Literature Survey of Performance Benchmarking Approaches of BPEL Engines

Cedric Röck, Simon Harrer

May 2014
Due to hardware developments, strong application needs and the overwhelming influence of the internet, distributed systems have become one of the most important topics for nowadays software industry. Owing to their ever increasing importance for everyday business, distributed systems have high requirements with respect to dependability, robustness and performance. Unfortunately, distribution adds its share to the problems of developing complex systems. Heterogeneity in both, hardware and software, frequent changes, concurrency, distribution of components and the need for inter-operability between systems complicate matters. Moreover, new technical aspects like resource management, load balancing and guaranteeing consistent operation in the presence of partial failures put an additional burden onto the developer.

Our long-term research goal is the development, implementation and evaluation of methods helpful for the realization of robust and easy-to-use software for complex systems in general while putting a focus on the problems and issues regarding distributed systems on all levels. This includes design methods, visual languages and tools for distributed systems development as well as middleware, SOA and cloud computing issues. Our current research activities focus on different aspects centered around that theme:

- **Implementation of Business Processes and Business-to-Business-Integration (B2Bi):** Starting from requirements for successful B2Bi development processes, languages and systems, we investigate the practicability and inter-operability of different approaches and platforms for the design and implementation of business processes.

- **Quality, esp. Robustness, Standard-conformance, Portability, Compatibility and Performance of Process-based Software and Service-oriented Systems:** In both, industry and academia, process languages have emerged, e.g., Windows Workflow (WF), Business Process Model and Notation (BPMN) and Web Services Business Process Execution Language (WS-BPEL). Although widely used in practice, current implementations of these languages and models are far from perfect. We work on metrics to compare such languages w.r.t. expressive power, conformance and portability as well as additional quality properties, such as installability, replaceability, adaptability and inter-operability. These metrics are developed and validated formally as well as evaluated practically. In the context of BPMN, we work on tools to check for and improve the standard compliance for human-centric process models on different layers of abstraction. Runtime environments for process languages with a focus on BPEL are investigated by means of a framework that eases the comparative test of different run-times and process engines.

- **Cloud Application Portability:** The hype surrounding the Cloud has lead to a variety of offerings that span the whole cloud stack. We examine important aspects of portability in cloud environments and enhance the portability of cloud applications by applying common standards between heterogeneous clouds. We make use of a holistic view of the cloud including important aspects like cloud specific restrictions, platform configurations, the deployment and life cycle of cloud applications.

- **Visual Programming- and Design-Languages:** The goal of this long-term effort is the utilization of visual metaphors and visualization techniques to make design- and programming languages more understandable and, hence, more easy-to-use. Currently, languages for designing and programming sensor networks are at the focus of this effort.

More information about our work can be found at www.uni-bamberg.de/en/pi/. If you have any questions or suggestions regarding this report or our work, don’t hesitate to contact us.

Bamberg, June 2014

Guido Wirtz
Literature Survey of Performance Benchmarking Approaches of BPEL Engines

Cedric Röck, Simon Harrer

Abstract  Despite the popularity of BPEL engines to orchestrate complex and executable processes, there are still only few approaches available to help find the most appropriate engine for individual requirements. One of the more crucial factors for such a middleware product in industry are the performance characteristics of a BPEL engine. There exist multiple studies in industry and academia testing the performance of BPEL engines, which differ in focus and method. We aim to compare the methods used in these approaches and provide guidance for further research in this area. Based on the related work in the field of performance testing, we created a process engine specific comparison framework, which we used to evaluate and classify nine different approaches that were found using the method of a systematical literature survey. With the results of the status quo analysis in mind, we derived directions for further research in this area.

