On the Evolution of BPMN 2.0 Support and Implementation

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

***Abstract*—The Business Process Model and Notation 2.0 (BPMN) standard has been hailed as a major step in business process modeling and automation. Recently, it has also been accepted as an ISO standard. The expectation is that vendors of business process management systems (BPMS) will switch to the new standard and natively support its execution in process engines.**

This paper presents an analysis of the current state and evolution of BPMN 2.0 support and implementation. We investigate how current BPMN 2.0 implementers deal with the standard, showing that native BPMN 2.0 execution still is an exception. Most BPMS do not support the execution format, despite claiming to be BPMN 2.0 compliant. Furthermore, building on past work, we evaluate three process engines that do provide native BPMN support and examine the evolution of their degree of support over a three-year period. This lets us delimit the areas of the standard that are considered important by the implementers. Since there is hardly an increase in supported features over the past three years, it seems that the implementation of the standard is more or less seen as finished by vendors and it is unlikely that features which are not available by now will be implemented in the future.

***Keywords*-BPMN, process engine, software evolution, conformance testing**

I. Introduction

Business process management (BPM) is a broad discipline with influence on other areas of computing research, such as service composition [1]. BPM is strongly tailored to the modeling of organizational processes in dedicated modeling languages and the subsequent implementation of process models in executable software. Among the many languages for process modeling that are available today (examples are discussed in [2]), the *Business Model and Notation* (BPMN) [3] has emerged as a widely accepted candidate. Its importance and potential for consolidating the market of process standards is emphasized by its recent acceptance as an ISO standard [4][1].

With the publication of BPMN 2.0 in January 2011, support for the native execution of BPMN process models and a standardized serialization format has been added to the standard. The intention underlying this addition is to mitigate the gap between modeled processes on the one hand and the execution of processes on the other hand [5]. When bridging this gap by translating a model into executable software, a variety of problems arise that are the topic of various publications, e.g., [6]–[9], regardless of the concrete modeling language used. Especially for industry-size process models, this translation step is non-trivial and can lead to a deviation of the actual implementation of the model from the desired execution semantics captured in the model. The direct execution of a process model intends to minimize the distance between desired and actual behavior. This avoids the necessity for a translation step in the first place and has the potential to preempt the deviation of a model from its implementation.

The specification of the execution semantics of a process model in a standards document, as achieved with BPMN 2.0 [4], is a first step towards the goal of direct execution. As a next step, it is necessary that the execution semantics are correctly and completely implemented by vendors of the standard. In the case of BPMN, the second step faces two major obstacles: Firstly, the implementation of the specified execution semantics needs to be feasible, which unfortunately cannot be guaranteed by the purely theoretical and informal discussion of the execution semantics in the BPMN 2.0 standard [10]. Secondly, the vendors need to actually implement the same semantics and not interpretations or customizations thereof. Since there is no certification authority for BPMN, any vendor can claim full implementation conformance and support for the standard without proof of this claim. Furthermore, it may well be possible that only a subset of BPMN is valuable from a practitioner's point of view, as indicated in [11]. As a result, vendors might want to implement only a limited part of the standard. The paper intents to cast light on the latter issue.

In prior work [12], we analyzed three well-known BPMN 2.0 engines and investigated the degree of standard conformance they provide. In this paper, we build upon this work with a comprehensive study of BPMN 2.0 implementers in the current market. We analyze the products of a broad list of vendors that claim to implement BPMN 2.0. This shows that a vast majority of current self-acclaimed BPMN 2.0 implementations is limited to visualization and does not actually implement the execution semantics of BPMN 2.0. Thereafter, we extend the standard conformance analysis from [12] with a more sophisticated set of tests, covering a

[1]For the remainder of the paper, we will refer to the ISO version of the standard [4], instead of [3].

higher degree of the executable part of the standard. Moreover, we also analyze whether common feature combinations, defined by the so-called workflow patterns [13], are supported. We compare multiple revisions of several engines that support the native execution of BPMN 2.0 over a three-year period. This lets us see how BPMN 2.0 support has evolved since its adoption as an ISO standard. The current development indicates that the implementation of BPMN 2.0 is consolidating at a level that excludes a large part of the actual execution semantics of the standard.

The remainder of this paper is structured as follows: The next section outlines related work on BPMN 2.0 execution and the evaluation and benchmarking of process engines. In Section III, we present and discuss data of 45 vendors that claim to implement the BPMN 2.0 standard. Thereafter, in Section IV, we detail our procedure for evaluating standard support in a BPMN engine and show the results of a detailed assessment of the supported feature set of three products. The results of our longitudinal study, showing the evolution of feature support over the course of the last three years, round off our findings in Section V. Finally, Section VI summarizes the main findings of the study and points to future work.

II. Related Work

Work related to this paper separates in three major areas: First, we discuss work that concerns the native serialization format and execution of BPMN 2.0 process models and problems experienced during this task. Second, we outline approaches for evaluating and benchmarking BPMN 2.0 engines. Third, we present approaches for building reproducible benchmarks, which is what we aim for in this paper.

A. Serialization Fromat and Native Execution of BPMN 2.0

The set of features that is to be supported by a BPMN 2.0 implementation is vast and so is the description of their execution semantics. As discussed in [10], this description is inaccurate and unambiguous in various places. As a result, the goal of fully standard-conformant BPMN 2.0 implementations with identical behavior is elusive, according to [10]. The same aspect is confirmed by [14], whose authors discuss underspecifications for a particular language construct, the service task, which are likely to result in differing behavior in different implementations. Also [15] targets this aspect and provides a comprehensive discussion of the execution semantics of BPMN 2.0 [4] and their shortfalls. To improve the situation, [15] refines the specification using abstract state machines. Finally, also [16] discusses issues in the specification of the serialization format of BPMN and provides a preliminary analysis of the BPMN support in modeling tools.

