Institute of Architecture of Application Systems

BPEL^{light}

Jörg Nitzsche, Tammo van Lessen, Dimka Karastoyanova, and Frank Leymann

Institute of Architecture of Application Systems
University of Stuttgart,
Universitätsstraße 38, 70569 Stuttgart, Germany
http://www.iaas.uni-stuttgart.de

in: 5th International Conference on Business Process Management (BPM 2007).
See also BibTeX entry below.

BibTeX:

```
@inproceedings{Nitzsche2007,
  author = {Joerg Nitzsche and Tammo van Lessen and Dimka Karastoyanova and Frank Leymann},
  title = {BPEL light},
  booktitle = {5th International Conference on Business Process Management (BPM 2007)},
  year = {2007},
  month = sep,
  publisher = {Springer},
}
```


document created on: 6th July 2007
created from file: bpellight.tex

Universität Stuttgart
Germany

BPEL^{light}

Jörg Nitzsche, Tammo van Lessen, Dimka Karastoyanova, and Frank Leymann

Institute of Architecture of Application Systems
University of Stuttgart
Universitaetsstrasse 38, 70569 Stuttgart, Germany
{joerg.nitzsche,tammo.van.lessen,dimka.karastoyanova,
frank.leymann}@iaas.uni-stuttgart.de
http://www.iaas.uni-stuttgart.de

Abstract In this paper we present BPEL^{light} which decouples process logic from interface definitions. By extending BPEL 2.0 with a WSDL-less interaction model, BPEL^{light} allows to specify process models independent of Web service technology. Since its interaction model is based on plain message exchange, it is completely independent of any interface description language. This fosters flexibility and reusability of process models and enables modelling platform and component model independent business processes. The presented approach takes a significant step towards narrowing down the gap between business level and IT level by facilitating a more business-oriented modelling of executable processes.

Key words: BPM, Workflow, BPEL, SOA, Web services, flexibility, reusability

1 Introduction

Business Process Management (BPM) and the workflow technology [1,2] in particular has become a very successful area with heavy impact on industry and research. Process orientation has been discussed for many years but with the emergence of Web Services [3,4] (WS) which is the most popular implementation of a service oriented architecture [5,6] (SOA) workflow technology and BPM got established to a great extent. The separation of business process logic and separate implementation of business functions enables programming on a higher, i.e. business process oriented level [7]. A workflow comprises 3 dimensions: process logic ('what' is to be done), organization ('who' does it) and infrastructure ('which' tools are used). There are two major standards for business processes. The execution centric Business Process Execution Language [8] (BPEL) has currently been approved as an OASIS[1] standard and the modelling focussed Business Process Modelling Notation (BPMN) [9] is standardized by OMG[2]. BPEL is part of the WS standard stack and is therefore based on WSs in particular on

[1] http://www.oasis-open.org/
[2] http://www.omg.org/

the Web Service Description Language [10] (WSDL). The 'who' dimension is not supported yet and the 'which' dimension is simply based on WSs.

In BPEL the 'what' and 'which' dimensions are strongly coupled since activities which are an aspect of the process logic ('what') directly refer to WSDL operations ('which'). This is a major drawback because it inhibits the reuse of processes or parts thereof in different contexts with different partners. Also this ties BPEL to WSDL for referring to activity implementations.

With BPEL^{light} we present an approach that gets over these deficiencies. First, we use BPEL's extensibility mechanisms to define a unified interaction model by introducing a new, single type of interaction activity resuming all interaction activities [3] currently defined by BPEL. Second, BPEL^{light} enables a strict decoupling of business logic and interface definitions (port types); as a result, interfaces in BPEL^{light} processes can be described via any interface definition languages (IDL, including WSDL). Without the fixed dependency on WSDL, BPEL^{light} can be used even in non-WS environments (*WSDL-less BPEL*). Especially, partner services even do not have to be described in terms of interface definitions at all: It is sufficient to describe how the process wants to interact with a partner in terms of a bilateral message exchange. Such a message exchange can be mapped to appropriate interfaces during deployment or even during runtime via proper tools and middleware. This results in a more business-like modelling style supported by BPEL^{light} and is a significant step towards narrowing down the gap between business level (e.g. BPMN) and IT level (BPEL).

Also, our approach fosters both, reusability and flexibility of processes. Since BPEL^{light} describes interactions in terms of message exchanges only, i.e. independent of interface definitions, processes or process fragments can be reused and bound to specific interfaces in any IDL. Binding may even happen during runtime, e.g. proper middleware can dynamically decide on an appropriate interface and corresponding implementation.

