A standard-driven implementation of WS-BPEL 2.0

Tim Hallwyl
University of Copenhagen
hallwyl@diku.dk

Fritz Henglein
University of Copenhagen
henglein@diku.dk

Thomas Hildebrandt
IT University of Copenhagen
hilde@itu.dk

ABSTRACT

We present a systematic study of the WS-BPEL 2.0 standard based on two complementary methods: the process of constructing a new high-level WS-BPEL implementation driven by the structure of the standard, and an empirical evaluation of existing interpretations of the standard reflected in five widely available WS-BPEL-implementations, both commercial and open source.

In doing so we uncover a number of new ambiguities. Most notably, WS-BPEL's integration of XPath 1.0, the data access component of WS-BPEL, turns out to be inconsistent with the XPath standard itself, which is evidenced by substantially differing results produced by existing implementations on test cases constructed to exercise their interpretation.

The core concepts in WS-BPEL have been formalized and analyzed successfully previously. Our choice to study the standard by constructing a high-level, standard-driven implementation rather than an abstract, mathematical formalization has made it feasible to cover the complete standard, notably the integration with XPath. Given WS-BPEL's design goal of being platform-independent the inconsistencies are arguably a serious concern since they cannot be attributed to the quality of any particular implementation.

Categories and Subject Descriptors

D.2.12 [**Software Engineering**]: Interoperability; D.3.2 [**Programming Languages**]: Language Classifications

Keywords

OASIS, WS-BPEL, XPath, standard-driven

1. INTRODUCTION

The *Web Services Business Process Execution Language Version 2.0 (WS-BPEL)* [1] is a language for the specification of executable and abstract business processes. It exports and imports functionality using Web Service interfaces exclusively to ensure that processes be reusable, deployable "in different ways and in different scenarios, while maintaining a uniform application-level behavior across all of them." [1, Section 1]. In particular, WS-BPEL is to ensure *portability* across different WS-BPEL execution engines, both existing and future ones.

For this reason, WS-BPEL is based on the *Extensible Markup Language (XML)* [2], which is designed to provide portability through a platform-neutral encoding of semi-structured data, and a number of web service standards that build on XML. The *WS-BPEL specification* consists of a number of XML schemas and namespaces defining the syntax of the language; and a *principal prose document* [1], which we henceforth informally refer to as the *(WS-BPEL) standard*, specifying its semantics.

The standard, being written in natural language, carries the obvious danger of containing ambiguities, be it inconsistent requirements or underspecifications. In the first category we include situations where a literal reading in the standard may be clear enough, but ostensibly inconsistent with underlying intentions.

1.1 Goal and method

Our goal has been to develop an implementation-driven method that scales to the complete WS-BPEL standard for identifying relevant ambiguities in WS-BPEL, complementing the analyses of WS-BPEL's process-theoretic core based on semantic formalizations.

But what is an ambiguity in *practice*? What may be a literal underspecification may have a canonical implied resolution. Likewise, what may be *a priori* logically inconsistent may have a commonly accepted resolution in practice, as is common in legal reasoning.

We can approach this question empirically: If different WS-BPEL engines exhibit different observable behavior this may indicate a potential ambiguity in the standard. But how do we find test cases that bring forth such differences? Not only are they bound to be difficult to find in practice where WS-BPEL processes typically execute only on a particular platform. A difference of behavior under different WS-BPEL engines may also be due to a buggy or incomplete WS-BPEL implementation or to an intended implementation freedom in the standard, neither of which the standard can be blamed for.

For this reason we approach our goal using two complementary methods: a *standard-driven* implementation of a WS-BPEL execution engine called *Beepell* for the *complete* WS-BPEL standard, including full process interaction with partners and implementation of referenced standards; and

SAC'10 March 22-26, 2010, Sierre, Switzerland.

a comparative evaluation of the interpretation of test cases exercising potential ambiguities by existing WS-BPEL engines.

Our implementation is standard-driven in the sense that each part of the standard is sought to be reflected as directly and at as high a level as possible in the source code. This approach is inspired by Karpf's *Isomorphism Principle* for formalization of legal systems [3, 4].

The purpose of developing an implementation whose design follows the structure of the standard is two-fold: It is to drive the systematic analysis of the standard so as to uncover potential inconsistencies and underspecifications; and it is to make the implementation and the standard easily interrelatable.