When it comes to the usage of the BPMN 2.0 execution semantics and the serialization format, [5] presents first results based on a survey. Already at the time shortly after the publication of BPMN 2.0, 40% of the survey respondents claimed to use the new native serialization format. It can be expected that this number has increased since the publication of [5].

B. Evaluating and Benchmarking BPMN 2.0 Engines

When it comes to the evaluation of BPMN 2.0 process engines, our prior work on standard conformance [12] is obviously related. Since we build on this approach, it is described in more detail in the following sections. A short case study that analyses several capabilities of a set of BPMN 2.0 engines is presented in [17]. At the time, practically none of the engines under focus was able to correctly import models in the native BPMN serialization format. This is an interesting result which is confirmed by our analysis of BPMN implementers in Sect. III. It contradicts the already mentioned results from [5]. There seems to be a discrepancy between modeling tools which are able to create and consume standard compliant BPMN models and execution engines which are unable to work with those models.

Apart from the evaluation of standard conformance, also other aspects of BPMN 2.0 engines are investigated. A notable approach for evaluating the performance characteristics of BPMN engines is [18]. Another comparative study is [19] which evaluates the three BPMN engines activiti, jBPM and camunda BPM with regard to their general architecture and their extensibility. However, the focus in [19] is on the integration of semantic techniques such as ontologies to improve the quality of the processes to be executed.

C. Computational Reproducibility

Reproducible research, especially computational reproducibility, is key in today's science [20]. Many scientific results are obtained with the help of extensive software tooling, without which it is not possible to reproduce or to check the correctness of results. To enable computational reproducibility, [20] suggests to make the documented, tested, and modular code on which scientific computations are based, as well as all related material and the results freely available, preferably on a version-control system. But even following these guidelines, researchers trying to reproduce the experiment still phase "significant barriers" [21, p. 72], e.g., imprecise documentation which is hard to follow or steep learning curves for the tools used in experiments.

To increase the reproducibility of scientific computations, [21], [22] suggest to use Docker[2] as the basis for computational environments, since it lowers several of the aforementioned barriers. In Docker, one can create lightweight Linux containers based on Docker images. These images can be shared, reused, archived, put under version control, and deployed on different platforms [21]. What is more, such an image can be built automatically by passing a

[2]See https://www.docker.io for details.

Dockerfile (i.e., a list of commands) to Docker itself, making the step to create the software environment for an experiment reproducible for third parties. Here, we perform a benchmark for obtaining core results. To enable reproducibility of this benchmark, it is executed based on Docker.

III. Analysis of BPMN 2.0 Implementers

To gather insights on the state of BPMN implementation, it is necessary to investigate the current market of systems that implement the standard. To this end, we evaluate public listings of BPMN implementers and judge their state of implementation when it comes to the execution of BPMN 2.0 processes in this section. Only a small subset of the existing implementations is mature enough to justify an evaluation of *the degree of standard conformance* they exhibit. This section describes the selection of such mature engines from the actual market. Sect. IV analyses the degree of standard conformance of the selected engines.

At a first glance, the market of BPM suites and engines supporting BPMN seems to be rather active. A list of BPMN implementers[3] contains 75 implementations at the time of writing. These implementers can be divided into diagramming tools, Business Process Management Systems (BPMS), Business Process Analysis (BPA) systems, and Case Management systems. With regard to BPMN execution semantics conformance, especially the fully fledged BPMS suites are of relevance. We identified 31 candidates which stated directly or implicitly that they also target the execution of BPMN processes. Another "list of active BPMN 2.0 Engines" at Wikipedia[4], which has been created and is mainly maintained by members of the BenchFlow project[5], currently lists 24 entries. Here, we compared and consolidated both lists, arriving at 45 BPM suites and engines, which claim to conform to the BPMN standard. These systems are the focus of the following sections.

A. Requirements for BPMN Engines

The standard document [4] is rather strict regarding *Process Execution Conformance*: *"The tool claiming BPMN Execution Conformance type MUST fully support and interpret the operational semantics and Activity life-cycle specified in sub clause 14.2.2. [...] Conformant implementations MUST fully support and interpret the underlying metamodel."* [4, p. 10] Following these statements, there are two main aspects relevant for BPMN engines and BPMS: Firstly, all language constructs defined in the BPMN meta-model must be supported. Secondly, for each language construct the underlying execution semantics must be implemented correctly. The only exceptions from this are a few "non-operational elements" defined in [4].

The purpose of the standardization of BPMN is to achieve portability and interoperability between different modeling tools and engines. A common workflow for implementing processes is to first model "basic" business process models in a BPMN modeling tool. Subsequently, these models are enriched with execution details specific to the BPMS used for execution. A strong indicator for the importance of this use case is the ongoing effort put into interchange of models by various vendors of modeling tools and engines in the *BPMN Model Interchange Working Group* (MIWG)[6]. Therefore, we require BPMN engines and BPMS to support the import of the standardized XSD-based serialization format as a third important requirement.

To summarize these requirements, a conforming BPMN implementation must:

1) support the usage of all constructs defined in the standard,
2) implement the behavior of the constructs as specified, and
3) be able to consume the standardized serialization format.

Only products that fulfill these requirements can be evaluated to support process execution conformance, as defined in the standard.

B. Requirements Evaluation for BPMS

We evaluated the requirements stated in the previous section for all 45 BPMN implementations mentioned before using a structured methodology. First, we checked the publicly available information on the websites of the different vendors. If a version of the BPMN implementation was available for free usage or evaluation purposes, we downloaded this version. For each available product, we checked whether existing integrated modeling tools support the BPMN meta-model at least partially, i.e., whether at least some BPMN elements with their defined attributes are available. If this was the case, or if direct deployment of BPMN files was possible, we tried to import and deploy an existing BPMN process model. Only products that pass all these tests can, in principle, be classified as an implementation of the execution semantics of BPMN.

