An Ontology Based Analysis of BPEL4WS and WSCI

Vinczellér Gábor, Birger Andersson, and Petia Wohed

Stockholm University/Royal Institute of Technology. Department of Information and Systems Sciences. Forum 100, 164 40 Kista, Sweden

Abstract. In this paper two languages for web services choreography, BPEL4WS and WSCI, are analysed and compared in order to increase our understanding of them. The analysis is based on ideas from the field of ontology. A reference model is used as a top-level ontology and the languages are analysed one by one using the reference model. The result is that the approach of using an ontology based analysis is suitable for gaining understanding of web services languages. The motivation for this work has been the on-going development in the field of web services technologies and the prominent place these technologies have gained for cross-organisational processes and applications integration.

1 Introduction

In spite of all standardization efforts, the Web Service domain still remains heterogenous. However, variety can be considered as a highly valuable property of this domain. Driving development and progress, the question is not how to eliminate variety, but rather how to understand and benefit from it.

One attempt is to lean on the foundational ideas of ontology. According to [1] an ontology is "... a shared understanding of some domain of interest, which is represented as a set of concepts (e.g. entities, attributes, and processes), their definitions and their inter-relationships". By adopting this definition we are agreeing on that firstly, an ontology can effectively be used as a tool for increasing the common understanding of a domain and secondly, that an ontology can be used as reference point when dealing with the heterogeneity within the domain.

The motivation behind this paper is the increased importance of the field of web services. As the number of languages/approaches for Web Services Composition increases, the need of a common and shared understanding of them increases as well. In this paper we are focusing on the comparison of two of these languages, Business Process Execution Language for Web Services (BPEL4WS)[2] and Web Service Composition Interface (WSCI)[3].

In order to be able to perform the analysis and talk about the domain in an unbiased and vendor independent way, we are relying on the reference model presented in [4]. This model was selected as it was particularly worked out for the purposes of comparing business process languages. Called a reference model, the way it was used and the way we will use it in this paper, is similar to
the suggested way an ontology could be used in [1]. Thereby, according to the
taxonomy of ontologies presented in[5], the reference model can be classified as a
top-level ontology and the analysis provided here as an ontology based analysis.

The method for doing this ontology based analysis, is to map the language
concepts onto the concepts of the reference model. First we map one language and
then the other, and the results of the mappings are used to do the comparisons.

Ontology based analysis has earlier been argued for and applied by [6,7] for
a number of purposes, e.g. evaluating the ontological completeness of a language
or for comparing different process modelling languages. Their argumentation
applies naturally in this paper as well.

An ontology based analysis has not yet been provided for the web services
domain. So far the work done has been concentrated on pattern based analysis
and comparison of some of the major standardization efforts proposed within
the domain [8,9]. These analyses focus primarily on evaluating the control-flow
aspects of BPEL4WS and BPML.

The structure of this paper is as follows. In subsection 1.1 we motivate our
choice of analyzed languages. In section 2 we introduce the reference model. In
section 3 we provide the analysis and mapping of BPEL4WS and WSCI into
the reference model. Section 4 contains a brief summarization of the analysis.
Finally, section 5 concludes the paper and gives directions for further research.

1.1 Choice of Languages

According to [10] there are two basic approaches for describing web service col-
laborations; the orchestration approach and the choreography approach. In the
orchestration approach, a service is described from the perspective of one design-
nated partner. This partner acts as a central process, coordinating and governing
the behavior of all collaborating services. The collaborating services need only
to respond to calls from the central process and do not need to be aware of their
different roles in the overall process. In contrast, in the choreography approach, a
process is constituted through the interactions of all the parties. There is no cen-
tral process and the process logic is distributed among the services participating
in a collaboration. The participating services need to be aware of their specific
roles within the collaboration. Web services composition languages can be char-
acterized according to these two collaboration approaches (see fig. 1, adopted
from [10]).

