University of Peloponnese

Department of Computer Science and Technology

Software and Database Systems Laboratory

Preprocessor transformations for implementing greedy and service provider-level QoS-based adaptation for BPEL scenario execution

Christos Kareliotis (ckar@di.uoa.gr)

Costas Vassilakis (costas@uop.gr)

Efstathios Rouvas (rouvas@di.uoa.gr)

Panayiotis Georgiadis (p.georgiadis@di.uoa.gr)

August, 2012

Tripoli, Greece

Table of Contents

1. Introduction

In this technical report, we describe and exemplify the transformations applied by the WS-BPEL preprocessor, in order to produce a BPEL scenario that can be adapted according to QoS specifications, in the architecture described in [1].

[1] adopts a *greedy* algorithm for performing adaptation, i.e. it uses only the QoS specifications pertaining to the first *invoke* activity *IA* to a specific service provider *S*, so as to decide the service provider to which both IA and further invocations to operations provided by *S* will be directed.

The greedy algorithm may result in suboptimal bindings, while in some cases it may even lead to situations where the middleware is unable to find any appropriate service selection for fully servicing the BPEL scenario, albeit such a path *does* exist. To this end, a service provider-level adaptation strategy can be employed: the transformed scenario may communicate to the ASOB middleware [1] the information concerning *all operation invocations* to a specific partner link *before* the first invocation an operation provided by the specific partner link is executed, and therefore the ASOB middleware can exploit this information to remedy the problems stemming from the greedy nature of the adaptation method specified in [1].

The rest of this technical report is structured as follows: section 2 outlines the architecture of the adaptation scheme presented in [1]. Section 3 presents the transformations employed by the preprocessor for accommodating the greedy strategy, and section 4 presents the pseudo-code for performing the adaptations in the ASOB middleware. Finally, section 5 presents the transformations employed by the preprocessor for accommodating the service provider-level strategy.

2. Adaptation scheme architecture

The architecture proposed in [1] introduces two additional modules in a BPEL execution environment.

The first one is a middleware layer named Alternate Service Operation Binding (ASOB), which undertakes the tasks of redirecting operation invocations to providers best matching the QoS specifications designated by the clients and also performing automated exception resolution.

The second module is a preprocessor, which transforms BPEL scenarios created by designers so as to (a) direct invocations to the middleware layer and (b) include in each invocation all the necessary information for performing adaptation according to the QoS characteristics specified by the BPEL designer.

The overall framework architecture is depicted in Figure 1. In brief, the modules of the ASOB framework have the following functionality:

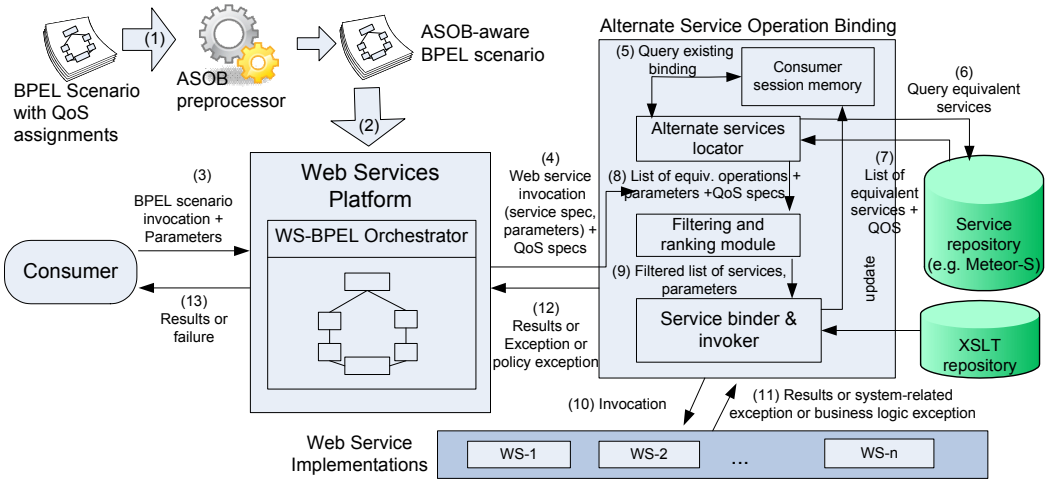

Figure 1 Architecture of the ASOB framework

1. *Alternate services locator*: locates services providing operations that are equivalent to the one currently invoked.