The results for the 45 tested products, depicted in Fig 1, are rather surprising: For 25 out of the 45 products, evaluation was not possible for various reasons. The development and distribution of five products has been discontinued. For ten products no suitable public information is available (too little information to meaningfully evaluate BPMN support, or information is not available in English). Two products offer some modeling and execution functionality but neither the shapes used nor the used serialization format correspond

[3]Available at http://www.bpmn.org/#tabs-implementers, the official BPMN website.

[4]The list is available at https://en.wikipedia.org/wiki/List_of_BPMN_2.0_engines.

[5]The project web site is available at http://www.iaas.uni-stuttgart.de/forschung/projects/benchflow.php.

[6]The work of the group, their intentions and preliminary results are presented at http://www.omgwiki.org/bpmn-miwg/doku.php

to the BPMN standard. Hence, it is unclear, why these tools are listed as BPMN implementers to begin with. For eight products, a closer analysis is not possible in the context of this paper, due to licensing restrictions. For instance, some vendors explicitly prohibit the benchmarking and evaluation of their products, which forces us to omit them from a closer investigation.

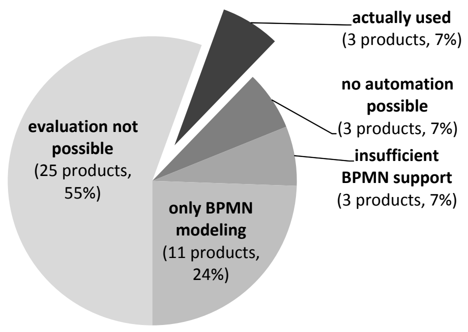

Figure 1. Results of the Product Analysis

The usage of graphical BPMN shapes to model processes is supported by eleven BPMS, but the definition of attributes does not conform to the standard and/or the import, and as a result, usage of standard compliant models is not possible. Therefore, of the original 45 products, only 9 support at least parts of the BPMN 2.0 meta-model, can import standard compliant models and provide enough information for a detailed assessment. Of those remaining BPMS, three engines require the usage of extensions that are not covered by the standard, or restrict the usage of essential features such as IDs and task types. As a result, the models consumed or produced by these systems are substantially different from the BPMN 2.0 standard, although they share some syntactical elements.

Considering all these aspects, only six systems justify a closer analysis regarding the degree to which they actually implement the BPMN standard. The remaining products are able to digest process models conforming to the BPMN standard, but for three of these, model import, deployment or the process execution is only possible via manual operations using a graphical interface.

This severely hampers computational reproducibility and, thus, is prohibitive for our approach of checking the conformance, which will be described comprehensively in the following section: We assess the conformance of the whole spectrum of BPMN. To achieve this, each BPMN language construct has to be tested in each possible configuration. Furthermore, we want to guarantee isolation of tests by providing a fresh process engine instance for each test run, and by running each test various times to check reproducibility of results. Those aspects are only feasible if automated deployment and execution is possible. Therefore, only three engines can be assessed here: These engines are *activiti*, *camunda BPM*, and *jBPM*[7].

IV. Assessment of Current BPMN Support

In this section, we benchmark the BPMN support of the three engines selected in the previous section. We use the newest version that was available on 2015-10-30, namely, activiti 5.18.0, jBPM 6.3.0, and camunda BPM 7.3.0, thereby updating the versions assessed in [12]. The setup of the benchmark is explained in Section IV-A, followed by the test suite, which consists of both, tests for native BPMN support and tests for workflow control-flow pattern support, in Section IV-B. The results of our assessment are given in Section IV-C.

A. Benchmark Setup

The benchmark is conducted with an improved version of the BPEL/BPMN Engine Test System (*betsy*). In previous work, betsy has been used to benchmark BPEL [23] as well as BPMN engines [12]. For this study, we have improved and extended betsy by 1) adding the ability to test several versions of activiti, jBPM, and camunda BPM, 2) enabling betsy to distinguish real concurrency and pseudo-parallelism during process execution, 3) adding a test suite for evaluating workflow control-flow pattern support in BPMN, and 4) porting betsy to Linux to enable execution with Docker.

Betsy has its own domain specific *test language* to express conformance tests and their results [12], [23]. A test corresponds to an executable process that uses a specific BPMN 2.0 language feature, e.g., a *ScriptTask* or an *ExclusiveGateway*. Each test has at least one *test case*, consisting of one or more *test steps*. These steps define inputs to the test, in the form of string or integer variables. Test inputs are injected into the process during execution as properties. Moreover, test steps define expected test outputs in form of *test assertions*. An assertion specifies execution traces of a process model and is used to verify the correctness of a test after its execution. Execution traces are written during execution by the means of BPMN 2.0 *ScriptTasks*. By comparing the expected execution trace with the actual one, it is possible to determine whether a test case was executed successfully. The testing of real concurrency within a process has been implemented by adding the ability to store labeled timestamps into the log trace. By checking if time stamps overlap, it is possible to detect concurrent execution.

The *test results* state for each test case, a) if the respective process could be deployed on a particular engine, and b) if the test case was executed successfully. A language feature is considered to be supported by an engine given the process can be deployed and all test cases are executed successfully.

On starting a benchmark, betsy executes a *test workflow* for each test and each engine under test. First, the native

[7]Further information is available at the respective homepages located at http://www.activiti.org/, http://www.camunda.org/, and http://www.jbpm.org/.

BPMN format of a test is adapted to the engine under test. For instance, a concrete scripting language needs to be set (this aspect is not treated by the BPMN standard). To ensure that each test is executed in isolation, a fresh instance of the engine under test is installed and started. Next, the process containing the language feature under test is deployed on the running engine. The deployment may require the creation of a deployment package, depending on the engine under test. Next, the test itself is executed by triggering all test cases with their test steps subsequently. Following this, execution traces are gathered from the log file and compared to the expected traces, thereby determining test success or failure. Last, the engine under test is shut down.

