list/sref:service-ref[$counter]</bpel:from>
                  <bpel:to>$isoB/current</bpel:to>
               </bpel:copy>
            </bpel:assign>
            <bl:interactionActivity name="sendRequest"
               inputVariable="##opaque"
               conversation="request-for-bid"
               partner="isoB">
            </bl:interactionActivity>
            <bl:pick>
               <bl:onMessage name="receiveBid"
                  outputVariable="##opaque"
                  conversation="request-for-bid"
                  partner="isoB">
               <bl:onMessage/>
               <bpel:onAlarm>
                  <mep:timingExpression name="timeout"
                     type="##opaque"
                     expression="##opaque"/>
                  <bpel:empty/>
               </bpel:onAlarm>
            </bl:pick>
         </bpel:sequence>
      </bpel:scope>
   </bpel:forEach>
</bpel:process>
```

Listing 7. “Request-for-Bid” MEP in abstract BPEL^{light}

stance multicasts to different transports. Another aspect of our future work is the identification of compatible (i.e. inverse) MEPs which is not necessarily a binary relation. Having identified the inverse MEP(s) for each MEP can ease service discovery and protocol mediation.

Achnowledments

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).

References

[1] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guizar, 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, January 2007.

[2] K. Ballinger, D. Ehnebuske, C. Ferris, M. Gudgin, C. Liu, M. Nottingham, and P. Yendluri. Basic Profile Version 1.1. *WS-I specification*, 8:1–1, 2004.

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

[4] S. O. Bradner. Key Words for Use in RFCs to Indicate Requirement Levels. Internet RFC 2119, March 1997.

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

[6] R. Chinnici, H. Haas, A. A. Lewis, J.-J. Moreau, D. Orchard, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts. *W3C Recommendation*, 2007.

[7] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. *W3C Recommendation*, 2007.

[8] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. *W3C Note*, 2001.

[9] J. Clark and S. J. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, November 1999.

[10] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. SOAP Version 1.2 Part 1: Messaging Framework. *W3C Recommendation*, 2007.

[11] M. Gudgin, M. Hadley, and T. Rogers. Web Services Addressing 1.0 — Core. W3C Recommendation, May 2006.

[12] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman, Amsterdam, October 2003.

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

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

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

[16] J. Nitzsche, T. van Lessen, and F. Leymann. WSDL 2.0 Message Exchange Patterns: Limitations and Opportunities. In *3rd International Conference on Internet and Web Applications and Services (ICIW)*, June 2008. Athens, Greece.

[17] T. van Lessen, J. Nitzsche, and F. Leymann. Formalising Message Exchange Patterns using BPEL^{light}. In *5th International Conference on Services Computing (SCC), To appear*, July 2008. Honululu, Hawaii, USA.

[18] 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.