Patterns for Workflow Engine Benchmarking

Simon Harrer[1], Oliver Kopp[2], and Jörg Lenhard[3]

[1] Distributed Systems Group, University of Bamberg, Germany
simon.harrer@uni-bamberg.de
[2] Institute for Parallel and Distributed Systems, University of Stuttgart, Germany
oliver.kopp@informatik.uni-stuttgart.de
[3] Department of Mathematics and Computer Science, Karlstad University, Sweden
joerg.lenhard@kau.se

Abstract. Workflow engines are frequently used in the service-oriented and cloud computing domains. Since engines have significant impact on the quality of service provided by hosted applications, it is desirable to compare and select the most appropriate engine for a given task. To enable such a comparison, approaches for benchmarking workflow engines have emerged. Although these approaches deal with different quality properties, such as performance or standard conformance, they face many reoccurring problems during the design and implementation phase, which they solve in similar ways. In this paper, we describe such common solutions to reoccurring problems in the area of workflow engine benchmarking as patterns. Our aim is to present pattern candidates that help benchmark authors to design and implement proper and valid workflow engine benchmarks and benchmarking tools.

Keywords: patterns, workflow engine, benchmarking

1 Introduction

An established part of the field of service-oriented computing is the construction of composite services on the basis of message exchanges between lower-level services [21]. This composition is often achieved by capturing the data- and control-flow between message exchanges of several services in a workflow [22]. The workflow is subsequently deployed on a *workflow engine*, which provides the middleware execution platform, context and cross-cutting functionality, message correlation, and many other features to the hosted workflow. Today, several standards for workflow definition and a multitude of engines have emerged, including implementations by multi-national middleware vendors, open source solutions, research prototypes, and even cloud-based engines. The range of solutions makes it important to the user to compare existing engines with the aim of selecting the best engine for her purpose. However, engines are highly complex products, resulting in an equally complex selection problem [9]. To address this problem, workflow engine benchmarking approaches have emerged [3, 6]. Several research groups are developing different benchmarking approaches and tools that

target varying quality properties of workflow engines, such as performance [3] or standard conformance [6].

Currently, approaches for workflow engine benchmarking are being developed in parallel. Their authors often face the same problems, regardless of the actual property that lies in the focus of the benchmark. Such common problems are, for instance, how to identify suitable tests or workloads for engine benchmarks, or how to ensure the correctness of test implementations. Moreover, solutions to such common problems are often similar, leading to the unfortunate situation that multiple groups invest significant effort to solve the same problem and re-implement the same solution. Since proven solutions to reoccurring problems exist and can be inferred from existing engine benchmarks, it is possible to capture these solution as *patterns* [1, 5]. By describing such solutions as patterns, it should be possible to reduce the effort for implementing new workflow engine benchmarks and also to ease the communication among benchmark authors through a common vocabulary.

This paper is a first attempt to provide pattern candidates within the domain of workflow engine benchmarking. Working on workflow engine benchmarks and tools for several years, we are confident that the presented patterns can help the authors of future benchmarks. It should be possible to extend and refine the proposed pattern candidates in the future. In the following, we develop the pattern candidates by describing the participants and challenges in workflow engine benchmarking, for which we propose a number of solutions. These solutions should provide guidelines to future benchmark authors when facing the same challenge.

The paper is structured as follows. First, we present related work in Sect. 2, providing the background for describing the participants and challenges in workflow engine benchmarking in Sect. 3. In Sect. 4, pattern candidates are presented which act as a set of alternative and competing solutions to the challenges outlined in Sect. 3. The work is concluded with an outlook on future work in Sect. 5.

2 Related Work

Work related to this paper can be roughly divided into three areas: i) The definition and usage of patterns, ii) work on workflows and workflow engines, and iii) work on benchmarking and benchmarking workflow engines in particular.