We used the following steps to ensure that the benchmark conducted as part of this work can be considered reproducible: Betsy itself including its test suites is open source and freely available on a version control system[8]. This holds true for the engines under test as well. In addition, the functional correctness and quality of betsy is ensured using a continuous integration pipeline. The results of our experiments can be found in version control as well[9]. The benchmark is done within a fixed Docker-based environment[10] for betsy, which required the porting of betsy to Linux. The pre-built image on which the experiment was conducted is also available[11].

B. Test Suites for BPMN Conformance Assessment

There are two *test suites* for assessing the conformance of engines to the BPMN standard: The first test suite bundles tests for the constructs and features described in the BPMN specification [4]. The second focuses on more complex constructs that are frequently needed in process models, captured in the form of workflow control-flow patterns [13].

1) Feature Tests for Language Constructs: The first test suite has already been used in [12] and covered 27 different language constructs (such as *ExclusiveGateway*) which have been tested by 70 feature tests divided in five different groups. Using feature tests we are able to determine whether all possible configuration possibilities (e.g, usage of a *default SequenceFlow* for *ExclusiveGateways*) are supported by the engines under test. Although the 27 language constructs represent the majority of the BPMN constructs, we added further constructs to tackle several limitations mentioned in [12]: First, we added a sixth construct group, *data*, which contains basic tests for the language constructs *DataObject* and *Property*. Second, we added further tests for inter-process communication performed with the help of *Signals*, *MessageEvents*, *SendTasks* and *ReceiveTasks*. Moreover, we now also test *MultipleEvents*, the usage of *EventDefintionRefs*, *AdHocSubProcesses* and different *startQuantity* and *completionQuantity* settings of activities (subsumed as *TokenCardinality*). Apart from those newly added language constructs we refined and added feature tests for various other language constructs. The resulting test suite for checking the native BPMN support consists of 38 language constructs tested by 113 feature tests. This is a 40% increase in the amount of language constructs covered. An overview over all covered constructs and the number of associated feature tests is shown as part of the results in Table I.

2) Workflow Control-flow Patterns: Standard conformance of all language features is necessary to execute arbitrary standard-conformant processes on a given engine. Any process that *can be expressed* in the standard, should be executable on an engine. Closely related to this aspect is the *expressive power* of the language dialect supported by an engine. Expressive power is captured by the ease with which structures that are frequently needed in a system can be expressed. The easier commonly needed structures can be built, the more expressive a language or system is. In the case of process languages, expressive power is frequently assessed by the means of workflow patterns [13]. Although the relevance of the concrete patterns is not undisputed [10], they are frequently applied in related studies. Additionally, using workflow patterns for assessing expressive power eases the comparison of this work to others.

In this paper, we use the original 20 workflow control-flow patterns from [13], and no extensions or derivations thereof, since these are most widely known. We built upon the pattern-based analysis for BPMN 1.0 presented in [24]. Most of the pattern implementations described in the paper can directly be applied to BPMN 2.0. In the rare cases, where a modification of a pattern implementation was necessary, we followed the rationale of [24] to provide a solution. Table II lists the patterns sorted by pattern category, along with the highest degree of pattern support that can be achieved for BPMN 2.0. Due to page constraints, we cannot describe every pattern here, but refer the interested reader to [13], [24]. Pattern support is rated in a trivalent rating of $+$ (direct support), $+/-$, (partial support), or $-$ (no direct support). Again, we follow [24] in the judgement of the degree of support that is possible in BPMN. It can be seen in Table II that two patterns (MI without A Priori Run-Time Knowledge and Milestone) cannot be directly implemented in BPMN. We were unable to find workarounds based on the extended vocabulary of BPMN 2.0 that could compensate for this. Thus, we exclude these patterns from further discussion. For the remaining patterns, we implemented at least one test case, according to the structures from [24], that led to pattern support. In case at least one engine failed the initially built test, we implemented an additional test that led to the same or a reduced support rating, to see if the engine supports an

[8]The project page is located at https://github.com/uniba-dsg/betsy.

[9]The complete results are available at https://github.com/uniba-dsg/2016sose-results

[10]The Dockerfile is available at https://github.com/uniba-dsg/betsy-docker

[11]To use the pre-built image, please execute the command `docker pull simonharrer/betsy-docker` on the command line. More information is available on the Docker Hub project at https://hub.docker.com/r/simonharrer/betsy-docker/

alternate structure. The reasoning for this is that an engine is considered to support a pattern if it implements at least one solution that grants support. Full support of all possible, equivalent solutions is not required. Finally, if an engine supports none of the solutions presented by [24] (fails all related tests), we consider it as not supporting a pattern.

C. Benchmark Results

The results are obtained by executing the benchmark setup, described in Section IV-A, for the three engines, selected in Section III, using the test suites from the previous section. The next section discusses the results for the test suite addressing native BPMN support, followed by the discussion of workflow control-flow pattern support.

1) Feature Tests for Language Constructs: The results of the evaluation of native BPMN support are shown in Table I. Activiti supports 51 out of the 113 features, camunda BPM supports 55, and jBPM provides the highest amount of support with 59 features. Translated to percentage values, support ranges from 45% up to 52%, being approx. half of the tested features. It is interesting that all three engines support roughly the same amount of features, differing by at most eight features. In comparison to previous work [12], we added 43 new feature tests, but the support of activiti has only risen by 12 features, of camunda BPM by 11 features, and of jBPM by 15 features. Hence, the data reveals that at most 35% of the added feature tests are supported.

Activiti and camunda BPM differ in their support of four features. Previously [12], camunda BPM supported all the features activiti that supported, but this is no longer the case. Activiti successfully detects two errors related to *Multiple-Events*, which are missed by camunda BPM at deployment time. However, taking all other differences into account, camunda BPM is better than activiti. Camunda BPM has higher support for *Compensation-*, *Signal-* and *TimerEvents*, as well as for detecting invalid loop conditions. The third engine, jBPM, shows strength with its support for twelve additional event types compared to camunda BPM, but fails to support conditional *SequenceFlows* and the *MultiInstance-Task*. Apart from these differences, jBPM behaves similar to the other two engines.