2. *Filtering and ranking module*: filters out alternate services that do not fulfill the QoS requirements specified by the client, and ranks remaining services according to the QoS specifications of the client.

3. *Service binder and invoker*: traverses the list of alternate services produced by the filtering and ranking module invoking each one in turn, until some service invocation succeeds (or the list is exhausted).

The Service repository (SR) provides information about (a) which operations are equivalent and (b) which are the values of QoS attributes of each operation. The XSLT repository contains transformation rules between semantically equivalent but syntactically different operations, adopting the approach of [2].

3. Transforming BPEL scenarios to accommodate greedy adaptation

As stated above, [1] adopts a *greedy* algorithm for performing adaptation, i.e. it uses only the QoS specifications pertaining to the first *invoke* activity *IA* to a specific service provider *S*, so as to decide the service provider to which both IA and further invocations to operations provided by *S* will be directed. To this end, each operation invocation simply needs to be redirected to the ASOB middleware, complemented with information regarding the QoS specifications stated by the BPEL designer, as well as (a) information regarding the service to be invoked and (b) the current *session id* [1], which is needed to maintain service selection affinity [1].

According to [1], the preprocessor takes as input the BPEL scenario *SC* crafted by the BPEL designer and produces as output an *ASOB-aware BPEL scenario* SC_{ASOB} by applying the following transformations:

1. SC_{ASOB} contains an additional *partnerLink* node, which corresponds to the ASOB middleware.
2. SC_{ASOB} includes, as its first operation, an invocation to a special web service operation of the ASOB middleware, namely *getSessionId*. This operation creates a value that is unique for a particular execution of the BPEL scenario, and returns it to the invoking scenario. Uniqueness is guaranteed by combining the requester's IP address, the current timestamp of the system and a random number from a sparse domain. As the operation name suggests, this value will be used as a session identifier for the particular execution of the BPEL scenario, in order to implement *service selection affinity*.
3. Each *invoke* node (i.e. each operation invocation) within SC is transformed as follows: firstly, WSDL file to which the *partnerlink* refers to is located and copied locally, and the *soapaction* address element for the particular invocation is amended to point to the ASOB middleware; the corresponding WSDL import is adjusted accordingly to point to the local copy. Secondly, the type of the *inputVariable* of the particular invocation is extended to accommodate five additional elements, namely *sessionId, origPLink, origAddress*, *ASOB_qoscons* and *ASOB_qosw*. To achieve the extension of *inputVariable*, the preprocessor downloads and appropriately modifies the files in which the type is defined (WSDL files for variables of type *messageType;* xml schemas for *element*) and amends import declarations to point to the modified files. For simple types (*type*), where the parameter is defined as a simple XML type (e.g. *string* or *integer*), the preprocessor creates an XML schema file defining a type containing the five aforementioned elements plus the element *ASOBvalue* of the appropriate type (*string, integer* etc), and amends the variable declaration to use the newly defined type; additionally, assignments (*copy* constructs) from/to this variable are modified accordingly. The transformed operation invocation is included in SC_{ASOB}.

According to these transformation rules, the BPEL scenario excerpt shown in Listing 1 will be transformed as shown in Listing 2.

```
<!-- set the value of the input parameter for the web service -->
<variable name="amount" type="xsd:integer″>
<assign name="assign1"> <copy>
  <from><literal>34</literal></from>
  <to variable="amount"/>
</copy></assign>
<!-- set the QoS specification -->
<assign name="QOSassign1">
  <copy>
    <from><literal>cost:0,2;sec:3,0</literal></from>
    <to variable="ASOB_QoSconstraints"/>
  </copy>
  <copy>
    <from><literal>cost:-3,sec:1,resp:2</literal></from>
    <to variable="ASOB_QoSweight"/>
  </copy>
</assign>
<invoke partnerLink="myLink" portType="thePort" operation="someOp"
    inputVariable="amount" outputVariable="theOutput"/>
```

