Open Source Versus Proprietary Software in Service-Orientation: The Case of BPEL Engines

Simon Harrer, Jörg Lenhard, and Guido Wirtz

Distributed Systems Group, University of Bamberg, Germany
{simon.harrer,joerg.lenhard,guido.wirtz}@uni-bamberg.de

Abstract. It is a long-standing debate, whether software that is developed as open source is generally of higher quality than proprietary software. Although the open source community has grown immensely during the last decade, there is still no clear answer. Service-oriented software and middleware tends to rely on highly complex and interrelated standards and frameworks. Thus, it is questionable if small and loosely coupled teams, as typical in open source software development, can compete with major vendors. Here, we focus on a central part of service-oriented software systems, i.e., process engines for service orchestration, and compare open source and proprietary solutions. We use the Web Services Business Process Execution Language (BPEL) and compare standard conformance and its impact on language expressiveness in terms of workflow pattern support of eight engines. The results show that, although the top open source engines are on par with their proprietary counterparts, in general proprietary engines perform better.

Keywords: open source, SOA, BPEL, patterns, conformance testing

1 Introduction

The comparison of open source and proprietary software is a topic that, depending on the audience, can quickly turn from a moderate discussion to a heated debate. Although it has been investigated a number of times, see for instance [10, 15, 26, 27], a definite answer is seldom found. Studies often focus on software such as operating systems [26] and despite the wide academic interest in such comparisons, little work on comparing open source and proprietary service-oriented software can be found. In the services ecosystem, highly specialized and inherently complex software that differs from operating systems in nature and level of abstraction prevails. Especially when it comes to middleware, vendors established large projects and created highly priced products and it is unclear whether open source alternatives can compete. This makes the comparison of open and proprietary service-oriented software especially interesting. What is more, direct comparisons of open source and proprietary software are typically impeded by the fact that truly comparable software is hard to find [26, p.260]. In the area of service-oriented computing, however, there exists a large set of detailed international standards that describe required functionality for Web

Services. Hence, a meaningful and precise comparison becomes feasible in this area. Here, we compare service-oriented middleware for service orchestration [23], in particular BPEL engines.

The BPEL 2.0 specification [20] defines a language to implement executable processes as interactions of Web Services in XML. A series of control- and data-flow activities are used to specify the order of the interactions, thereby orchestrating invoked Web Services [23]. A typical example for a BPEL process is that of a travel agency service which books accommodation, flight and transportation in a single transaction by reusing multiple external services. BPEL is tightly integrated into the Web Services ecosystem and relies heavily on other standards, e.g., the Web Service Description Language (WSDL), XPath, and SOAP. As such, BPEL is a natural choice for implementing processes within a Web Services-based service-oriented architecture (SOA) [22]. It is frequently used in scenarios such as business-to-business integration where multiple partners participate in cross-organizational business processes. Within such scenarios, service choreographies define the global perspective of a shared process by specifying the public collaboration protocol, whereas orchestrations implement the local perspective of a single partner [23]. BPEL particularly fits to implement these orchestrations due to its inherent usage of vendor-independent technologies such as Web Services and XML, and has been used in industry standards, for example [24], and various studies, e.g., [9, 11].

These approaches rely, among other criteria, on the existence of fully conformant BPEL engines, which provide two of the key selling points of BPEL, namely platform independence and portability of process definitions. Therefore, standard conformance is a highly relevant selection criterion for projects in industry or academia that leverage BPEL. For this reason, we use standard conformance as the central comparison factor in this study. A comparison of additional quality factors, such as performance, is also valuable, but we defer this to future work. Directly related to standard conformance and potentially more insightful is the factor of expressive power. Expressive power refers to the ease with which concepts typically needed in a language or system can be expressed in that system. The more concepts supported and the easier they can be expressed, the more suitable the system is. For process languages such as BPEL, expressiveness is typically measured in terms of workflow pattern support [30]. Such patterns are derived from real world systems and usage scenarios, describing features of processes that are repeatably used. Thereby, patterns can provide meaningful insights into the capabilities of engines. If an engine is not fully standard conformant and lacks several features, it may also suffer from a reduction of expressiveness. Hence, we consider expressiveness of engines as the second comparison factor. In previous work [8], we evaluated the standard conformance of five open source engines and could show that it varies strongly among them. Here, we complement this evaluation by extending the comparison to a) proprietary engines and b) workflow pattern support. These proprietary engines often claim to excel in terms of performance and are part of large and optimized middleware solutions. We evaluate three proprietary engines and contrast the results to [8]. Furthermore, we present a test suite for automatically evaluating workflow patterns [30] support to determine the effects of standard conformance of engines on their expressiveness. In summary, we pose two research questions:

RQ1: Do proprietary BPEL engines outperform open source engines in terms of standard conformance?

RQ2: How do variances in standard conformance influence the expressiveness of the language subsets supported by the engines?

