BPMN Conformance in Open Source Engines

Matthias Geiger, Simon Harrer, Jörg Lenhard, Mathias Casar, Andreas Vorndran and Guido Wirtz
Distributed Systems Group
University of Bamberg
Bamberg, Germany
{*matthias.geiger, simon.harrer, joerg.lenhard, guido.wirtz*}*@uni-bamberg.de*

***Abstract*—Service-oriented systems are increasingly implemented in a process-based fashion. Multiple languages for building process-based systems are available today, but the *Business Process Model and Notation* (BPMN) is becoming ubiquitous. With BPMN 2.0 released in 2011, execution semantics were introduced, supporting the definition of executable processes. Nowadays, more and more process engines directly support the execution of BPMN processes. However, the BPMN specification is lengthy and complex. As there are no official tests and no certification authority, it is very likely that engines a) implement only a subset of the language features and b) implement language features differently. In other words, we suspect that engines do not *conform* to the standard, despite the fact that they claim support for it. This prohibits the porting of processes between different BPMN vendors, which is an acclaimed goal of the language. In this paper, we investigate the standard conformance of open source BPMN engines to provide a clear picture of the current state of the implementation of BPMN. We develop a testing approach that allows us to build fully BPMN-compliant tests and automatically execute these tests on different engines. The results demonstrate that state-of-the-art BPMN engines only support a subset of the language. Moreover, they indicate that porting BPMN processes is only feasible when using basic language constructs.**

***Keywords*-BPMN, engine, conformance testing**

I. Introduction

Process-aware information systems [1] are facing increasing adoption in practice. This is particularly relevant for service-oriented systems where processes are frequently used for building service *orchestrations* [2]. Such an orchestration aligns calls to existing services in a process-based manner and thereby provides a value-added service. To foster an optimization of a process or orchestration, it is considered as a best practice to apply the Business Process Management (BPM) lifecycle [3]. This lifecycle increases the value of the services even further by feeding back insights gained during process enactment into the next version of the process.

Many languages for implementing process-based service-oriented systems exist today, but currently the BPMN specification [4] is the most promising. BPMN has been accepted as an ISO standard in revision 2.0.2. It provides a notation for modeling processes of various types, but also for implementing executable processes that can be used for the task of service orchestration. BPMN is even expected to supersede the *Business Process Execution Language* (BPEL) [5], when it comes to the construction of service orchestrations. More and more vendors are currently implementing the specification, resulting in a variety of freely available or commercial modeling tools and engines[1]. BPMN *engines* are able to consume and execute processes provided in the correct format. Since BPMN standardizes the format and semantics of processes, their execution behavior should not differ on different engines. However, this only works if all engines fully implement BPMN in the same manner with respect to semantics, which is an unrealistic assumption. If engines do not fully *conform* to the standard, i.e., if they skip certain parts of it or implement some features in a limited or differing way, the portability of processes can no longer be taken for granted. In this case, strictly speaking, an engine only supports a dialect of BPMN. A lack of standard conformance in the implementations removes the benefits of the standard and defeats the purpose of the standardization in the first place. Since there is no certification authority that checks standard conformance of BPMN implementations, vendors can easily state that they support the standard.

In similar work, we investigated the standard conformance of BPEL engines [6], [7]. Here, we build upon this work and extend it for benchmarking the standard conformance of BPMN engines. We develop a testing approach for this task and, thereby, extract a testing methodology that is independent of a concrete process language. We implement the approach with a testing tool and provide a rich test suite of standard conformant BPMN processes. We use this approach and test suite to assess several freely available BPMN engines. The results show that multiple BPMN features are rarely implemented and, thus, should be treated with care in practice.

The next section discusses related work and delineates this paper from similar studies. Thereafter, we detail our testing approach and methodology in Section III, including a description of the domain model, the testing workflow and testing suite, and the engines under test. Section IV presents the results of the test execution. We interpret the results and discuss threats to validity in Section V. Finally, Section VI ends the paper with a summary and an outlook on future work.

[1]Currently, more than 70 implementers are listed at http://www.bpmn.org/.

II. Related Work

Related work separates in three areas. These are process languages for implementing executable process-based systems in Section II-A, conformance benchmarking and testing of process engines in general in Section II-B, and, the evaluation of the conformance of BPMN processes and environments in particular in Section II-C.

A. Languages for Building Executable Processes

A large variety of process languages is available today. For instance, [8] discusses more than 19 business process modeling languages, which have different areas of application. Especially for executable languages, standard conformance is of paramount importance for avoiding vendor lock-in. For this reason, we do not consider process languages for which the sole implementation is the standard, e.g., the Windows Workflow Foundation[2] (WF) 4.5 or the jBPM Process Definition Language[3] (JPDL). The three most relevant standardized ones are the OASIS standard BPEL 2.0 [5], the WfMC standard *XML Process Definition Language* (XPDL) 2.2 [9], and the OMG/ISO standard BPMN 2.0 [4]. BPEL is primarily aimed at implementing Web Service-based service orchestrations. It only has an XML format and lacks a graphical one. As opposed to this, BPMN and XPDL do have a graphical representation as part of their specification, as well as an XML serialization format. Especially BPMN is targeted towards business users, enabling the modeling of processes on various levels with different technical detail. Whereas BPEL 2.0 uses a block-oriented approach for control-flow definition, both XPDL and BPMN are graph-oriented [10]. These three languages are often used in combination. BPMN defines transformation rules to derive a BPEL representation [11, Chapter 14], and XPDL acts as an interchange format that is compatible with BPMN and BPEL. As BPEL has no graphical representation, some tools, e.g., the IDE bundled with OpenESB, make use of BPMN shapes to visualize BPEL processes.