Our paper is structured as follows. Section 2 provides an overview of BPEL's interaction model. The subsequent section (3) introduces and discusses two different approaches of WSDL-less BPEL. In section 4 BPEL^{light} is presented. Section 5 shows how to realize BPEL's interaction semantics using BPEL^{light} in conjunction with WSDL. Section 6 discusses and summarizes the advantages of BPEL^{light} compared to conventional BPEL.

2 BPEL

BPEL is the de facto standard for specifying business processes in a WS world and has gained broad acceptance in industry and research. It enables both, the composition of WSs [3] and rendering the composition itself as WSs. Thus, BPEL provides a recursive aggregation model for WSs. Currently, extensions to BPEL are developed to support human interactions (BPEL4People [11]) and use of subprocesses (BPEL-SPE [12]). The composition of WSs can be specified as a flow between operations of WS. Therefore BPEL provides several so called *structured activities* that facilitate prescribing the control flow between the *interaction*

activities. BPEL does not support explicit data flow; instead, data is stored in shared variables that are referenced and accessed by interaction activities and manipulation activities (e.g. <assign> activity). The control flow between activities can be structured either block-based by nesting structured activities like <sequence> (for sequential control flow), <flow> (for parallel control flow) and <if> (for conditional branches in the control flow) activities, or graph-based by defining <links> (i.e. directed edges) between activities in a <flow> activity; both styles can be used intermixed.

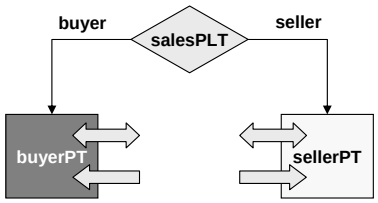
Since BPEL processes are intended to support robust applications, transactionality and fault handling are an integral part of BPEL and are defined by means of scopes, compensation handlers and fault handlers. Scopes represent units of works with compensation-based recovery semantics. Fault and compensation handlers are attached to a scope: fault handlers define how to proceed when faults occur, compensation handlers define how to compensate already completed activities in a custom manner.

WSs rendering process instances typically have state. Therefore it is important that messages can be sent to a particular process instance. This can be either achieved by using a standard BPEL mechanism called correlation sets, or by using WS-Addressing [13]. Correlation sets are based on pointing to key fields embedded in messages exchanged between the process instance and its partners.

In order to enable communication with other services or processes BPEL introduces the concept of a partner link type which is defined as an extension to WSDL. A partner link type binds two port types, namely a port type the process offers to a partner and a port type the process requires from the corresponding partner.

```
<wsdl:definitions
   targetNamespace=...
   xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
   ...
   <plnk:partnerLinkType name="salesPLT">
      <plnk:role name="buyer"
                 portType="buyerPT" />
      <plnk:role name="seller"
                 portType="sellerPT" />
   </plnk:partnerLinkType>
   ...
</wsdl:definitions>
```

(a) code snippet



(b) scenario

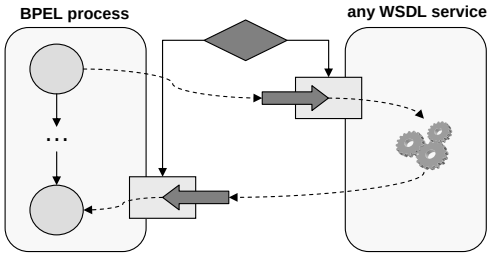
Figure 1. The WSDL extension <partnerLinkType>

Figure 1 shows an example of such a partner link type. It defines a channel (*salesPLT*) between two abstract business partners (roles) called *buyer* and *seller* through which the partners exchange messages; these roles are defined as port types, in the example *buyerPT* and *sellerPT*. In cases of a process synchronously interacting with a partner, such a channel is just unidirectional, i.e. the corresponding partner link type contains a single role. In order to establish a contract (i.e. an agreement between two partners which message channel to use), BPEL's partner links reference a partner link type and specify which role is taken by the process itself (myRole) and which role is taken by the partner (partnerRole).