The rest of the paper is structured as follows: First, we discuss related work and, thereafter, outline our testing approach with a focus on the testing of expressiveness. In section 4, we evaluate, analyze, and discuss the test results and their implications, and answer both research questions. Last, a summary and an outlook on future work is given in section 5.

2 Related Work

Related work is subdivided into four different areas: i) alternative process and workflow languages and systems, ii) testing and verification of BPEL, iii) evaluations of the expressiveness of process languages using patterns, and iv) approaches for comparing the quality of open versus proprietary software.

i) Process languages: Although BPEL has received immense attention in the last decade, there are a variety of other process or workflow languages and engines. Yet Another Workflow Language (YAWL) [29] is a formally defined workflow language based on Petri nets with direct support for workflow patterns [30]. At this time, only one implementation, namely, the YAWL workflow engine exists. Another notable competitor to BPEL is the Windows Workflow Foundation [4]. For this language, there is also only a single implementation, but it is closed source and does not ship with an accompanied specification. In recent years, the Business Process Model and Notation (BPMN) 2.0 [21] has gained rising attention. Although the focus of BPMN resides on its visual notation for business processes, it ships with a mapping to BPEL 2.0 [21, pp. 445–474]. Today, several implementations of BPMN have arrived. However, nearly all of them only provide *modeling conformance* [21, p.1], meaning they can be used for visualization, but not *execution conformance*, required for constructing executable processes. What is more, the BPMN specification offers a lot of room for interpretation concerning executable processes models. This makes it more easily adaptable to a different technological context, as opposed to BPEL which is tailored to Web Service orchestration, but complicates the construction of processes that can be executed on more than a single engine [6]. Last, the XML Process Definition Language (XPDL) 2.2 [31] from the Workflow Management Coalition (WfMC) is a serializable meta model to exchange process definitions between different products. As opposed to BPEL, XPDL includes and serializes visual information and is well suited for exchanging BPMN models but does not provide execution semantics. In summary, we focus on BPEL here, as it provides, in contrast to other process languages, precisely defined execution semantics, as well as a variety of open source and proprietary implementations, which are directly comparable.

ii) Testing of BPEL: Testing and verification of SOAs and Web Services has been extensively studied. See for instance [3] for a comprehensive overview. When it comes to conformance testing, a distinction has to be made between approaches that assert the conformance of concrete services, possibly implemented in BPEL, to a communication protocol, such as [5, 14], and the testing of an engine to a language specification, which we do here. Concerning BPEL, research primarily focuses on unit testing, performance testing, or formal verification of BPEL processes, and not engines. When it comes to unit testing, BPELUnit is of most importance [17]. Performance testing approaches for services, such as SOABench [1], which also benchmarks BPEL engines, and GENESIS2 [13], are based on generating testbeds from an extensible domain model which can then be used to gather performance metrics. Here, we conduct standard conformance testing, thus, instead of testing the correctness of a BPEL process, we test the correctness of a BPEL engine, and focus on different kinds of metrics. We build upon the tool *betsy*[1], which we also use in [8], but extend it with capabilities for testing several proprietary engines and a test suite for the evaluation of workflow pattern support.

iii) Expressiveness and patterns: Workflow patterns aim to "provide the basis for an in-depth comparison of a number of commercially available workflow management systems" [30, p.5]. Patterns capture a distinct feature or piece of functionality that is frequently needed in workflows and processes and which should therefore be supported as directly as possible in process languages. The more patterns a language or system supports, the more expressive it is. The original pattern catalog [30] consists of 20 workflow control-flow patterns (WCPs) which are subdivided into *basic* control-flow, *advanced branching and synchronization*, *structural*, *state-based*, *cancellation*, and *multi-instance* patterns. Although the appropriateness of these patterns is not undisputed [2], they have been extensively used for benchmarking, designing and developing languages and systems, as demonstrated by a large array of additional pattern catalogs and studies, for instance [16, 19, 28]. For this reason, the usage of the workflow patterns here facilitates the comparison of this study to related work. The Workflow Patterns Initiative[2] already provides evaluations of the expressiveness of BPEL 1.1 and two proprietary BPEL 1.1 engines. Here, we focus on BPEL 2.0 only, as it is the latest published version of the standard for six years. In [19], multiple pattern catalogs, including the workflow patterns, have been implemented for WF, BPEL 2.0 and the BPEL 2.0 engine OpenESB. We base our work on [19] by adapting these BPEL 2.0 implementations of the original 20 workflow patterns to allow for an automatic and repeatable benchmark of eight BPEL engines and thereby evaluate the effects of BPEL standard conformance on language expressiveness.

[1] This tool can be found at `https://github.com/uniba-dsg/betsy`.

[2] See the project page at `http://www.workflowpatterns.com/`.

iv) Comparing quality: Software quality comparisons typically focus on *internal quality* [26, 27], for instance by computing and comparing source code metrics for different pieces of software, or *external quality* [15], by investigating the usefulness of the software for its end users. Here, we do not look at source code, which in case of proprietary engines is not available. Instead, we focus on conformance as an external quality attribute. Conformance determines the degree to which prescribed functionality is available to the users of an engine. This has a direct effect on the kinds of patterns that can be implemented on an engine, and thereby its expressiveness. The more standard-conformant an engine is, and the more patterns it directly supports, the higher its external quality is.