Once a potential ambiguity is located in this process, a paradigmatic test case is constructed and submitted to existing WS-BPEL engines. If they produce different results we take this as supporting evidence that we have uncovered a practically relevant ambiguity.

Since we use the engines to analyze the standard, not to compare them with respect to each other for other purposes such as completeness, performance, integration support, etc., it is not significant which particular versions we use. A difference in behavior is supporting evidence for a potential ambiguity in the standard, even if implementors subsequently communicate with each other and agree on a common resolution.

1.2 Contributions

The key features regarding the control flow of WS-BPEL processes are already well investigated using formalizations based on Abstract State Machines [5, 6], Petri nets [7] and process calculus and labelled transition semantics [8]. Mapping WS-BPEL to an abstract, mathematical model is very valuable, since it provides an unambiguous definition of the semantics amendable to formal analysis. However, the abstraction level of the formalizations inevitably leads to omissions or deviations from the standard. Our choice to use a high-level, standard-driven implementation of the WS-BPEL standard rather than an abstract, mathematical formalization made it feasible to go beyond the core of WS-BPEL. In particular, the use of existing rich libraries and software components facilitates scaling the analysis to include referenced standards that have previously been abstracted away or modeled independently of the standard. In this fashion we uncovered a series of problems primarily related with the standard's treatment of XPath without specifically targeting XPath. These are summarized in Section 2. A full description of Beepell and the issues identified are given in the first author's Master's thesis [9]. The source code for the Beepell implementation is available at http://sf.net/projects/ws-bpel/.

1.3 Standard Driven

Being standard driven means that the concepts described in the WS-BPEL specification are easily recognized in the implementation. For example, the specification tells us that "each `<variable>` is declared within a `<scope>` and is said to belong to that scope". If we look at the implementation, we will find a Scope class holding a hash table of named Variable objects. The specification explains that "variable access follows common lexical scoping rules". The Scope class has a method `getVariable(String name)` that first looks in the Scope object's own hash table of Variable objects—and if not found here it calls `getVariable` on the parent Scope object.

Other examples are the implementation of *activities*, for example the Sequence-activity that "contains one or more activities that are performed sequentially". The implementation has a Sequence-class with a List of child activities and a `run` method that iterates though the list executing the children one by one. The source code for the Sequence-activity is outlined in the listing below. Only the constructor, the package declaration and imports are omitted.

```
public class Sequence
             extends AbstractStructuredActivity {

    private final List<Activity> activities;
    ...
    protected synchronized void run() {
      for (Activity activity : activities) {
        if (this.getState() == TERMINATING)
          return;

        execute(activity);
      }
   }
}
```

2. RESULTS

We present five issues, each with a presentation of related requirements, analysis of the issue, the resolution we implemented and a test case exploring the other implementations.

2.1 Issue 1: XPath Context

The WS-BPEL standard prescribes that XPath queries must have either "node-list or object" as the context node. However, XPath accept neither a node-list nor an object as the context node: according to the XPath specification the context node is a single node [10].

In the case of the node-list, the standard requires it to be "a node-list containing a single node" [1, Section 8.2.6]. This issue is easy to overcome, using the single node in the list as the context node.

> "If the type is a simple type, the context node MUST point to the XPath object specified in section 8.2.2." [1, Section 8.2.6]

The quote above tells us that the standard insists on using an XPath object as the context node when a query on a simple typed 'variable property alias' is evaluated[1]. This is clearly not consistent with the XPath specification. The question is how to implement an impossible requirement.

We constructed a test case in an attempt to reveal how other implementations go about this. But it is difficult to explore because we cannot select the context node as an object. We try to explore it with an assignment operation, copying from a `xsd:boolean` variable using a query: 'boolean(.)' where the dot selects the current node.

If the Boolean variable is passed as an XPath object, then it should return the Boolean value of the variable: *false*. If

[1]A variable property alias is used to refer to a value within a variable of an particular type (structure)

Table 1: Types of expressions, their return type and applied conversion

WS-BPEL Expression Type	Return Type	Conversion
Boolean expressions	xsd:boolean	boolean(object)
Deadline expressions	xsd:date and xsd:dateTime	string(object)
Duration expressions	xsd:duration	string(object)
Unsigned Integer expressions	xsd:unsignedInt	number(object)
General expressions	any	none

Table 2: Attempt to use an object as the context node

Implementation	Result
Beepell	true
Apache ODE	false
JBoss jbpm	failed
Sun BPEL SE (GlassFish)	false
Active BPEL	false
BEA AquaLogic SOA Suite	failed

it is passed as a node, the Boolean value *true* is expected, disregarding the variable's value. The last option is to fail.

