Lessons Learned from Evaluating Workflow Management Systems

Jörg Lenhard[1], Vincenzo Ferme[2], Simon Harrer[3], Matthias Geiger[3], and Cesare Pautasso[2]

[1] Department of Mathematics and Computer Science, Karlstad University, Sweden, joerg.lenhard@kau.se
[2] Software Institute, Faculty of Informatics, USI Lugano, Switzerland, {firstname.lastname}@usi.ch
[3] Distributed Systems Group, University of Bamberg, Germany, {firstname.lastname}@uni-bamberg.de

Abstract. Workflow Management Systems (WfMSs) today act as service composition engines and service-oriented middleware to enable the execution of automated business processes. Automation based on WfMSs promises to enable the model-driven construction of flexible and easily maintainable services with high-performance characteristics. In the past decade, significant effort has been invested into standardizing WfMSs that compose services, with standards such as the Web Services Business Process Execution Language (WS-BPEL) or the Business Process Model and Notation (BPMN). One of the aims of standardization is to enable users of WfMSs to compare different systems and to avoid vendor lock-in. Despite these efforts, there are many expectations concerning portability, performance efficiency, usability, reliability and maintainability of WfMSs that are likely to be unfulfilled. In this work, we synthesize the findings of two research initiatives that deal with WfMSs conformance and performance benchmarking to distill a set of lessons learned and best practices. These findings provide useful advice for practitioners who plan to evaluate and use WfMSs and for WfMS vendors that would like to foster wider adoption of process-centric service composition middleware.

Keywords: Workflow Management Systems, Standards, Lessons Learned, Evaluation Research, Benchmarking, Service Composition

1 Introduction

Workflow management systems (WfMSs) are a core middleware technology for engineering service-oriented applications and for building service orchestrations [21]. Recently, WfMSs are being adapted to cloud computing environments to enable the development of scalable and elastic service-oriented systems.

The most critical part of a WfMS is probably the language in which a user can implement applications and services (i.e., workflows), that run on top of a WfMS. Selecting an unsuitable language or one that cannot easily be transferred to another system can have severe implications for a user, such as an

inability to express business requirements or vendor lock-in [2]. This situation has been addressed by global standardization consortia. To this end, several organizations proposed workflow standards, as for example OASIS with the Web Services Business Process Execution Language (WS-BPEL) [20], or OMG with the Business Process Model and Notation (BPMN) [16].

Workflow standards [16,20] define the language that can be used to implement workflows and the lifecycle of workflow instances. They are meant to clarify the exact scope, building blocks, constraints, and semantics of the language in a precise and unambiguous fashion. As a result, users may select a WfMS with respect to the standard it supports. Unfortunately, in many real-world systems, this assumption is flawed. For example: 1) the quality of standards is often not as high as expected. Especially the BPMN 2.0 standard has been shown to contain many inconsistencies, ambiguities, and editorial flaws [2, 9]. 2) A certification process is not available for any of the standards. As a consequence, any WfMS vendor can claim compliance to a standard without providing proof of this claim. Thus, many vendors just claim support for a standard [8]. 3) Even in the cases where standards provide an unambiguous specification, WfMSs do not necessarily follow that specification, but only implement parts of it [8, 12]. The standard-support related flaws, also lead to performance evaluation pitfalls in WfMSs, such as: 1) the execution performance of the same workflows differs significantly between WfMSs [22] and 2) the miscellaneous usage and implementations of workflow languages are obstacles for the construction of a standard benchmark [23].

As a consequence of the aforementioned flaws, the selection of a suitable WfMS constitutes a challenging task in practice. This paper aims to provide guidance for practitioners (e.g., users and vendors of WfMSs) by highlighting key issues during WfMS evaluation. The material is a cumulative report based on findings we collected over a period of more than five years of experience and derived through independent research initiatives regarding WfMSs[4]. The novelty in this work comes from the aggregation of these results in a joint fashion as lessons learned and we aim at answering the following research questions:

RQ1 Which are the most common expectations and pitfalls during the usage of standard-based WfMSs for automated service composition?