3 Testing Approach

To be able to compare different engines and answer the research questions, we need a mechanism for an in-depth and isolated analysis of each engine. The approach for achieving this kind of analysis is described in this section. Thereafter, the results are aggregated for the comparison in the following section.

The testing approach consists of the testing setup in general and the expressiveness test suite in particular. Within the testing setup, we list the engines under test, elaborate on the standard conformance test suite, and the steps of a typical test run.

3.1 Testing Setup

Our testing setup is an adapted and extended version of the setup proposed in [8] which relies on the publicly available testing tool betsy. The tool can be used to automatically manage (download, install, start and stop) several open source engines. Moreover, it provides a conformance test suite which comprises more than 130 manually created test cases, in the form of standard-conformant BPEL processes. Since the publication of [8], three open source engines received major or minor updates. Here, we updated betsy to use the latest versions of these engines, namely, Apache ODE 1.3.5, bpel-g 5.3, OpenESB v2.3, Orchestra 4.9.0 and Petals ESB 4.1[3]. Furthermore, we added support for testing the conformance of three proprietary BPEL engines. These engines come from major global SOA middleware vendors that also participated in crafting the BPEL specification. Due to licensing reasons, we cannot disclose the names of the proprietary engines, and, therefore, refer to them as $P1$, $P2$ and $P3$.

The tests are derived manually from the normative parts of the BPEL specification which are indicated with the keywords MUST and MUST NOT, as defined in [12]. The test suite is subdivided into three groups resembling the structure of the BPEL specification, namely, basic activities [20, pp. 84–97] (e.g., assign, empty, exit, invoke, and receive), scopes [20, pp. 115–147] (e.g., fault-, compensation-, and termination handlers) and structured activities [20, pp. 98–114] (e.g., if, while, flow, and forEach). The various configurations of the BPEL

[3] Download links available at `https://github.com/uniba-dsg/betsy#downloads`

activities of each group form the basis of the test cases, including all BPEL faults. Hence, every test case of the standard conformance test suite asserts the support of a specific BPEL feature. Every test case consists of a test definition, the BPEL process and its dependencies (WSDL definitions, XML Schemas, etc.), and a test case configuration, specifying the input data and assertions on the result. All processes are based upon the same process stub, which is shown in Listing 1, and implement the same WSDL interface containing a one-way and two request-response operations for exchanging basic data types via the document/literal binding over HTTP[4]. To assert the correctness of the process execution, each test must provide observable output, which is implemented via a receive-reply pair.

```
<process>
    <partnerLinks/>
    <variables/>
    <sequence>
        <receive createInstance="true" />
        <!-- feature under test -->
        <assign /> <!-- prepare reply message -->
        <reply />
    </sequence>
</process>
```

Listing 1. Process stub for conformance tests adapted from [8, p.4]

The tests aim at checking the conformance of a feature in isolation. This is not completely possible, as the basic structure depicted in Listing 1 and basic input and output is always required, otherwise the correctness of a test cannot be asserted. However, all features in the stub could be verified to work in a basic configuration on all engines and therefore have no impact on the test results.

During a full test run, our tool automatically converts the engine independent test specifications to engine specific test cases and creates required deployment descriptors as well as deployment archives. Next, these archives are deployed to the corresponding engines and the test case configurations are executed. At first, every test case configuration asserts successful deployment by determining whether the WSDL definition of the process has been published. Next, the different test steps are executed, sending messages and asserting the responses by means of correct return values or expected SOAP faults. When all test cases have been tested, an HTML report is created from the test results.

The quality and correctness of the conformance test cases were ensured by validating them against the XML Schema files of their specifications, e.g., BPEL 2.0, WSDL 1.1 and XML Schema 1.1, and by reviewing them within our group. In addition, all test cases are publicly available and, as a result, already have been improved by the developers of two of the engines, Apache ODE and bpel-g. Finally, only 4 of the test cases fail on all engines, hence, approx. 97% of all test cases succeed on at least one engine, indicating their correctness.

[4] This is the preferred binding for achieving interoperability, as defined by the WS-I Basic Profile 2.0.

3.2 Pattern Test Suite

Table 1 shows the test case implementations for the automatic testing of workflow patterns support of the original 20 workflow patterns from [30]. According to [30] and related studies, a pattern is *directly* supported (denoted as +) in a language or system, if at least one direct solution using a single language construct (activity in BPEL) for the pattern can be found. If the solution involves a combination of two constructs, it is counted as *partial support* (denoted as $+/-$) for the pattern, otherwise, there is *no direct support* (denoted as $-$). The BPEL 2.0 column in Table 1 shows the workflow patterns support by the specification [19][5].