The interaction activities [3] (<receive>, <reply>, <invoke>, <pick>) and the event handlers are used to define the actual message exchange corresponding to a partner link, i.e. data transmitted and style of communication (synchronous vs. asynchronous). For that purpose, interaction activities reference a partner link and a WSDL operation. Receiving activities (i.e. <receive> and <pick>) and the <reply> activity as well as the event handler reference an operation of the process's port type, whereas the <invoke> activity references an operation of the partner's port type. Note, that a synchronous invocation of a process is specified via a receiving activity and a matching reply activity.

```
<flow>
   <links>
      <link name="send-to-receive" />...
   </links>...
   <invoke name="orderItem"
           partnerLink="salesPL"
           operation="getOrder"
           inputVariable="item">
      <sources>
         <source linkName="send-to-receive" />...
      </sources>...
   </invoke>...
   <receive name="receiveConfirmation"
            partnerLink="salesPL"
            operation="getConfirmation"
            variable="confirmation">
      <targets>
         <target linkName="send-to-receive" />...
      </targets>...
   </receive>
</flow>
```

(a) code snippet



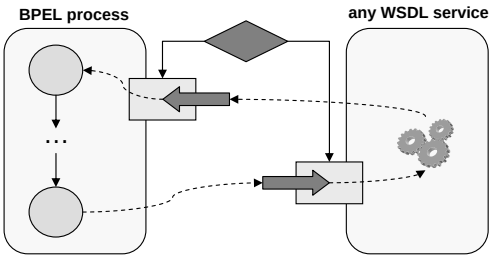
(b) scenario

Figure 2. Asynchronous invocation of a WSDL service

Figure 2 illustrates the use of an <invoke> and a <receive> activity to model an asynchronous invocation of a partner via two one-way operations. The partner link used within this example references the partner link type given in Figure 1 and defines myRole="buyer" and partnerRole="seller".

```
<flow>
   <links>
      <link name="receive-to-send" />...
   </links>...
   <receive name="receiveOrder"
            partnerLink="salesPL"
            operation="order"
            variable="confirmation">
      <targets>
         <target linkName="receive-to-send" />...
      </targets>...
   </receive>...
   <invoke name="sendConfirmation"
           partnerLink="salesPL"
           operation="getOrder"
           inputVariable="item">
      <sources>
         <source linkName="receive-to-send" />...
      </sources>...
   </invoke>
</flow>
```

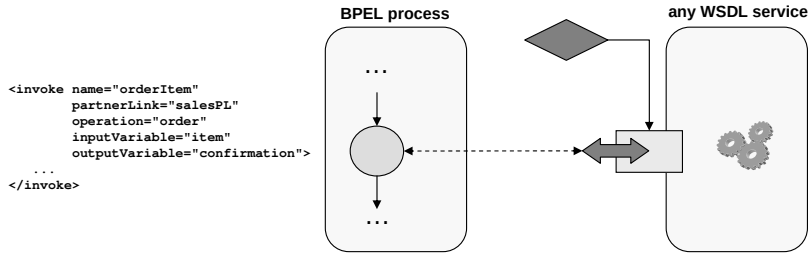
(a) code snippet



(b) scenario

Figure 3. Asynchronous invocation of a BPEL process

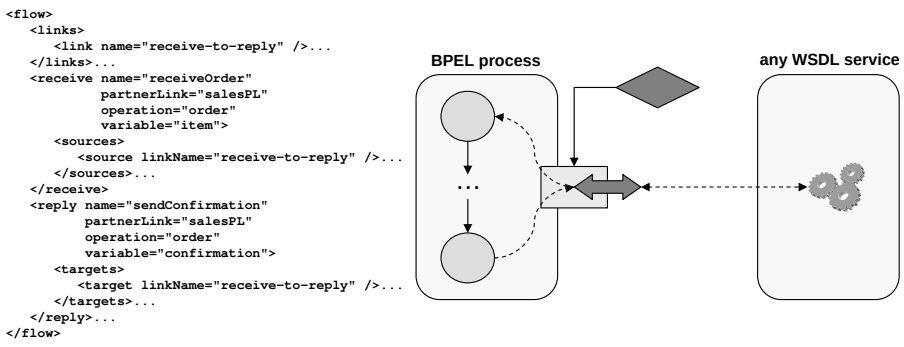
An example of an asynchronous invocation of the process is shown in Figure 3. In this example the partner link type presented in Figure 1 is also used but the partner link defines `myRole="seller"` and `partnerRole="buyer"`.



(a) code snippet (b) scenario

Figure 4. Synchronous invocation of a WSDL service

The simple synchronous use cases are illustrated in Figure 4 and Figure 5. The former shows how a synchronous invocation of a service can be modelled: The `<invoke>` activity of the process uses a request-response operation (*order*) provided by the partner service. In this case only the partner role of the *salesPL* is specified. The latter depicts a synchronous invocation of the process. It is realized by a `<receive>`-`<reply>` pair referencing the *order* operation the process offers. The partner link only specifies the `myRole` part of the partner link definition.



(a) code snippet (b) scenario

Figure 5. Synchronous invocation of a BPEL process

The `<pick>` activity and the `<eventHandler>` play a special role with respect to the WSDL dependency since they do not depend on WSDL itself but

encapsulate elements which references a WSDL operation, the `<onMessage>` element and `<onEvent>` element respectively.

