Detecting Portability Issues in Model-Driven BPEL Mappings

Jörg Lenhard and Guido Wirtz
Distributed Systems Group, University of Bamberg
An der Weberei 5, 96047 Bamberg, Germany
E-mail: {joerg.lenhard | guido.wirtz}@uni-bamberg.de

Abstract

Service orchestration languages, like the Web Services Business Process Execution Language (BPEL), have been frequently used to provide an implementation platform for model-driven development approaches. As avoidance of vendor lock-in and portability of process definitions are central aims of BPEL, most approaches claim to support a large set of different runtime environments. But, even though today various runtimes for BPEL are available, every runtime implements a different language subset, thus hampering portability. Our idea is to improve this situation by using techniques, the Web Services Interoperability Organization (WS-I) has used to improve services interoperability. We describe a portability profile for BPEL that can detect portability issues in process definitions. Using this profile, we evaluate the portability of BPEL mappings used in several model-driven development approaches.

Keywords: SOA, BPEL, portability, profile, mapping

1. Introduction

Software portability is the ability to move software from one runtime platform to another without having to rewrite it fully or in part. It is a central characteristic of software quality [5]. Next to interoperability, it is also one of the core aims of service-oriented processes [6]. Being an OASIS standard that describes an open and platform-independent XML language for programming executable processes based on services, BPEL [8] is a main driver of services portability. For this reason, it is used as execution language in various model-driven mappings, such as [10, 12]. In these mappings, higher level and generally more abstract process or domain models, such as business process models, are transformed into executable BPEL code in the form of one or more process definitions. Through their property of being portable, BPEL process definitions enable these approaches to work on many different systems.

Shared standards are the basis for interoperability and portability of applications that run on heterogeneous platforms. However, these characteristics are difficult to achieve. Various case studies [7, 13] show that interoperability of heterogeneous systems still is limited. In the Web Services ecosystem, the WS-I is established as the main driver of interoperability. Its working mode is to define profiles which are standard documents that describe restrictions and assertions on existing standards, such as the *basic profile 2.0* [14]. Such assertions delimit the expressiveness of a standard or clarify the interpretation of it with the aim of making implementations more likely to interoperate. Although this approach does not guarantee interoperability, it relieves the problem and is accepted in practice.

Whereas the WS-I profiles deal with enforcing interoperability, portability has been neglected so far. The idea we present in this paper is to apply the concept of profiles from the area of *services interoperability* to the area of *services portability*. We address BPEL, as it is designed to build portable and executable programs. In previous work [4], we could show that current runtimes for BPEL (i.e., BPEL *engines*) implement different parts of the specification and the portability of process definitions among them is problematic. Here, we present a portability profile for BPEL that works similar to WS-I profiles [14] and addresses common portability issues. The issues are identified through an extensive benchmark of current engines. Using the profile, we can detect portability limitations in approaches that use a model-driven mapping to BPEL [9, 10, 12].

In the next section, we outline related work and in the following present the *BPEL Portability Profile*, focusing on the test assertions defined by it. Thereafter we discuss issues in model-driven approaches using BPEL. Finally, we summarize the paper and present areas of future work.

2. Related Work

Related work separates in approaches that try to mitigate portability issues in BPEL and work on BPEL mappings.

[1] also identifies the problem of portability among different BPEL engines which is ascribed to the informality of the specification and resulting ambiguities. The authors address the problem by defining the language B*lite* which en-

hances BPEL with a formal definition and refines the behavior of problematic constructs. B*lite* programs can be compiled to executable BPEL code for a specific runtime [1]. Such an approach can preempt portability errors, but requires the usage and understanding of another language ontop of BPEL, which in the case of a formal notation can be hard to learn. We do not define a new language, but provide assertions based on empirical data that can be used to detect portability issues in existing code.

Being an open international standard, BPEL has been used as target language in numerous model-driven mappings. A well-known one is part of the Business Process Model and Notation (BPMN) [9]. This is a standard developed by the OMG for modeling and visualizing several perspectives of business processes. It defines a notation for process models and a mapping of these process models to executable BPEL code. Since BPMN 2.0, this mapping is updated to the most recent revision of BPEL [9, pp. 445–474]. Before that, academic approaches tried to map BPMN 1.0 to BPEL 2.0 [10].