RQ2 How does a practitioner experience the consequences of these pitfalls, in particular regarding the behavior and performance of the WfMS, and how can the pitfalls be addressed?

To answer these questions and to support an easy understanding and transfer to practice, we formulate the aggregated knowledge as lessons learned. We mainly look at situations in which a WfMS is integrated into a more complex environment and is used as a service by many other systems.

This paper is based on an extended abstract [5], which motivates the reporting of lessons learned and briefly mentions a set of five lessons. Here, we report on an extended set of lessons learned, providing additional evidence and details gained with a larger set of WfMSs.

[4] BenchFlow - `http://benchflow.inf.usi.ch` and Betsy - `https://github.com/uniba-dsg/betsy`

In the next section, Sect. 2, we discuss related work. Thereafter, in Sect. 3, we outline the process followed to derive lessons learned, the schema for their presentation in this paper, and the set of resulting lessons. Sect. 4 concludes the paper with a summary and an outline for future research directions.

2 Related Work

In this section, we discuss work related to our lessons learned, related workflow languages, and benchmarking approaches.

Two of the most prominent languages for modeling and executing workflows related to services and also relevant to our work here are WS-BPEL [20] and BPMN 2.0 [16]. The two languages serve similar goals but differ in their expressive power and the language constructs they provide: WS-BPEL is dedicated to the orchestration of web services [20], whereas BPMN 2.0 supports service invocations and message exchanges, but also human activities [16]. Here, we focus on automated workflows only and the human aspects of BPMN 2.0, such as collaborative process modeling, are deliberately out of scope.

Closely related to this paper is the work by Bianculli et al. [1]. The authors propose SOABench as an approach for evaluating the performance of WS-BPEL WfMSs. Here, our focus is broader, since we also take BPMN 2.0 into account and are not limited to the evaluation of performance efficiency aspects only, but cover other characteristics of software quality as well. Furthermore, Wohed et al. [27] evaluate several WfMSs and older versions of BPMN and BPEL for their support for workflow patterns. We also leverage workflow patterns in our work [8, 12, 22], but evaluate newer versions of said workflow languages. Similarly, Garcês et al. [7] present a survey of open source WfMSs. However, the authors address the problem from a different direction. They derive comparison criteria a priori from the workflow reference model and evaluate these criteria. The key difference to this work is that we present lessons learned, i.e., we did not derive a priori criteria but did a post-hoc study based on the observations made during our evaluations. Moreover, the set of systems evaluated is quite different from the systems we consider here. The same applies to Delgado et al. [3] who also build on similar quality models as we do here, but take a more generic stance. The authors state themselves in their study that they differ from our work in many aspects, such as the systems to be evaluated. Being similar to benchmarking methods and tools, approaches for test generation and testbed generation for service-oriented systems, such as [19], are related to our work. For instance, López et al. [19] propose a framework for black-box and property-based testing of web services. Although this framework is intended to test actual applications running on WfMSs, it could be leveraged to evaluate WfMSs as well.

3 Findings and Lessons Learned

A commonly accepted classification of research papers in software engineering is presented in the work by Wieringa et al. [26]. Using this classification, our work

qualifies as *evaluation research*. More precisely, we aim at increasing knowledge by combining existing findings derived from multiple research groups into a common view. This can be achieved by a joint specification of *lessons learned* [26, p. 105].

The *lessons learned* presented in this work are based on cumulative research results, data collection, and experience derived by using and benchmarking different workflow language standards and WfMSs in a number of empirical studies [4, 6, 8, 9, 11–14, 17, 22, 23]. We conducted these studies over a period of several years independently of each other in separate groups, different environments, and with a different evaluation focus. To generate the lessons learned presented here, the group of authors met in person and in video conferencing sessions over an extended period of time. We discussed key lessons that we learned independently through our research in WfMSs evaluation and captured the ones that we learned jointly. The resulting list of lessons was prioritized through a joint voting process and the top lessons are presented in this paper. This research process is very similar to the way in which pattern languages are being developed.