Since BPMN 2.0, the "BPMN execution semantics have been fully formalized" [4, p. 10] and are provided in [4, Chapter 13]. An engine "claiming BPMN Execution Conformance [...] must fully support and interpret the operational semantics" [4, p. 10]. However, several studies report that building executable and, at the same time, standard-conformant processes is hardly feasible due to ambiguities and underspecification, e.g., [12], [13]. Gutschier et al. [12] state that, for this reason, BPMN cannot be used as a service orchestration language. They circumvent the problems in the specification through wrappers, but admit that this leads to the dependence on specific execution engines. Our work is similar to [12] in that we analyze how certain engines implement specific features of BPMN. However, we do not address the executability of the standard per se or try to fix problems in the specification, but analyze which features are implemented in current engines in a standard-conformant way. Whereas [12] focuses on *ServiceTasks* only, we evaluate a whole range of *Activities*, *Gateways*, and *Events*.

[2]The official documentation is available at http://msdn.microsoft.com/en-us/library/dd489441(v=vs.110).aspx.

[3]The official documentation is available at http://docs.jboss.com/jbpm/v3/userguide/jpdl.html.

B. Conformance Benchmarking and Testing of Process Engines

This work is an extension of previous work on the conformance benchmarking of process engines, albeit for another language, BPEL [5]. In former work, we implemented a conformance benchmarking tool for this language, the *BPEL engine test system* (betsy) [14] and used it to analyze the standard conformance of a plethora of BPEL engines [6], [7], [15] in an isolated fashion [16]. The difference between this and former work is the focus on a new language, BPMN [4]. Hence, we heavily adapted the testing tool to support multiple process languages instead of a single one, implemented a new test suite for BPMN, and added support for three BPMN engines.

The benchmarking of process engines is not limited to standard conformance, of course. A central area of interest is performance benchmarking for comparing competing systems [17] and to verify that nonfunctional performance requirements are met [18]. An overview of the available approaches targeting BPEL engines is given in [19]. To conduct performance benchmarking, choosing the right workload [20], as well as the setup of the test bed is critical. In [21], a workload model for benchmarking BPEL engines is proposed, whereas [22], [23] propose test beds for BPEL engines. Here, we benchmark BPMN engines, and focus solely on standard conformance where our workload is simply a set of processes with predefined inputs, and the test bed is set up by betsy.

C. Conformance Checking of BPMN

The conformance checking of BPMN processes manifests itself in various aspects in related work: Part of this work concentrates on the verification of the standard-compliance of concrete models. This may refer to whether there are issues regarding the execution semantics (e.g., [24], [25]) or whether the serialization of a model is correct and compliant regarding the standardized format. Examples for the latter category are [26], [27]. These studies show that a schema validation of BPMN models with the XSD-based serialization format is not sufficient, because the standard defines many constraints which require more sophisticated checks.

Also the standard conformance of modeling tools is of particular interest [26]. The Object Management Group (OMG) currently puts effort in the investigation of interoperability issues of BPMN models and has founded the BPMN Model

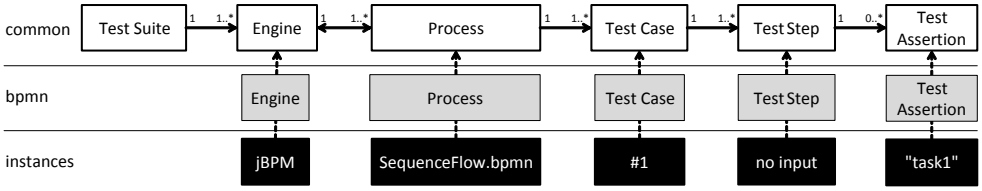

Figure 1. Betsy's Domain Model Applied on a BPMN Test

Interchange Working Group[4] (BPMN MIWG). The focus of this group is different from the work at hand, as BPMN MIWG is tackling interchange of models between modeling tools and not checking the standard conformance of BPMN engines.

III. Approach and Testing Methodology

The intention of our approach is to test the conformance of a set of BPMN engines to the features defined in the specification in an isolated and reproducible fashion. On the one hand, this requires the construction of an automated test workflow that allows for the isolated testing of language features on different engines. On the other hand, a set of feature tests is needed.

To implement the testing workflow, we extended our existing conformance benchmarking tool for BPEL, the betsy tool (cf. Section II-B)[5]. We extracted common core parts from betsy which are used for the testing of both, BPEL and BPMN engines. This allows us to reuse a lot of existing code and procedures while keeping the extension for BPMN small and concise. Moreover, betsy can now easily be extended for the testing of even more process languages. For implementing a suite of feature tests, we extended the domain model of betsy and provided a set of 70 standard-conformant BPMN processes.

The following subsections, describe the domain model, testing workflow, and conformance test suite. Furthermore, we briefly introduce and classify the engines under test in Section III-D.

A. Testing Domain Model

The domain model for conformance tests, shown in Fig. 1, is identical to the one used in betsy [6, p. 3]. Briefly said, a conformance benchmark consists of a *test suite*, which comprises the set of *engines* and *processes* to be tested. For each combination of *engine* and *process*, at least one *test case* is executed. A *test case* consists of a number of *test steps*, which corresponds to zero or more *test assertions*. Only if all *test assertions* can be validated after a test execution, the *test case* is recorded as successful, i.e., the test verifies the conformance of an *engine* to a feature of BPMN. An example of an instantiation of the domain model is shown in the last row of Fig. 1. This instantiation represents a single test case in which betsy verifies whether the *SequenceFlow* construct is supported by the jBPM engine. This involves a single test case that has no input and ensures that an expected trace, in this case the token `task1`, shows up in the process log. Thereby, the test verifies that each *SequenceFlow* in the process is followed as expected.