Both the interaction activities and the grouping mechanism that allows modelling complex message exchanges depend on WSDL. For this reason, reusability and flexibility of BPEL processes or parts of processes (process fragments) are very limited. In the next section we present two approaches that decouple process logic from the WSDL interfaces (or any other interface descriptions) and thus increase reusability and flexibility of the processes and process fragments.

3 The notion of WSDL-less BPEL

There are two major shortcomings of BPEL: limited reusability of (parts of) processes and lack of flexibility in terms of interfaces.

Hard-coding (partner) interfaces in the process logic limits reusability. Assume a process sends a message to a partner service, receives a response and dependent on this response different branches are taken in the subsequent flow. This combination of an interaction activity and a decision taken occurs very often in processes [14] and thus is a good candidate for reuse ("process fragment"). However, since such a fragment is bound to specific WSDL operations it cannot be reused in other scenarios where the same logic is required but the interaction is specified in terms of different WSDL operations and different WSDL port types.

The other downside of hard-coding interfaces in process logic is lack of flexibility. Only services that implement the predefined WSDL interface can be used during process execution, whereas services that provide the same functionality but implement a different interface are excluded from the service discovery procedure.

In addition, explicitly specifying partner interfaces in a process definition results in tight-coupling of the two dimensions of WS composition, namely process logic ('what') and activity implementations ('which'). For instance, modelling a two-way invoke enforces the use of exactly a WSDL request-response operation at the partner side. However, such a two-way message exchange pattern [15] could be realized by two one-way operations, increasing flexibility by weakening the assumption on the concrete type of WSDL operation to be used. Thus, it should be sufficient to model an activity with one outgoing message and one incoming message, and leaving the selection of the proper WSDL operation(s) and interaction style to the supporting middleware. In this case, the BPEL navigator must understand that the activity is only completed after a response message has been received. We argue that this drawback is eliminated by decoupling the 'what' and 'which' dimensions, and thus separate process logic and communication.

We identify two possible approaches for discarding the static specification of port types and operations in processes and thus improving process reusability and flexibility. These alternative approaches will be presented in the following sections.

3.1 Profile for abstract BPEL

BPEL enables the definition of profiles for abstract processes. These profiles enable omitting information that is necessary for describing an executable BPEL process. The BPEL 2.0 specification identifies two use cases for abstract processes: (i) definition of the *observable behaviour* of a process, (ii) definition of process *templates*.

However, it is envisioned that there are other use cases for profiles of abstract processes. Thus, the BPEL specification defines a starting point for abstract profiles, the *Common Base* that defines the syntactic form to which all abstract processes have to conform. The Common Base defines that activities, expressions, attributes and from-specifications may be hidden by replacing them with opaque tokens. However, it does not define the semantics of an abstract process. The semantics have to be defined in a profile. Additionally, the profile defines more precisely which information may be omitted. The profile for observable behaviour for instance defines, that the interaction activities themselves and the attributes `partnerLink` and `operation` must not be opaque.

Following the generic approach of abstract profiles enables specifying a profile that allows omitting WSDL specific details and thereby increases reusability of process models. This approach is for instance used in [16] to define a language for choreographies using BPEL.

3.2 Extensions to executable BPEL

BPEL is designed to be extensible. BPEL 1.1 [17] can be extended by defining new attributes and standard elements. However, in BPEL 1.1 the extensibility is limited since there is no way to introduce new activity types without violating the BPEL Schema and specification and losing BPEL compliance. In order to eliminate this drawback BPEL 2.0 [8] introduces the `<extensionActivity>` that is the designated extension point for new activity types. Additionally, BPEL 2.0 facilitates defining custom assign operations. In both specifications the extensions must not change the semantics of the BPEL language. The extensibility features of BPEL can be used to define new interaction activity types that do not reference WSDL interfaces. In particular this can be done using the `<extensionActivity>` mechanism. This implies that a new partner link definition (WSDL-less partner link) is necessary, which also does not refer to a WSDL definition. This way BPEL enables defining a WSDL-less interaction model by introducing new WSDL-less activity types and partner links.

3.3 Discussion

The approach of creating a profile for abstract BPEL allows omitting WSDL specific details during design time and thereby increasing reusability of abstract processes. However, when completing such an abstract process into an executable process the 'what' dimension and the 'which' dimension are coupled, and the BPEL process depends on WSDL again. For this reason, the approach using

profiles for abstract BPEL results in design time flexibility only, improving the reusability of process definitions only during the modelling phase.