Keywords  SOA, BPEL, process engines, performance testing

Contact:

cedric.roeck@uni-bamberg.de, simon.harrer@uni-bamberg.de
Contents

1 Introduction ........................................... 1

2 Method .................................................... 2

3 Approaches ............................................... 3

4 Comparison Framework and its Criteria ................. 4
   4.1 Primary Criteria ...................................... 4
      4.1.1 Types of Performance Tests ......................... 5
      4.1.2 Workload: Definition, Categories and Injection .......... 5
      4.1.3 Performance Metrics ................................. 6
   4.2 Secondary Criteria ..................................... 7

5 Results .................................................... 8

6 Discussion and Further Suggestions .................... 11

7 Conclusion and Future Work ........................... 13

Bibliography ............................................... 14

8 List of previous University of Bamberg reports .... 17
List of Figures

1. Influences on the performance of a software component [20] 4
2. Benchmark hierarchy [25, p. 266] 6
3. Relevant timings of a synchronous request-response transaction [31] 7
List of Tables

<table>
<thead>
<tr>
<th>#</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Literature analysis with primary criteria</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Literature analysis with secondary criteria</td>
<td>9</td>
</tr>
</tbody>
</table>
1 Introduction

Over the past few years, the research concerning the Web Services Business Process Execution Language (BPEL) \cite{27} made huge progress and focused on arising chances and challenges for businesses \cite{7}. Based on service-oriented architectures (SOAs), one of the major trends in the development of business information systems, BPEL steadily became the standard for Web Service based business orchestration \cite{21}. In addition, BPEL is also used to provision software in the cloud \cite{6} and to implement partner specific views of a choreography, in the domain of business to business (B2B) integration \cite{29}. Strongly connected to the growing popularity is the development of more complex systems, which also leads to an increased error-proneness of the developed software \cite{3}. Therefore, the need to intensively test those SOAs also gains more importance. However, there still is a tremendous deficit in terms of proper testing tool support, which has been considered to be one of the major problems of SOAs \cite{8}.

A classic goal of performance testing has always been to compare competing platforms, allowing the selection of the best fitting product for particular needs \cite{38}. Despite the grown acceptance of SOAs, performance testing in this area still lacks some major aspects. While the number of approaches steadily grows, the majority focuses solely on the evaluation of single Web services instead of middleware components or even complete systems \cite{19}. However, “the middleware used to build a distributed application often determines the overall performance of the application” \cite{11}, p. 2] and should therefore be considered at least as carefully as the choice of partner services involved in a process. Looking at performance testing of BPEL processes and their engines, multiple studies have been conducted in industry and academia. However, there is no standardized benchmark, let alone a commonly agreed upon testing methodology \cite{10}. As a consequence, the current view on the status quo and further research topics is clouded. In this work, we aim to provide a clear view by means of a selection of comparison criteria and a literature survey of the current approaches, as well as their classification into our framework.

Our contribution comprises three parts. First, a literature survey to find the available approaches that perform performance benchmarking of BPEL engines. Its method is outlined in Section 2 whereas the nine found approaches are described in Section 3. Second, a framework to compare performance benchmarking approaches of process engines. The framework and its criteria are detailed in Section 4 and applied onto the found approaches from Section 3, for which the results are given in Section 5. Third, in Section 6 directions on improving performance benchmarking of process engines in general and BPEL engines in particular based on an evaluation of the status quo comparison are given. Our work concludes in Section 7 which also outlines future work.
2 Method

To ensure that we find the relevant performance benchmarking approaches of BPEL engines, we conducted an extensive and systematical literature survey. At first, we searched in the archives of the ACM\(^1\) as well as IEEE\(^2\) and with the meta search engine Google scholar\(^3\). Additionally, we also leveraged the search engine Google itself for finding approaches in industry as well. For our search, we used the keyword combinations \textit{bpel performance}, \textit{bpel performance benchmark}, \textit{bipel performance testing}, \textit{middleware performance} and \textit{web service performance} as our search queries. Next, we filtered down the search results in two phases: firstly, by reading the title, and secondly, by skimming the abstract. Thereafter, we performed forward and backward searches on the remaining references.

Using our method for the literature survey, we found nine BPEL middleware performance test projects, which are briefly described in Section \ref{section:method}. While the benchflow\(^4\) project would be the perfect candidate at creating a benchmark for BPEL engines, it is still in its early stages. During our literature survey, we also came across a plethora of additional related studies which either focus on other middleware products, e.g., benchmarking the performance of enterprise service buses (ESBs) with ESBPerformance\(^5\) or focus on performance modeling as well as prediction \cite{33, 32, 36}, and simulation \cite{9}. However, these are out of scope.

