Modeling Web Service Interaction Using Extended BPEL Language

Márk Kaszó, Csaba Legány, Tihamér Levendovszky

Budapest University of Technology and Economics, Hungary
mkaszo@aut.bme.hu, legcsabi@aut.bme.hu, tihamer@aut.bme.hu

Abstract: Nowadays web services and SOA (Service Oriented Architecture) systems have fundamental significance in computer science, especially in platform integration. The most common consideration is system workload estimation. Interacting composite web services can be modeled as interacting BPEL processes. These web services are specified in BPEL and communicate with each other using asynchronous XML messages. In order to estimate system workload (such as system throughput, resource usage, request process time) this paper presents an extension of the standard BPEL language. This extension enables us to compute key factors of the system workload even in design time.

Keywords: BPEL, extension, web services, modeling, SOA, XML

1 Introduction

Business Process Execution language provides us a standard way to specify our business processes. This paper deals with problem of performance prediction of SOA systems in design time. BPEL is suitable to describe SOA systems, but the original version it is unable to describe performance parameters; thus a new BPEL extension with performance parameters is presented in this paper. These performance parameters are absolutely necessary to predicate SOA systems performance in design time. Of course other factors are also needed like proper workload model.

The structure of the article is as follows: Section summarizes BPEL Language. Our extension to the language can be found in Section 3. Finally Section 3 summarizes our work.

2 Summary of BPEL Language

BPEL [1] language resembles usual programming languages having variables, data operations, external function and service calls. The BPEL model specifies processes communicating with each other asynchronously using messages. Every message is made up of several parts, any part can be of a single type (e.g. xsd:string)[3][4] or of a complex type (e.g. sns:customerInfo).

Listing 1 contains three messages: OrderFaultMessage is a message containing only one single part, while InvMessage and POMessage are made up of complex parts.

```
<message name="POMessage">
    <part name="customerInfo" type="sns:customerInfo"/>
    <part name="purchaseOrder" type="sns:purchaseOrder"/>
</message>
<message name="InvMessage">
    <part name="IVC" type="sns:Invoice"/>
</message>
<message name="OrderFaultMessage">
    <part name="problemInfo" type="xsd:string"/>
</message>
```

Listing 1
Messages

Any BPEL process can have multiple port types. Port types specify available operations for each process and the required input and output messages. Listing 2 contains an example BPEL port type specification: purchaseOrderPT porttype can perform sendPurchaseOrder operation on an input POMessage. Normally it has to reply with and InvMessage or if a fault occurs with an OrderFaultMessage message.

```
<portType name="purchaseOrderPT">
    <operation name="sendPurchaseOrder"><input message="pos:POMessage"/>
    <output message="pos:InvMessage"/> <fault name="cannotCompleteOrder"
message="pos:OrderFaultMessage "/>
    </operation>
</portType>
```

Listing 2
Port types

Processes communicate with each other using partnerlinks according to BPEL model. Partnerlinks belong to partnerlink types (like the well-known object-class model), they define roles for each porttype. Listing 3 contains an example for

partner link types: purchasingLT partner link type uses purchaseOrderPT port type in purchaseService role for communication. The introduction of roles means that any partner link type can use different port types in different roles, which means that a process can communicate with different processes depending on its current role.

```
<plnk:partnerLinkType name="purchasingLT">
    <plnk:role name="purchaseService">
    <plnk:portType name="pos:purchaseOrderPT"/>
    </plnk:role>
</plnk:partnerLinkType>
```

Listing 3
Partner links

There are four major sections in the definition of any process.

- <variables>
- <partnerLinks>
- <partners>
- <faultHandlers>

The <variables> section defines data variables used by the process, which are required to assign states to processes. These states are necessary to track process lifecycle.

The <partnerLinks> section defines those processes which interact with the given process. A name can be assigned to each partner link using <partner> term.

The <faultHandlers> section contains fault handlers defining activities to be performed in response to faults. It is important to note that compared to WSDL 1.1, BPEL requires a uniform naming model for faults and fault handlers which means that BPEL provides a better fault-naming model. Listing 4 contains these sections for purchaseOrderProcess process. This process interacts with two other processes using purchasing and invoicing partner links. The process uses three variables, the third one for fault handling.