The WSDL-less interaction model defined using BPEL's extensibility mechanism provides for flexibility of process models at modelling time and at execution time. This results in reusable executable processes and process fragments. The flexibility of executable processes is further increased because WSDL interfaces of partners are no longer part of process models: Activities are bound during deployment or even as late as runtime to proper implementations. Obviously, this requires proper tooling and runtime support [18]. Moreover, interface definitions are not restricted to be specified in WSDL, but rather any other IDLs can be used.

Since our second approach is more powerful the rest of the paper is focussed on that. The extended BPEL language we introduce in the next section is a light-weight version of BPEL that can be applied for specifying business processes not only in WS-* environments. We call this language BPEL^{light}.

4 BPEL^{light}

BPEL^{light} is an extension of BPEL 2.0 [8], i.e. the existing semantics of the language remains unchanged, including variable handling and typing. It defines a new mechanism to describe the interaction between two partners without dependency on WSDL and therefore it decouples the two dimensions of BPEL processes, namely 'what' and 'which'. BPEL^{light} introduces new elements in a separate namespace[3] which represent a WSDL-less conversation between partners using WSDL-less interaction activities. We describe the BPEL^{light} interaction model and enhance and adapt the concept of uniquely identifiable partners to support stateful WSDL-less conversations.

4.1 The BPEL^{light} interaction model

We define the BPEL^{light} interaction model in terms of two elements, namely `<conversation>` and `<interactionActivity>`.

The `<conversation>` element plays the role of a WSDL-less partner link not referencing a partner link type. Thus it defines a contract between two partners independent of their WSDL port types, i.e. interfaces. The `<conversation>` element allows grouping of interaction activities and thus enables defining a complex message exchange between two partners. Hence the requirements to the partner service is not expressed using WSDL port types, but rather by the ability to send messages to and receive messages from a process during a conversation.

Similarly to the `<partnerLink>` which is defined in the `<partnerLinks>` section, every `<conversation>` is defined within a `<conversations>` element. The syntax is shown in Listing 1.

In order to decouple the interaction activities from the activity implementation dimension ('which') we define interaction activities that do not refer to WSDL

[3] `xmlns:bl=http://iaas.uni-stuttgart.de/BPELlight/`

```
<bl:conversations>
  <bl:conversation name="NCName"/>+
</bl:conversations>
```

Listing 1. The <conversation> element

interfaces. The interaction activities defined in BPEL (<receive>, <reply>, <invoke> and <onMessage> within a <pick>) cannot be used since the WSDL specific attributes partnerLink and operation are mandatory. These new interaction activities can model simple and complex message exchanges with a partner by referencing a <conversation> element.

In BPEL^{light} we utilize the <extensionActivity> mechanism to introduce a new activity type – the <interactionActivity> (see Listing 2). This activity type is capable of modelling all interaction activities defined in BPEL. Additionally, it can be configured to represent an activity that receives a message and is not completed before sending a response message.

```
<extensionActivity>
   <bl:interactionActivity name="NCName"
                           inputVariable="NCName"?
                           outputVariable="NCName"?
                           mode="in-out|out-in"?
                           conversation="NCName"
                           createInstance="yes|no"?
                           standard-attributes>
      standard-elements
   </bl:interactionActivity>
</extensionActivity>
```

Listing 2. BPEL^{light}'s <interactionActivity>

The activity types that are covered by the <interactionActivity> are summarised in the following:

1. activities that only receive a message (like a BPEL <receive>)
2. activities that only send a message (like a BPEL <invoke> or <reply>)
3. activities that first send a message and then receive a message (like a BPEL synchronous/two-way <invoke>)
4. activities that first receive a message and then send a message

The BPEL^{light} <interactionActivity> is comparable to a BPMN task. Similarly to BPEL^{light}, BPMN [9] does not define different task types but rather specifies one task and this task may have incoming and outgoing messages. However, BPMN is only a modelling notation, whereas BPEL^{light} is executable.