The BPMN 2.0 WfMSs we consider in this paper are *Activiti*, *Bonita*, *Camunda*, and *jBPM*, which according to the vendor's websites, are widely used in the industry. In total, we attempted the evaluation of 47 BPMN 2.0 WfMSs, but most of them could not be integrated into our evaluation approaches due to various reasons such as licensing issues, missing standard compliance, or the unavailability of management APIs [8]. For WS-BPEL, our lessons learned are based on the usage of *Apache ODE*, *OpenESB*, *bpel-g*, *Orchestra*, *Petals ESB*, and three commercial WfMSs whose names we are not allowed to disclose due to licensing. We decided to include pseudonymized results for the commercial WfMSs in this paper, although we acknowledge that this is less informative. The data collected are publicly available on an interactive dashboard[5]. To continuously survey the standard-compliant WfMSs landscape, we created a Wikipedia page for BPMN 2.0 WfMSs[6] which is kept up-to-date by public contributors and ourselves. In addition to helping practitioners in their decision-making, the dashboard and the Wikipedia pages help with improving transparency by reporting the actual state of the WfMSs ecosystem back to the vendors. To document the lessons learned, we use the following structured schema:

Expectation: Users of WfMSs have a number of expectations that are often perceived as true in practice and that are communicated in this fashion by WfMS vendors.

Observation: Although many of the expectations towards WfMSs can be fulfilled, some cannot. This also applies to aspects of considerable importance. In the observations sections, we present and discuss existing evidence that an expectation is not met. Furthermore, we report cases in which it is met.

Consequence: Not meeting a certain expectation has consequences, which are discussed in this part of the schema.

The final list of lessons identified in this study comprises seven items. For some of them, we identified WfMSs that avoid the pitfalls by following what

[5] `http://peace-project.github.io`, last visited at September 27, 2017

[6] `https://en.wikipedia.org/?curid=43305615` , last visited at September 27, 2017

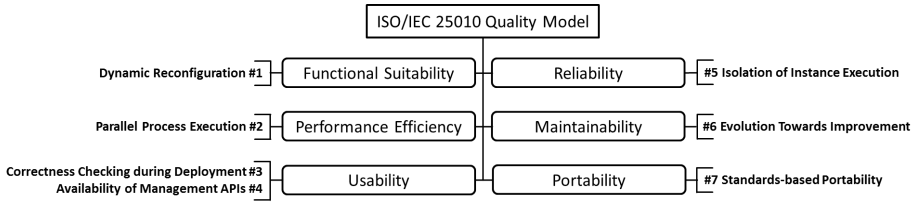

Fig. 1. Findings categorized using the ISO/IEC 25010 Quality Model

we consider a good approach. For structuring purposes, we categorized these lessons using the ISO/IEC 25010 quality model [15] as shown in Fig. 1. According to this quality model, the lessons we defined can be grouped with respect to *functional suitability*, *performance efficiency*, *usability*, *reliability*, *maintainability*, and *portability*. The ISO/IEC 25010 standard provides two more categories, namely, *compatibility* and *security*, that were not considered in this work, since we did not experience pitfalls specific to these categories. More specifically, a dedicated evaluation of security properties is not part of our initiatives and not planned for future work. Nevertheless, an evaluation dedicated to this property might very well discover new pitfalls.

3.1 Functional Suitability Findings

Lesson 1: Dynamic Reconfiguration

Expectation: WfMSs are expected to support dynamically changing environments and flexible dynamic service bindings [25]. This entails the reconfiguration of workflow instances at runtime when they interact with late-bound external systems and alternative service providers. It should be possible to pass the address of a service to a workflow instance and the instance should be able to redirect its communication channels to this service. As an example, in a workflow that processes orders, a buyer could send the address of his own web service to the workflow instance so that he will be notified when the order is complete.

Observation: In WS-BPEL, dynamic reconfiguration is possible in a standard conformant way, by updating the endpoint of specified partner links. However, one out of three proprietary WfMSs evaluated in [12] and four out of five open source WfMSs, namely, Apache ODE[7], OpenESB, Orchestra, and Petals ESB, evaluated in [11] and [12] do not support standard-conformant dynamic reconfiguration and only bpel-g does so. We did not check non-standard vendor-specific extensions which could provide similar functionality. In BPMN 2.0, dynamic reconfiguration is defined only in an abstract way, but no WfMS actually supports it based on the standard.