The notion of patterns originated from the field of architecture [1], where patterns were used to describe reoccurring structures in buildings. Years later, the idea to describe reoccurring structures in the design of software in the form of patterns [5] had a huge impact on software development. Since then, patterns have been applied in many areas and contexts, and a multitude of pattern catalogs and languages have been published. Workflow engine benchmarking is an area, where, to the best of our knowledge, patterns are still lacking. The huge momentum in the development of pattern languages, has also led to work that theorizes on pattern structure [17,18] and how to write a pattern [19]. Here, we build on these works to specify our pattern candidates. Meszaros and Doble [19] propose *name*,

problem, *context*, *forces*, and *solution* as the mandatory elements of patterns. *Examples* and *relations* are considered as optional elements. In our work, we use *name*, *problem*, *solution*, *relations*, and *example* as elements for pattern description. Our paper aims to provide a list of pattern candidates we discovered in our work with workflow enginge benchmarking. Due to space limitations, we excluded *context* and *forces* from the presentation of our pattern candidates.

Workflows and workflow engines, or more abstract, process-aware information systems [25], are commonly used in the service-oriented computing domain to orchestrate services [22]. In short, a workflow is the machine-readable and executable representation of a business process in whole or part and a workflow engine is the software runtime environment that manages and controls the execution of workflow instances [27]. Today, two language standards are predominantly used for workflow specification and execution. These are the Web Services Business Process Execution Language (BPEL) [20] and the Business Process Model and Notation (BPMN) [16].

Benchmarks are an important tool in computer science that is needed to compare and analyze the quality provided by software systems [15]. Many aspects of software can be benchmarked, but often the focus resides on performance-related aspects, such as latency or throughput. When it comes to workflow engines, two major aspects have been in the spotlight. As indicated above, one of these is performance [2, 3]. The second aspect is standard conformance, which reflects the fact that workflow engines are often standards-based products [7, 10]. Benchmarking approaches for both aspects exist for both languages mentioned in the previous paragraph, for BPEL [2,10] and BPMN [3,7]. The BPEL/BPMN Engine Test System betsy[4] and BenchFlow[5] are implementations of these benchmarking approaches.

3 Problems in Workflow Engine Benchmarking

In the following subsection, we clarify the reoccurring challenges one faces when building a benchmark for workflow engines. These challenges are the crucial sources and motivation for gathering workflow engine benchmarking patterns.

3.1 Big Picture

In the big picture of workflow engine benchmarking, there are four elements: *tests*, the *engines* to be tested, the *benchmarking procedure*, and the benchmark *results*.

When a benchmark is conducted, *tests* are used to specify requirements or desired behavior of the engines under test. Next to the tests, these *engines* are the second input to the benchmark. They are the objects of study that are to be evaluated in a comparable fashion. The *benchmarking procedure* is the tool

[4] `https://github.com/uniba-dsg/betsy`
[5] `https://github.com/benchflow/benchflow`

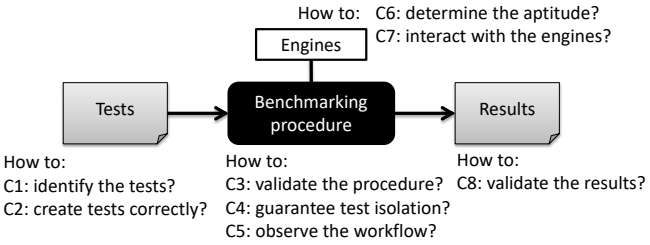

Fig. 1. Big Picture of Workflow Engine Benchmarking

that instruments both, engines and tests. It evaluates the first using the latter and produces benchmark *results*. These results should be constructed in an easily comprehensible fashion to allow for a straight-forward interpretation.

3.2 Challenges

During workflow engine benchmarking, a number of challenges arise related to each element listed in Sect. 3.1. These challenges are non-trivial and reoccurring problems that need to be solved for every benchmark, hence, they are the problems for which we propose patterns as a solution in this paper. In total, we identified eight challenges, numbered from C1 to C8, which we present in the following.

Regarding the tests, the major issues are how to *identify the tests (C1)* and *create tests correctly (C2)*. The tests should be suitable and representative of a realistic usage scenario. If this is not the case, the results produced by the benchmark are of no use. Since realistic tests can be non-trivial, it is important to ensure that they are free of issues, since even minor issues could have considerable impact on the benchmark results.