Looking at the supported language constructs by group, we can see that the *data* group is supported completely. Within the *gateways* group, all three engines support *EventBased-*, *Exclusive-*, *Inclusive-*, and *MixedGateway* combinations. However, all fail to support *ComplexGateways*. *Parallel-Gateways* are only partially supported, since none of the engines actually supports concurrency, but only pseudo-parallelism. The language constructs of the *basics* group, namely, *Lanes*, *Participants* and *SequenceFlows* are supported by every engine. Conditional *SequenceFlows* are supported partially by activiti and camunda BPM, whereas jBPM has no support for this construct. Regarding the *activities* group, three features, namely, *ReceiveTask*, *SendTask* and *AdHocSubProcess*, are unsupported. Nevertheless, every engine supports *SubProcesses* and *Transactions*. Regarding the *LoopTask*, only one out of six different tests is supported by every engine. In contrast, support for *MultiInstanceTasks* is relatively good, with five out of eight successful tests by activiti and camunda BPM, but it is completely unsupported by jBPM. The usage of tokens is barely supported as only activiti and camunda BPM pass one out of four tests. In previous work, all feature tests within the *errors* group were supported. As opposed to this, most of the newly added error tests fail, except for two checks for the usage of multiple events and invalid loop conditions by activiti and jBPM. Within the *events* group, *MessageEvents*, *Conditional-Events*, and *EventDefinitionRefs* are unsupported by every engine, whereas widespread support can be diagnosed for *Error-* and *TerminateEvents*. The engine jBPM supports all *EscalationEvents*, which are unsupported by the other two engines. Furthermore, it supports all *SignalEvents*, which are supported with five and six successful tests out of nine by activiti and camunda BPM, respectively. Approximately half of the tests for *TimerEvents* and *CompensationEvents* are passed by the engines. Failures of the *TimerEvents* in jBPM and activiti are mainly caused by using timers for start events. *Cancel-* and *LinkEvents* are unsupported by jBPM and activiti, respectively, but supported by the camunda BPM. Half of the tests for *MultipleEvents* are passed by jBPM only.

Looking at the number of language features that were not supported, we can distinguish between two different cases: rejection at deployment and failure at runtime. Activiti rejects 21, camunda BPM 24, and jBPM 23 valid language features at deployment. Of the unsupported language features, activiti accepts 41, camunda BPM 34, and jBPM 31 language features at deployment, but at runtime the corresponding tests reveal wrong execution semantics. This is a real issue for users of the engines as they might encounter unexpected behavior in their own deployed processes, which is even worse in case it remains undetected.

2) Workflow Control-flow Patterns: The support of workflow control-flow patterns is shown in Table II. Overall, every engine supports at least 13 out of the 20 original workflow patterns. However, as mentioned before, the patterns WCP-15 and WCP-18 are excluded from the analysis, since they cannot be directly supported in BPMN to begin with. Eleven patterns are supported as expected by all three engines. The three patterns WCP-9, WCP-12 and WCP-17, for which partial support is possible in BPMN, are unsupported by all engines under test. For four patterns the results vary from engine to engine, but are always supported by two engines. WCP-13, WCP-14 and WCP-19 are directly supported by activiti and camunda BPM, but not supported by jBPM. This reveals a major problem with the jBPM engine. The pattern WCP-10 is correctly executed on jBPM and activiti, but not on camunda BPM. This is also the only difference of the pattern support between camunda BPM and activiti.

Table I
NATIVE BPMN SUPPORT

group	language construct	features	activiti	camunda BPM	jBPM
Σ	**38**	**113**	**51**	**55**	**59**
gateways	**6**	**14**	**12**	**12**	**11**
	ComplexGateway	1	0	0	0
	EventBasedGateway	2	2	2	2
	ExclusiveGateway	3	3	3	2
	InclusiveGateway	2	2	2	2
	MixedGatewayCombinations	4	4	4	4
	ParallelGateway	2	1	1	1
basics	**4**	**6**	**5**	**5**	**3**
	Lanes	1	1	1	1
	Participant	1	1	1	1
	SequenceFlow	1	1	1	1
	SequenceFlowConditional	3	2	2	0
data	**2**	**2**	**2**	**2**	**2**
	DataObject	1	1	1	1
	Property	1	1	1	1
activities	**9**	**27**	**10**	**10**	**4**
	AdHocSubProcess	2	0	0	0
	CallActivity	2	1	1	1
	LoopTask	6	1	1	1
	MultiInstanceTask	8	5	5	0
	ReceiveTask	2	0	0	0
	SendTask	1	0	0	0
	SubProcess	1	1	1	1
	TokenCardinality	4	1	1	0
	Transaction	1	1	1	1
errors	**5**	**10**	**5**	**4**	**5**
	InvalidGatewayCombinations	2	2	2	2
	InvalidLoopConditions	2	0	1	0
	InvalidTokenQuantity	3	0	0	0
	MultipleEvents	2	2	0	2
	ParallelGatewayConditions	1	1	1	1
events	**12**	**54**	**17**	**22**	**34**
	CancelEvent	1	1	1	0
	CompensationEvent	6	2	3	5
	ConditionalEvent	5	0	0	0
	ErrorEvent	4	4	4	4
	EscalationEvent	7	0	0	7
	EventDefinitionRef	4	0	0	0
	LinkEvent	1	0	1	1
	MessageEvent	3	0	0	0
	MultipleEvents	4	0	0	2
	SignalEvent	9	5	6	9
	TerminateEvent	1	1	1	1
	TimerEvent	9	4	6	5