Consequence: Because of the lack of dynamic reconfiguration, the creation of self-adapting systems is hindered in WS-BPEL WfMSs. Instead of changing the channels within the workflow, each external web service needs to be encapsulated

[7] The developers of ODE fixed this in a later version.

through a proxy service which itself can be dynamically reconfigured, leading to higher development and maintenance costs [25]. In BPMN 2.0 WfMSs, it is not possible to implement dynamic reconfiguration in a standards-based fashion.

3.2 Performance Efficiency Findings

Lesson 2: Parallel Process Execution
Also affects the functional suitability.

Expectation: One of the major advantages of workflow languages is that one can specify the control-flow of workflows in a declarative way. Then, any WfMS that implements the execution semantics of the workflow language should execute instances of these workflows by following the specified control-flow definitions. When modeling parallelism, the user expects that the underlying WfMS will execute the constructs that are marked as parallel to each other in a concurrent manner. Parallelism can be expressed with various control flow constructs in both standards. Using WS-BPEL [20] it is possible to define parallel execution using the *forEach* and *flow* constructs. Moreover, *eventHandlers* are always executed in parallel for a workflow scope that is attached to them. BPMN 2.0 [16] provides a similar variety of constructs for expressing parallelism. The most commonly used one is the splitting *parallelGateway* that acts as a fork operation and causes the parallel execution of the connected branches. Other possibilities are the use of *inclusiveGateways*, *eventBasedGateways*, or the definition of *MultiInstanceLoopCharacteristics* for tasks and sub-processes.

Observation: In [11,12], we studied the behavior of WS-BPEL [20] workflow models containing the parallel *forEach* element. The research was conducted for five open source (Apache ODE, bpel-g, OpenESB, Orchestra, and Petals ESB) and three proprietary WS-BPEL WfMSs. The results showed that two open source (OpenESB, Petals ESB) and one proprietary WfMSs ignored the parallel semantics on the *flow* and *forEach* elements and one (Orchestra) on the *forEach* element only, *silently* resulting in a sequential execution.

Likewise, we evaluated three open source BPMN 2.0 WfMSs [22], namely Camunda, Activiti and jBPM regarding their support for workflow patterns. These patterns are considered as pieces of functionality that should be easily expressible in any workflow language. We tested the WfMSs against five fundamental control-flow workflow patterns, two of which contain parallelism. The results show that all benchmarked WfMSs implement parallelism in a pseudo-parallel, non-deterministic way, which is also reported in another study [8]. jBPM uses a random execution order of the parallel elements, while Camunda and Activiti always execute the parallel elements in the order in which they are defined in the workflow model. Moreover, jBPM shows a significant drop in performance if the parallelism construct is used [22].

Consequence: Due to pseudo-parallel execution, it is not possible to speed up the execution of independent control-flow branches by using the language constructs dedicated to parallelism. Apart from the performance aspect, this is also problematic as in some situations the functional correctness might depend on a truly parallel execution. For instance, several workflow patterns, such as

multiple instances without a priori runtime knowledge, build on truly parallel execution, and thus, "the lack of truly parallel execution in a WfMS is the biggest obstacle to pattern support" [12, p. 111].

3.3 Usability Findings

Lesson 3: Correctness Checking during Deployment

Expectation: Both workflow languages, WS-BPEL [20] and BPMN 2.0 [16], define constraints regarding the correctness of modeled workflows. WS-BPEL explicitly lists 94 rules named *static analysis rules*, which describe issues that should be detected by any standard compliant WfMS. Thus it is to be expected that WfMSs are capable of detecting invalid workflow models at deploy time.

Observation: Deploying invalid workflows that violate static analysis rules or BPMN 2.0 constraints revealed that most WfMSs are not capable of this kind of detection [8,14]. Regarding WS-BPEL WfMSs, we evaluated several systems for their coverage of static analysis rules [14]. Ignoring the reference implementation of BPEL which we do not consider here, we found that a single WfMS (OpenESB) performs no detection at all, and the rest have a highly varying detection rate of at most 75%. For BPMN 2.0, a common omission can be found in the missing validation of *timer* conditions. None of the three WfMSs benchmarked in [22], namely Camunda, Activiti, and jBPM, validated *timer* conditions at deploy-time, although there is a mandatory definition of the format [16].