BPEL4WS is a representative of both the choreography and orchestration
approaches. It builds on IBM’s Web Services Flow Language (WSFL)[11] and
Microsoft’s XLANG[12]. Accordingly, it combines the features of a block struc-
tured process language (XLANG) with those of a graph-based process language
(WSFL). BPEL4WS is intended for modelling two types of processes: executable
and abstract processes. An abstract process is a business protocol specifying the
message exchange behavior between different parties without revealing the in-
ternal behavior of any of them and is thereby classified as a choreography ap-
proach. An executable process specifies the execution order between a number
of constituent activities, the partners involved, the messages exchanged between
these partners, and the fault and exception handling mechanisms, i.e. it is a representative for the orchestration approach.

WSCI is a representative of the choreography approach. It is an initiative supported by BEA systems, Intalio, SAP, and SUN Microsystems. WSCI is a choreography language that describes the messages exchanged between web services that participates in a collaboration. In contrast to BPEL4WS, it does not address the definition of executable business processes. These can instead be specified through the Business Process Modelling Language (BPML), which was developed by the Business Process Management Initiative [13]. The first draft of BPML incorporated the WSCI protocol and both languages share the same underlying process execution model and a similar syntax.

As BPEL4WS, WSCI and BPML are considered to be the main standardization efforts within the domain, we concentrate on them. The analysis presented in this paper is on the abstract process of BPEL4WS and WSCI, i.e. representatives of the same collaboration approach. An ontology based analysis for the BPEL4WS executable process and BPML remains to be done.

2 Introduction of the reference model

2.1 Core concepts

All process modeling languages can at least represent the following concepts: activity, state, event, and time point. These concepts are in turn used to form a business process. The intuitive meaning behind these concepts is that an activity is a performance of some work. A state is the situation of some entity, usually modeled as a set of properties. An event is a noteworthy occurrence of a change in some entity. A time point is an instant in time, not further decomposable. Most process modeling languages use these concepts in a well defined and precise way but their relations differ from language to language.

2.2 Complete reference model

Figure 2 shows the reference model with core and additional concepts. The additional ones were added to make the model more useful in a business setting. This
A process is modelled at the knowledge level as a structure of logical dependencies between activities. These activities use one or more resources as input, and produces one or more resources as output. One specific type of resource, the role, is regarded to be responsible for that one or more activities will be performed.

The execution of a process is regarded to be a time-bound series of events, caused by an actor. These events may result in a state change of a resource. An actor who causes an event always has an intention with his/her actions to achieve a specific state for a resource. The state can be either different from the resource's current state, or it can be the same state as the current one.

An actor is regarded as an instantiation of a role, a state is viewed as an instance of a resource, and an event is an instance of an activity. In the latter case, our perception of an activity is through the event that initiates and/or terminates it. An event always occurs at a certain time point or between two time points at a certain location.
3 Comparison of BPEL4WS and WSCI

In this section we focus on comparing three concepts: actor and event from the operations level, and activity dependency from the knowledge level. These concepts were chosen for the following reason: the actor concept is completely different, the event concept is quite different, and the activity dependency concept is almost similar in the two languages. Thus, the concepts were chosen to highlight the level of difference between the concepts. Comparisons between the remaining concepts are reported in [14], but these are made from a different perspective. A brief summarization of the results of all the comparisons is presented in section 4.

3.1 Actor in BPEL4WS

An actor in BPEL4WS is a Web Service. The representation of an actor in BPEL4WS is through the `<partner>` construct. A relationship with a partner is usually peer-to-peer, but it is common to handle more than one conversational relationship with a partner in one business process. One conversational relationship is described by a `<partnerLink>` element. It contains information about the nature of the particular conversation, or more precisely, it shows the roles which the participants act according to in the given interaction. A `<partner>` in this manner is a collection of `<partnerLinks>`, which represent the capabilities required from a business partner. The `<partner>` element is optional as sufficient information is in the `<partnerLink>` element. There is one actor type, the machine actor, represented in BPEL4WS.