Table II
WORKFLOW CONTROL-FLOW PATTERN SUPPORT

Control-Flow Pattern		BPMN 2.0	activiti	camunda BPM	jBPM
	Basic Control-Flow Patterns				
WCP-1	Sequence	+	+	+	+
WCP-2	Parallel Split	+	+	+	+
WCP-3	Synchronization	+	+	+	+
WCP-4	Exclusive Choice	+	+	+	+
WCP-5	Simple Merge	+	+	+	+
	Advanced Branching and Synchronization Patterns				
WCP-6	Multi-Choice	+	+	+	+
WCP-7	Structured Synchronizing Merge	+/-	+/-	+/-	+/-
WCP-8	Multi Merge	+	+	+	+
WCP-9	Structured Discriminator	+/-	-	-	-
	Structural Patterns				
WCP-10	Arbitrary Cycles	+	+	-	+
WCP-11	Implicit Termination	+	+	+	+
	Multiple Instances (MI) Patterns				
WCP-12	MI Without Synchronization	+	-	-	-
WCP-13	MI With A Priori Design-Time Know.	+	+	+	-
WCP-14	MI With A Priori Run-Time Know.	+	+	+	-
WCP-15	MI Without A Priori Run-Time Know.	-	-	-	-
	State-Based Patterns				
WCP-16	Deferred Choice	+	+	+	+
WCP-17	Interleaved Parallel Routing	+/-	-	-	-
WCP-18	Milestone	-	-	-	-
	Cancellation Patterns				
WCP-19	Cancel Activity	+	+	+	-
WCP-20	Cancel Case	+	+	+	+

Looking at the pattern groups, it is obvious that all *basic* control-flow patterns are supported. Both, *structural* and *cancellation* patterns, are also supported on almost each engine, except for one failure in each of the groups. Within the group of *advanced branching and synchronization* patterns, only one pattern is completely unsupported, but the remaining three are directly supported by every engine. The same holds true for the *state-based* patterns. The remaining group of *multiple instances* patterns is least supported, as WCP-12 is completely unsupported by all engines and both WCP-13 and WCP-14 are unsupported by jBPM. When looking at the reasons for failures in pattern support, the most common issue is that the BPMN process containing the pattern is not deployable, because a construct used therein (such as an *AdHocSubProcess*) is not supported by an engine. Because of this, jBPM supports none of the *multiple instances* patterns due to missing support for the *MultiInstanceTask* and the *WCP-19 Cancel Activity* due to missing support for the *CancelEvent*. The pattern *WCP-10 Arbitrary Cycles* should in general be supported by camunda BPM as all used language features are supported, but the test fails with a runtime exception caused by the composition of these language features.

To sum up, pattern support of the three engines can be considered high and balanced, as it is ranging between 13 up to 15 out of 18 patterns. It is interesting that activiti supports 14 out of the 18 patterns with only 51 out of 113 language features. In contrast, the engine jBPM supports 59 out of the 113 language features, but only supports 13 patterns. Hence, only a moderate degree of support for language features is required to implement a large set of the patterns.

What is more, the lack of pattern support by jBPM is mainly caused by rejecting processes containing the pattern already at deployment time. Out of the ten failed tests, nine pattern tests were being rejected during deployment. For activiti and jBPM, only one out of six and three out of seven are rejected upon deployment.

V. EVOLUTION OF BPMN SUPPORT

As seen in the previous section, there are substantial limitations in the current implementation of the BPMN standard. These limitations could be attributed to the fact that implementation is still in progress and support of the standard will increase with time. An alternative interpretation is that implementations are limited to the features of the standard that are relevant in practice and the remaining part of the standard will likely never be implemented. The latter interpretation is supported by studies that show that the features used in process models by practitioners are limited to a modest part of the standard [11], [25]. Here, we try to address this aspect from the direction of process engines. If, on the one hand, the feature set supported by engines constantly increases, this is an indication that the implementation of the standard is still in progress. On the other hand, if the feature set supported by engines stays rather constant over time, this is an indication that the implementation of the standard has stopped in practice and language features that have not been supported by now are considered as irrelevant.

To address this aspect, we investigate the evolution of BPMN support over time in this section. We benchmark different versions of the engines discussed in Section IV that have been published over a three year period, using the exact same methodology as before. An overview of the engine versions we consider is given in Section V-A. The results of the benchmarks, describing the evolution of native BPMN support and workflow control-flow pattern support, are shown in Section V-B and Section V-C, respectively, and are summed up in Section V-D. The results support the second hypothesis, i.e., that the implementation of the BPMN standard is concluded in practice.

A. Compared Engine Versions

Section IV details the benchmarks of the latest versions of the engines activiti, jBPM, and camunda BPM. Here, we add benchmarks for three additional prior versions of each said engine. The concrete version numbers of the engines are shown in Table III, along with their release date and the number of days each version has served as latest stable release. The engine vendors apply semantic versioning for setting release numbers[12]. Version numbers are encoded in the form `MAJOR.MINOR.PATCH` and numbers are incremented depending on the nature of the changes made in a revision. According to semantic versioning, an increment of the patch level should only reflect bug fixes, the minor level is incremented when adding functionality which is backwards compatible, and the major version is only changed when introducing API-breaking changes. Within the last three years, several minor or patch level updates have been released for all engines, but no major release has been made. Here, we focus on the last four minor version updates and always use the highest available patch level[13]. Each of the releases are at least three month and on average six months apart. Therefore, we can expect to see signs of progress in BPMN feature implementation, given the implementation has not been concluded yet.

[12]See http://semver.org/ for details.

[13]The only exception to this rule is activiti 5.16.4 which contained a bug that severely hampered process deployment. Since the fix for this bug has not been backported to activiti 5.16.X, we used the previous patch level, namely, activiti 5.16.3.