Major challenges regarding the benchmarking procedure are how to *validate the procedure (C3)*, *guarantee test isolation (C4)*, and *observe the workflow (C5)*. As for the tests, quality assurance needs to be in place to make sure that there are no errors in the benchmarking procedure that might have an impact on the benchmark results. As realistic test sets might be large, it is important to make sure that tests can be executed independently, regardless of the execution order, and regardless of whether execution takes place sequentially or in parallel. Finally, a mechanism needs to be in place that helps to identify how and if the benchmark and singular tests are progressing. Since a benchmark might push an engine to its limits, it can easily be the case that an engine fails to make progress during execution, which needs to be detected and acted upon.

Regarding the engines, the major issues are how to *determine the aptitude (C6)* of an engine and how to *interact with the engines (C7)*. The sixth challenge concerns the ability of the engine to participate in the first place. The engine needs to be in normal working mode at the start of the benchmark, otherwise meaningful results are unlikely. C7 concerns the ability to control and monitor the engine during execution. Test execution requires the evaluation of assertions

or observation of behavior, so it is necessary that the engine has facilities in place that allow to communicate its state to the outside.

Finally, regarding the results, the major issues are how to *validate the results (C8)*. Although the benchmark procedure might have worked without problems, it is possible that there are errors in the conversion of raw benchmark data to interpretable results. Therefore, quality assurance is needed to make sure that the results are actually correct.

4 Workflow Engine Benchmarking Pattern Catalog

In the following section, we describe our candidates for workflow engine benchmarking patterns. The candidates are structured into several categories that mirror the elements of a benchmark from Sect. 3.1 and are based on the challenges described in the previous section. *Test patterns* concern the identification and quality assurance of test cases for a benchmark, and *procedure patterns* group patterns for automating the benchmark environment. *Engine patterns* describe ways to instrument workflow engines for using them in a benchmark. Finally, we present *results patterns*, which provide solutions for the validation of result data. This section concludes with a short discussion of the patterns.

For each pattern, we provide a unique *name* and list the *problems* it addresses, which directly corresponds to the challenges from Sect. 3.2. Furthermore, we describe the *solution* to said problem, which outlines what the pattern does, in an abstract form, whereas the *example* describes its known usage and *relations* to other patterns.

Test Patterns: To *identify the tests (C1)*, one can determine the constructs of a language and apply *Configuration Permutation (P1)* or use *Reoccurring Constructs (P2)* to identify the most used constructs and their configuration. To *create tests correctly (C2)*, one can derive tests using *Stub Extension (P3)* or *Mutated Existing Test (P4)*, which both ensure a certain degree of correctness. Applying *Open Sourcing (P5)*, *Expert Review (P6)*, and *Automatic Static Analysis (P7)* helps to strengthen the degree of correctness further. What is more, the latter patterns can be applied independently of each other.

Name Configuration Permutation (P1)
Problem Identify the tests (C1)
Solution Identify a construct. Permutate all configurations of a construct. Each permutation is a test.
Example This pattern was applied in betsy for BPEL and BPMN standard conformance tests. For instance, in BPMN, there is the construct *exclusive gateway* which can be configured in three ways resulting in three tests: (i) *standard* with all outgoing sequence flows having conditions, (ii) exclusive gateway with a sequence flow without a condition and marked as *default*, and (iii) one as a *mixed* gateway with both branching and merging capabilities.

Name Reoccurring Constructs (P2)
Problem Identify the tests (C1)

Solution Gather a large corpus of workflows. Identify the reoccurring elements in these workflows. Tests are created based on the most important (i.e., reoccurring) elements.

Example This pattern was applied in betsy for BPEL and BPMN expressiveness tests as the test suite is based on the workflow control-flow patterns [26, 28] which are created from analyzing multiple workflow management systems. In BenchFlow, a large corpus of workflows is used to construct workloads for performance tests [23, 24].

Name Stub Extension (P3)

Problem Create tests correctly (C2)

Solution Use a workflow stub, a minimal workflow, which is extended for all tests, so that the extension contains solely the feature under test. The stub itself provides extension points, where the feature under test can be put. The rest is minimal overhead required to observe the feature under test. This way, all tests follow the same structure, and when looking at the difference between the test and the stub, the feature under test can be easily identified.

Example This pattern was applied in betsy for both, BPEL and BPMN.

Related If the stub is fully functional, it can act as an Aptitude Test (P8).