Table 1. List of workflow patterns from [30] along with number of test cases and degree of support

Basic Control Flow Patterns		BPEL 2.0	Tests
WCP01	Sequence	+	1
WCP02	Parallel Split	+	1
WCP03	Synchronization	+	1
WCP04	Exclusive Choice	+	1
WCP05	Simple Merge	+	1
Advanced Branching and Synchronization Patterns		BPEL 2.0	Tests
WCP06	Multi-Choice	+	2
WCP07	Synchronizing Merge	+	2
WCP08	Multi-Merge	-	0
WCP09	Discriminator	-	0
Structural Patterns		BPEL 2.0	Tests
WCP10	Arbitrary Cycles	-	0
WCP11	Implicit Termination	+	1
Patterns with Multiple Instances		BPEL 2.0	Tests
WCP12	Multiple Instances Without Synchronization	+	3
WCP13	Multiple Instances With a Priori Design Time Knowledge	+	2
WCP14	Multiple Instances With a Priori Runtime Knowledge	+	1
WCP15	Multiple Instances Without a Priori Runtime Knowledge	-	0
State-based Patterns		BPEL 2.0	Tests
WCP16	Deferred Choice	+	1
WCP17	Interleaved Parallel Routing	+/-	1
WCP18	Milestone	+/-	1
Cancellation Patterns		BPEL 2.0	Tests
WCP19	Cancel Activity	+/-	1
WCP20	Cancel Case	+	1

The BPEL implementations used here are adopted from [18, 19] and modified to be automatically testable with betsy, that is, to use the same WSDL definition and partner service as the conformance test suite. The pattern implementations work similar to the other test cases and are based on the process stub presented in Listing 1. Each pattern test case contains an implementation

[5] The pattern support evaluation from [19] differs from [25] which evaluates BPEL 1.1. Please refer to [19] or the technical report [18] for explanatory details.

of a workflow pattern in BPEL. Given an engine successfully deploys the process and returns the asserted result on invocation, it demonstrates that it supports the related workflow pattern. Four of the patterns, *Multi-Merge*, *Discriminator*, *Arbitrary Cycles* and *Multiple Instances Without a Priori Runtime Knowledge*, are left untested. These patterns cannot be implemented directly in BPEL (i.e., they would require the usage of too many constructs), due to the structuredness of its control-flow definition and the inability to create cycles using links [19]. Moreover, the tests for three patterns, *Interleaved Parallel Routing*, *Milestone*, and *Cancel Activity* provide at most partial support, as there is no single activity in BPEL that directly implements these patterns. Four of the patterns, *Multi-Choice*, *Synchronizing Merge*, *Multiple Instances Without Synchronization* and *Multiple Instances With a Priori Design Time Knowledge* are implemented in more than one test case. The reason for this is that multiple alternative implementations of the pattern, with a differing degree of support, are available in BPEL. For instance, the *Multi-Choice* pattern is typically implemented in BPEL using `links` in a `flow` activity to activate different parallel control-flow paths at the same time. However, `links` are not supported by all engines under test and these engines would consequently fail to support that pattern. An alternative implementation of the *Multi-Choice* pattern that grants partial support can be achieved by nesting multiple `if` activities in a `flow` activity. By including such alternative tests, we can provide a precise classification of all engines under test.

In summary, if an engine passes a test case, it provides either direct or partial support depending on the type of the test case. If there are multiple test cases, the engine is granted the degree of support of the most direct test case.

4 Results and Implications

In this section, we present the test results of a full test run evaluating the standard conformance and the expressiveness test suite on both, proprietary and open source engines[6]. The results are subdivided into standard conformance results in Table 2 and expressiveness evaluations in Table 3. In the following, we first discuss the conformance results of the three proprietary engines. Next, we compare these to the results of the five open source engines and discuss the implications of this comparison. Last, based on workflow pattern support, the effects of standard conformance on expressiveness are evaluated and presented for both engine types.

4.1 Commercial Engines

The conformance results in Table 2 consist of the aggregated number of successful tests per BPEL activity for each engine as well as the percentage and average values per engine, engine type, and activity group. In addition, the *deployment*

[6] We executed this test run on a Windows 7 64 bit system with 16 GB of RAM and an i7-2600 processor.

rate, the percentage of successfully deployed test cases, is given at the bottom of the table.

Engine P1: Engine $P1$ ranks first place when compared to the other proprietary products and conforms to the BPEL specification to a degree of 92%, failing only in eleven of 131 test cases. Nine of these failed tests concern basic and two concern structured activities, respectively. Language features related to scopes are fully supported. The major shortcomings of this engine lie in fault handling and detection. Faults that are expected to be thrown under certain circumstances are not thrown. For example, the engine does not throw the expected `invalidExpressionValue` fault when the `startCounter` of the `forEach` activity is too high or its `completionCondition` is negative. In addition, XSL transformations and the invocation of Web Service operations that do not expect input are not implemented in a standard-conformant fashion.