Table III
THE BPMN ENGINES UNDER TEST

Version	Release Date	Newest Version for # Days
activiti		
5.18.0	2015-07-31	>92
5.17.0	2014-12-18	225
5.16.3	2014-09-17	92
5.15.1	2014-04-01	169
jBPM		
6.3.0	2015-09-28	>33
6.2.0	2015-03-09	203
6.1.0	2014-08-19	202
6.0.1	2014-05-14	97
camunda BPM		
7.3.0	2015-05-29	>155
7.2.0	2014-11-28	182
7.1.0	2014-03-31	242
7.0.0	2013-08-31	212

Each engine has a different release strategy, as can be seen in the release dates and the number of days between each release in Table III. Whereas activiti has a relatively short release cycle, both jBPM and camunda have longer release cycles. As we only use the last four minor version releases, we do not cover the same amount of time for each engine. This is not a problem, since we are interested in increasing feature support between releases, and not within a fixed period of time.

B. Evolution of Native BPMN Support

Since page space is limited, we cannot present the complete benchmark results for all engine versions. Instead, an overview of the changes in native BPMN support is shown in Table IV. Feature regressions, i.e., features that were supported in the earliest version, stopped working in an intermediate version, and resumed functioning in a subsequent version, are excluded from this overview.

Table IV
EVOLUTION OF NATIVE BPMN SUPPORT

Feature	Change	Engine
4 events	supported since	camunda BPM 7.1.0
2 events	supported since	camunda BPM 7.3.0
1 gateway and 4 events	supported since	jBPM 6.1.0

First of all, it becomes obvious that there is very little change in the support of BPMN features. Feature support for all three engines increased by the number of merely eleven additional features during the past three years. Especially when considering the immense number of potential features and the moderate degree to which engines implement the standard so far, as described in Section IV, this number is very low.

Regarding activiti, we do not see any evolution of native BPMN support. In the groups of *activities*, *basics*, *data*, and *errors*, we cannot diagnose any changes at all. There are minor improvements in the *gateways* and *events* group, but the changes are minimal. For instance, one test shifts from being undeployable to a partial support of some variants of a language feature, which does not justify to be judged as an increase in feature support. The only factual change that does happen is a regression in activiti 5.16.3 which is fixed in activiti 5.17.0 for a single event type.

In contrast to activiti, there are more differences between the tested versions of camunda BPM. However, the support for the *basics*, *data*, *errors*, and *gateway* groups stays the same for all versions. With the change from camunda BPM 7.0.0 to 7.1.0, four new features within the *events* group are supported and with the latest version 7.3.0, camunda BPM gains support for two additional features in the *events* group. A negligible change, similar to the changes for activiti, can be found in the *activities* group, between camunda BPM 7.0.0 and 7.1.0. Overall, six new BPMN events are supported in the latest version in contrast to the earliest version we have tested.

Looking at jBPM, we can see that all the versions have the same support for *activities*, *basics*, *data*, and the *errors* group. Upgrading jBPM 6.0.1 to 6.1.0 brought support for five new features, namely four new events and one gateway. Upgrading from jBPM 6.1.0 to 6.2.0, however, results in five regression errors as again four events and one gateway option fails. With jBPM 6.3.0, these failures are fixed again. To sum up, over the time of the last four minor versions, jBPM introduced five new features in total.

C. *Evolution of Workflow Control-flow Patterns Support*

An overview of the evolution of workflow control-flow pattern support is shown in Table V. As it can be seen, there are even fewer changes and these are not limited to an increase in feature support, but instead also include a decrease.

Table V
EVOLUTION OF PATTERN SUPPORT

Pattern	Change	Engine
WCP06	supported since	jBPM 6.1.0
WCP10	supported until	camunda BPM 7.1.0

For activiti, we have no visible improvements in pattern support over the last four minor releases. This can be expected, since there is also no increase in native language features. The only change is that processes which contain complex gateways are no longer marked as *not deployable*, affecting the variants of WCP06 and WCP09 that make use of complex gateways. This change happens between the versions 5.16.3 and 5.17.0. However, this has no effect on pattern support, as said pattern variants do not work as expected at runtime anyway. Regarding jBPM, we can see a single improvement in the variant of WCP06 that is implemented with an inclusive gateway. This is unsupported in jBPM 6.0.1 but supported in every subsequent version we have tested. Looking at camunda BPM, the support for workflow patterns even declines. While WCP10 is still supported in both versions 7.0.0 and 7.1.0, versions 7.2.0 and 7.3.0 lack support of this pattern.

D. *Summary*

Overall, it seems that the evolution of feature support in BPMN engines has stopped. Improvements to feature support are confined to BPMN events. as shown by increases for camunda BPM and jBPM. Fig. 2 visualizes this development. It can be seen that feature support is hardly rising. Interestingly, activiti, which supports the least number of BPMN events of the three engines, has not improved in that area within the last four releases.

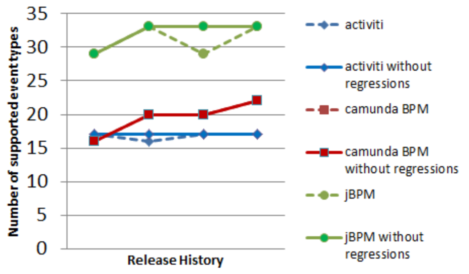

Figure 2. Evolution of Native BPMN Support for Events

This halt in feature improvements of these three major BPMN engines can be seen as a sign that the implementation of the BPMN standard has concluded at the current level. It seems that neither the implementers of these engines view the remaining features of the standard as valuable for their implementations, nor are they pressured by their customer base to implement further features. Otherwise, we would see a stronger increase in new features. Since all three engines are successful in the market, we can conclude that the feature set they provide is sufficient for BPM-systems in practice.

VI. Conclusion and Future Work

In this paper, we have presented a comprehensive analysis of the current state and the evolution of the implementation of the BPMN 2.0 standard. The results of this analysis can be summarized as follows: Evidence suggests that the implementation of the standard in practice has concluded at the current level. Remaining features of the standard that are not yet supported seem to be irrelevant in practice.