In service-oriented computing, a notable distinction of process models is made between orchestration and choreography models [11]. Choreography models describe a global view on a distributed process between multiple autonomous parties. In a model-driven setting, a local executable process for each of the different parties can automatically be derived from the global choreography. Such a local service-oriented process is called an orchestration. An example of a choreography to BPEL mapping is given in [12] based on the ebXML Business Process Specification Schema (ebXML BPSS).

3. The BPEL Portability Profile

The BPEL portability profile follows the scheme of the WS-I profiles [14]. It defines test assertions that can be seen as invariants of the standard specification [8]. Each test assertion defines a normative requirement of the profile that should be adhered to, if the goal of the profile (in this case portability of the code) is to be reached.

The assertions for the profile are based on data of an analysis of the BPEL conformance of a large set of BPEL engines [4]. The conformance assessment was performed using the tool *betsy*[1] [3]. The benchmark in this paper comprises seven engines: ActiveBPEL, bpel-g, Apache ODE, OpenESB BPEL Service Engine, Easy BPEL, Orchestra, and the engine of a leading middleware vendor whose name we cannot disclose for licensing reasons. The conformance test produces a data set indicating the support for every feature of the language specification by each engine. Based on this, it is possible to calculate the relative proportion to which each feature of the language is supported by today's runtimes. For each feature that is not fully supported by all engines and the usage of which might consequently result in a portability issue, a test assertion can be derived. These test assertions in turn enable the identification of portability issues in process definitions.

Test Assertions: Test assertions follow a predefined structure and contain several decisive elements. For the WS-I profiles and tools, not all test assertions are testable, because for several assertions the required information cannot be collected with the tools. The assertions of this profile base on the structure of process definitions only and as a consequence all assertions are testable. We defined and implemented a test assertion for every feature of the specification that was not supported by all engines under test. In total, this amounts to almost 70 assertions, each checking for the usage of specific BPEL elements or activities, their combination, or their configuration. Assertions that check for similar aspects, such as the usage of `toParts` and `fromParts`, are grouped together. The testing is based on XPath 2.0 expressions, similar to the mechanism used by WS-I profiles. Each assertion defines such an expression which selects all elements in the code that violate the criterion the assertion is checking. Based on the amount of engines that do not support a given feature, a test assertion can be classified according to a level of severity. The lower the amount of engines that support a feature, the more of a barrier to portability this feature will be.

The assertions are specified in the XML format for assertions defined by the WS-I. Crucial elements are the *target* and *predicate* which are both XPath 2.0 expressions. The target selects all elements in a process definition that violate the test assertion. This is necessary for being able to produce a list of all violations of an assertion when using the profile. The predicate determines whether on evaluation the assertion as a whole is counted as passed, which is the case if there are no elements found by the target. The *diagnostic* part specifies the severity of the test assertion.

Portability Levels: Based on the severity of the test assertions violated, it is possible to classify a process definition into different levels. This classification can be used to discriminate high-quality process definitions in terms of their portability from low-quality ones. We define the portability levels i) *portable*, ii) *widely portable*, iii) *partially portable*, iv) *limited portability*, and v) *nonportable*.

The severity, $Sev(ta)$, of an assertion ta depends on the degree of support of the feature, $S(ta)$, tested by the assertion. If all engines support a feature, it is fully *portable*. If at least 80 % of all engines support the feature, which can be considered an acceptable level of portability [2], it is classified as *widely portable*. If less than 80 %, but more than 50 % support the feature, it is classified as *partially*

[1] Betsy is a conformance testing tool for BPEL. For more information, see the project page: `https://github.com/uniba-dsg/betsy`. Betsy is also used in [4], but here we consider a larger number of engines.

portable. If less than 50 %, but more than one engine support the feature, which here amounts to at least 16 %, it is classified as being of *limited portability*. Finally, if only a single or no engine supports the feature, it is classified as *nonportable*.

$$Sev(ta) = \begin{cases} Portable, & \text{if } S(ta) = 100\% \\ Widely\ Portable, & \text{else if } S(ta) \geq 80\% \\ Partially\ Portable, & \text{else if } S(ta) \geq 50\% \\ Limited\ Portability, & \text{else if } S(ta) \geq 16\% \\ Nonportable, & \text{otherwise} \end{cases}$$