Table 2. Number of successfully passed conformance tests, aggregated by activity, group, and engine

	Prop. E.				Open Source Engines							
Activity	P1	P2	P3	Ø	bpel-g	ODE	OpenESB	Ø	Orch.	Petals	Ø	Σ
Basic Activities												
Assign	15	7	15		15	10	13		11	8		19
Empty	1	1	1		1	1	1		1	1		1
Exit	1	1	1		1	1	1		1	1		1
Invoke	11	6	7		11	7	3		8	5		12
Receive	4	3	3		4	3	1		1	1		5
ReceiveReply	8	6	6		8	5	6		5	1		11
Rethrow	3	0	1		3	2	1		0	0		3
Throw	5	0	4		5	5	4		0	0		5
Validate	2	0	2		2	0	2		0	0		2
Variables	3	1	1		3	2	2		1	1		3
Wait	3	2	3		3	3	3		2	1		3
Σ	56	27	44		56	39	37		30	19		65
	86%	41%	68%	65%	86%	60%	57%	68%	46%	29%	56%	
Scopes												
Compensation	5	5	5		5	4	5		2	0		5
CorrelationSets	2	0	2		2	2	1		0	0		2
EventHandlers	8	5	7		8	6	6		6	0		8
FaultHandlers	6	5	6		6	6	6		2	5		6
MessageExchanges	3	1	1		3	1	1		1	0		3
PartnerLinks	1	0	1		1	1	1		1	0		1
Scope-Attributes	3	2	3		3	2	3		1	1		3
TerminationHandlers	2	0	0		2	0	2		2	0		2
Variables	2	2	2		2	2	2		2	0		2
Σ	32	20	27		32	24	27		17	6		32
	100%	63%	84%	82%	100%	75%	84%	86%	53%	19%	66%	
Structured Activities												
Flow	9	6	7		9	9	2		7	0		9
ForEach	9	4	6		9	3	9		0	2		11
If	5	4	4		5	4	4		4	4		5
Pick	5	5	5		5	5	4		4	1		5
RepeatUntil	2	2	2		2	1	2		2	0		2
Sequence	1	1	1		1	1	1		1	1		1
While	1	1	1		1	1	1		1	1		1
Σ	32	23	26		32	24	23		19	9		34
	94%	68%	76%	79%	94%	71%	68%	77%	56%	26%	62%	
Σ of Σ	120	70	97		120	87	87		66	49		131
	92%	53%	74%	73%	92%	66%	66%	75%	50%	26%	62%	
Deployment Rate	98%	88%	100%		98%	92%	100%		95%	57%		

Engine P2: The second engine is the lowest-ranking proprietary product, as it only supports roughly half of the test cases. Although it supports approximately two third of the scope and structured activity test cases, less then half of the basic activities are implemented correctly. This is mostly due to the way faults are propagated in engine $P2$. If a fault is not handled at root level, the process fails silently and does not propagate the fault to external callers that are still waiting for a response. Thus, an external caller gets no hint on the cause of the error. This contradicts common fault handling principles known in higher level programming languages and hampers distributed fault handling [7]. As a consequence, $P2$ fails all fault related tests, e.g., for the `throw`, `rethrow`, and `validate` activities, the handling of incoming faults from invoked services, and tests for standard BPEL faults. This amounts to ten tests for structured activities, two for scopes and 30 for basic activities, and adds up to 32% of all tests in total. Another factor that impacts its standard conformance rating is its deployment rate. Twelve percent of the standard-conformant test cases are rejected by $P2$ during deployment resulting in an upper bound of 88% for its conformance rating. Furthermore, multiple features of the `assign` activity seem to be unimplemented: the XPath extension functions for BPEL, `getVariableProperty` and `doXslTransform`, the assignment of a `partnerLink` or a `property`, as well as the `keepSrcElementName` attribute of a `copy` element. `Invoke` activities cannot be used with `correlationSets` or empty messages, and the same applies to embedded fault handlers. Engine $P2$ does not implement `terminationHandlers` and the definition of `correlationSets` or `partnerLinks` at the level of a `scope` is unsupported, although both constructs work when used on the `process` level. Event handling is supported in a basic fashion. However, the `onEvent` activity does not support the `fromParts` syntax and the `onAlarm` activity does not support the `until` element. The initiation of a `correlationSet` with an asynchronous operation leads to a failure to correlate on this set in an `onEvent` message handler. If used within a `scope` or at root level, `faultHandlers` work in most cases. The only exception to this is the failure to catch a fault that carries additional fault data using a `faultElement`. The `forEach` activity is implemented but lacks support for configuration related to the `completionCondition`. In a similar fashion, the `flow` activity supports links, but no conditional activation with `joinConditions`.

Engine P3: The proprietary engine with the second highest degree of standard conformance, successfully completing 74% of the tests, is engine $P3$. It supports all structured activities in their basic configuration but fails to support several special cases, such as `links` in a `flow` activity that use `joinConditions`, and `forEach` activities that use a `completionCondition` with `successful-BranchesOnly`. In addition, the `forEach` activity is always executed sequentially even if the `parallel` attribute is set. $P3$ does not support `terminationHandlers`, throwing or re-throwing `faultData`, the `keepSrcElementName` option of the `copy` element and the specification of `toParts` for messaging activities. Moreover, embedded fault- or compensation handlers for the `invoke` activity are not supported. Finally, the remaining tests fail, because certain standard faults, such as `correlationViolation` or `missingReply`, are not thrown as required.