Name Mutated Existing Test (P4)

Problem Create tests correctly (C2)

Solution Instead of starting from scratch, use correct tests and modify them by introducing a mutation [12]. This is especially useful for creating tests for faulty conditions: An existing test is mutated by injecting a single isolated fault, to see if a feature works correctly even in the face of errors [13].

Example The pattern was applied in betsy for creating erroneous workflows that have to rejected upon deployment for both BPEL and BPMN. In addition, it was applied to create a test suite for determining robustness [12].

Relations Similar to Stub Extension (P3) as the workflow model of another test as the basis for a new workflow test.

Name Open Sourcing (P5)

Problem Create tests correctly (C2), validate the procedure (C3), and validate the results (C8)

Solution Open source tests, procedure, and results, and put it under the scrutiny of the public. Public availability can help to find errors the original authors did not find. Also, this can help to build a community for the benchmark.

Example Both betsy and benchflow are open source. In case of betsy, this has lead to contributions by experts and also engine vendors.

Relations May result in Expert Review (P6).

Name Expert Review (P6)

Problem Create tests correctly (C2), validate the procedure (C3), and validate the results (C8)

Solution Ask experts to review the benchmark artifacts. Experts can be domain experts, engine developers, or benchmark engineers.

Example For betsy, the maintainers of Apache ODE and bpel-g helped to improve the test cases through their feedback, looking at the results and checking why the behavior of their engine was different from what they expected.

Name Automatic Static Analysis (P7)
Problem Create tests correctly (C2), validate the procedure (C3), and validate the results (C8)
Solution Create static analysis checks which detect mistakes automatically. As most workflow languages are XML-based, an XML well-formedness check as well as schema validation with the XSD of the workflow language is straight-forward. If possible, apply additional static analysis based on workflow language rules and best practices.
Example The pattern was applied in betsy by checking the correctness of workflows regarding naming conventions, XML well-formedness, XSD validity regarding the workflow language schema, and even more sophisticated static analysis rules with BPELlint[6] and BPMNspector[7].

Benchmarking Procedure Patterns: To *validate the procedure (C3)*, one can apply *Open Sourcing (P5)*, *Expert Review (P6)*, and *Aptitude Test (P8)*. *Reinstallation (P9)*, *Virtual Machines (P10)*, and *Containers (P11)* can be applied to *guarantee test isolation (C4)*. To *observe the workflow (C5)*, *Message Evaluation (P12)*, *Partner-based Message Evaluation (P13)*, *Execution Trace Evaluation (P14)*, *Engine API Evaluation (P15)*, *Concurrency Detection (P16)*, and *Detailed Logs (P17)* are applicable.

Name Aptitude Test (P8)
Problem Validate the procedure (C3) and determine the aptitude (C6)
Solution Define an aptitude test as a minimal requirement for participation in the benchmark. An engine must pass this test. The test should check the minimal amount of features required.
Example In betsy, there are two aptitude tests, one for BPEL named *Sequence*, containing a receive-assign-reply triplet (see Message Evaluation (P12)), and one for BPMN, named *SequenceFlow*, containing a start and end event, with corresponding script tasks to allow to observe the events (see Execution Trace Evaluation (P14)), connected through sequence flows.
Relations Can be used as a stub for Stub Extension (P3).

Name Reinstallation (P9)
Problem Guarantee test isolation (C4)
Solution Install and start the engine anew for each test case, providing a fresh engine instance. Although a reinstallation can consume a lot of time, it ensures that one test case cannot interfere with another one.
Example In betsy, this is the default mode.
Relations Virtual Machines (P10) and Containers (P11) are alternatives.

[6] `https://github.com/uniba-dsg/BPELlint`
[7] `http://bpmnspector.org/`

Name Virtual Machines (P10)
Problem Guarantee test isolation (C4)
Solution Create a virtual machine with a snapshot of a running engine upfront. This may require some time and effort once per engine. But with a snapshot in place, each test can be executed in isolation. The snapshot can easily be restored before each test and be discarded afterwards, resulting in test isolation with a low temporal overhead. However, for virtual machines, there is typically a substantial RAM and HDD overhead.
Example Since 2014, betsy also supports this pattern [14].
Relations Reinstallation (P9) and Containers (P11) are alternatives.