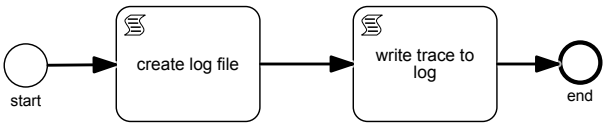

Figure 2. Process Test Stub

The *SequenceFlow* test process, depicted in Fig. 2, is of central importance to our approach. It serves as the test stub which we extend for every other conformance test for BPMN. The language constructs used in this test work on every BPMN engine we benchmark in this paper. Hence, there is no influence of this base test on other tests. The constructs used are, a *none StartEvent*, *SequenceFlows*, *ScriptTasks*, and a *none EndEvent*. Said *Start-* and *EndEvents* are the simplest events in BPMN that involve no specific kind of trigger. *ScriptTasks* are needed for writing the traces of the process execution to a log file. The correctness of these traces in the log file is verified after each test. The usage of execution traces is common for correctness checking of process execution [28], but is a significant difference to the conformance testing of BPEL engines [6]. For BPEL, we actively communicate with the process through messages. This mechanism was not possible for BPMN, since there is no standard-conformant and engine-independent way of specifying message exchanges that works on all engines under test. This problem is also discussed in [12]. For this reason, we had to draw on the evaluation of execution traces. In the process tests, the *ScriptTasks* are initially empty. Upon executing a test for a specific engine, we inject a script into the body of the script task that works on the engine. This script then creates a log file or writes an execution trace. The engine specific adaption of the script body is needed, as the specification does not require BPMN engines to support a specific scripting language [4, Section 10.3.3]. Using this

[4]More information is available at http://www.omgwiki.org/bpmn-miwg/doku.php.

[5]The project is freely available at GitHub: https://github.com/uniba-dsg/betsy; A more thorough description of betsy is given in [14].

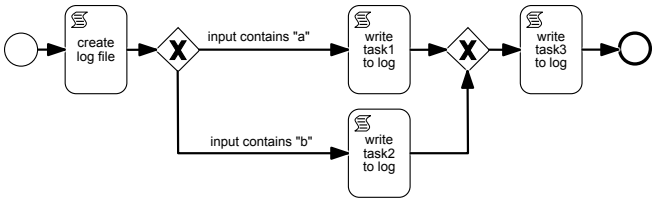

Figure 3. Example: BPMN process for *ExclusiveGateway*

Table I
EXAMPLE: TEST CASES FOR THE FEATURE *ExclusiveGateway*

#	input	expected log trace
1	a	task1,task3
2	b	task2,task3
3	ab	task1,task3
4	c	runtime error

approach does not violate the standard conformance of the process test as both, the empty script and an executable script body, are standard compliant.

Further tests are implemented by extending the test stub in Fig. 2 with specific BPMN constructs. For instance, to test the language construct *ExclusiveGateway*, we require several feature tests. One of them is the plain *ExclusiveGateway* without any additional configuration. To test this feature, we add this gateway along with additional *SequenceFlows* and *ScriptTasks* to check all possible execution traces of the process, as shown in Fig. 3. The four test cases that verify the correctness of this feature are given in Table I, covering every possible branch and condition.

Many BPMN constructs have a variety of configuration options that cannot be checked in a single test case or process. The *ExclusiveGateway*, for example, may include a *default SequenceFlow* and a *gatewayDirection*. We build a different process for each setting of configuration options to exhaustively test every feature. Each process then corresponds to a file in the serialization format defined in [4]. Hence, an engine must be able to consume this format, otherwise a conformance benchmark is not possible.

B. Test Workflow

The purpose of the test workflow is to allow for isolated testing and reproducible results. Since the engines are complex middleware products, this is relatively complicated. A parallel test execution is not possible, because engines consume system resources, such as ports, and might conflict with each other when tested in parallel. Moreover, as previous deployments and test runs might impact the results of subsequent test runs, an engine is installed anew for every test case execution. The test workflow is adapted from [6] and shown in Fig. 4. After some preparation, e.g., the creation of folders and the download of required files, each BPMN feature is tested in isolation, followed by the generation of HTML and CSV reports based on the results. At the beginning of a feature test, the standard conformant BPMN process is enriched with vendor-specific information, such as required namespace declarations and scripts. Moreover, a deployment package is created along with all additional files required by an engine. Next, a Java class containing a unit test is generated. This unit test is used to verify the correctness of the process execution trace. Thereafter, the engine is downloaded, installed, started, and the deployment package is deployed using an engine-specific deployment method. In case a deployment is not successful, a corresponding trace is written into the process log. The testing itself involves several steps: a) starting the process via a REST API call and passing an optional input parameter, b) optionally waiting for a specified amount of time, c) searching the engine log file for exceptions as well as errors and marking any findings in the process log trace, and d) compiling and executing the generated unit test to verify the process log trace. A correct log trace corresponds to the correct implementation of the BPMN feature. Finally, the engine under test is stopped.

C. Conformance Testing Suite

The remaining part of our approach is the test suite of conformance tests. If processes are executed automatically on process engines it is crucial that an engine a) supports all language features in general and b) sticks to the defined execution semantics. These two aspects determine which features should be tested and how the concrete tests have to be designed. In this work, we stick to the most commonly used BPMN language features for processes. In total we define a set of 70 feature tests for 27 language constructs which can be divided in five categories:

Basics (BA): The basis for all tests are connections between BPMN elements through *SequenceFlows*. This test category focuses mainly on different configurations of *SequenceFlows*, for instance conditional and default configurations. Further aspects in this category are the usage of *Participants* and *Lanes*. We created 6 tests covering these basic aspects.

Activities (ACT): The standard introduces different types of *Activities* [4, Chapter 10.3]. *Activities* can be *Tasks*, different kinds of *SubProcesses*, and *CallActivitities*. BPMN also provides the ability to define special execution behavior of *Activities*, like looping or multiple instantiation. We test different types of *Activities* and their execution behavior in 12 tests.