4.2 Comparison of Proprietary and Open Source Engines

This section compares the standard conformance and its effects on expressiveness of open source engines with that of proprietary engines and answers the two research questions posed in the introduction. The results of the five open source engines presented in Table 2 vary slightly from previous analyses [8], because the engines under test as well as the conformance test suite were updated.

Proprietary engines successfully pass between 53% and 92% of the conformance tests. For open source engines, these numbers vary from 26% to 92%. On average, proprietary engines pass 73% of the conformance test suite, whereas the open source engines only achieve 62%. We used a binomial test to verify if this difference is significant. We tested if the number of successfully passed tests for open source engines is equal to the corresponding value for proprietary engines at a significance level of 5%. With a p-value of $2.5e^{-9}$, this hypothesis can be safely rejected in favour of the alternative: Open source engines pass significantly less tests than their counterparts. A reason for this observation may be that our test set of open source engines includes engines that could be considered experimental or premature. This is supported by the fact that the lowest ranking engine only deploys 57% of the tests and passes 26%. But because we lack market data on engine usage, we are unable to make a clear distinction on this issue. The overall situation changes, however, when looking at the top three open source engines which, to our experience, also are the ones most widely used in practice. Considering the top three open source engines, standard conformance ranges at 75%, two percentage points above the corresponding value for proprietary engines. Using binomial tests as before, we could confirm that there is no significant difference between the proprietary and the top three open source engines. The number of successful tests is clearly not lower (p-value of 0.81), but also not significantly higher (p-value of 0.23) for open source engines. In summary, based on this data, the answer to $RQ1$, whether proprietary engines outperform open source ones, has to be confirmed. In total, proprietary engines provide a higher degree of support, although the difference balances when only considering mature open source engines.

Table 3 details the results for workflow pattern support based on the expressiveness test suite using the trivalent rating described in section 3.2. Insights on pattern support can be gained by comparing the engines with the workflow pattern support of BPEL 2.0 shown in the BPEL column. We consider the number of times an engine is *compliant* to BPEL (i.e., the engine has the same degree of pattern support as BPEL), the engine *deviates* (i.e., the engine only provides partial support while BPEL directly supports the pattern) and the engine fails to directly support the pattern, in relation to the total number of patterns. We exclude the four patterns that cannot be implemented directly in BPEL from these calculations, as we cannot diagnose support for them in the first place. The results show that compliance to BPEL in pattern support ranges from 56% to

Table 3. Workflow patterns support per engine, aggregated by pattern, pattern group and engine

		Comm. Eng.			Open Source Engines					
Pattern	BPEL	P1	P2	P3	bpel-g	ODE	OpenESB	Orch.	Petals	
Basic Control-Flow Patterns										100%
WCP01 Sequence	+	+	+	+	+	+	+	+	+	100%
WCP02 Parallel Split	+	+	+	+	+	+	+	+	+	100%
WCP03 Synchronization	+	+	+	+	+	+	+	+	+	100%
WCP04 Exlusive Choice	+	+	+	+	+	+	+	+	+	100%
WCP05 Simple Merge	+	+	+	+	+	+	+	+	+	100%
Advanced Branching and Synchronization Patterns										88%
WCP06 Multi-Choice	+	+	+	+	+	+	+/-	+	+/-	75%
WCP07 Synchronizing Merge	+	+	+	+	+	+	+/-	+	+/-	75%
Structural Patterns										100%
WCP11 Implicit Termination	+	+	+	+	+	+	+	+	+	100%
Patterns with Multiple Instances										50%
WCP12 MI Without Sync.	+	+	+	+/-	+	+	+/-	+/-	+/-	50%
WCP13 MI W. Design T. Know.	+	+	+	-	+	+	+/-	+/-	+/-	50%
WCP14 MI W. Runtime Know.	+	+	+	-	+	+	-	-	-	50%
State-based Patterns										90%
WCP16 Deferred Choice	+	+	+	+	+	+	+	+	+	100%
WCP17 Interl. Parallel Routing	+/-	+/-	+/-	+/-	+/-	+/-	-	-	-	63%
WCP18 Milestone	+/-	+/-	+/-	+/-	+/-	+/-	+/-	-	-	75%
Cancellation Patterns										100%
WCP19 Cancel Activity	+/-	+/-	+/-	+/-	+/-	+/-	+/-	+/-	+/-	100%
WCP20 Cancel Case	+	+	+	+	+	+	+	+	+	100%
compliance		100%	100%	81%	100%	100%	63%	69%	56%	84%
deviation				6%			25%	13%	25%	9%
no direct support				13%			13%	19%	19%	8%
Ø of compliance per engine group			94%			88%		72%		

100% for open source engines, wheras proprietary engines excel their competitors with support ranging from 81% up to 100%. Two open source engines, Apache ODE and bpel-g, and two proprietary engines, $P1$ and $P2$, are completely compliant to BPEL in terms of support. $P3$ ranks second, whereas the remaining open source engines Orchestra, OpenESB, and PetalsESB come last.