Listing 1. Original BPEL scenario excerpt.

```
<!-- type_amount.xsd is a preprocessor-generated file in which the type type_amount is
defined, having the parts sessionId, origServiceAddress, ASOB_qoscons, ASOB_qosw and
ASOBvalue, the latter corresponding to the actual value -->

<import location="type_amount.xsd" importType="http://www.w3.org/2001/XMLSchema" />
<partnerLinks>
  <partnerLink name="ASOB" partnerLinkType="ASOBns:middleware"
    partnerRole="ASOBrole" />
</partnerLinks>
<variable name="ASOBsessionId" type="xsd:string">
<variable name="amount" type="ns:type_amount">

<!-- retrieve session id -->
<invoke partnerLink="ASOB" portType="ASOBport" operation="getSessionId"
outputVariable="ASOBsessionId" />

<!--assign the value to be passed to actual web service - modified-->
<assign name="assign1"> <copy>
  <from><literal>34</literal></from>
  <to variable="$amount.parameters/ASOBvalue"/>
</copy></assign>

<!--assign "QOSassign1" assigning literal values to ASOB_QoSconstraints and
ASOB_QoSweight is left intact and is not repeated here -->
<!-- plant extra fields in the input message -->
<assign name="ASOB_ASSIGN1">
  <copy> <from variable="ASOBsessionId"/>
    <to variable="$amount.parameters/sessionId"/> </copy>
  <copy> <from><literal>"http://addr.com/path"</literal></from>
    <to variable="$amount.parameters/origAddress"> </copy>
  <copy> <from><literal>"someOp"</literal></from>
    <to variable="$amount.parameters/origOperation"> </copy>
  <copy> <from variable="ASOB_QoSconstraints" />
    <to variable="$amount.parameters/ASOB_qoscons" /> </copy>
  <copy> <from variable="ASOB_QoSweight" />
    <to variable="$amount.parameters/ASOB_qosw" /> </copy>
</assign>
<invoke partnerLink="ASOB" portType="ASOBport" operation="proxyInvoke"
inputVariable="amount" outputVariable="theOutput"/>
```

Listing 2. Preprocessed BPEL scenario excerpt.

4. Processing requests in the ASOB middleware

As stated above, the preprocessor effectively redirects operation invocation to the ASOB middleware. Upon receiving an operation invocation request, the ASOB middleware executes the algorithm illustrated in Listing 3. The procedure for invoking a single alternative operation (invoked by the code in Listing 3) is detailed in Listing 4.

In listing 5, after each invocation to a service, function *update_SR_QoS* is called; this function arranges for informing the service repository about the observed behavior of individual operations (availability, response time etc), and the repository may in turn update the QoS database accordingly. This update takes place asynchronously, and it may also be optimized to minimize the number of messages towards the service repository, e.g. amassing a number of updates and forwarding them to the service repository in a single batch.

```
sessionID ← getPart(request, “ASOBsessionId”);
origPLink ← getPart(request, “origPLink”);
origAddress ← getPart(request, “origAddress”);
origOperation ← getPart(request, “origOperation”);
payload ← extractActualPayloadFromRequest(request);
qosConstraints ← getPart(request, “ASOB_qoscons”);
qosWeights ← getPart(request, “ASOB_qosw”);

boundService ← SessionMemory.query(sessionID, origPLink);
if (boundService != null)
    /* query the repository to find the operation of the boundService that is equivalent to
        the operation originally invoked; only one will be returned */
    candidateList ← Repository.getEquivOperation(origPLink, origOperation, boundService);
else
    /* query the repository for services that are equivalent to the one providing the
    operation originally invoked. Each service record returned contains the QoS
    parameters of the operation as well. */
    candidateList ← Repository.getAlternateOperations(origPLink, origOperation);
end if

/* Prune elements not satisfying the QoS requirements */
candidateList ← filterList(candidateList, qosConstraints);
/* Rank remaining elements according to the weights */
candidateList ← rankList(candidateList, qosWeights);

while (!empty(candidateList))
    serviceToTry ← removeFirstElement(candidateList));
    tryService(serviceToTry, origAddress, origOperation, payload);
end while

/* control reaches this point if the candidate list has been exhausted, without having either (a)
      successfully invoked some service or (b) faced an application-logic exception. */
return PolicyFault to BPEL engine;
```