Consequence: As invalid workflows are not rejected on deployment, errors are not detected early in the development process. Thus, errors may be hidden for a long time in production use only to be found later, which is costly. Invalid workflows may fail at runtime creating runtime errors in the WfMSs. To make things worse, in some cases the workflows do not crash observably, but instead complete with non-deterministic results. If the used WfMS is weak in detecting violations of standard-defined rules, the users creating workflows for deployment should consider using external tools for validating the workflows prior to deployment.

Lesson 4: Availability of Management APIs

Expectation: In production, WfMSs are usually part of a more complex software ecosystem and typically interact with other services. They are also integrated more and more into a continuous integration and delivery lifecycle [24]. It is to be expected that WfMS vendors provide management APIs [4] to support continuous integration and delivery.

Observation: Only four (Camunda, Bonita, jBPM, and Activiti) out of a set of 47 BPMN 2.0 WfMSs which we analyzed allow an automatic deployment and execution of workflows through a REST API [8]. Camunda and Activiti expose complete REST APIs to the clients so that the interaction with those WfMSs is straightforward. Bonita and jBPM provide partial support for interaction through the provided REST APIs. For example, the former misses the possibility to log

into the API, forcing the user to log in using the so-called Web REST API[8], while the latter misses an API to deploy workflows, leading to the need for a workaround. Out of the remaining 43 WfMSs, many require human interaction with a user interface at various stages. Examples range from a manual import of standard-compliant BPMN 2.0 workflows prior to deployment, over manual deployment using a web front-end, to manual creation of new workflow instances. In contrast, the WfMSs supporting WS-BPEL do not support REST APIs, but require file handling or other APIs (ODE, bpel-g, Orchestra, Petals ESB) [10,11]. One (OpenESB) even lacks a remotely accessible API.

Consequence: Given the limitations or lack of the WfMSs' APIs, it is often hard or impossible to integrate the products in a fully automated continuous integration lifecycle. For instance, it is often not possible to quickly detect errors introduced in revisions of existing workflows by automated tests. This hinders the application of agile development methods that rely on short feedback loops [24].

3.4 Reliability Findings

Lesson 5: Isolation of Instance Execution

Expectation: Workflow instances should be executed in a sandbox. Users expect that instances cannot influence each other only because they are executed in the same environment. Moreover, faulty instances should not have an impact on the stability and integrity of the WfMS itself. Facilities need to be in place to restrict hostile instances from breaking out of their runtime environment, overloading the WfMS performance-wise, or taking down the entire WfMS [13,18]. Furthermore, the WfMS should also try to minimize performance interference of, and between, different workflow instances [18].

Observation: For some Apache ODE versions, we observed that individual workflow instances are able to jeopardize the execution of other workflow instances or even severely affect the underlying WfMS stability. If a process instance enters a state of busy waiting, the resulting CPU load crashes the whole WfMS [12]. In the case of BPMN 2.0 WfMSs, we found that single versions of Activiti, jBPM, and Camunda were crashing due to memory leaks if the executed workflows are using infinite loops to execute script tasks. Moreover, the default configurations of Activiti and Camunda were not stable when using workflows containing loops and 1500 concurrently interacting users [22].

Consequence: The effect of missing isolation during workflow instance execution is similar to a single process crashing a complete operating system. It is obvious that this should not happen. The WfMSs should be safe from possible crashes of workflows instances, protecting other running instances. This can be achieved, for example, by detecting excessive resource usage (e.g., RAM, CPU, I/O) and suspending the critical workflow instances. Another complementary approach could be the usage of independent WfMS installations for different, independent sets of workflows instances. This is facilitated as the WfMSs vendors are starting to support virtualization techniques such as Docker containers.