3.2 Actor in WSCI

An actor in WSCI is a Web Service, but an explicit language construct that models an actor is missing in WSCI. The language element used to invoke the operations of the partner Web Services is the `<action>` element. This is a very general element. It defines the WSDL PortType and an operation. The peer-to-peer conversational relationships are modeled as WSDL operation pairs (<connect> element in the Global model (<model>)). There is no explicit language element in WSCI that describes and groups the services offered by partners.

3.3 Event in BPEL4WS

An event in BPEL4WS is an indication of a message reception or sending, an alarm expiration, the occurrence of a fault, or the performance of an activity. The following simple event types are represented in BPEL4WS:

- External event: It is realized as a message reception from or a message sending to a partner or as an external fault message.
- Internal event: It can be an internal fault occurrence, an alarm or an internal action.
The following non-exhaustive list of event types are combinations of the simple event types defined in the reference model:

- Time point event + External event: A message reception. For example, reception of a message from a partner.
- Time point event + Internal event: An occurrence of a fault. For example, the effect of a <throw> activity.
- Time duration event + Internal event: Internal action that takes time. For example, the execution of an internal waiting action:

  `<wait for="P0Y0D0T0H0M50S"/>

- Time point event + Pre-activity event + External event: A reception of a message from a partner starts the execution of an activity. For example, reception of a message starts the invocation of a partner’s service:

  `<sequence>
  <receive> <!-- Message reception -->
  <invoke partner="supplier">
  <!-- Partner WS invocation-->
  </invoke>
  </receive>
  </sequence>

- Time point event + Pre-activity event + Internal event: An expiration of a timer starts the execution of an activity. For example, if no message arrives in 30 seconds, a notification message should be sent out.

  `<eventHandlers>
  <onAlarm for="P0Y0D0T0H0M30S">
  <!-- Sending out the notification -->
  <reply partner="customer" name="SendNotification"/>
  </onAlarm>
  </eventHandlers>

- Time point event + Post-activity event + External event: An external fault stops the execution of an activity.
- Time point event + Post-activity event + Internal event: An internal fault stops the execution of an activity.

In BPEL4WS the normal and the abnormal events are distinct. The normal execution of events is described by the activities that form the process and the event handlers (<eventHandlers>). When a normal event occurs (for example a message arrives or an alarm fires) the activity that is associated with this event will be performed concurrently with the normal activities of the current scope. For instance, when implementing a process for order handling, the <eventHandler> mechanism would be applied for taking care of order status requests, which may arrive a varying number of times (including zero) during a normal order execution.
The abnormal events are listed in fault handlers and compensation handlers. When an abnormal event happens, all activity in the current scope will be implicitly terminated and after that the associated activities will be performed. In other words, it is possible to accept normal events concurrently with the activities that executes in the current scope, and it is possible to stop the execution of the scope by throwing or receiving a fault.

Explicit variable assignments are also possible in BPEL4WS.

3.4 Event in WSCI

An event in WSCI indicates a message reception or sending, a fault occurrence or expiration of a timer or a performance of an activity.

The following simple event types are represented in WSCI:

- External event: For example, a normal or a fault message is received from a partner.
- Internal event: For example, the expiration of a timer or the performance of an internal activity.

The following non-exhaustive list of event types are combinations of the simple event types defined in the reference model:

- Time point + External event: A reception of a message. For example, a cancel of a request:

```
<action name="CancelRequest" role="Supplier" operation="tns:CustomerToSupplier/CancelRequest" />
```

- Time point + Internal event: For example, throwing a fault from inside the WSCI description:

```
<fault code="RequestFault"/>
```

- Time duration + Internal event: For example, performing a delay action:

```
<delay property="duration-value" type="duration"/>
```

- Time point event + Pre-activity event + External event: For example, a reception of a message starts an action:

```
<choice>
  <onMessage>
    <!-- reception of a cancel request message ? -->
    <action name="CancelRequest" role="Supplier" operation="tns:CustomerToSupplier/CancelRequest"/>
    <!-- starts a notification sending action-->
    <action name="NotifyCancel" role="Supplier" operation="tns:SupplierToCustomer/NotifyCancel"/>
  </onMessage>
</choice>
```
Time point event + Pre-activity event + Internal event: For example, the expiration of a timer starts an activity:

```xml
<exception> <!-- The expiration of a timer ? -->
<onTimer property="tns:expiryTime" type="duration">
  <!-- starts a compensation activity-->
  <compensate name="CompensateReservation" transaction= "seatReservation"/>
</onTimer>
</exception>
```