Name Containers (P11)
Problem Guarantee test isolation (C4)
Solution Create an image with the engine already installed and configured. Create a new container for each test and discard the container afterwards, effectively ensuring test isolation. This is similar to Virtual Machines (P10), but with considerably less overhead. At this point in time, however, support for RAM snapshots of containers is not existing, but HDD snapshots are available.
Example This pattern is used in both, betsy [8] and BenchFlow [4].
Relations Reinstallation (P9) and Virtual Machines (P10) are alternatives

Name Message Evaluation (P12)
Problem Observe the workflow (C5)
Solution Send messages to the workflow and compare responses with an expected response. Use small interfaces with only few methods to keep different message types and possibilities low.
Example Betsy communicates with BPEL instances only through four different SOAP messages and observes the behavior by checking the responses.
Relations Partner-based Message Evaluation (P13) builds upon this pattern. Execution Trace Evaluation (P14) and Engine API Evaluation (P15) are alternatives.

Name Partner-based Message Evaluation (P13)
Problem Observe the workflow (C5)
Solution The workflow under test sends messages to an external service which the benchmarking system controls. The calling pattern of the service can be checked and compared to the expected interaction.
Example In betsy, this pattern is used to mock any partner service a BPEL workflow is required to communicate with. Moreover, concurrency detection was implemented with a mocked partner service as well.
Relations This pattern is an extension of Message Evaluation (P12). Alternatives are Execution Trace Evaluation (P14) and Engine API Evaluation (P15). Concurrency Detection (P16) can be implemented using this pattern.

Name Execution Trace Evaluation (P14)
Problem Observe the workflow (C5)

Solution The workflow writes log traces to the disk. The benchmarking framework then reads the log traces and compares them with expected ones. Use a small set of different standardized log traces. One can even inspect Detailed Logs (P17) and convert log statements to log traces.
Example This pattern is used in process mining, but also in betsy for observing the behavior of BPMN workflows, as Message Evaluation (P12) does not work because of the lacking support for sending and receiving messages. In script tasks, log traces are written to a log file. Moreover, engine specific logs are checked and additional log traces are created based on them. This is useful for conditions like the detection of whether a workflow did exit correctly.
Relations Partner-based Message Evaluation (P13), Message Evaluation (P12), Engine API Evaluation (P15), Concurrency Detection (P16), and Detailed Logs (P17)

Name Engine API Evaluation (P15)
Problem Observe the workflow (C5)
Solution Use the API provided by the engine to query the deployment state of the workflow model and the current state as well as the history of specific workflow instances. As this is engine dependent, it profits from applying Engine Layer Abstraction (P18) as well.
Example In both betsy and BenchFlow, the BPMN engines are queried about their deployment and final states.
Relations Alternative to Partner-based Message Evaluation (P13), Message Evaluation (P12), and Execution Trace Evaluation (P14), but works fine together with Engine Layer Abstraction (P18).

Name Concurrency Detection (P16)
Problem Observe the workflow (C5)
Solution Identify the parallel branches in the workflow under test. Upon entering and exiting each branch, a timestamp has to be stored alongside a branch identifier at runtime. If the enter-exit pairs of parallel branches overlap, real concurrency has been detected. The concurrency traces can either be tracked through an an external service or a log file.
Example Betsy applies this pattern relying on Partner-based Message Evaluation (P13) for BPEL and Execution Trace Evaluation (P14) with a separate concurrency detection log file for BPMN.
Relations Can be used either with Partner-based Message Evaluation (P13) or Execution Trace Evaluation (P14).

Name Detailed Logs (P17)
Problem Observe the workflow (C5)
Solution Configure the engine to use verbose logging. Otherwise, it might not be possible to observe everything that is important regarding the state of a workflow.
Example In betsy, detailed logs are enabled for several engines by replacing the log configuration file with a more verbose one.
Relations Execution Trace Evaluation (P14) and Concurrency Detection (P16).