Table 2 shows the results of this test case: Three of the implementations seem to have solved this issue in some way while two fail. Our own implementation, which is listed as 'Beepell' simply passes the value as a text node.

2.2 Issue 2: Return Values

Table 1 shows the types of XPath expressions that are used in WS-BPEL. The return types listed are "conforming" types. As XPath is not aware of XML Schema types, the value returned is only required to *conform* with the specified type. A deadline expression, for example, returns a Text node conforming with `xsd:date`.

All typed WS-BPEL expressions return a sequence of character information items (CII) in the InfoSet model, equivalent to a Text node in the Document Object Model (DOM). However, WS-BPEL requires specific conversion methods to be applied, as listed in the 'Conversion' column of Table 1.

When selecting a Boolean value—for example from the `xsd:boolean` typed 'active' attribute in the listing below—then XPath selects a Text node.

```
<foo:account active="false">
  <foo:name>Leased Equipment</foo:name>
</foo:account>
```

Applying the `boolean` XPath function is required by WS-BPEL because XPath does not know of the XML Schema types. This will, however, convert a selected value of 'false' into a Boolean value *true*.

The `boolean` function does, according to the XPath specification, convert all non-zero length Text nodes or String objects to *true*. Only empty strings are converted to *false*.

Obviously, this has a significant impact on how the language can be used. The question is if implementors choose to implement another—more practical—conversion.

To explore this, we try evaluating a Boolean expression that evaluates to an XPath String object of value 'false'. Using the conversion methods required by the WS-BPEL standard, this should return the Boolean value *true*. Table 3 shows the results.

Table 3: Implicit conversion of Boolean expressions

Implementation	Result
Beepell	true
Apache ODE	true
JBoss jbpm	false
Sun BPEL SE (GlassFish)	true
Active BPEL	false
BEA AquaLogic SOA Suite	false

As it turns out that half of the tested implementations return *true* and the other half *false*, we are forced to conclude that portability of process descriptions is severely reduced by this issue.

The implementations that return *true* are consistent with the WS-BPEL standard and the definition of the XPath `boolean` function. These implementations make it rather difficult to base Boolean expressions on `xsd:boolean` values selected within variables, however.

2.3 Issue 3: getVariableProperty

The WS-BPEL standard defines an XPath function `getVariableProperty`, for use in expressions, as follows:

> The return value of this function is calculated by applying the appropriate `<vprop:propertyAlias>` for the requested property to the current value of the submitted variable. [1, Section 8.3]

This gives a good idea of the intention of the function, but leaves us with the understanding that the XPath function should return the same as when a variable property is referred to directly in an assignment: a single information item other than CII, or a sequence of zero or more CIIs.

There is nothing invalid about returning a sequence of CIIs; in XPath this will be mapped to a String object. It does, however, impose some problems having all simple type values represented as XPath String objects. This is especially clear in simple Boolean expressions, as shown in the example below:

```
<bpel:if>
  <condition>
    bpel:getVariableProperty('shipRequest',
                             'props:shipComplete')
  </condition>
  ...
</bpel:if>
```

A conversion using the XPath `boolean` function will implicitly be added [1, Section 8.3]. However, if the function returns a String object, for example 'false', the conversion will

Table 4: Retrieve a Boolean value 'false' using getVariableProperty

Implementation	Result
Beepell	true
Apache ODE	true
JBoss jbpm	failed
Sun BPEL SE (GlassFish)	N/A
Active BPEL	true
BEA AquaLogic SOA Suite	failed

Table 5: Are XPath functions allowed in join conditions

Implementation	Result
Beepell	Allowed
Apache ODE	Allowed
JBoss jbpm	Allowed
Sun BPEL SE (GlassFish)	N/A
Active BPEL	Rejected
BEA AquaLogic SOA Suite	N/A

return *true* — a string is *true* if and only if its length is non-zero. Thus, the expression in the above example will always return *true*.

