Process Engine Selection Support

Simon Harrer

Distributed Systems Group, University of Bamberg, Germany
simon.harrer@uni-bamberg.de

Abstract. Nowadays, business processes and their execution are corner stones in modern IT landscapes, as multiple process languages and corresponding engines for these languages have emerged. In practice, it is not feasible to select the best fitting engine, as engine capabilities are mostly hidden in the engine implementation and a comparison is hampered by the large differences and high adoption costs of the engines. We aim to overcome these problems by a) introducing an abstract layer to access the functionality of the engines uniformly, b) by revealing the engine capabilities through automated and isolated tests for typical requirements, and c) support the user in their selection of a process engine by determining and explaining the fitness of the engines for a single process or a given set of processes using policy matching against previously revealed engine capabilities. Early results show the general feasibility of our approach for BPEL engines for a single capability.

Keywords: BPM, process engines, engine selection, execution requirements, testing

1 Introduction

Today, process-centric information systems [11] are corner stones in the current IT landscapes in industry, creating a $6.6 billion dollar market of Business Process Management (BPM) solutions [1]. Within this market, at least 19 different business process modeling languages have emerged [6]. Typically, these languages define execution semantics, are implemented in process engines onto which processes are deployed, and instances of these processes are executed by interacting with the engines through message passing. In this work, we focus on such executable process languages, e.g., the OASIS standard BPEL [7] and the OMG standard Business Process Modeling and Notation 2.0 (BPMN) [8], as both are defined in a standard, widely used, executable, and have multiple implementations. There are at least 31 BPMN engines[1] and more than ten BPEL engines available on the market, which vary greatly in their feature set [2]. Hence, the selection of an engine for a given process is a real practical issue in various situations. A typical scenario is that a developer requires an engine with a small footprint which may have slow performance for development whereas the production server may use an engine with a large footprint but with high performance characteristics, hence, the best engine depends on more than functional properties of the process. And a company may use more than one engine, as having multiple runtimes is typical in state-of-the-art

[1] See `http://www.bpmn.org/#tabs-implementers`.

practice and feasible with current cloud and virtualization techniques. In a typical web application, multiple programming languages, database systems and different cloud services are already used extensively.

To foster a constant improvement of processes in industry, it is standard practice to apply the Business Process Management (BPM) lifecycle, which consists of four steps within a closed loop. First, the process is modeled using the vocabulary of a process language. Second, the engine, i.e., the runtime environment, is selected, setup and configured in the *system configuration* step. Third, the process is deployed onto and its instances are executed on the previously provisioned engine, the so called *process enactment*. Fourth, the execution of the process instances is diagnosed via audit trails, which are used to model an improved process in the next iteration.

The issue of this BPM lifecycle is that while we design processes using a standardized process language, the selection of the best fitting engine for a given process is not feasible at the current status-quo. For both BPEL and BPMN, there are many engines that vary greatly in their capabilities, including the level of standard conformance, performance, robustness, installability, security, monitoring functionality, licensing cost and vendor support available. Hence, a *model once, run anywhere* paradigm does not hold here, requiring adaption costs. What is more, due to the complexity of these process languages and their engines, a comparison is very time intensive and requires detailed knowledge of the engines. Moreover, although these engines support standard conform and vendor-independent processes, the management of the engines is vendor-dependent, i.e., every engine has its custom installation, startup and shutdown routines, and different deployment methods as well as descriptors. Hence, an automated selection of an engine for a given process and its execution on the selected engine is not feasible to date, resulting in ill-informed selection decisions for engines with possible hidden costs.

In this work, we aim to overcome these problems by improving the *system configuration* step of the BPM lifecycle. Hence, we tackle the research question: *How to determine and explain the fitness of a set of engines for a set of processes?*

This work is structured as follows. In Sect. 2, related work is outlined. The research hypotheses are given in Sect. 3, and the approach is explained in Sect. 4. The preliminary results are given in Sect. 5, followed by an evaluation in Sect. 6. A discussion and future work is outlined in Sect. 7. Section 8 concludes this work.

2 Related Work

The selection or comparison of process engines is neglected in research so far, except for our preliminary work which is explained in Sect. 5. Engines have only been compared according to their support for workflow patterns. Such a pattern is a reusable solution to a reoccurring problem in a specific context and the support of a pattern is determined by the effort required to implement it. A plethora of pattern catalogs and corresponding evaluations exist[2], e.g., the control-flow [10] patterns. Such evaluations reveal helpful engine capabilities, but render only partially relevant for our case as this would require determining the used patterns in a given process. In addition, they only focus on the functionality of a process, neglecting non-functional aspects.

[2] See `http://www.workflowpatterns.com/`.

Instead of selecting engines, there are studies to compare and select process languages which we aim to learn from. While Kopp et al. [4] distinguish between block- or graph-based process modeling languages, Lu et al. [5] categorize several business process modeling approaches as either rule-based or graph-based approaches, and then compare them with five different criteria. In [9], the fitness of a process language for a given set of requirements is evaluated.

3 Research Hypotheses

The research question formulated in Sect. 1 is translated into the following three research hypotheses which build on one another.
H1: Uniform Process Engine Management The lifecycle and management functionality of different engines and their processes is vendor-dependent, but can be managed uniformly and vendor-independently.
H2: Revealing Engine Capabilities Through Tests Methodically created tests are suitable to reveal functional and non-functional capabilities of engines.
H3: Explained Engine Selection Through Policy Matching Policies are suitable to formalize the execution requirements of a process and the engine capabilities, and therefore can be used to determine and explain the fitness of process engines for a set of processes.

4 Approach

In this section, we elaborate the overall approach to support the hypothesis from Sect. 3 and how they build upon another.