Engine Patterns: To *determine the aptitude (C6)*, one can apply the *Aptitude Test (P8). Engine Layer Abstraction (P18)*, *Failable Timed Action (P19)*, *Timeout Calibration (P20)* and *Detailed Logs (P17)* can be used to *interact with the engines (C7).*

Name Engine Layer Abstraction (P18)
Problem Interact with the engines (C7)
Solution Create an abstract layer which a) converts engine independent artifacts to engine dependent ones and vice versa, and b) provides uniform methods to interact with each engine. This handles converting engine specific logs to engine independent log traces, installation, deployment, starting, and other engine specific assertions such as how to behave after an abortion of a workflow.
Example The Uniform BPEL Management Layer (UBML) [11] has been extracted from betsy and it is an engine independent layer to (un)install, start, and stop the engine as well as to deploy workflows and collect log file. The engine adapters of this layer heavily rely on Failable Timed Action (P19), Timeout Calibration (P20), and Detailed Logs (P17).
Relations Can rely upon Failable Timed Action (P19), Timeout Calibration (P20), and Detailed Logs (P17). In addition, eases Reinstallation (P9).

Name Failable Timed Action (P19)
Problem Interact with the engines (C7)
Solution The test system executes a specified action. Then it waits for a specific period during which success and failure conditions are checked every X milliseconds. The action fails if time is exceeded or if failure condition is met. It succeeds if success condition is met within the specific period.
Example As most engines do not support a synchronous API, betsy needs to rely on Failable Timed Action (P19). The act of deploying a workflow often involves copying the artifact to a specific location on the file system, after which the engine deploys it automatically, and then evaluating success through log inspection (see Detailed Logs (P17)).
Relations May require Detailed Logs (P17), should be used with Timeout Calibration (P20).

Name Timeout Calibration (P20)
Problem Interact with the engines (C7) and validate the results (C8)
Solution Before an actual machine is used for benchmarking, calibrate the timeouts that are required in the tests itself, in a Failable Timed Action (P19), or for Reinstallation (P9). The calibration of timeouts in the tests is necessary, in case a test produces non-deterministic results depending on different timeout settings.
Example Betsy implements a mechanism using the Aptitude Test (P8) to calibrate typical timeout values with a security range.
Relations Ensures that the timeouts in Reinstallation (P9) and Failable Timed Action (P19) are suitable.

Results Patterns: To *validate the results (C8)*, one can apply *Open Sourcing (P5)*, *Expert Review (P6)*, *Aptitude Test (P8)*, *Timeout Calibration (P20)* and *Single Success (P21)*.

Name Single Success (P21)
Problem Validate the results (C8) and create tests correctly (C2)
Solution Compare the benchmarking results of one test between engines. If the test does not succeed on at least one engine, it is necessary to investigate the test itself, since the test or the testing procedure might be broken.
Example In betsy, this pattern was applied multiple times to detect issues in the tests, the procedure and the interaction with engines.

Discussion: Most of the patterns are only used for a single challenge. The pattern Detailed Logs (P17) can be applied twice, once to observe the workflow (C5) and to interact with the engines (C7). Three patterns, namely, Open Sourcing (P5), Expert Review (P6), and Aptitude Test (P8), are applicable to three challenges each. Open Sourcing (P5) and Expert Review (P6) are more general patterns not specific to workflow benchmarking, but they helped us tremendously with quality assurance, namely, to create tests correctly (C2), validate the procedure (C3), and validate the results (C8). The Aptitude Test (P8) is more specific to the benchmarking domain, and can be used to validate the procedure (C3), determine the aptitude (C6), and validate the results (C8).

5 Conclusion and Future Work

In this paper, we introduced pattern candidates for workflow engine benchmarking. We identified the central elements in this domain, as well as major challenges one faces when building a benchmark. The patterns we proposed are categorized, according to these aspects, into test, procedure, engine, and results patterns.

In the future, we want to improve and refine the pattern candidates with domain experts, i.e., the authors of other workflow engine benchmarks or workflow engines. In the long term, these patterns could provide a useful basis for future benchmark authors and could help to establish a common vocabulary in the domain. Last, we see potential in generalizing these specific patterns for benchmarking workflow engines to more generic patterns for benchmarking other standard-based software or even to patterns for benchmarking in general.