There are two hints in the standard that suggest an implicit conversion of variable properties into proper XPath objects. The first hint is that the method signature returns an object. The second hint is that the example above is from the general examples section in the WS-BPEL standard [1, Section 15.1.3]. The `props:shipComplete` property is of `xsd:boolean` type.

We suspect that the same conversion method as used with variables in XPath expressions [1, Section 8.2.2], is intended — though it is not specified.

In a test case we use the `getVariableProperty` function to retrieve a *false* Boolean value, as part of a Boolean expression. The results are listed in Table 4. Besides our own implementation, it seems only Apache ODE and Active BPEL support this function. They both return *true* meaning that they agree with us on the strict interpretation that `getVariableProperty` does not apply any implicit conversion.

2.4 Issue 4: Use of Functions in Join Conditions

According to the section on static analysis in the WS-BPEL standard, a join condition expression must be constructed using only Boolean operators and the status values of the activity's incoming links [1, SA00073]. Section 8.2.5 in the standard, on the XPath context for join conditions, does however allow access to XPath functions.

In our implementation, we allow XPath functions in join conditions. However, this is only because we do not implement static analysis.

We constructed a test case using core XPath functions within a join condition to investigate if other implementations are allowing this. Table 5 below shows the results: The BEA and Sun implementations do not support synchronization links. Active BPEL rejected the process description at deployment while Apache and JBoss allowed it.

2.5 Issue 5: 'Identical' Values in Correlation Sets

A *correlation set* is initiated with values from a message. It is then used to route incoming messages to the right process instance.

The correlation semantics is based on two constraints that must be observed: the initiation and consistency constraints. The consistency constraint is defined as follows:

> After a correlation set is initiated, the values of the properties for a correlation set must be identical for all the messages in all the operations that carry the correlation set [1, Section 9.2]

On the one hand it is easy to understand the intentions, but on the other hand the concept of 'identical value' is a bit vague.

Because correlation only uses simple type properties, the result cannot be anything but a Text node (a sequence of CIIs in the Infoset model).

Since it is the same property that is applied to different messages, the type is per definition identical. But the WS-BPEL standard does not offer any guidelines on how to decide if two values are identical. Since Infoset is used to explain the concept of 'value', we could assume that identical simple type values have identical CII sequences.

However, comparing the textual values directly may not be the intention. For example, a property of type `xsd:decimal` may retrieve two values using different aliases, such as '42.10' and '42.1'. Are the values identical?

The standard does in general represent a simple type value as an InfoSet CII sequence, and in that sense the answer must be 'no'—they do not have identical Infoset CII sequences. However, it seems reasonable to expect that decimal simple type values are compared by their numerical value, especially since the properties are typed.

This issue regards a wide range of data types. Another example is the Boolean value *false* that may be expressed as either 'false' or '0'. In our implementation we follow the implications of the WS-BPEL standard and compare the textual values, ignoring the data type altogether.

Testing how this is implemented by others is a time consuming task because we need to manipulate the messages that are meant to correlate. For this reason we have set up only one test case comparing our implementation (Beepell) with ActiveBPEL.

Table 6 shows the test results of an attempt to correlate with two properties, a decimal value '42.10' and a Boolean value 'false'. 'Yes' means that a message with the literal values specified was routed to the instance 'No' means that the message did match the correlation set.

While our implementation (Beepell) strictly requires identical CII sequences, ActiveBPEL does seem to compare Boolean values on their value rather than their representation, but not with the decimal numbers. We have also tried with 'true' instead of 'false', and this gave the same results. The results emphasize the need for canonicalization of correlation values when designing processes.

3. DISCUSSION

We have presented a systematic semantic study of the WS-BPEL standard that goes beyond the core of WS-BPEL and includes other standards referenced in the WS-BPEL standard, notably XPath. The process (not just the result) of

Table 6: Attempt to correlate messages with 'identical' values.

Decimal	Boolean	Beepell	ActiveBPEL
42.1	false	No	No
42.1	0	No	No
42.10	false	Yes	Yes
42.10	0	No	Yes
42.100	false	No	No
42.100	0	No	No

constructing the implementation from the standard helped identifying potential ambiguities in the standard outside the process-theoretic core of WS-BPEL. By performing a comparative analysis, constructing and applying a set of test cases to our implementation and five widely available implementations of WS-BPEL, we have provided empirical evidence that the issues identified indeed give rise to inconsistencies between the different implementations in practice.