Table 1 shows how the interaction activity has to be configured to model the different activity types listed above. Activities that receive a message must specify the output variable whereas activities that send a message must specify

the input variable. Activities that send a message only must not define the output variable, and activities that only receive a message must not define the input variable. For these activities the value for the attribute "mode" is not evaluated. Activities that do both, receive and send a message, must specify the attribute `mode`. The value has to be set to `in-out` for activities that first receive a message and `out-in` for activities that first send a message. The default value for the attribute `createInstance` is `no`. Activities that start with a receiving message may specify this attribute, for the other activity types this attribute is not evaluated.

	input variable	output variable	mode	create instance
Activity that only receives a message	MUST NOT	MUST		MAY
Activity that only sends a message	MUST	MUST NOT		
Activity that first receives and then sends a message	MUST	MUST	in-out	MAY
Activity that first sends and then receives a message	MUST	MUST	out-in	

Table 1. Modelling different interaction activity types

In order to be aligned with BPEL we introduce a new `<pick>` activitiy with semantics similar to the pick activity in BPEL. This is required since the BPEL specification enforces to have at least one conventional `<onMessage>` element specified, whose dependency on WSDL breaks the idea of BPEL^{light}. Instead the new activity allows to specify WSDL-less `<onMessage>` elements that reference a conversation just as the interaction activity. Additionally and a new WSDL-less `<onEvent>` element for the `<eventHandler>` is defined.

To close the description of the BPEL^{light} interaction model Listing 3 illustrates how the sample BPEL process showed in Figure 2a is modelled using BPEL^{light}.

4.2 The notion of partners

BPEL 1.1 includes a `<partner>` element that groups a subset of partner links to identify a partner within a process. This way the `<partner>` element postulates that several partner links have to be established with one and the same business partner. Thus it specifies what capabilities (in terms of port types) a specific partner has to provide. In BPEL 2.0 the partner element has been removed.

In BPEL^{light} we introduces a new `<partner>` element that enables grouping the WSDL independent `<conversation>`s. Thus it can be defined that several

```
<bl:conversations>
   <bl:conversation name="salesConv"/>...
</bl:conversations>...
<flow>
   <links>
      <link name="send-to-receive" />...
   </links>...
   <extensionActivity>
      <bl:interactionActivity name="orderItem"
                              conversation="salesConv"
                              inputVariable="item">
         <sources>
            <source linkName="send-to-receive"/>...
         </sources>...
      </bl:interactionActivity>
   </extensionActivity>
   <extensionActivity>
      <bl:interactionActivity name="receiveConfirmation"
                              conversation="salesConv"
                              outputVariable="confirmation">
         <targets>
            <target linkName="send-to-receive"/>...
         </targets>...
      </bl:interactionActivity>
   </extensionActivity>...
</flow>
```

Listing 3. Asynchronous invocation of a service using BPEL^{light}

conversations have to take place with one business partner. The new <partner> element, which is referring to a <conversation> instead of a partner link is illustrated in Listing 4. This is a way to impose constraints on a partner to support multiple conversations, i.e. message exchanges, and thereby multiple business goals. Assume a flight should be booked after the price for this particular flight has been checked. In this case it is required that both activity implementations are using the same partner to avoid checking the price at Lufthansa and then booking a British Airways flight. Since the granularity of these business goals and thereby the granularity of the conversations cannot be standardised the <partner> element is needed to support the explicit specification of different granules in a user-friendly manner.

```
<bl:partners>?
  <bl:partner name="NCName"
              businessEntity="QName"?>+
    <bl:conversation name="NCName">+
  </bl:partner>
</bl:partners>
```

Listing 4. The <partner> element in BPEL^{light}

In addition, the <partner> element may define the concrete partner instance that has to be used for a set of conversations. This is realized using the businessEntity attribute, which specifies the name of an organisation.

Consequently BPEL^{light} comes with an extension to the `<assign>` activity that enables copying a partner identification into the `<partner>` element. Therefore the empty `<to>` specification is extended with a `<partner>` attribute that defines to which partner definition the partner instance information is copied. Note, that a partner can only be set if its corresponding conversations have not been established yet. This is similar to copying an endpoint reference to a partner link.

5 Using BPEL^{light} in WS-* environment

As already discussed, BPEL^{light} decouples from WSDL. Since WS-* is the most popular service-based integration technology we show how BPEL^{light} can be used to support WSs based compositions. Therefore when using BPEL^{light} in a WS-* environment it emulates the interaction model of BPEL. This way BPEL^{light} also provides a recursive aggregation model for WSs analogously to conventional BPEL, i.e. a BPEL^{light} process can expose its functionality as a WS and it can invoke conventional WSs.

Note that even though the communication semantics of BPEL can be emulated, the two dimensions are still decoupled since the mapping of the technology neutral interaction model of BPEL^{light} to WSDL is external, i.e. not within the process logic.

To enable interaction among WSDL-based services, a conversation serves the role of a partner link and is associated with a partner link type. The role of a partner service and the role the process itself takes are also specified. In addition to the association of the conversation to the partner link type, all interaction activities have to be mapped to WSDL operations.