In WSCI the normal and abnormal events are sharply separated, but only the executions of the activities that form the process are considered to be normal events (there are no separate declaration for normal events, like the <eventHandlers> in BPEL4WS). When a normal event occurs concurrently with other normal events, the corresponding <process> element is instantiated and runs in parallel.

Abnormal events are listed in exception handlers (<exception>). When an abnormal event occurs, the service waits for the in-progress atomic activity to finish or cancels its work. Subsequently the associated <exception> activity runs. This means that a fault or a message or the expiration of a timer stops the execution of the activities that are associated with the current context.

In WSCI the values of properties can only be changed when a message is sent or received. An internal event cannot change the state of a Web Service.

### 3.5 Activity dependency in BPEL4WS

All types of activity dependencies are present in BPEL4WS. The structured activities of BPEL4WS corresponds to the activity dependency concept in the reference model. These activities define the choreography of the basic activities, thus giving the structure and all possible execution paths of a business process.

- Sequence: The corresponding language construct is the <sequence> construct. It means that all of the activities listed inside this construct must be performed in the given order.
- Selection: The <switch> and <pick> constructs provide this functionality.
- Repetition: The element which supports the repeated execution of activities is the <while> element.
- Fork: The <flow> element is introduced for this purpose. The <flow> element allows the parallel execution of activities, but it is statically defined at design time how many instances of a particular activity that can execute concurrently. It is not possible to dynamically at runtime change the number of running activities.
- Synchronization: Links inside a <flow> block support this feature. This mechanism provides a way to express synchronization dependencies between parallel enabled activities. Every activity can be the source or the target of
a link. Every link has a status which can, when evaluated, enable or disable the execution of the target activity. Source activities can provide transition condition expressions to set the status of the outgoing link. Target activities can have join condition expressions to enable or disable themselves.

3.6 Activity dependency in WSCI

All types of activity dependencies from the reference model are present in WSCI. The complex activities in WSCI, together with <join> and <spawn>, provide the choreography of activities corresponding to the dependency of activities. The control flow of the message exchange is described by these complex elements.

- Sequence: This concept is explicitly given by the <sequence> element. Moreover, activities in a <process> element are performed sequentially by default.
- Selection: Two constructs allow the selection of exactly one action from an action set. One is the element <switch>, the other is the element <choice>.
- Repetition: <foreach>, <until>, <while> are loop constructs that provide repetitions.
- Fork: Activities in an <all> element may execute in parallel. One process could instantiate another process by issuing the <spawn> command. This child process runs parallel to its parent. The <all> element provide the function of parallel execution of activities that were defined at design time. Furthermore, WSCI supports the dynamically spawning of processes at runtime. Since the <process> element is the unit of reuse in WSCI, it is possible to refer to it by its name. It is possible to repeat the <spawn> element several times, for example in a <while> loop where the condition is determined at runtime.
- Synchronization: The element <join> is for this purpose.

4 Overview of the supported concepts

Figure 3 gives an overview of the differences and similarities of BPEL4WS an WSCI according to the reference model. A '+' sign means that the language has a direct support for the current concept. A '-' sign means that there is no realization of the given concept in the language. A '+/-' sign shows that the concept is partially realized by the language element set. Some brief comments about the table are given since there are differences between the realizations of the concepts, even if both languages support them. The actor, event and activity dependency concepts has already been discussed in section 3, so they are not discussed here.