A key finding is that the source of what we consider to be the most significant inconsistencies uncovered here is attributable to the XPath and WS-BPEL standards having different data models and type systems for XML documents and the lack of an explicit consistent mapping of one data model into the other.

The choice of using a high-level Java implementation has made it feasible to cover the entire (mandatory part of the) language specification including referenced standards such as XPath, which have been left out of previous formalizations based on mathematical models. By using Java as our "formalization" language we forfeit *mathematical* reasoning. Also, there may be ambiguities or under-specifications inherited from Java. For instance, we rely on Java thread scheduling for handling execution of concurrent flows. We are not aware of any mathematical model language which at the same time supports a high-level, standard-driven representation of the *complete* standard and also allows execution of processes, however. An intermediate step towards this would be a mathematical, executable model language with support for integration of XPath as an external component. As part of the CosmoBiz project[2] we are presently working along this direction, replacing parts of the Java implementation with an executable formalization based on the bigraph formalism [11] and implementing an engine for executing bigraphs that allow integration of XPath as expression language. As demonstrated in [12, 13] bigraphs are able to represent XML data and WS-BPEL processes very directly, making the formalism promising for a standard-driven formalization.

Acknowledgements

This work has been partially supported by the Danish National Advanced Technology Foundation under Project *3gERP* (3d generation Enterprise Resource Planning systems, 3gERP.org), by the Danish Research Council for Technology and Production under Project *Cosmobiz* (Grant no. 274-06-0415, cosmobiz.org) and by Sirius IT (siriusit.com).

[2]See http://www.cosmobiz.org.

4. REFERENCES

[1] 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. Technical report, OASIS Web Services Business Process Execution Language (WS-BPEL) TC (April 2007)

[2] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F., Cowan, J.: Extensible Markup Language (XML) 1.1 (Second Edition). W3C Recommendation, W3C (1999)

[3] Karpf, J.: Quality Assurance of Legal Expert Systems. Jurimatics **8** (1989) Copenhagen Business School.

[4] Bench-Capon, T.J.M., Coenen, F.P.: Isomorphism and legal knowledge based systems. Artificial Intelligence and Law **1** (1992) 65–86

[5] Fahland, D.: Complete Abstract Operational Semantics for the Web Service Business Process Execution Language. Informatik-Berichte 190, Humboldt-Universität zu Berlin (September 2005)

[6] Fahland, D., Reisig, W.: ASM-based semantics for BPEL: The negative Control Flow. In Beauquier, D., Börger, E., Slissenko, A., eds.: Proceedings of the 12th International Workshop on Abstract State Machines (ASM'05), Paris XII (March 2005) 131–151

[7] Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0 and its compiler BPEL2oWFN. Informatik-Berichte 212, Humboldt-Universität zu Berlin (August 2007)

[8] Lapadula, A., Pugliese, R., Tiezzi, F.: A formal account of WS-BPEL. In: Proceedings of the 10th International Conference on Coordination Models and Languages (COORDINATION'08). Volume 5052 of Lecture Notes in Computer Science (LNCS)., Springer (2008) 199–215

[9] Hallwyl, T.: Evaluating the BPEL standard specification. Master's thesis, Department of Computer Science, University of Copenhagen (May 2008) DIKU TOPPS Technical Report D-609.

[10] Clark, J., DeRose, S.: XML Path Language (XPath) Version 1.0. W3C Recommendation, W3C (1999)

[11] Milner, R.: The Space and Motion of Communicating Agents. Cambridge University Press (March 2009)

[12] Hildebrandt, T., Niss, H., Olsen, M.: Formalising business process execution with bigraphs and Reactive XML. In: COORDINATION'06. Volume 4038 of LNCS., Springer (2006) 113–129

[13] Bundgaard, M., Glenstrup, A.J., Hildebrandt, T., Højsgaard, E., Niss, H.: Formalizing higher-order mobile embedded business processes with binding bigraphs. In: Proceedings of the 10th International Conference on Coordination Models and Languages (COORDINATION'08). Volume 5052 of Lecture Notes in Computer Science (LNCS)., Springer (2008) 83–99