[8] `http://documentation.bonitasoft.com/?page=rest-api-overview#toc2`, last visited at September 27, 2017

3.5 Maintainability Findings

Lesson 6: Evolution Towards Improvement
Also affects performance efficiency and functional suitability.

Expectation: WfMSs should improve and evolve over time towards a higher degree of maturity. When upgrading to a newer version, users expect that it will be better than the previous one. For WfMSs, usual expectations are improvements in i) functionality available to the users, ii) number of language features supported, iii) performance and iv) reductions of workflow instances execution cost.

Observation: In [8], we investigated the evolution of the three BPMN 2.0 WfMSs Camunda, Activiti, and jBPM over the period of three years. All the three WfMSs evolved in terms of functionality available. Besides the evolution in functionality, we also discovered regressions over the years. For example, in the cases of Activiti and jBPM there were features that stopped working after upgrading to the next release. What is more, all three WfMSs made only marginal progress in the support of more BPMN 2.0 features. Also, research in performance benchmarking of the Activiti and Camunda WfMSs revealed that over the versions the performance of the WfMSs decreased [6]. In particular, the time needed by the WfMSs to execute single workflow instances constantly increased over time, by approximately 8% per year.

Consequence: There are two consequences for users: 1) they should be careful when upgrading to a newer version as regressions (in terms of supported features and performance) are possible, and 2) users should not expect that language feature support (especially for BPMN 2.0) will increase from version to version. This creates the necessity for users to benchmark WfMSs extensively before deciding about adopting newer versions.

3.6 Portability Findings

Lesson 7: Standards-based Portability

Expectation: One of the goals of standardizing workflow languages and WfMSs is to establish a commonly agreed set of functionality and a serialization format for specifying the workflows that enables their portability [16, 20]. Given that a standard is supported by multiple WfMSs, users should be able to move workflows implemented in the standard between any of these systems. This protects from vendor lock-in. Moreover, the execution semantics of a standards-based workflow should be identical on any WfMS supporting the standard.

Observation: In the absence of certification authorities, standard acronyms are rather hollow. For instance, there are many vendors that state to support BPMN 2.0. In [8], we tested 47 products stating to implement BPMN 2.0 and discovered that only three of them (Activiti, Camunda, and jBPM) were able to import, deploy, and execute workflows expressed directly in the standardized execution format. Instead, many vendors rely on custom serialization formats, as for example Bonita[9], and only provide a subset of the visual BPMN 2.0 shapes for modeling.

[9] `http://documentation.bonitasoft.com/?page=build-a-process-for-deployment`, last visited at September 27, 2017

Table 1. Summary of the Findings: + (good approach), − (pitfall present), ∼ (pitfall partially present), N/A (no observations)

WfMS	1. Dynamic Reconfiguration	2. Parallel Execution	3. Correctness Checking	4. Management APIs	5. Instance Isolation	6. Improved Evolution	7. Standard Portability
BPMN							
Activiti	N/A	− [8, 22]	∼ [8, 22]	+ [8]	∼ [22]	∼ [6, 8]	∼ [8]
Bonita	N/A	N/A	N/A	∼	N/A	N/A	−
Camunda	N/A	− [8, 22]	∼ [8, 22]	+ [8]	∼ [22]	∼ [6, 8]	∼ [8]
jBPM	N/A	− [8, 22]	∼ [8, 22]	∼ [8]	+ [22]	∼ [8]	∼ [8]
WS-BPEL							
ODE	+ [11]	+ [12]	∼ [14]	+ [10]	− [12]	N/A	∼ [11]
OpenESB	− [11]	− [12]	− [14]	− [10]	+ [12]	N/A	∼ [11]
bpel-g	+ [11]	+ [12]	∼ [14]	+ [10]	+ [12]	N/A	∼ [11]
Orchestra	− [11]	∼ [12]	∼ [14]	+ [10]	+ [12]	N/A	∼ [11]
Petals ESB	− [11]	− [12]	∼ [14]	+ [10]	+ [12]	N/A	∼ [11]
3 Commerc.	∼ [12]	∼ [12]	N/A	N/A	+ [12]	N/A	∼ [12]

Even if execution using a standardized language is supported in principle, this support is often limited to selected features of the language. Details of feature support are reported in [11,12] for WfMSs using WS-BPEL and in [8] for WfMSs using BPMN 2.0. Essentially, for both standards, the number of language features that is commonly supported by a large majority of the tested WfMSs is limited to around 40% of the features defined in the respective standards. The supported features are limited to the basic part of the standards, such as enabling sequential execution, conditional branching, and basic looping.