- Role. The role concept is closely connected to the actor concept, according to the reference model. Role in BPEL4WS is clearly defined and explicitly described by the <partnerLinkType> element. It defines the WSDL port types of each role containing operations. The actor who plays the specified
<table>
<thead>
<tr>
<th>Reference concept</th>
<th>BPEL4WS</th>
<th>WSCI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Who</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actor</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Role</td>
<td>&lt;partnerLinkType&gt; element represents this concept.</td>
<td>The &lt;actor&gt; element has a role attribute, but it is just a reference to a role that is defined outside of WSCI.</td>
</tr>
<tr>
<td><strong>Why</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>State</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Resource</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>What</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Activity</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Temporal dependency</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Activity dependency</td>
<td>+/-</td>
<td>+/-</td>
</tr>
<tr>
<td>Process</td>
<td>+/-</td>
<td>+/-</td>
</tr>
<tr>
<td><strong>How</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time point</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Location</td>
<td>--</td>
<td>+/-</td>
</tr>
</tbody>
</table>

**Fig. 3.** Comparison table
role is responsible to perform these operations. In WSCI, the role concept is out of the scope. There is a role attribute of the <action> element, but this is just a reference to an externally defined role. WSCI does not define the responsibilities of the given role.

- Goal. Neither BPEL4WS nor WSCI has implemented the goal concept.
- State. The state concept is realized in a very similar way in both languages. In BPEL4WS the state of a process is described by the actual values of variables. These values can be derived from message parts, from other variables (for example using the <assign> element), from expressions (<from expression="Constant string"/> or from "opaque" sources. In WSCI the state is described by the values of properties. These properties get their value when receiving or sending a message and the values can be derived also from message parts or from expressions. It is possible to reference to a value that is defined inside the Web Service implementation, it is similar to the "opaque" assignment in BPEL4WS.
- Resource. The resource concept is realized almost in the same way in the two languages. Resources are messages and partner Web Services. Only machine and information resources are present.
- Activity. The activity concept is covered by the primitive (basic or atomic) activities in both languages.
- Temporal dependency. Both languages support three kinds of temporal dependencies. The sequential and parallel execution of events provides the "after", "before", and "concurrently" orderings.
- Process. The BPEL4WS abstract process description shows the externally observable behaviour of one participant in a multi-participant business process but there is no concept in BPEL4WS that gives an overall picture of the whole process. Every participant has its own separate BPEL4WS document, which holds no reference to any other document. Understanding the overall process is therefore difficult. In WSCI, the behaviour of every participant is described by its WSCI document, but there is a "Global model" concept (the <model> element) that connects these documents. In this way, it is possible to get an overall picture of the given message exchange that helps to understand the whole business process. So, while BPEL4WS does not implement entirely the process concept, WSCI has a "Global model" concept that corresponds to the process concept of the reference model.
- Time point. Both languages realize the time point concept with the date-time and duration XML Schema datatype. Basically the time points are for defining the expiration time of timers and for defining waiting time. Both languages support the description of time point and time duration expressions and they also support the definition of absolute as well as reference time, but there are differences in details. However, WSCI has a more sophisticated representation of the reference time, as it allows the start point of a reference time definition to be set to another time point than the one from which it starts to execute. For instance, the start of a reference time can be set to the start or to the end of an activity (e.g., $reference="orderrequest@start"$, $reference="orderrequest@end"$) and it is also possible to set the reference
time to any other specific time point. Both languages have the same limitations when using complex expressions to specify time points.

- Location. In the world of Web Services the location information are in WSDL documents. In this manner this concept is out of the scope of BPEL4WS and WSCI. However, since WSDL provides only static information about the locations of Web Services, sometimes it is needed to dynamically locate a partner Web Service at runtime. The standardization bodies behind both specifications consider this an important issue, but at this moment only the WSCI specification implements the feature by the "dynamic participation" concept. So, while BPEL4WS just aims to support the dynamic location of Web Services, WSCI has already provided an extension for this purpose.

4.1 Missing concepts

There are two concepts, message correlation and transactional/compensational behaviour, which are specific to the domain. They are implemented in both of the languages and not covered by the reference model. Since the reference model is extensible, it is possible to add these missing concepts. Adding them would specialize the model and bring it closer to the level of an application ontology [5].