The first part of this study, presented in Section III, was a comprehensive analysis of products that claim to implement the BPMN 2.0 standard. Of the 45 products that classify as BPMS or process engines, only three remain that are able to consume the required serialization format and permit a closer evaluation. In Section IV, we performed such a closer evaluation of feature support provided by the most recent versions of these engines. We analyzed conformance to the BPMN standard and, on top of that, support for common workflow control-flow patterns. Following this analysis, we extended the study to three additional prior versions of the engines that have been published over the last three years in Section V. We could show that hardly any features have been added over this time. Hence, we can conclude that neither the implementers, nor the users of the standard are interested in the remaining features of the standard.

Multiple directions of future work follow from this. Firstly, it would be desirable to investigate further BPMN engines. We hope to obtain access to further products, but this depends on the availability of academic licenses. Secondly, we work on extending the coverage of our test suite. Currently, this test suite does not cover all aspects of the standard and a higher level of coverage would be desirable. With respect to the results presented in this paper, it can be expected that a higher level of feature coverage leads to a diagnosis of even more insufficiencies in contemporary implementations. The refinement of the standard can be seen as a third direction. Currently, the BPMN standard is quite extensive and studies that analyse its usage in practice, such as this one, consistently show that many of the features are simply not needed. A reduction of the feature set of BPMN to a core subset could help in many ways, e.g., by easing the implementation of the standard and reducing the complexity for its users. Similar approaches exist for related standards [26] and could be valuable for BPMN as well.

ACKNOWLEDGMENTS

We would like to thank the students of the University of Bamberg who contributed to the benchmarking system used in this paper during software development projects. Thanks go to Adrian Bazyli, Annalena Bentele, Christian Kremitzl, Lea-Louisa Maaß, Frederik Müller and Severin Sobetzko.

REFERENCES

[1] W. M. P. van der Aalst, “Business Process Management: A Comprehensive Survey,” *ISRN Software Engineering*, 2013.

[2] 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 CSUR*, vol. 43, no. 1, 2010.

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

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

[5] M. Chinosi and A. Trombetta, “BPMN: An introduction to the standard,” *Comp. Stand. & Inter.*, vol. 34, no. 1, pp. 124–134, 2012.

[6] B. Hofreiter and C. Huemer, “A Model-driven Top-down Approach to Inter-organizational Systems: From Global Choreography Models to Executable BPEL,” in *IEEE EEE*, 2008.

[7] I. Weber, J. Haller, and J. Mulle, “Automated Derivation of Executable Business Processes from Choreographies in Virtual Organisations,” *IJBPIM*, vol. 3, pp. 85–95, 2008.

[8] C. Ouyang, M. Dumas, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and J. Mendling, “From Business Process Models to Process-Oriented Software Systems,” *ACM TOSEM*, vol. 19, no. 2, 2009.

[9] A. Schönberger, “The CHORCH B2Bi Approach: Performing ebBP Choreographies as Distributed BPEL Orchestrations,” in *SC4B2B*, 2010.

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

[11] M. zur Muehlen and J. Recker, “How Much Language is Enough? Theoretical and Practical Use of the Business Process Modeling Notation,” in *CAiSE*, 2008.

[12] M. Geiger, S. Harrer, J. Lenhard, M. Casar, A. Vorndran, and G. Wirtz, “BPMN Conformance in Open Source Engines,” in *SOSE*, San Francisco Bay, CA, USA, March/April 2015.

[13] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, “Workflow Patterns,” *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, July 2003.

[14] C. Gutschier, R. Hoch, H. Kaindl, and R. Popp, “A Pitfall with BPMN Execution,” in *WEB*, 2014, pp. 7–13.

[15] F. Kossak, C. Illibauer, V. Geist, J. Kubovy, C. Natschläger, T. Ziebermayr, T. Kopetzky, B. Freudenthaler, and K.-D. Schewe, *A Rigorous Semantics for BPMN 2.0 Process Diagrams*. Springer, 2014, ISBN: 978-3-319-09930-9.

[16] M. Geiger and G. Wirtz, “BPMN 2.0 Serialization - Standard Compliance Issues and Evaluation of Modeling Tools,” in *EMISA*, 2013.

[17] M. Dirndorfer, H. Fischer, and S. Sneed, “Case Study on the Interoperability of Business Process Management Software,” in *S-BPM ONE*, 2013.

[18] M. Skouradaki, D. H. Roller, F. Leymann, V. Ferme, and C. Pautasso, “On the Road to Benchmarking BPMN 2.0 Workflow Engines,” in *ACM/SPEC ICPE*, 2015.

[19] K. Kluza, K. Kaczor, G. J. Nalepa, and M. Slazynski, “Opportunities for Business Process semantization in open-source process execution environments,” in *FedCSIS*. IEEE, 2015, pp. 1307–1314.

[20] Z. Merali, “Computational science: Error, why scientific programming does not compute,” *Nature*, vol. 467, no. 7317, pp. 775–777, 2010.

[21] C. Boettiger, “An introduction to Docker for reproducible research,” *ACM OSR*, vol. 49, no. 1, pp. 71–79, 2015.

[22] R. Chamberlain and J. Schommer, “Using Docker to Support Reproducible Research,” Invenshure, LLC, Tech. Rep., 2014.

[23] S. Harrer, J. Lenhard, and G. Wirtz, “BPEL Conformance in Open Source Engines,” in *SOCA*. IEEE, 2012, pp. 237–244.

[24] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and N. Russell, “On the Suitability of BPMN for Business Process Modelling,” in *BPM*, 2006, pp. 161–176.

[25] J. Lenhard, M. Geiger, and G. Wirtz, “On the Measurement of Design-Time Adaptability for Process-Based Systems,” in *SOSE*. IEEE, 2015.

[26] E. Højsgaard and T. Hallwyl, “Core BPEL: Syntactic Simplification of WS-BPEL 2.0,” in *ACM SAC*, 2012, pp. 1984–1991.