The classification, $Level(p)$, of a mapping or process definition p depends on the severity of the test assertions that it violates. The set V is the set of assertions violated by p and N_V is its size. Effectively, a process definition is assigned to the portability level of the most severe test assertion it violates. It is classified as portable only if no issues could be detected. This means, if all violations that are found concern test assertions of the severity *widely portable*, then the complete process definition is assigned to this level. If there is a single violation of the level *nonportable*, then the complete process definition is classified as *nonportable*.

$$Level(p) = \max_{i=1...N_V} (Sev(ta_i)) \; where \; ta_i \in V$$

The Bpp Tool: We have implemented the assertions and classification in the *bpp* tool[2]. The tool is written in Java and takes fragments of BPEL code, being complete process definitions or not, as input. The test assertions are encoded in it, and can be printed in the schema of a WS-I profile. At runtime, bpp checks each test assertion for the input code and records violations. Finally, the tool produces reports that conform to the WS-I report schema.

4. Evaluation of BPEL Mappings

In the following, we discuss issues in mappings from BPMN 1.0 [10], BPMN 2.0 [9, pp. 445–474], and ebXML BPSS [12] to BPEL.

4.1. BPMN 1.0

A notable contribution in the area of BPMN to BPEL mappings is [10] which focuses on revision 1.0 of BPMN, but claims to map to the up-to-date revision of BPEL. BPMN 1.0 on the other hand maps to an outdated version of BPEL. The mapping in [10] takes the form of BPMN constructs that are translated to fragments of BPEL code and two examples of complete mappings.

[2]For more information and a description on how to use the tool, see the project page: `https://github.com/uniba-dsg/bpp`. The BPEL fragments that underpin the discussion in section 4 are included as well.

Issues: The approach focuses on a control-flow oriented mapping to BPEL and omits several details. Whereas missing namespaces can be easily fixed, a missing declaration on how data handling works (i.e., by referencing variables or using the `parts` syntax) is problematic when it comes to the portability of the mapping. The `parts` syntax for dealing with data in BPEL is rarely supported, so the underspecification of data handling results in a portability issue here. A more severe problem is that the mapping uses elements unknown to BPEL 2.0 to direct the flow of control, as for example the `if-case` elements in the order fulfillment process mapping. These elements seem to be a variation of the BPEL 1.0 `switch-case` activity and their usage renders the mapping nonportable. Issues of minor severity in the mapping are the usage of `links` in the `flow` activity, as for example in the order fulfillment process mapping, which are only partially supported. A minority of engines does not support the `flow` activity, `onAlarm` timeout handlers and `onMessage` handlers when used in a `pick`.

Discussion: The severe issues in this mapping can be fixed by using the syntax defined by BPEL. By enhancing the mapping with a definition of how data items are handled in `invoke` activities, also this issue can be tackled. The usage of `links` is a crucial aspect of the mapping, so replacing it is likely to be a non-goal, even if it limits its portability.

4.2. BPMN 2.0

In revision 2.0 of BPMN, the mapping to BPEL is updated to the most recent revision of the specification [9, pp. 445–474]. The mapping describes elements of BPMN process diagrams and presents a fragment of BPEL code for each element. We analyzed each code fragment separately and, barring spelling errors in the BPMN specification, could detect portability issues in half of these code fragments. None of them are classified as nonportable, but 25 % of the issues are of limited portability.

Issues: One issue that reduces the overall mapping to limited portability are the fragments for the send and service tasks [9, pp. 448/449] and message end events [9, p. 457]. Similar to the mapping in [10], theses mappings use an `invoke` activity in BPEL, but do omit data flow and, therefore, fail to specify for instance an `inputVariable`. Depending on the Web Service operation that is invoked, this is legal in BPEL, but omiting variables is not supported by a majority of the engines. The data associations mapping [9, pp. 467/468] readdresses this issue, but uses the `from-` and `toParts` syntax of BPEL to assign variables instead of `assign` activities, which is also rarely supported. Areas of partial portability in the mapping are compensation intermediate events and compensation end events [9, pp. 457/458] which both use the `compensate` or the `compensateScope` activity. Error end events [9, pp.