All engines share the degree of support with BPEL for nine patterns (WCP01-WCP05, WCP11, WCP16, WCP19-20). For three patterns, several engines deviate from BPEL, whereas five patterns are not directly supported by at least one engine. As shown in the right-most column in Table 3, engines comply with BPEL for the basic control-flow patterns, the structural patterns, and the cancellation patterns. Patterns with multiple instances show most deviations and only in 50% of the cases, the engines achieve the same rating as BPEL. For advanced branching and synchronization and state-based patterns, engines provide 88% and 90% of compliance, respectively. What is more, support for advanced branching and synchronization patterns is in place for all engines, although two open source engines only support the patterns using a workaround solution. A similar situation applies to the the *Multiple Instances* patterns, where two patterns, WCP12 and WCP13, can be implemented by workaround solutions by three engines. One proprietary engine fails to support two of the *Multiple Instances* patterns (WCP13 and WCP14), and deviates from BPEL for the third *Multiple Instance* pattern (WCP12). Several open source engines also fail to support three patterns, namely, the *Multiple Instances With a Priori Runtime Knowledge* pattern (WCP14), the *Interleaved Parallel Routing* pattern (WCP17) and the *Milestone* pattern (WCP18). Interestingly, the patterns for which open source and proprietary engines deviate from BPEL are almost disjunctive, only WCP14 is not directly supported in both groups on at least one engine. Moreover, WCP14 is the least supported pattern as the corresponding test case fails on four engines. In total, the proprietary engines implement more workflow patterns (94%) than their open source counter parts (72%). As before, when comparing the three proprietary engines with the top three open source engines, this difference shrinks to an insignificant level (94% vs. 88%). The proprietary engines provide no direct support in two cases and deviate from BPEL in one case, whereas the top three open source engines provide no direct support in two cases and deviate from BPEL in four cases. These results reinforce the answer to $RQ1$.

The 21 cases of deviation from BPEL are caused by a lack of support for the `flow` activity in combination with `links` (six times), the `forEach` activity (three times) in combination with parallel execution (nine times), message correlation (twice) and isolated scopes (once). These results let us answer $RQ2$, concerning the impact of standard conformance on workflow pattern support. All in all, 18 cases of deviation (or 86% of the deviations) are a result of the lack of a standard conformant implementation of the `flow` and the `forEach` activity. Put differently, the lack of truly parallel execution in an engine is the biggest obstacle to pattern support. Nevertheless, the impact of standard conformance on pattern support seems little. Apache ODE, with 66% of successful conformance tests, supports all workflow patterns that can be directly implemented in BPEL. Even the worst engine in terms of standard conformance, PetalsESB with only 26% of successful conformance tests, provides direct or partial support for 13 out of 16 workflow patterns (81%). To frame an answer to $RQ2$: *Workflow patterns can be directly implemented with only a moderate degree of standard conformance, but support for truly parallel execution of activities is a decisive factor.*

5 Conclusion and Future Work

In this paper, we presented a comparison of open source and proprietary BPEL engines in terms of standard conformance and language expressiveness. The results demonstrate, that proprietary engines provide a slightly higher degree of standard conformance and language expressiveness than their open source counterparts, and thus are of higher quality. This observation changes when considering the top three open source engines which are equal to their proprietary counterparts. The effect of standard conformance on language expressiveness turned out to be moderate, although parallel execution is a crucial factor.

Future work comprises two aspects: i) adding additional conformance and expressiveness test suites to get a more precise picture and ii) enhancing the test suite for testing other criteria, to provide a more comprehensive comparison of open source and proprietary products. Firstly, the BPEL specification [20, appendix B] contains a list of 94 static analysis rules specifying which BPEL processes must be rejected by a standard conformant engine. A test suite based on these rules that helps to verify if erroneous processes are correctly rejected would be desirable. Concerning language expressiveness, as outlined in section 2, many additional pattern catalogs do exist for which automatic testing would be beneficial. Secondly, in addition to standard conformance and expressiveness, performance is also a very important selection criteria for a process engine and a major quality criterion. The existing infrastructure could be used to provide valuable insights on the performance of certain activities and combinations thereof, as well as of workflow pattern implementations.