Gateways (GW): The third category contains the constructs that control the routing behavior of a process, i.e., gateways. BPMN defines different types of gateways and we test *Exclusive*, *Inclusive*, *Parallel*, *Complex*, and *EventBased* gateways [4, Chapter 10.6] and their combination in 13 tests.

Events (EV): BPMN defines 3 major types of Events [4, Chapter 10.5]. These are *Start* events, *Intermediate* events and *End* events. Each event type can be enhanced by none, one, or multiple *EventDefinitions* to determine what kind of

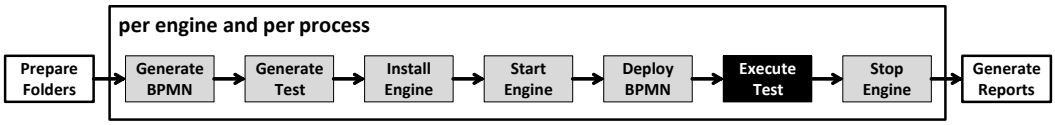

Figure 4. Test Workflow, taken from [6, p. 4]

event is triggered or caught. We test *Cancel*, *Compensation*, *Conditional*, *Error*, *Escalation*, *Link*, *Signal*, *Terminate* and *Timer EventDefinitions* in the form of *Start*, *Intermediate* and *End* events, depending on the allowed combinations (see [4, p. 259–260]), resulting in 36 tests.

Errors (ERR): The error category does not correspond to a certain set of BPMN constructs, but bundles several faulty processes which should be detected by an engine according to [4]. It comprises three tests of invalid condition usages and invalid branching/merging combinations of gateways which result in blocked processes.

D. Engines Under Test

The market of process engines supporting the execution of BPMN is rather fragmented. There are various vendors claiming to support BPMN 2.0. To be used in this study, two main requirements have to be fulfilled: First, we want to test open source engines. So the vendors have to provide a version of their tool licensed under one of the major open source licenses. If there is both an open source and a commercial version available we use the freely available open source version. Second, we check the standard conformance regarding the most recent version of BPMN [4]. This is only possible if the engines support BPMN natively, i.e., they are able to import, deploy and execute processes defined in the standardized serialization format. This excludes some engines which use BPMN for visualization purposes but require a proprietary serialization format. From the remaining set of engines, we choose three engines, namely *jBPM*, *Activiti* and *camunda BPM*. The three engines are briefly described in the following:

jBPM: Originally, jBPM[6] was not developed as a distinguished BPMN engine but as a more general BPM platform. Based on a Process Virtual Machine (PVM), it supports several process languages (e.g., jPDL and BPEL [5]). Since version 4.3, jBPM also supports the execution of processes in native BPMN. The version under test is 6.0.0-Final.

Activiti: Activiti is an open source BPM platform[7]. In 2010, developers already working on jBPM decided to build a new BPM engine from scratch exclusively designed for BPMN execution and this engine is the result. Activiti is supported by various companies, however most core developers are associated with Alfresco who provides an enterprise edition of Activiti. At the time of writing the most recent version of Activiti was 5.16.3 which is used here.

camunda BPM: camunda BPM[8] is a fork of Activiti which is now developed and distributed by the German BPM software vendor camunda. Next to the open source version of camunda BPM, also an enterprise edition is available. In our work we use the most recent open source version 7.1.0-Final.

Table II
FEATURES OF ENGINES UNDER TEST

	jBPM	**Activiti**	**camunda BPM**
General			
Version	6.0.0-Final	5.16.3	7.1.0-Final
License	Apache	Apache	Apache
Date of Release	11/2013	09/2014	03/2014
Developed in	Java	Java	Java
Installation			
Requirements	Ant	-	Maven
Java Version	Java 7	Java 8	Java 7
Container	JBoss AS 7.1.1	Tomcat 7.0.53	Tomcat 7.0.50
Deployment			
Method Used	CLI tool	REST API	container
Package Format	jar	bpmn	war
Files in Package	4	1	4
Vendor Specifics			
Scripting Engine	Java	Groovy	Groovy
Process Variables	explicit	implicit	implicit

Table II gives an overview on all tested engines. All are developed in Java, are published under the Apache open source license, and were released at most twelve months ago. Activiti is the only one that can already be run on Java 8. Both camunda BPM and jBPM do run on at most Java 7. What is more, Activiti has no installation requirements, whereas jBPM requires the availability of the build tool Apache Ant and camunda BPM requires Apache Maven. In our setup Activiti and camunda BPM are executed in a Tomcat container, and jBPM in the JBoss application server. Interestingly, every engine requires a different package format for a BPMN process. For jBPM and camunda BPM, this is a jar or war archive with 4 files, respectively, while for Activiti a plain `.bpmn` file is sufficient. As a consequence, we use a different deployment method for each engine. For camunda BPM, the war file is deployed on Tomcat and not directly on the engine itself. In contrast, for Activiti, the BPMN file is deployed via a REST API and, for jBPM, the jar package is deployed via a command line interface (CLI) call to a jBPM deployment script. As mentioned in Section III-B, we had to take vendor-specific characteristics into account, to generate a BPMN file that is executable on a specific

[6]The project website is located at http://www.jbpm.org/.

[7]The project homepage can be found at http://www.activiti.org/. An introduction to the usage of Activiti is given in [29].

[8]The website is available at http://www.camunda.org/.

engine. Activiti and camunda BPM use Groovy, and jBPM Java for executing script tasks and evaluating *conditions* and *conditionExpressions*. Another aspect is whether process variables have to be modeled explicitly in the process or are implicitly created on demand. jBPM uses the explicit approach, and Activiti as well as camunda BPM use the implicit one.