Process activity (sending and receiving messages) is detailed below.

```
<process name="purchaseOrderProcess"
    <partnerLinks>
        <partnerLink name="purchasing" partnerLinkType="lns:purchasingLT"
        myRole="purchaseService"/>
        <partnerLink name="invoicing" partnerLinkType="lns:invoicingLT"
        myRole="invoiceRequester" partnerRole="invoiceService"/>
    </partnerLinks>
    <partners>
```

```
        <partner name="purchasingPartner">
            <partnerLink name="purchasing"/>
        </partner>
        <partner name="invoicingPartner">
            <partnerLink name="invoicing"/>
        </partner>
    </partners>
    <variables>
        <variable name="PO" messageType="lns:POMessage"/>
        <variable name="Invoice" messageType="lns:InvMessage"/>
        <variable name="POFault" messageType="lns:OrderFaultMessage"/>
    </variables>
    <faultHandlers>
    <catch faultName="lns:cannotCompleteOrder" faultVariable="POFault">
        <reply partnerLink="purchasing" portType="lns:purchaseOrderPT"
        operation="sendPurchaseOrder" variable="POFault"
        faultName="cannotCompleteOrder"/>
    </catch>
    </faultHandlers>
<!—Process activity -->
</process>
```

Listing 4
Process structure

The process activity section defines the way it performs operations. The most important activities are the following:

- <sequence> : sequential operations
- <flow> : parallel operations
- <scope> : inlay activity
- <request> :request message handling
- <reply> : reply message handling

All operations inside a <sequence> term must be performed sequential (one after the other). In contrary, all operations inside a <flow> term must be performed side-by-side. The process activity section usually contains three sequential steps: receiving messages, processing them, sending reply messages. Listing 5 contains this activity structure: first the incoming message is stored to an input variable PO, it is processed in the <flow> steps (detailed below) finally a message is sent using the output variable Invoice.

```
<sequence>
    <receive partnerLink="purchasing" portType="lns:purchaseOrderPT"
    operation="sendPurchaseOrder" variable="PO">
    </receive>
    <flow>
    ...
    </flow>
    <reply partnerLink="purchasing" portType="lns:purchaseOrderPT"
    operation="sendPurchaseOrder" variable="Invoice"/>
</sequence>
```

Listing 5
Major steps of process activity

The following terms can be used in the <flow> section:

- <assign> - variable management
- <invoke> - invoking another process
- <receive> - receive reply
- <throw> - throw an exception to indicate internal errors
- <wait> - waiting
- <switch> - conditional branch
- <while> - cycle
- <pick> - message, alarm management
- <empty> - no operation
- <sequence> and <flow>: sequential or parallel sub-operations

The <assign> term can be used for variable management, Listing 6 provides an example for copying a part of a variable to another one.

```
<assign>
    <copy>
        <from variable="PO" part="customerInfo"/>
        <to variable="shippingRequest" part="customerInfo"/>
    </copy>
</assign>
```

Listing 6
Use of <assign> term

The <invoke> term can be used to invoke (call) another process. In order to invoke another process, the current process should provide the following: link, port type, operation to be performed, input and output variables. Listing 7 contains an example for invoking

```
<invoke partnerLink="shipping" portType="lns:shippingPT" operation="requestShipping"
inputVariable="shippingRequest" outputVariable="shippingInfo" />
```

Listing 7
Invoking

Receiving a reply message can be performed using <receive> term. Listing 8 contains an example for receiving the result of sendInvoice operation into variable Invoice.

```
<receive partnerLink="invoicing" portType="lns:invoiceCallbackPT"
operation="sendInvoice" variable="Invoice"/>
```

Listing 8
Receiving reply messages

In some cases, a dependency set of invoke calls can be estimated. It means that an invoke call cannot happen prior to another one. Listing 9 contains an example for invoke dependencies: the result of shipping is stored in a variable shippingInfo which is used by invoicing. This dependency is described as a link from shipping to invoicing.

```
<links>
    <link name="shipping-to-invoicing"/>
</links>

<invoke partnerLink="shipping" portType="lns:shippingPT" operation="requestShipping"
    inputVariable="shippingRequest" outputVariable="shippingInfo">
    <source linkName="shipping-to-invoicing"/>
</invoke>

<invoke partnerLink="invoicing" portType="lns:computePricePT"
operation="sendShippingPrice" inputVariable="shippingInfo">
    <target linkName="shipping-to-invoicing"/>
</invoke>
```

Listing 9
Invoke dependencies

It can be noted that BPEL language contains further terms like <correlationSet>-s or <eventHandler>-s.

3 Extending BPEL

Section 2 contained a detailed description of BPEL specification. Now we are focusing on its extension in order to gain additional information about our system. This extension will be used in Section 4 to calculate key workload factors.