References

1. C. Alexander. A Pattern Language, Aug. 1978.
2. D. Bianculli, W. Binder, and M. L. Drago. SOABench: Performance Evaluation of Service-oriented Middleware Made Easy. In *ICSE*, pages 301–302. ACM, 2010.
3. V. Ferme, A. Ivanchikj, and C. Pautasso. A Framework for Benchmarking BPMN 2.0 Workflow Management Systems. In *BPM*. Springer, Aug. 2015.

4. V. Ferme, A. Ivanchikj, C. Pautasso, M. Skouradaki, and F. Leymann. A Container-centric Methodology for Benchmarking Workflow Management Systems. In *CLOSER*, 2016.
5. E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Amsterdam, 1995.
6. M. Geiger, S. Harrer, and J. Lenhard. Process Engine Benchmarking with Betsy – Current Status and Future Directions. In *ZEUS*, pages 37–44, Jan. 2016.
7. M. Geiger, S. Harrer, J. Lenhard, M. Casar, A. Vorndran, and G. Wirtz. BPMN Conformance in Open Source Engines. In *SOSE*, Mar. 2015.
8. M. Geiger, S. Harrer, J. Lenhard, and G. Wirtz. On the Evolution of BPMN 2.0 Support and Implementation. In *SOSE*, pages 120–128, Mar. 2016.
9. S. Harrer. Process Engine Selection Support. In *OTM 2014 Workshops*, volume 8842 of *LNCS*, pages 18–22. Springer, 2014.
10. S. Harrer, J. Lenhard, and G. Wirtz. BPEL Conformance in Open Source Engines. In *SOCA*, pages 237–244. IEEE, Dec. 2012.
11. S. Harrer, J. Lenhard, G. Wirtz, and T. van Lessen. Towards Uniform BPEL Engine Management in the Cloud. In *CloudCycle*, LNI. GI e.V., Sept. 2014.
12. S. Harrer, F. Nizamic, G. Wirtz, and A. Lazovik. Towards a Robustness Evaluation Framework for BPEL Engines. In *SOCA*, pages 199–206. IEEE, Nov. 2014.
13. S. Harrer, C. Preißinger, and G. Wirtz. BPEL Conformance in Open Source Engines: The Case of Static Analysis. In *SOCA*, pages 33–40. IEEE, Nov. 2014.
14. S. Harrer, C. Röck, and G. Wirtz. Automated and Isolated Tests for Complex Middleware Products: The Case of BPEL Engines. In *ICSTW*, 2014.
15. K. Huppler. The Art of Building a Good Benchmark. In *Performance Evaluation and Benchmarking*, pages 18–30. Springer, 2009.
16. ISO/IEC. *ISO/IEC 19510:2013 – Information technology - Object Management Group Business Process Model and Notation*, Nov. 2013. v2.0.2.
17. C. Kohls. The structure of patterns. In *PLOP*. ACM, 2010.
18. C. Kohls. The structure of patterns - part ii - qualities. In *PLOP*. ACM, 2011.
19. G. Meszaros and J. Doble. A pattern language for pattern writing. *Pattern languages of program design*, 3:529–574, 1998.
20. OASIS. *Web Services Business Process Execution Language*, Apr. 2007. v2.0.
21. M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: A Research Roadmap. *IJCIS*, 17(2):223–255, 2008.
22. C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, Oct. 2003.
23. M. Skouradaki, V. Ferme, C. Pautasso, F. Leymann, and A. van Hoorn. Micro-Benchmarking BPMN 2.0 Workflow Management Systems with Workflow Patterns. In *CAiSE*, pages 67–82. Springer, June 2016.
24. M. Skouradaki, D. H. Roller, F. Leymann, V. Ferme, and C. Pautasso. On the Road to Benchmarking BPMN 2.0 Workflow Engines. In *ICPE*. ACM, 2015.
25. W. M. P. van der Aalst. Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, pages 1–37, 2013.
26. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, July 2003.
27. WfMC. *The Workflow Reference Model*, Jan. 1995. v1.1.
28. P. Wohed, M. Dumas, A. H. M. T. Hofstede, and N. Russell. Pattern-based Analysis of BPMN - An extensive evaluation of the Control-flow, the Data and the Resource Perspectives (revised version). BPM Center Report BPM-06-17, 2006.