\footnotesize
\begin{itemize}
\item \(^1\)http://dl.acm.org
\item \(^2\)http://www.computer.org/portal/web/search
\item \(^3\)http://scholar.google.com
\item \(^4\)http://www.iaas.uni-stuttgart.de/forschung/projects/benchflowE.php
\item \(^5\)See http://esbperformance.org/
\end{itemize}
3 Approaches

In this section, the nine BPEL performance benchmarking approaches found using the method from Section 2 are briefly described. Their full evaluation using the comparison framework from Section 4 is available in Section 5.

Multiple studies have originated from the engine vendors themselves. This includes Intel and Cape Clear with their Cape Clear BPEL Engine [18], Sun Microsystems with its OpenESB [35], and ActiveEndpoints with ActiveVOS [1]. In 2007, Intel and Cape Clear published a White Paper [18] in which they demonstrate their method to evaluate the performance and horizontal scaling capabilities of their BPEL engine Cape Clear 7. More specifically, they focus on the clustering capabilities of their engine, and among others on the ability to recover from single server crashes within a cluster. In contrast, ActiveEndpoints tested their BPEL engine ActiveVOS to verify the maximum load its engine is able to handle, whereas Sun Microsystems benchmarked their BPEL Engine within the OpenESB bundle [35] to spot performance degradation during development.

In academia, there are on the one hand approaches that benchmark the performance of BPEL engines to proof the validity of their contribution [30, 4, 13, 22] and on the other hand approaches that solely focus on benchmarking the performance of BPEL engines [23, 12]. Roller [30] worked on optimizing the performance of workflow engines in general and applied his techniques onto BPEL engines in a case study. To measure the success of his proposed improvements, he conducted performance tests before and after applying his optimizations. Hackmann et al. [13] developed a lightweight BPEL engine for mobile devices, named Sliver, and used performance benchmarking as a means to evaluate whether their engine has the expected characteristics to make it suitable for being executed on mobile devices in comparison to the typical BPEL engine that is targeted for powerful server systems. Liu et al. [22] propose a framework for fault-tolerant composition of transactional web services (FACTS) which can automatically introduce fault-handling logic into BPEL processes. To determine the performance impact of the additional fault-handling logic, they evaluated the performance costs corresponding to each fault-handling strategy on a single BPEL engine. Bianculli et al. [5, 4] developed SOABench, a testbed generation framework to benchmark the performance of arbitrary service-oriented middleware. To proof the applicability of their approach, they benchmarked the performance of BPEL engines in a case study. In contrast, Din et al. [12] created a configurable workload model to specifically benchmark the performance of different configurations of a BPEL engine including an evaluation, whereas Längerer et al. [23] solely conducted a performance comparison in their more practical study.
Performance testing refers to the usually technical examination of a system under test (SUT) with the aim to analyze and validate a product’s characteristics. While it is known for allowing a measurement-based comparison of different products and platforms under similar conditions, hence offering valuable information for purchase decisions\textsuperscript{38}, performance testing can also refer to the model or simulation based evaluation, which can be realized at much earlier stages during the development cycle. Furthermore, performance testing is also used to identify bottlenecks and verify the requested quality of service (QoS) attributes in nearly finished software, i.e., whether the predefined nonfunctional performance requirements, which are often party of contractually binding service level agreements (SLA), are met\textsuperscript{39}.

According to Koziolek\textsuperscript{20}, the performance of a software component is influenced by five factors: its implementation, the usage profile, deployment platform, required services and resource contention. These influences are also illustrated in Figure 1. Focusing on the test of BPEL middleware, neither the usage profile, nor implementation details of the component are important for our work, as they should be identical for all systems under test. Instead, we focus mainly on the influence of the deployment platform, namely the BPEL engine, while adjusting and monitoring the behavior of required services, and available resources.

![Figure 1: Influences on the performance of a software component\textsuperscript{20}](image)

Based on these influences, we created a framework to enable the classification of BPEL engine performance tests. In the following, we present four primary (see Section 4.1) and three secondary criteria (see Section 4.2), according to which BPEL performance test approaches can be analyzed and made comparable.