Table 1. Summary of Portability Issues

Mapping	Classification	Major Issues	Minor Issues
BPMN 1.0 [10]	nonportable	non-BPEL elements, data handling	timeout handling, links
BPMN 2.0 [9]	limited portability	data handling, usage of parts-syntax	compensation, links, timeout handling
ebXML BPSS [12]	widely portable	–	timeout handling

457/458] are mapped to a `throw`. The interpretation of this activity, in case it is used to terminate a process instance, varies among engines. Another problematic part is the mapping of message handlers [9, p. 452] and message boundary events [9, pp. 458/459] which map to `onMessage` event handlers of a BPEL `scope`. While such event handlers are widely supported when used in `pick` activities, this is not the case when they are attached to a `scope`. Finally, the mapping of error boundary events [9, p. 459], multiple boundary events [9, pp. 460/461], and the inclusive decision pattern [9, p. 463] use `links`, partly in combination with `transitionConditions`, to direct the flow of control which is only of partial portability. Minor issues in the mapping are timeout handlers in event sub-processes [9, p. 452] and the exclusive event-based decision pattern [9, p. 462] which use `onAlarm` event handlers and timer intermediate events [9, p. 456] which rely on the `wait` activity. These timing-related activities are unsupported in a minority of engines. Also, the `forEach` activity, used in the multi-instances mapping [9, p. 455], is not ubiquitous.

Discussion: The main portability problems in the mapping result from data handling, either from not specifying it or from using a syntax that is only of limited portability. The more critical issues could be resolved by using `assign` activities instead of the `parts` syntax in BPEL.

4.3. ebXML BPSS

[12] defines a mapping from the ebXML Business Process Specification Schema to BPEL. The models translated are state-machine-based choreography definitions and BPEL fragments for translating states are presented.

Disussion: Only few issues could be detected in the mapping. No nonportable, limited, or partially portable elements were found, so the overall classification of the approach is *widely portable*. The only issues relate to timeout handling, implemented through `onAlarm` event handlers which are not supported by all, but by a majority of engines.

4.4. Summary

Table 1 summarizes the results from the previous sections. Portability issues could be detected in all mappings and two out of three produce BPEL code that will only be executable on a minority of today's BPEL engines. The most severe issues in the mappings relate to data handling which can be expressed in several different ways in BPEL, of which only a subset is widely supported.

5. Outlook and Future Work

In this paper, we proposed a mechanism for detecting portability issues in BPEL code, applied it to three model-driven development approaches that target the language, and made recommendations on how to fix detected issues.

Future work centers on two areas: Improving the portability profile by gathering more data on BPEL support and complementing the qualitative assessment here with a quantitative one based on formally defined portability metrics.

References

[1] L. Cesari, A. Lapadula, R. Pugliese, and F. Tiezzi. A tool for rapid development of ws-bpel applications. In *ACM SAC, Sierre, Switzerland*, March 22-26 2010.

[2] M. Glinz. A Risk-Based, Value-Oriented Approach to Quality Requirements. *IEEE Computer*, 25(8):34–41, 2008.

[3] S. Harrer and J. Lenhard. Betsy – A BPEL Engine Test System. Bamberger Beiträge zur WI und AI, no. 90, University of Bamberg, July 2012.

[4] S. Harrer, J. Lenhard, and G. Wirtz. BPEL Conformance in Open Source Engines. In *IEEE SOCA*, Taipei, Taiwan, December 17-19 2012. IEEE.

[5] ISO/IEC. *Systems and software engineering – System and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*, 2011. 25010:2011.

[6] R. Khalaf, A. Keller, and F. Leymann. Business processes for Web Services: Principles and applications. *IBM Systems Journal*, 45(2):425–446, 2006.

[7] S. Kolb, J. Lenhard, and G. Wirtz. Bridging the Heterogeneity of Orchestrations - A Petri Net-based Integration of BPEL and Windows Workflow. In *IEEE SOCA*, Taipei, Taiwan, December 17-19 2012. IEEE.

[8] OASIS. *Web Services Business Process Execution Language*, April 2007. v2.0.

[9] OMG. *Business Process Model and Notation (BPMN) Version 2.0*, January 2011.

[10] C. Ouyang, M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, and J. Mendling. From Business Process Models to Process-Oriented Software Systems. *ACM Transactions on Software Engineering and Methodology*, 19(2), 2009.

[11] C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, October 2003.

[12] A. Schönberger, C. Pflügler, and G. Wirtz. Translating Shared State Based ebXML BPSS models to WS-BPEL. *IJBIDM*, 5(4), 2010.

[13] A. Schönberger, J. Schwalb, and G. Wirtz. Interoperability and Functionality of WS-* Implementations. *International Journal of Web Services Research*, 9(3):1–22, 2012.

[14] WS-I. *Basic Profile Version 2.0*, November 2010.