Listing 3. Main Request Processing of ASOB.

```
function tryService(serviceToTry, origAddress, origOperation, payload)
    if (! areSyntacticallyEquivalent(origOperation, serviceToTry.operationField))
        payloadXSLT ← fetch XSLT for transforming the original payload to the payload of the
                service to be invoked
        finalPayload ← XSLTransform(payloadXSLT, payload);
    else
        finalPayload ← payload;
    end if
    result ← invoke_bind_WS(serviceToTry, finalPayload);

    if (not timeout_occured)
        if (is_normal_reply(result))
            /* update consumer session memory to reflect the binding */
            SessionMemory.insert(sessionID, origPLink, serviceToTry.PLinkField);
            if (! areSyntacticallyEquivalent(origOperation, serviceToTry.operationField)
                resultXSLT ← fetch XSLT for transforming the result received to the result format
                    of the service originally invoked
                result ← XSLTransform(resultXSLT, result);
            end if
            update_SR_QoS(serviceToTry.operationField, respTime, true, null);
            return result to BPEL engine; /* request has concluded */
        else /* an exception has been raised during the invocation*/
            knownExceptions ← getExceptionsFromWSDL(serviceToTry.WSDL_Location
                    serviceToTry.operationField);
            if ((result.faultCode = “Sender”) or (result.faultCode = “Client”) or (result in
                    knownExceptions))
                /* Known, application-logic exception, or malformed client message.
                These cases cannot be remedied through substitution, thus the exception
                is directly returned to the BPEL engine. */
                update_SR_QoS(serviceToTry.addressField, respTime, true, result.faultCode);
                return result to BPEL engine;
        end if
    else /* timeout occurred */
        update_SR_QoS(serviceToTry.addressField, null, false, “timeout”);
    end if
end function

function update_SR_QoS(WS, respTime, isAvail, errType)
    /* start a new thread to forward QoS information to SR; return immediately */
end function
```

Listing 4. Invocation Routines.

5. Transforming BPEL scenarios to accommodate the service provider-level adaptation strategy

In order to accommodate the service provider-level adaptation strategy, the preprocessor arranges that *all* the operations that are listed within particular BPEL scenario for a specific partner link are sent to the ASOB middleware *before* any operation on the specific partner link is invoked. This can be arranged by collecting all pertinent information from the original BPEL scenario and placing appropriate calls to the *partnerLinkInformation* operation; such invocations can be placed in the proprocessor's output, right after the invocation to the *getSessionId* operation.

Listing 5 illustrates a BPEL scenario excerpt invoking multiple operations on the same service provider, while Listing 6 presents the output of the preprocessor for the particular BPEL scenario excerpt.

```
<!-- set the value of the input parameter for invoking the checkAvailability operation -->
<variable name="availPeriod" type="xsd:string">
<assign name="assign1"> <copy>
    <from><literal>"2009/Dec/12-2009/Dec/20"</literal></from>
                        <to variable="availPeriod"/> </copy></assign>
<!-- set the QoS specification for invoking the checkAvailability operation -->
<assign name="ASOB_QOS_invoke1">
    <copy> <from><literal>"sec:2,0"</literal></from>
                        <to variable="ASOB_QoSconstraints"/> </copy>
    <copy> <from><literal>"perf:1"</literal></from>
                        <to variable="ASOB_QoSweight"/> </copy>
</assign>
<invoke name="ASOB_INV_invoke1" partnerLink="HotelB" portType="HotelB_Port"
                    operation="checkAvailability" inputVariable="availPeriod"
                    outputVariable="availableRooms"/>

<!-- set the value of the input parameter for invoking the getQuote operation -->
<assign name="assign2"> <copy>
    <from><variable="$availableRooms.parameters/roomId"></from>
                        <to><variable="roomToGetQuote"></copy></assign>
<!-- set the QoS specification for invoking the getQuote operation -->
<assign name="ASOB_QOS_invoke2">
    <copy> <from><literal>"sec:5,0"</literal></from>
                        <to variable="ASOB_QoSconstraints"/> </copy>
    <copy> <from><literal>"perf:1"</literal></from>
                        <to variable="ASOB_QoSweight"/> </copy>
</assign>
<invoke name="ASOB_INV_invoke2" partnerLink="HotelB" portType="HotelB_Port"
                    operation="getQuote" inputVariable="roomToGetQuote"
                    outputVariable="priceQuote"/>
```