3.1 Process Model

Interacting web pages can be modeled as follows. Each web page call requires several processes to run. In our abstract model, a process can be a web page, a web service, an application layer function call or even a stored procedure. For example if a new user registers on a web site by filling in the registration form, this form sends the registration data to the *register_user* web service which calls the appropriate stored procedures. In our model, every process determines the server it is running on. For example the *register_user* web service runs on *S* web server. Each process is made up of several sub-processes: sub-processes of a process run on the same server. Processes communicate with each other using input and output messages. Figure 1 depicts summarizes this model of interacting web pages and services. Processes are denoted with *P*, sub-processes with P_{sub} and servers with S_i.

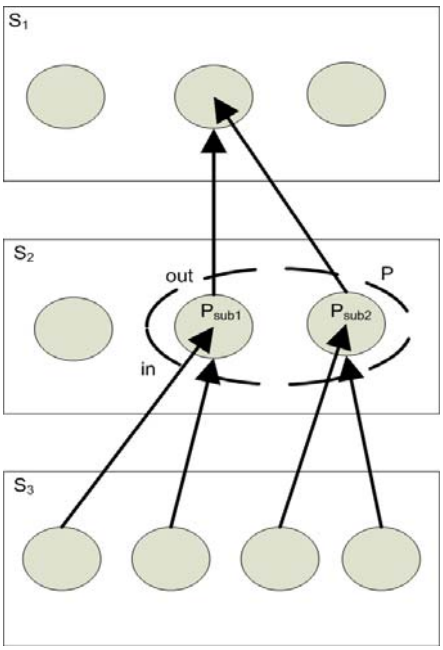

Figure 1
Model of interacting web pages using processes

3.2 BPEL Extensions

Our first extension to BPEL is the extension of BPEL messages as follows. All of the newly added attributes and terms are displayed with bold in our listings.

Each message contains several message parts according to the BPEL specification detailed in Section 2. We have to add further attributes for each message to describe its size properly (to estimate the cost of each message on the server as described later). For example if a message contains two int variables, the size of each variable is 32 bytes and there are two units of int variables in the message (Listing 10).

```
<message name="POMessage">
        <part name="ints" type="xsd:int" size=”32” unit=”2”/> ...
</message>
```

Listing 10
Message extension

Next we introduce a new term, called as <server>. Now it only describes the cost of each variable type on the server. The serialization, CPU, IO and memory cost of each variable type can be measured and these costs only depend on the server used. For example the serialization cost of an int variable can be measured by serializing a large array of int variables and dividing the result by the size of the array (average cost). Listing 11 contains an example for the new <server> term, while Figure 2 depicts the meta-model of it.

```
<server name=”S1”>
    <type name=”xsd:int”>
        <cost name=”serialization” value=”120”/>
        <cost name=”cpu” value=”140”/>
        <cost name=”io” value=”10”/>
        <cost name=”memload” value=”12”/>
    </type>
        ...
</server>
```

Listing 11
Newly introduced server term

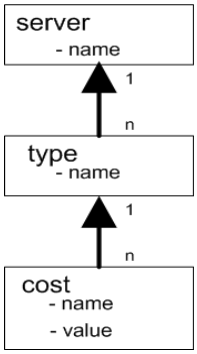

Figure 2
Meta-model of server term

Processes interact with each other using partner links. An attribute was added to each link type to describe the average delay of it. Listing 12 contains the newly added attribute to partner link types.

```
<plnk:partnerLinkType name="purchasingLT" avg_delay=”120” >...
</plnk:partnerLinkType>
```

Listing 12
Adding delay to links

Finally, <process> term was extended. As described previously a process defines the server it is running on. Thus, a <server> term was added to processes. Since it is difficult to model business logic of processes, we suppose that the runtime of processes mainly depends on their data transformation time and the time they spend with receiving and sending input and output messages. Data transformation time is introduced as the <data_transformation> tag. Listing 13 contains the two additional tags of processes.

```
<process name="purchaseOrderProcess"
        <data_transformation value=”120” />
        <server name=”S1” />...
</process>
```

Listing 13
Extending processes

BPEL is an XML document, so it is easy to extract the additional information using XML, and XPATH[5].

Conclusions and Future Works

This paper introduced a method to extend BPEL language to provide such a model of interacting web services that can be used to calculate important system workload factors.

The initial model of interacting web services was extended using standard XML elements. Our future work is to calculate key parameters of interacting web services using extended BPEL language in practice.

References

[1] Business Process Execution Language for Web Services, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

[2] Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl, March 2001

[3] Extensible Markup Language (XML). http://www.w3c.org/XML

[4] XML Schema. http://www.w3c.org/XML/Schema

[5] XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath, 1999