IV. Results

With the combination of the testing approach and conformance test suite described in the previous section, we are able to run a fully automated and reproducible benchmark of the standard conformance of the three BPMN engines. We executed the benchmark multiple times on a single desktop computer running a Jenkins build server on Windows 7 Professional SP1 with an Intel Core i7-2600 processor, 16 GB RAM, and a 1TB HDD. On this machine, each run of our experiment took approx. 7 hours, resulting in identical benchmark data. Table III shows the results[9] of such a test run. Each row lists the number of supported *features* of a language *construct* for a specific engine. The language *constructs* are grouped according to their *category*. The column *features* indicates the total number of features tested per construct, whereas the last row computes the sum of each column.

A *feature* is supported when all the associated test cases are executed correctly, and unsupported when all the test cases failed. In our result set, the engines either support a feature, or do not support it. The case in which only some but not all test cases are executed successfully did not arise.

A single language *construct* is fully supported if all assigned features are supported. If all features are unsupported, the construct is unsupported as well. In case some, but not all, features for a language construct are supported, the language construct is partially supported. In Table III, no support is indicated with the number zero, whereas full support is achieved when the number of supported features equals the number of total features in the middle column. The remaining constructs are partially supported.

Whereas Table III presents the raw data, Fig. 5 shows aggregated values grouped by language construct category for each engine. This figure shows that all three engines are able to handle all erroneous tests (ERR). Also, the support for the basics (BA) is high with at least 75% (three out of four) of the language constructs being fully supported. The only limitation here is the usage of conditional *SequenceFlows*. The engine jBPM does not support conditional *SequenceFlows* originating from Tasks at all. Activiti and camunda BPM do not implement a special case described in BPMN specification [4, p. 427] correctly. In the following, we describe the results specifically for each engine under test, focusing on the activities (ACT), events (EV) and gateways (GW).

[9]The data set which is the basis for this and all following figures is available at https://github.com/uniba-dsg/sose2015-bpmn-conformance-results.

A. Conformance of Activiti

Activiti passes 56% (39 out of 70) of the feature tests. This is the lowest amount of passed tests of the three engines. Its main strengths lie in the "gateways" and its weaknesses in the "events" category.

Activities (ACT): Regarding the tested activities, only *SubProcesses* and *Transactions* are supported fully by Activiti. Calling *GlobalTasks* or using *CallActivities* is not possible. Furthermore, looping of tasks and the creation of multiple instances using the *MultiInstance* attributes is only supported partially. Whereas *MultiInstance* behavior is actually supported in a few test cases, the *LoopTask* is basically unsupported as no loop iteration is implemented. However, in a special case where the contents of the loop have to be executed only once, the behavior of the *LoopTask* is implemented correctly.

Events (EV): The category of events is least supported. Only about 36% of all event tests pass and only three out of nine event types are fully supported (*Cancel*, *Error* and *Terminate* events). Processes using *Conditional*, *Escalation* and *Link* events either cannot be deployed or the event definitions are ignored by Activiti, resulting in incorrect execution traces. Compensation is only supported if it is triggered by a *Cancel* event or an intermediate *Compensation* event, which is the case in two out of six tests. The usage of signals causes problems if they are used as start or end events in *(Event)SubProcesses*. Furthermore, a *SubProcess* is not interrupted correctly if a boundary event is used. As a result only two out of six of the *Signal* tests pass on Activiti. Finally, Activiti is not able to handle *Timer* start events defined for an *EventSubProcess*. These tests are rejected during deployment.

Gateways (GW): Activiti clearly has its strength in this category, where nearly all tested gateway types and their combinations are supported. Only the single type *ComplexGateway* is not supported.

B. Conformance of camunda BPM

The engine camunda BPM passes 63% (44 out of 70) feature tests, the highest amount of the three engines. The results of camunda BPM are largely identical to the results of Activiti, both in the number of passed and failed tests. The only exception is the "events" category, as camunda BPM supports more event types. Therefore, we discuss only the results for "events" omitting a repetition of the results of the other categories.

Events (EV): *Cancel*, *Error*, *Link*, *Terminate* and *Timer* events are fully supported. This stands in contrast to Activiti, which provides no support for *Link* events, and only partial support for *Timer* events. Analogous to Activiti, only intermediate *Compensation* and *Cancel* events can be used to trigger the *Compensation* (two out of six tests). *Signal* events are almost fully supported, but the usage as an interrupting start event for an *EventSubProcess* and as an interrupting boundary

Table III
RESULTS: OVERVIEW ON TEST RESULTS PER FEATURE FOR ALL ENGINES

Category	Construct	Features	Activiti	camunda BPM	jBPM
basics	Lanes	1	1	1	1
	Participant	1	1	1	1
	SequenceFlow	1	1	1	1
	SequenceFlow_Conditional	3	2	2	0
activities	CallActivity	2	0	0	0
	LoopTask	3	1	1	1
	MultiInstanceTask	5	3	3	0
	SubProcess	1	1	1	1
	Transaction	1	1	1	1
events	Cancel	1	1	1	0
	Compensation	6	2	2	5
	Conditional	5	0	0	0
	Error	4	4	4	4
	Escalation	7	0	0	6
	Link	1	0	1	1
	Signal	6	2	4	5
	Terminate	1	1	1	1
	Timer	5	3	5	3
gateways	Complex	1	0	0	0
	EventBased	2	2	2	2
	Exclusive	3	3	3	2
	Inclusive	2	2	2	1
	MixedGatewayCombinations	4	4	4	4
	Parallel	1	1	1	1
errors	InvalidGatewayCombinations	2	2	2	2
	ParallelGateway_Conditions	1	1	1	1
Σ		70	39	44	44