Listing 5. A BPEL scenario excerpt invoking multiple operations on the same service provider.

```
<!-- retrieve session id -->
<invoke partnerLink="ASOB" portType="ASOBport" operation="getSessionId"
                        outputVariable="ASOBsessionId" />

<!-- invoke the partnerLinkInformation operation of ASOB -->
<assign name="ASOB_PLI_1">
    <copy>
            <from><variable="ASOBsessionId" /></from>
            <to variable="$PartnerLinkOperations.parameters/sessionId“ />
    </copy>
    <copy>
        <from><literal>"HotelB"</literal></from>
        <to variable=”$PartnerLinkOperations.parameters/partnerLinkName“ />
    </copy>
    <copy>
        <from><literal>"http://HotelB/BookRoom"<literal></from>
        <to variable="$PartnerLinkOperations.parameters/endPointAddress" />
    </copy>
<!-- QOS parameters for invocation to checkAvailability -->
    <copy>
        <from><literal>"checkAvailability"</literal>
        <to variable="$PartnerLinkOperations.parameters/QoS[1].opname" />
    </copy>
    <copy>
        <from><literal>"Cons=sec:2,0"</literal></from>
        <to variable="$PartnerLinkOperations.parameters/QoS[1].Cons" /></copy>
    <copy>
        <from><literal>"perf:1"</literal></from>
        <to variable="$PartnerLinkOperations.parameters/QoS[1].Weight" />
    </copy>
<!-- QOS parameters for invocation to getQuote -->
    <copy>
        <from><literal>"getQuote"</literal></from>
        <to variable="$PartnerLinkOperations.parameters/QoS[2].opname" />
    </copy>
    <copy>
        <from><literal>"Cons=sec:5,0"</literal></from>
        <to variable="$PartnerLinkOperations.parameters/QoS[2].Cons" /></copy>
    <copy>
        <from><literal>"perf:1"</literal></from>
        <to variable="$PartnerLinkOperations.parameters/QoS[2].Weight" />
    </copy>
</assign>
<invoke name="ASOB_PLI_invoke_1" partnerLink="ASOB" portType="ASOBport"
                        operation="partnerLinkInformation” inputVariable="PartnerLinkOperations"
                        />

<!-- assignments for availableRooms not listed for brevity -->
<invoke name="ASOB_INV_invoke1" partnerLink="ASOB" portType="ASOBport"
                        operation="proxyInvoke" inputVariable="availPeriod"
                        outputVariable="availableRooms" />
<!-- assignments for roomToGetQuote not listed for brevity -->
<invoke name="ASOB_INV_invoke2" partnerLink="ASOB" portType="ASOBport"
                        operation="proxyInvoke" inputVariable="roomToGetQuote"
                        outputVariable="priceQuote" />
```

Listing 6. The result of preprocessing the BPEL scenario of listing 5.

In listing 6 we can notice that the input parameter for the invocation to the *partnerLinkInformation* operation includes the session id, which will be later used by

the middleware to correlate incoming operation invocations from the same BPEL scenario execution to the information conveyed to it through this invocation. The field *QoS* of the *partnerLinkInformation* operation's input parameter is a repeating element, where each instance contains information for an operation that will be invoked for the partner link designated through the *origServiceAddress* and *origOperation* fields, together with the QoS attribute constraints and weights associated with the particular operation invocation.

6. References

[1] Kareliotis C, Vassilakis C, Rouvas E, Georgiadis P (2009) QoS-Driven Adaptation of BPEL Scenario Execution. Proceedings of ICWS 2009, July 6-10, 2009, Los Angeles, CA, USA.

[2] Kareliotis C, Vassilakis C, Rouvas E, Georgiadis P (2009). QoS-aware Exception Resolution for BPEL Processes: A Middleware-based Framework and Performance Evaluation. International Journal on Web and Grid Services (IJWGS), 2009 - Vol. 5, No.3 pp. 284 - 320