Consequence: Modeling a workflow in compliance to a standard does not guarantee that the workflow can be executed by a WfMS. The situation is further complicated by the fact that some vendors serialize workflows with custom language extensions specific to their product, which introduces vendor lock-in.

3.7 Summary

We presented seven lessons for which we formulated common expectations and discussed good approaches and potential pitfalls [RQ1]. In Table 1, we summarize these lessons learned. The table reports the findings for all the WfMSs providing sufficient APIs [4], as discussed in lesson 3.3, enabling us to automate the analysis producing the data on which most of the findings are based on. We use a trivalent rating that classifies WfMSs as using a good approach (+), containing a pitfall (-), or partially containing a pitfall (∼). As evident from the table, each WfMS has advantages and disadvantages and no single WfMS provides a good approach for all of the discussed lessons.

The aggregated results highlight the relationship between workflow standards, WfMSs, and the user expectations that follow from this relationship. To answer RQ1, we identified pitfalls in the areas of functional suitability, performance

efficiency, usability, reliability, maintainability, and portability. Despite the existing standards, expectations about the available functionality or portability of workflow models are often not fulfilled. The usability of WfMSs in terms of correctness checking and available APIs is limited in many cases and new versions do not necessarily lead to improvements. Lastly, scalability and workflow instance isolation can be problematic.

As consequences for users, referring to RQ2, this study shows that thorough research and evaluation before selecting a WfMS is inevitable despite the existence of workflow standards. It is unlikely to find a system that actually supports a complete standard and the danger of vendor lock-in is still real. Even when only updating a WfMS, a careful evaluation is needed because of potential regressions in terms of supported features or performance characteristics.

4 Conclusion and Future Work

In this paper, we presented a catalog of lessons learned regarding the usage of WfMSs, synthesized from findings obtained by two independent research initiatives over a five-year period. This synthesis uncovers a number of common expectations and pitfalls in WfMSs usage. No WfMS we investigated follows only good approaches, but they all lack important aspects. Most prominently, the standard conformance, and thus the portability of workflows, is severely hampered, despite the claimed support for standards. In this regard, it can be diagnosed that standards have largely failed their target of a harmonized WfMSs landscape. Ultimately, a WfMS selection will require a prioritization approach that ranks features according to their importance for the intended specific usage scenario. Based on such a ranking, the presented results can support the selection.

In conducting this study, we also learned lessons about aggregating benchmarking results as lessons learned. We have found the aggregation of lessons feasible and valuable despite the fact that our initiatives were focused on different quality attributes, i.e., conformance and performance, even if this meant we had to discard lessons that were important in one area, but not present in the other.

References

1. Bianculli, D., Binder, W., Drago, M.L.: Automated Performance Assessment for Service-Oriented Middleware: a Case Study on BPEL engines. In: 19th WWW. pp. 141–150. Raleigh, North Carolina, USA (Apr 2010)
2. Börger, E.: Approaches to Modeling Business Processes. A Critical Analysis of BPMN, Workflow Patterns and YAWL. Softw Syst Model 11(3), 305–318 (2012)
3. Delgado, A., Calegari, D., Milanese, P., Falcon, R., García, E.: A Systematic Approach for Evaluating BPM Systems: Case Studies on Open Source and Proprietary Tools. In: 11th OSS. Florence, Italy (May 2015)
4. Ferme, V., Ivanchikj, A., Pautasso, C., Skouradaki, M., Leymann, F.: A Container-centric Methodology for Benchmarking Workflow Management Systems. In: 6th CLOSER. Rome, Italy (2016)
5. Ferme, V., Lenhard, J., Harrer, S., Geiger, M., Pautasso, C.: Workflow Management Systems Benchmarking: Unfulfilled Expectations and Lessons Learned (extended abstract). In: 39th ICSE Companion, Poster Track (2017)