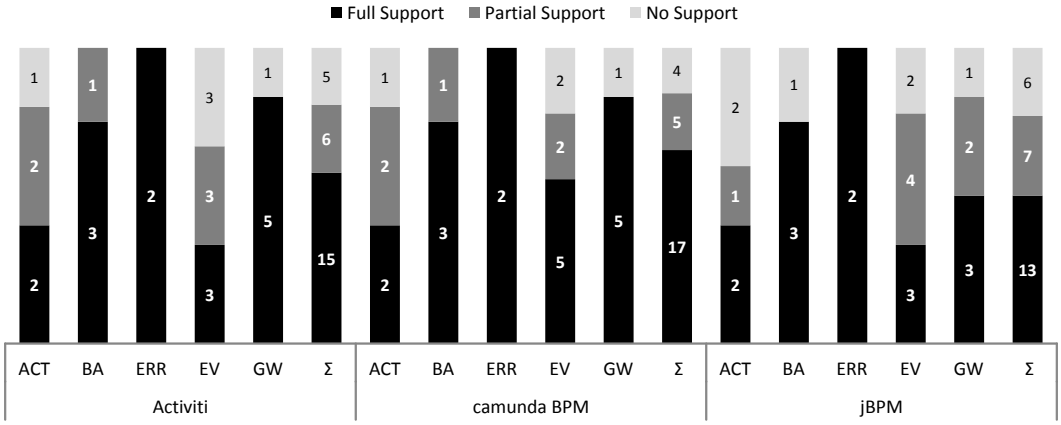

Figure 5. Number of constructs with full, partial and no support grouped by category and engine.

event fails due to runtime exceptions (four successful tests out of six). Thus, camunda BPM passes two tests more than Activiti for this event type. *Conditional* and *Escalation* events are not supported at all which is indicated by meaningful log messages, stating that the engine does not support the event definition.

C. Conformance of jBPM

Although jBPM supports the same amount of feature tests as camunda BPM (44 out of 70), it does not support the exact same test cases. As a result, its strengths lie in the support for the "gateway" category, and its weaknesses in the support for the "activities" category.

Activities (ACT): Apart from the default usage of "activities" (*SubProcess* and *Transaction*), the support for more advanced configuration is limited in jBPM. It passes only three out of twelve tests. Multiple instances and *CallActivities* are not supported at all. Looping behavior is partially supported, although in a severely limited fashion where the loop is executed only once, similar to Activiti.

Events (EV): The results for the event tests are ambivalent. Although jBPM provides basic support for most of the tested

event definitions (69% of all feature tests pass for jBPM) only *Error*, *Link* and *Terminate* events are fully supported. Both *Conditional* and *Cancel* events are completely unsupported. The missing support for canceling transactions hampers full *Compensation* support of jBPM: All *Compensation* tests pass, except compensation triggered by a canceled transaction. Moreover, jBPM is the only engine under test which supports *Escalation* events at least partially with six out of seven test cases. In addition, it also has the highest number of successful tests cases for *Signal* events with five out of six. However, jBPM does not pass all tests for those two event types, as the engine does not interrupt *SubProcesses* properly. *Timers* are partially supported by jBPM, as these are not correctly implemented if used as start events in *EventSubProcesses*.

Gateways (GW): Similar to the other engines, jBPM is strong regarding the support of the different gateway types and their execution semantics. However, *ComplexGateways* and gateways using a *mixed GatewayDirection* (i.e., the gateway is merging and diverging at the same time) are not supported. Moreover, the feature of using a *default SequenceFlow* is not supported in combination with an *InclusiveGateway*.

V. Interpretation and Discussion

In the following, we interpret the outcomes of the benchmark in Sections V-A and V-B, and discuss threats to validity and limitations of the approach in Section V-C.

The results presented in Section IV can be interpreted from two different viewpoints: First, the overall conformance of all BPMN engines in terms of commonly supported features is analyzed. Second, the implications of this level of conformance on process portability are discussed.

A. Overall Support for BPMN Features

The accumulated successful feature tests shown in Table III do not distinguish an obvious winner with regard to supported language features. Both, camunda BPM and jBPM pass 44 of 70 tests (64%), whereas Activiti passes only 56% of the tests. Hence, all engines support only a subset of the language features defined in the BPMN specification [4]. It can be seen from the tests that a) the engines do reject many processes if they detect unsupported features, and b) there are tests which can be deployed but fail nonetheless, as the implementation of the feature is not correct. As a result, none of the engines under test can claim to be fully BPMN compliant.

Looking more closely at the data reveals that the results for Activiti and camunda BPM are roughly equal. This is not surprising as the source code of camundaBPM is forked from the Activiti code base. In fact, Activiti supports a subset of the features supported by camunda BPM.

Although camunda BPM and jBPM have the same overall score, camunda BPM clearly performs better in terms of language constructs (see Fig. 5) because 17 out of 27 constructs tested are fully supported by camunda BPM.

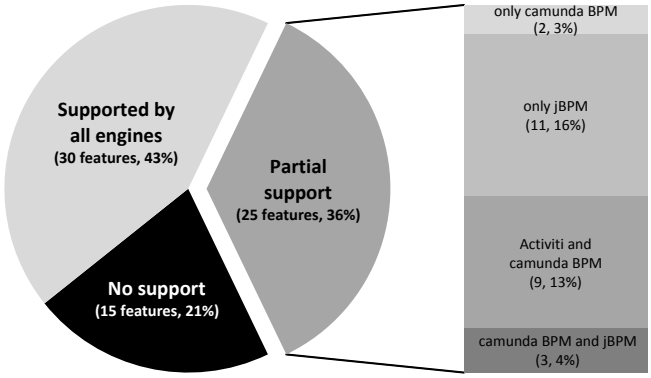

Figure 6. Support Level By Single Feature Test

jBPM is even outranked by Activiti (15 constructs), as it only provides full support for 13 language constructs. This implies that jBPM generally supports more constructs than the other engines in a basic configuration. However, the implementation often does not cover all aspects of a specific language construct. An example already mentioned is the lack of support when using *default SequenceFlows* with an *InclusiveGateway*. At the same time, it can be stated that if a language construct is supported by Activiti and camunda BPM, it is generally supported in a more comprehensive way, respecting all possible configurations.