References

1. 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 World Wide Web Conference (WWW)*, pages 141–150, Raleigh, North Carolina, USA, April 2010.
2. E. Börger. Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL. *Software & Systems Modeling*, 11(3):305–318, 2012.
3. M. Bozkurt, M. Harman, and Y. Hassoun. Testing & Verification In Service-Oriented Architecture: A Survey. *Software Testing, Verificaton and Reliability*, 00:1–7, May 2012.
4. B. Bukovics. *Pro WF: Windows Workflow in .NET 4*. Apress, June 2010. ISBN-13: 978-1-4302-2721-2.
5. M. Geiger, A. Schönberger, and G. Wirtz. Towards Automated Conformance Checking of ebBP-ST Choreographies and Corresponding WS-BPEL Based Orchestrations. In *23rd International Conference on Software Engineering and Knowledge Engineering, Miami, Florida, USA*. KSI, 7.-9. July 2011.
6. 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.
7. C. Guidi, I. Lanese, F. Montesi, and G. Zavattaro. On the Interplay Between Fault Handling and Request-Response Service Interactions. In *8th International Conference on Application of Concurrency to System Design (ACSD)*, pages 190–198, Xi'an, China, June 2008.
8. 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, December 17-19 2012. IEEE.
9. S. Harrer, A. Schönberger, and G. Wirtz. A Model-Driven Approach for Monitoring ebBP BusinessTransactions. In *Proceedings of the 7th World Congress on Services 2011 (SERVICES2011), Washington, D.C., USA*. IEEE, July 2011.
10. J. Hoepman and B. Jacobs. Increased Security Through Open Source. *Communications of the ACM*, 50(1):79–83, January 2007.
11. B. Hofreiter and C. Huemer. A model-driven top-down approach to inter-organizational systems: From global choreography models to executable BPEL. In *Join Conf CEC, EEE*, Hong Kong, China, 2008.
12. IETF. *Key words for use in RFCs to Indicate Requirement Levels*, March 1997. RFC 2119.

13. L. Juszczyk and S. Dustdar. Programmable Fault Injection Testbeds for Complex SOA. In P. Maglio, M. Weske, J. Yang, and M. Fantinato, editors, *Service-Oriented Computing*, volume 6470 of *Lecture Notes in Computer Science*, pages 411–425. Springer Berlin Heidelberg, 2010.
14. K. Kaschner. Conformance Testing for Asynchronously Communicating Services. In G. Kappel, Z. Maamar, and H. Motahari-Nezhad, editors, *Service-Oriented Computing*, volume 7084 of *Lecture Notes in Computer Science*, pages 108–124. Springer Berlin Heidelberg, 2011.
15. J. Kuan. Open Source Software as Lead User's Make or Buy Decision: A Study of Open and Closed Source Quality. In *Proceedings of the 2nd Conference on The Economics of the Software and Internet Industries*, Toulouse, France, January 2003.
16. A. Lanz, B. Weber, and M. Reichert. Workflow Time Patterns for Process-Aware Information Systems. In *Enterprise, Business-Process, and Information Systems Modelling: 11th International Workshop BPMDS and 15th International Conference EMMSAD in conjunction with CAiSE*, pages 94–107, Hammamet, Tunisia, June 2010.
17. D. Lübke. Unit Testing BPEL Compositions. In L. Baresi and E. D. Nitto, editors, *Test and Analysis of Service-oriented Systems*, pages 149–171. Springer, 2007. ISBN 978-3-540-72911-2.
18. J. Lenhard. A Pattern-based Analysis of WS-BPEL and Windows Workflow. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik, no. 88, Otto-Friedrich Universität Bamberg, March 2011.
19. J. Lenhard, A. Schönberger, and G. Wirtz. Edit Distance-Based Pattern Support Assessment of Orchestration Languages. In *On the Move 2011 Confederated International Conferences: CoopIS, IS, DOA and ODBASE,*, Hersonissos, 2011.
20. OASIS. *Web Services Business Process Execution Language*, April 2007. v2.0.
21. OMG. *Business Process Model and Notation*, January 2011. v2.0.
22. M. P. Papazoglou and D. Georgakopoulos. Service-oriented Computing. *Communications of the ACM*, 46(10):24–28, October 2003.
23. C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, October 2003.
24. RosettaNet. *MCC Web Services Profile*, June 2010. R11.00.00A.
25. N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst, and N. Mulyar. Workflow Control-Flow Patterns: A Revised View. Technical report, BPM Group, Queensland University of Technology; Department of Technology Management, Eindhoven University of Technology, 2006.
26. D. Spinellis. *Quality Wars: Open Source Versus Proprietary Software*. O'Reilly Media, Inc., 2011. Making Software, ISBN: 978-0-596-80832-7.
27. I. Stamelos, L. Angelis, A. Okionomou, and G. L. Bleris. Code quality analysis in open source software development. *Information Systems Journal*, 12(1):43–60, April 2002.
28. L. H. Thom, M. Reichert, and C. Iochpe. Activity Patterns in Process-aware Information Systems: Basic Concepts and Empirical Evidence. *International Journal of Business Process Integration and Management (IJBPIM)*, 4(2):93–110, 2009.
29. W. van der Aalst and A. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245 – 275, 2005.
30. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases, Springer*, 14(1):5–51, July 2003.
31. WfMC. *XML Process Definition Language*, August 2012. v2.2.