The semantics of the BPEL interaction activities can be achieved using the following mappings: an interaction activity with both variables specified and `mode="out-in"` is mapped to a request-response type operation the partner provides, which is similar to a synchronous BPEL `<invoke>`. An interaction activity with the output variable specified, which corresponds to a BPEL `<receive>`, can be assigned to a one-way operation the process provides. However, it can also be assigned to a request-response operation the process provides. In this case there must be a successive interaction activity with the input variable specified that is also assigned to that particular operation. Together, these two activities provide the semantics of a synchronous `<receive>`-`<reply>` pair in BPEL. The interaction activity with only the input variable specified may also be assigned to a one-way operation the partner service provides, which corresponds to an asynchronous `<invoke>` in BPEL.

Additionally, an interaction activity with both variables specified and the attribute `mode` set to `"in-out"` can be assigned to a request-response operation provided by the the process. This scenario has no direct counterpart in BPEL (instead, a receive-reply pair is used).

The assignment of the activities to a communication infrastructure can be done (i) using an *assignment file* that is interpreted during deployment (ii) using

WS-PolicyAttachment [19] or (iii) by delegating all communication issues to the underlying middleware, e.g. the ESB [20]. In this paper we focus only on the first approach.

The assignment of the process logic to WSDL operations via an assignment file is depicted in Listing 5. The `<conversation>` is associated with a partner link type and it is specified which role is taken by the partner service and which is taken by the process itself using the `myRole` and `partnerRole` attribute. This is similar to the association of a partner link to a partner link type. Additionally, the activities have to be mapped to corresponding WSDL operations using the `operation` attribute like in BPEL.

```
<assignmentFor process="QName">
   <conversation name="NCName"
                 partnerLinkType="QName"
                 myRole="NCName"
                 partnerRole="NCName" />*
   <activity name="NCName"
             operation="NCName" />*
</assignmentFor>
```

Listing 5. Assignment file

So far we have shown that BPEL^{light} can be used to express the communication semantics of BPEL. However, a consequence of this approach is again a tight coupling of the 'what' and 'which' dimensions since there is a direct dependency of the behaviour of the navigator and the kind of communication used: a blocking activity implies a synchronous request-response operation. However, there is an alternative and maybe even more promising approach of applying WSDL to BPEL^{light}. The idea is not to map the activities directly to operations but rather map the input- and output variable to an operation. This way a blocking interaction activity that first sends and then receives a message can for instance be mapped to two one-way operations, one provided by the partner service and one by the process.

Since the consequences of this approach have to be investigated in depth we consider it future work.

6 Discussion and Assessment

BPEL^{light} decouples process logic and interface definitions. Interfaces are defined separately and bound to activities in processes separate from BPEL^{light}. Moreover interfaces can be specified in any IDL.

BPEL^{light} eases modelling of business processes. By separating interfaces and process logic the amount of IT artefacts a process modeller must understand is reduced. It is sufficient to describe how the process wants to interact with a partner. This behaviour can be mapped separately during deployment or at runtime by proper middleware to an interface of a partner. Figure 6 illustrates

how BPEL^{light} improves the business process modelling lifecycle: The grey parts indicate the artefacts a process modeller needs to know. On the top of the figure the situation with traditional BPEL is shown whereas in the bottom it is depicted that BPEL^{light} frees the modeller from IT specific details.

BPEL^{light} decouples activity definitions from component models. Different IDLs can be mixed and matched within one and the same process model. Based on the concept of assignment files, proper middleware can be configured that allows to bridge to the hosting environment of the corresponding components.

BPEL^{light} increases reusability and flexibility of process models. One and the same piece of process logic can be bound to completely different interfaces dependent on the set of interfaces available in a target environment.

BPEL^{light} eases mapping of BPMN to an execution environment. By supporting the specification of message exchange patterns directly, BPEL^{light} removes the impedance mismatch of message orientation and interface orientation which causes problems in mapping BPMN to conventional BPEL.

BPEL^{light} eases the construction of matching partner processes, e.g. in choreographies. It is straightforward to construct to a given activity the matching dual activity in a partner process: the requested activity can be found by simply mirroring the conversation.