The number of constructs that are fully supported by all engines is rather small when compared to the amount of constructs that BPMN does define [4]. Apart from the basis for our tests (unconditional *SequenceFlow*, *none StartEvent*, *none EndEvent* and *ScriptTask*) only 8 further constructs are supported by all engines: *SubProcesses*, *Transactions*, *Lanes*, *Participants*, *ErrorEvents*, *TerminateEvents*, *EventBasedGateways* and *ParallelGateways*. Furthermore, all engines are capable to execute combinations of different gateway types and to detect all tested erroneous processes. Although these numbers are not very promising, all but one of the eight constructs most frequently used in BPMN processes are supported [30]. Solely using *ExclusiveGateways* may cause problems, as jBPM does not support a *mixed GatewayDirection*.

CallActivities, *ConditionalEvents*, and *ComplexGateways* are unsupported by all engines. Moreover, the tests show that *interrupting* events often cause problems during execution and are not supported correctly.

B. Implications on Process Portability

With respect to the portability of processes, Fig. 6 shows that porting is problematic. Only 30 of the 70 tested features (i.e., 43%) are supported by all three engines. Hence, only these features can be considered to be fully portable for the three engines under test. Those, and only those, features can be used within a process without causing problems when porting a process from one engine to another. 25 features

(or 36%) are supported only by a subset of engines. Their portability depends on the source and the target engine. Out of the 25 features, two are only supported by camunda BPM and eleven only by jBPM. These 13 features can be considered as non-portable, as they are only supported by a single engine, i.e., the source and target engine must be the same. Nine features are supported by both Activiti and camunda BPM, and three by jBPM and camunda BPM. In contrast, Activiti and jBPM do not have any features in common except the ones that are fully portable. The remaining 15 features are not supported at all. Because of this, any process making use of such a feature is not executable on any of the engines.

The numbers show that the porting of processes from one engine to another is likely to cause problems. In this respect, the situation for BPMN is no different from that of other process languages [31]. Only the porting from Activiti to camunda BPM is unproblematic, since camunda BPM supports all 39 features supported by Activiti. In all other cases there are several features which are supported by the source but not by the target engine. Porting processes from jBPM to Activiti is most problematic, as there are only 30 common features, i.e., 14 features working on jBPM are not executable on Activiti.

This illustrates that feature support and portability are two sides of the same coin: Using an engine that supports many language features enables the execution of more diverse and advanced processes. However, migrating from this engine to another one is often not possible, as some required features are not supported by the target engine. This effectively results in the undesirable condition of a vendor lock-in.

C. Limitations and Threats to Validity

Despite the amount of tested features, several untested aspects and threats to validity remain which limit the generalizability of the interpretation.

Currently not all aspects of BPMN are covered. BPMN provides the ability to define *multiple EventDefinitions* (either parallel or not) which is hard to test due the combinatorial explosion of test cases. Furthermore, we do not deploy more than one deployment package at a time for reasons of test isolation. Although, using BPMN it is generally possible to define more than one process in one `.bpmn` file. However, the engines under test do not support this feature. This implies that is not possible to test some features which require inter-process communication. In particular, these are signals used between processes, messages sent from process to process and *CallActivities* calling another process deployed before. Regarding *MessageEvents* (and *Service*, *Send* and *Receive Tasks*) the standard requires modeling tools and engines to support WSDL based interface and operation definitions [4, pp. 6 and 52]. However, none of the engines under test is able to actually call or provide Web Services in a standard compliant manner. Instead, engine-specific means for message exchange are used. For this reason, we currently do not test said events and tasks. Further limitations in the test suite are the omission of advanced data handling aspects and the handling of erroneous processes.

Moreover, we cannot prove that the test suite and the betsy tool are free of errors that might influence the benchmark. We try to minimize the amount of errors in the generated tests by validating the test processes with respect to schema validity and standard conformance[10]. Furthermore, we peer-reviewed the tests in our group and exposed them to public scrutiny. Finally, we use unit testing and continuous integration to improve the quality of the benchmarking tool.

Another threat to the generalizability of our results lies in the selected engines. Without market data on engine usage it cannot be proven that the engines under test really are representative for BPMN execution in practice. We are confident that the engines are important in the area of open source BPMN engines, but the results might look different when analyzing proprietary BPMN engines. For this reason, we explicitly limit the scope of the paper to open source BPMN execution.

VI. Conclusion and Future Work

The purpose of this paper is to investigate the current state of standard conformance of open source BPMN engines. By extending prior work [6], [7], we were able to check BPMN process engines in a fully automatic fashion. We developed an extensive test suite comprising 70 tests for the most important BPMN language constructs. Using the enhanced tool and the test suite we were able to evaluate the conformance of the three BPMN engines Activiti, camunda BPM, and jBPM.

Our results show that the engines under test currently support at most 64% of the tested features. So, more than a third of the features is either rejected during the deployment of a process, or implemented in a way that conflicts with the specification. An analysis regarding process portability shows that only 43% of the tested features are portable among all three engines. Even a basic construct such as *ExclusiveGateway* is not fully supported by all engines.

The limitations of the approach, as listed in Section V-C, are, at the same time, promising areas of future work. First of all, the test suite does not fully cover all aspects of the specification. Therefore, we plan to extend the test suite by providing support for the remaining language constructs and feature combinations. Moreover, we also aim to take more BPMN engines into account to get a broader picture of the portability of BPMN processes. Especially a benchmark of several proprietary engines would be interesting.

[10]For this task we used a self-developed tool. For more information, see http://www.uni-bamberg.de/pi/bpmn-constraints.

References

[1] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede, *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley, 2005.

[2] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, October 2003.

[3] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, "Business Process Management: A Survey," in *Proceedings of the International Conference on Business Process Management*. Eindhoven, The Netherlands: Springer Berlin Heidelberg, June 2003, pp. 1–12.