- Message correlation. In a long running stateful business interaction participants send and wait for messages or other events. They exist for a long time and they have to keep their state (e.g. which messages were sent or received). When a Web Service holds multiple conversations with one or multiple partners, then every single conversation has to keep its own state. Thus there are multiple instances of the given Web Service running concurrently. When a message is received by the service, it has to be routed to the appropriate instance of the service. The identification mechanism used for this purpose is called message correlation. When a message is received, some of its contents can be used as instance identification information (for example, customer id). When a Web Service instantiates, it keeps this identification information and every message with this identifier will be routed to this Web Service instance. This is the basic idea behind the message correlation concept of both BPEL4WS and WSCI where the <correlation> and <correlationSet> elements provide this functionality.

In a long running stateful business process it is common to decompose the whole process into smaller, often nested units of works. These units of works can be used as individual parts of work and they are often associated to transactions. Before we go into details, two concepts need to be introduced. When a transaction is executing (not finished yet) and a fault occurs (during execution) then the already completed parts of the transaction must be undone. This is called cancellation. When a transaction has completed (finished its entire work), sometimes it is needed to undo its effects. This is called compensation. There are only small differences in the realization of the transactional/compensational behaviour concept in the two languages.
Transactional/compensational behaviour in BPEL4WS: A unit of work in BPEL4WS is defined as a <scope>. One <scope> may contain several activities. When these activities complete normally, the <scope> is considered as normally completed. Otherwise, if a fault occurs, the scope stops executing the remaining activities. According to the BPEL4WS specification, faults are always treated as a "reverse work" that aims to undo the effects of the completed activities that are enclosed in the current <scope>. In this way faults provide the description of the cancellation mechanism. When a <scope> completes normally, but later its effects are needed to be undone, BPEL4WS provides the compensation mechanism for this purpose. A <scope> may define a <compensationHandler> to describe the compensation mechanism.

Transactional/compensational behaviour in WSCI: A unit of work in WSCI is an activity set. It contains several activities and their context. It is possible to propagate the transactional behaviour of the current activity set. WSCI models two kinds of transactions. One is an atomic transaction the other is the open nested transaction. The type attribute of the <transaction> element shows explicitly the transactional behaviour of the current activity set. The cancellation mechanism is described by the <exception> element, that lists the activities that will be executed during a cancellation of an activity set. The description of the compensation mechanism in WSCI is connected to transactions. A <transaction> element contains the <compensation> element that shows how to compensate a transaction that was normally completed before.

5 Summary and Future Research

We have in this paper systematically analyzed and compared two competing languages for Web Services Composition using an ontology based analysis method. The languages were the BPEL4WS abstract process and the WSCI. The analysis aimed at highlighting the similarities and the differences between these languages. The results of the comparison is that although there are similarities there are also differences between them. While some differences were obvious, e.g. the absence of an actor concept in WSCI, others, like the different uses of the event concept, were subtle and the analysis method proved beneficial for their identification.

We have presented and used a reference model for doing our comparisons. Using the reference model facilitated the reading and understanding of the language specifications. Searching for correspondences to high level concepts like activities or resources, enabled us to be systematic when reading the specifications. The pronoun ("who", "when", . . . ) view of the model was also helpful to gain understanding of the specifications. These positive results indicate that using a reference model when analyzing languages, facilitates the learning and understanding of languages and reduces the time to do so. This is beneficial when coping with a large variety of languages.
We found the reference model adequate for doing a high level comparison of the languages. To make finer comparisons the reference model needs to be extended with domain specific concepts, e.g., concepts for capturing compensation handling and message correlation. Extensions like this would specialize the reference model making it a domain/task ontology as defined in [5]. Domain/task ontological analysis would thereby provide a natural continuation of the work reported here.

Another line for further work, aiming at reducing ambiguity and increase clarity, is to formalize the reference model. Two interesting issues in this direction would be to: a) investigate to what extent the FRISCO framework[15] intersect with the reference model and the possibility of using in as a point of departure for the formalization and b) to compare the formalized reference model with the Bunge-Wand-Weber ontology[16].

References