6. Ferme, V., Skouradaki, M., Ivanchikj, A., Pautasso, C., Leymann, F.: Performance Comparison Between BPMN 2.0 Workflow Management Systems Versions. In: 18th BPMDS (2017)
7. Garcês, R., Jesus, T., Cardoso, J., Valente, P.: BPM & Workflow Handbook, chap. Open Source Workflow Management Systems: A Concise Survey. Future Strategies (2009)
8. Geiger, M., Harrer, S., Lenhard, J., Wirtz, G.: BPMN 2.0: The state of support and implementation. Future Generation Computer Systems (Jan 2017)
9. Geiger, M., Wirtz, G.: BPMN 2.0 Serialization - Standard Compliance Issues and Evaluation of Modeling Tools. In: 5th EMISA (Sep 2013)
10. Harrer, S., Lenhard, J.: Betsy–A BPEL Engine Test System. Tech. Rep. 90, Otto-Friedrich Universität Bamberg (Jul 2012)
11. Harrer, S., Lenhard, J., Wirtz, G.: BPEL Conformance in Open Source Engines. In: 5th IEEE SOCA. pp. 237–244 (Dec 2012)
12. Harrer, S., Lenhard, J., Wirtz, G.: Open Source versus Proprietary Software in Service-Orientation: The Case of BPEL Engines. In: 11th ICSOC (Dec 2013)
13. Harrer, S., Nizamic, F., Wirtz, G., Lazovik, A.: Towards a Robustness Evaluation Framework for BPEL Engines. In: 7th IEEE SOCA. pp. 199–206 (Nov 2014)
14. Harrer, S., Preißinger, C., Wirtz, G.: BPEL Conformance in Open Source Engines: The Case of Static Analysis. In: 7th IEEE SOCA. pp. 33–40 (Nov 2014)
15. ISO/IEC: ISO/IEC 25010:2011; Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models (2011)
16. ISO/IEC: ISO/IEC 19510:2013 – Information technology - Object Management Group Business Process Model and Notation (2013), v2.0.2
17. Lenhard, J., Wirtz, G.: Portability of Executable Service-Oriented Processes: Metrics and Validation. Service Oriented Computing and Applications 10(4) (Dec 2016)
18. Leymann, F.: BPEL vs. BPMN 2.0: Should You Care? In: 2nd Intl. Workshop on BPMN. pp. 8–13 (2010)
19. López, M., Ferreiro, H., Francisco, M., Castro, L.: Automatic Generation of Test Models for Web Services Using WSDL and OCL. In: 11th ICSOC. Berlin, Germany (Dec 2013)
20. OASIS: Web Services Business Process Execution Language (2007), v2.0
21. Peltz, C.: Web Services Orchestration and Choreography. Computer 36(10), 46–52 (Oct 2003)
22. Skouradaki, M., Ferme, V., Pautasso, C., Leymann, F., van Hoorn, A.: Micro-Benchmarking BPMN 2.0 Workflow Management Systems with Workflow Patterns. In: 28th CAiSE 2016. pp. 67–82 (Jun 2016)
23. Skouradaki, M., Roller, D.H., Leymann, F., Ferme, V., Pautasso, C.: On the Road to Benchmarking BPMN 2.0 Workflow Engines. In: 6th ACM/SPEC ICPE. pp. 301–304. ACM (2015)
24. Thiemich, C., Puhlmann, F.: An agile BPM project methodology. In: Business Process Management, pp. 291–306. Springer (2013)
25. Tsai, W.T., Song, W., Paul, R., Cao, Z., Huang, H.: Services-Oriented Dynamic Reconfiguration Framework for Dependable Distributed Computing. In: COMPSAC. pp. 554–559 (2004)
26. Wieringa, R., Maiden, N., Mead, N., Rolland, C.: Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. RE 11(1) (2006)
27. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., Russell, N.: On the Suitability of BPMN for Business Process Modelling. In: 4th BPMN. pp. 161–176. Vienna, Austria (Sep 2006)