[4] ISO/IEC, *ISO/IEC 19510:2013 – Information technology - Object Management Group Business Process Model and Notation*, November 2013, v2.0.2.

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

[6] S. Harrer, J. Lenhard, and G. Wirtz, "BPEL Conformance in Open Source Engines," in *Proceedings of the 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA'12), Taipei, Taiwan*. IEEE, 17–19 December 2012, pp. 237–244.

[7] ——, "Open Source versus Proprietary Software in Service-Orientation: The Case of BPEL Engines," in *International Conference on Service Oriented Computing*, vol. 8274. Berlin, Germany: Springer Berlin Heidelberg, 2013, pp. 99–113.

[8] H. Mili, G. Tremblay, G. B. Jaoude, E. Lefebvre, L. Elabed, and G. E. Boussaidi, "Business Process Modeling Languages: Sorting Through the Alphabet Soup," *ACM Comput. Surv.*, vol. 43, no. 1, pp. 4:1–4:56, December 2010.

[9] WfMC, *XML Process Definition Language*, August 2012, v2.2.

[10] O. Kopp, D. Martin, D. Wutke, and F. Leymann, "The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages," *Enterprise Modelling and Information Systems Architectures*, vol. 4, no. 1, pp. 3–13, 2009.

[11] OMG, *Business Process Model and Notation*, January 2011, v2.0.

[12] C. Gutschier, R. Hoch, H. Kaindl, and R. Popp, "A Pitfall with BPMN Execution," in *Second International Conference on Building and Exploring Web Based Environments*, Chamonix, France, April 2014, pp. 7–13.

[13] E. Börger, "Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL," *Software & Systems Modeling*, vol. 11, no. 3, pp. 305–318, 2012.

[14] S. Harrer and J. Lenhard, "Betsy–A BPEL Engine Test System," Otto-Friedrich Universität Bamberg, Tech. Rep. 90, July 2012.

[15] S. Harrer, C. Preißinger, and G. Wirtz, "BPEL Conformance in Open Source Engines: The Case of Static Analysis," in *Proceedings of the 7th IEEE International Conference on Service-Oriented Computing and Applications (SOCA'14)*. IEEE, 17–19 November 2014, (to appear).

[16] S. Harrer, C. Röck, and G. Wirtz, "Automated and Isolated Tests for Complex Middleware Products: The Case of BPEL Engines," in *Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on*, Cleveland, Ohio, USA, April 2014, pp. 390 – 398, Testing Tools Track.

[17] E. J. Weyuker and F. I. Vokolos, "Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study," *IEEE Trans. Softw. Eng.*, vol. 26, no. 12, pp. 1147–1156, December 2000.

[18] M. Woodside, G. Franks, and D. C. Petriu, "The Future of Software Performance Engineering," in *Future of Software Engineering, 2007. FOSE'07*. IEEE, May 2007, pp. 171–187.

[19] C. Röck, S. Harrer, and G. Wirtz, "Performance Benchmarking of BPEL Engines: A Comparison Framework, Status Quo Evaluation and Challenges," in *26th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Vancouver, Canada, July 2014, pp. 31–34.

[20] A. Avritzer, J. Kondek, D. Liu, and E. J. Weyuker, "Software Performance Testing Based on Workload Characterization," in *Proceedings of the 3rd International Workshop on Software and Performance (WOSP '02)*. New York, NY, USA: ACM, 2002, pp. 17–24.

[21] G. Din, K.-P. Eckert, and I. Schieferdecker, "A Workload Model for Benchmarking BPEL Engines," in *IEEE International Conference on Software Testing Verification and Validation Workshop, 2008. ICSTW '08.*, April 2008, pp. 356–360.

[22] D. Bianculli, W. Binder, and M. L. Drago, "Automated Performance Assessment for Service-oriented Middleware: A Case Study on BPEL Engines," in *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*. New York, NY, USA: ACM, 2010, pp. 141–150.

[23] L. Juszczyk, H.-L. Truong, and S. Dustdar, "GENESIS - A Framework for Automatic Generation and Steering of Testbeds of ComplexWeb Services," in *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on*, March 2008, pp. 131–140.

[24] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008.

[25] P. Van Gorp and R. Dijkman, "A Visual Token-based Formalization of BPMN 2.0 Based on In-place Transformations," *Inf. Softw. Technol.*, vol. 55, no. 2, pp. 365–394, Feb. 2013.

[26] M. Geiger and G. Wirtz, "BPMN 2.0 Serialization - Standard Compliance Issues and Evaluation of Modeling Tools," in *5th International Workshop on Enterprise Modelling and Information Systems Architectures*, St. Gallen, Switzerland, September 2013, pp. 177–190.

[27] ——, "Detecting Interoperability and Correctness Issues in BPMN 2.0 Process Models," in *ZEUS, Rostock, Germany, February 21-22, 2013*, ser. CEUR Workshop Proceedings. CEUR-WS.org, Feb 2013, pp. 39–42.

[28] W. M. P. van der Aalst, B. F. can Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters, "Workflow mining: A survey of issues and approaches," *Data & Knowledge Engineering*, vol. 47, no. 2, pp. 237–267, February 2003.

[29] T. Rademakers, *Activiti in Action: Executable Business Processes in BPMN 2.0*. Greenwich, CT, USA: Manning Publications Co., 2012.

[30] M. zur Muehlen and J. Recker, "How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation," in *Advanced Information Systems Engineering, 20th International Conference, CAiSE 2008, Montpellier, France, June 16-20, 2008, Proceedings*. Springer Berlin Heidelberg, 2008, pp. 465–479.

[31] J. Lenhard and G. Wirtz, "Measuring the Portability of Executable Service-Oriented Processes," in *Proceedings of the 17th IEEE International EDOC Conference*. Vancouver, Canada: IEEE, September 2013, pp. 117 – 126.