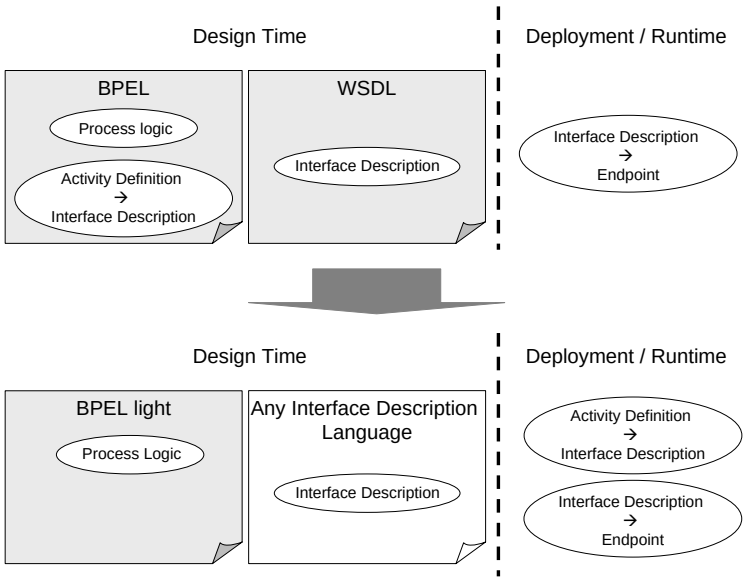


Figure 6. Modelling with BPEL^{light} and BPEL: Improvements

7 Conclusion

The work in this paper strives improving the flexibility and reusability of process-based compositions especially in service-oriented environments. We presented a language for composition, called BPEL^{light}, that facilitates decoupling of the two dimensions of process compositions - business logic ('what') and the activity implementations ('which'). The advantage of BPEL^{light} over conventional BPEL is that it allows to specify process models independent of WS technology; in fact it is independent of any other interface technology used to implement activities. As a result, BPEL^{light} improves both, reusability and flexibility of process models.

Reusability of process definitions is improved since the process definitions can be used with different service technologies while the business logic remains unchanged. In addition, a mixture of service technologies may be utilized when a process model is executed. Moreover, any BPEL^{light} process definition can be used to generate partner interfaces automatically based on information about message exchange patterns defined in BPEL^{light}. This is enabled by the inherent symmetry of the message exchange patterns of interacting partners. BPEL^{light} improves the flexibility of process definitions because both deployment time configuration and run time discovery and binding to appropriate interfaces is enabled. In our view, the process language introduced in this paper is an adequate answer to the needs businesses have with respect to reusability and flexibility of processes. It is an extension of the de facto standard for service composition (BPEL) and therefore the industry relevance and application of these language extensions is not hampered.

Currently we are developing a process modelling tool with native BPEL^{light} extension support. Building an engine for executing BPEL^{light} processes is part of our future work.

8 Acknowledgments

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

References

1. Leymann, F., Roller, D.: Production workflow. Prentice Hall (2000)
2. van der Aalst, W., van Hee, K.: Workflow management. MIT Press (2002)
3. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.: 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)
4. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applications. Springer (2004)

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

5. Burbeck, S.: The Tao of e-business services. IBM Corporation (2000)
6. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series). Prentice Hall PTR Upper Saddle River, NJ, USA (2004)
7. Leymann, F., Roller, D.: Workflow-based applications. IBM Systems Journal **36**(1) (1997) 102–123
8. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C.K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0. Committee specification, OASIS Web Services Business Process Execution Language (WSBPEL) TC (January 2007)
9. White, S.: Business Process Modeling Notation (BPMN) Version 1.0. Object Management Group/Business Process Management Initiative, BPMN.org (2004)
10. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1 (2001)
11. Kloppmann, M., Koenig, D., Leymann, F., Pfau, G., Rickayzen, A., von Riegen, C., Schmidt, P., Trickovic, I.: WS-BPEL Extension for People – BPEL4People. Joint white paper, IBM and SAP, July (2005)
12. Kloppmann, M., Konig, D., Leymann, F., Pfau, G., Rickayzen, A., von Riegen, C., Schmidt, P., Trickovic, I.: WS-BPEL Extension for Sub-processes – BPEL-SPE. Joint white paper, IBM and SAP (2005)
13. Box, D., Curbera, F., et al.: Web Services Addressing (WS-Addressing). W3C Member Submission (August 2004)
14. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51
15. Barros, A., Dumas, M., ter Hofstede, A.: 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)
16. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: ICWS 2007, IEEE Computer Society (July 2007)
17. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services (BPEL) 1.1. http://www.ibm.com/developerworks/library/specification/ws-bpel/ (2003)
18. Karastoyanova, D., van Lessen, T., Nitzsche, J., Wetzstein, B., Wutke, D., Leymann, F.: Semantic Service Bus: Architecture and Implementation of a Next Generation Middleware. In: 2nd International Workshop on Services Engineering (SEIW). (April 2007) Istanbul, Turkey.
19. Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Malhotra, A., Maruyama, H., et al.: Web Services Policy Attachment (WS-PolicyAttachment). W3C Member Submission (April 2006)
20. Chappell, D.A.: Enterprise Service Bus. O'Reilly (2004)