We propose to support H1 by introducing a Uniform Process Management Layer (UPML) for managing engines uniformly. UPML provides mappings for the engines in different versions, different configurations (e.g., executing processes in-memory or not), and different environments (e.g., locally, in a virtual machine or in the cloud). Such a layer is required for the BPM lifecycle steps *system configuration* and *process enactment*, as both require that the selected engine has to be available (installed, configured and started) and the process has to be executed (deployed and instances of it executed). Hence, our layer includes lifecycle methods of the engine (install, start, stop, and uninstall) and of the process (deploy and undeploy).

To support H2, we use a methodical approach to determine engine capabilities through tests. For each typical process requirement, we create executable tests that are executed on top of the UPML via a testing tool. A test consists of a process, a list of messages the process that are going to be sent as requests along with expectations to the corresponding responses. This method consists of a generic part applicable for every requirement and a customized part which has to be fitted for each requirement. The produced test results are converted to engine capabilities which conform to the policy type that provides a mean to formalize different degrees of fulfillment of the typical process requirement. To ensure that we cover the right requirements, we conduct a literature study, create tests for the found requirements, and execute them reusing the UPML.

H3 is supported by providing a framework that allows formalizing the process requirements in terms of policies. A policy contains constraints, e.g., a specific language feature has to be available, or the engine must start in less than 5 seconds. By encoding all the engine capabilities and the process requirements as XML-based policies, we can determine the fitness of an engine for a process, as well as provide estimations on adaption costs for possible alternatives , e.g., portability costs to change a process for another engine. The latter is especially interesting in the cases of finding no or more than one engine that is considered fit, as well as finding a single one which is not acceptable for the user. When determining the fitness for a set of processes, we will aggregate the atomic fitness results to enable informed decisions as well. We use the policy serialization and matching algorithm of Web Services Policy 1.5 Framework [12] for our needs.

5 Preliminary Results

To validate the approach outlined in Sect. 4, we conducted a case study for BPEL engines in [3] which builds upon additional previous work cited there. We showed that it is possible to overcome the huge API differences of seven open source BPEL engines by means of a uniform management API (H1), to determine standard conformance capabilities of engines using automated tests (H2), to select the best fitting engine for a process using previously revealed standard conformance capabilities as well as to automatically execute the process on the selected engine (H3).

6 Evaluation

We use multiple methods to ensure the validity of the results. Regarding H1, we evaluate the correctness of UPML by checking whether both, the simplest possible process and the process that only uses features that are supported by every engine, can be deployed and executed on every of the supported engines. Regarding H2, we focus on ensuring that the tests *are correct* by checking syntax and semantics via XML and XML Schema validations automatically, *test the correct features* by conducting peer-reviews within our group and with engine experts, and *produce the correct results* by fully automating the test procedure for reproducible results, repeating the test procedure multiple times to prevent any alternations, providing a fresh engine instance for each single test to avoid side-effects between tests, and compare the results to one another to detect any anomalies, e.g., take the log files into account for tests that are not successful on any engine. Regarding H3, we evaluate the produced prototype with a large set of processes as training data to test the correctness of the selection procedure.

7 Discussion and Future Work

Instead of using only workflow patterns for a manual comparison of engines, we provide an approach that reveals the capabilities of engines and determines the fitness of a set of engines for a given set of processes, and can automatically execute them on the selected

engine. In the future, we aim to complete the BPEL case study in Sect. 5 for multiple engine capabilities. Furthermore, we aim to conduct a second case study using BPMN, proving the applicability of our approach on processes and engines using another process language.

8 Conclusion

We have presented an approach to help in the selection of the best fitting engine for processes and their execution on selected engines, enabling to use the best available runtime for given processes, or provide guidance for process adaption to target other engines. Early results show the general feasibility of the approach for the process language BPEL, seven open source BPEL engines and the selection criteria standard conformance, hence, answering the research question partly.

References

1. C. L. Clair, A. Cullen, and J. Keenan. Prepare For 2013's Shifting BPM Landscape. Technical report, Forrester Research, January 2013.
2. 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*, pages 1–8. IEEE, 17–19 December 2012.
3. S. Harrer, J. Lenhard, G. Wirtz, and T. van Lessen. Towards Uniform BPEL Engine Management in the Cloud. In *INFORMATIK*, September 2014. (to appear).
4. O. Kopp, D. Martin, D. Wutke, and F. Leymann. The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. *Enterprise Modelling and Information Systems Architectures*, 4(1):3–13, 2009.
5. R. Lu and S. Sadiq. A Survey of Comparative Business Process Modeling Approaches. In *In Proceedings 10th International Conference on Business Information Systems (BIS), number 4439 in LNCS*, pages 82–94. Springer, 2007.
6. H. Mili, G. Tremblay, G. B. Jaoude, E. Lefebvre, L. Elabed, and G. E. Boussaidi. Business Process Modeling Languages: Sorting Through the Alphabet Soup. *ACM Comput. Surv.*, 43(1):4:1–4:56, December 2010.
7. OASIS. *Web Services Business Process Execution Language*, April 2007. v2.0.
8. OMG. *Business Process Model and Notation*, January 2011. v2.0.
9. S. Thöne, R. Depke, and G. Engels. Process-oriented, flexible composition of web services with UML. In *Advanced Conceptual Modeling Techniques*, pages 390–401. Springer, 2003.
10. W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, July 2003.
11. W. M. P. van der Aalst, A. H. ter Hofstede, and M. Weske. Business Process Management: A Survey. In *Proceedings of the International Conference on Business Process Management*, Eindhoven, The Netherlands, 2003. Springer Berlin Heidelberg.
12. W3C. *Web Services Policy Framework (WS-Policy)*, September 2007. v1.5.