### 4.1 Primary Criteria

The primary criteria include the performance test type (Section 4.1.1), the workload along its categorization and injection strategies (Section 4.1.2), and the metrics according to which the workload execution on the system under test is measured (Section 4.1.3).
4.1 Primary Criteria

4.1.1 Types of Performance Tests

Measurement based performance tests can be executed in several ways depending on the intended purpose of the test. These different execution strategies establish the *types* of performance tests, namely *baseline*, *stress* and *load* tests [24].

*Baseline tests* measure the response time behavior of an application with only a single request (or at most a single concurrent user), and therefore represent the best-case scenario. Consequently, it can either be used for product comparison itself or as a benchmark baseline for other test types, e.g., for load tests [26, pp. 38-40].

A *load test* verifies the application behavior in typically expected situations, which also include load peaks with multiple concurrent users. It is often used to prove that an application meets performance related quality of service (QoS) attributes [24, 9] [25, p. 291].

Pushing an application beyond the expected level of load and peak conditions is called *stress testing*. Its goal lies in revealing bugs, as well as finding SLA violations and unexpected application behavior that only appear in extreme conditions [24]. The results are also referred to as worst-case scenario and hint at the application’s capacity limits [26, pp. 38-40] [25, p. 291].

4.1.2 Workload: Definition, Categories and Injection

Workloads are defined as the sum of all inputs that are received by the system under test. They are considered to be one of the key aspects for assuring the validity of performance test results [34, 39, 38, 25]. In the context of BPEL engines, the workloads consists of the invoked processes and their dependent Web services. This also includes the requests starting the BPEL processes and the strategy for sending these requests.

According to Menascé [25, pp. 265-266], the workload of software performance benchmarks can be categorized into four groups, which are illustrated in Figure 2. Basic operations, the innermost circle, refer to the smallest offered operations, or all supported and standardized activities in the context of a BPEL engine and supply more fine-grained performance characteristics. Toy-benchmarks, usually implementing classic puzzles or being proof-of-work concepts, are of not much use for performance tests. Kernels, which are shown in the third circle, are core parts of real programs. They represent the most important or time consuming parts of applications. In our context, they can be seen as processes implementing patterns, e.g., the workflow patterns [36] that were extracted from a large corpus of real world processes. They provide an intermediate level in terms of granularity and realism. Real programs, or real processes in our case, are often seen as the most important category of workloads because their results are most accurate for estimating the performance of production programs or processes [2, 34]. Because of this, real workloads are also used in other domains, e.g., for benchmarking SQL databases with TPC-C[6].

The previously specified workloads also have to be injected into the system under test. These injection strategies can be distinguished by their time to fully inject a complete workload and

---

to measure the execution at different points in time during the injection.

The workload injection strategies can be subdivided into a) continuously injecting the workload and b) injecting the workload with an arrival rate. Continuously injected workloads test the engine with a constant load, once reaching the plateau phase. This plateau, however, can either be reached with a big bang (default approach), or stepwise, by slowly increasing the number of concurrent users over a specific period [26, p. 41]. The arrival rate can be either fixed, e.g., 20 requests are sent every minute, or dynamic to simulate even more realistic peaks [34].

If the system under test and load generating clients are co-located, one has to closely observe the system for overloads. Therefore, they are often installed on separate systems, i.e., distributed. Moreover, a separation allows the load to be created from distributed clients, providing more realistic situations and assuring the saturation of the system under test.

4.1.3 Performance Metrics

The evaluation of software performance usually considers multiple metrics. In this study, we use three main performance metrics: latency, throughput and utilization.

The latency or response time [11] defines the time span between sending a request and having received the full response. As shown in Figure 3, the response time is influenced by multiple factors. The execution time of partner processes $t_e$, the network delay $t_l$ and the wrapping and parsing of XML messages $t_w$ [31]. For BPEL processes, the classic response time must further be split-up into three distinguishable measures, the execution and the creation as well as the update response times. The execution response time refers to the classic response time, instantiating the process upon receiving the request as well as terminating it upon sending the response. In contrast, when having long running interactions, the creation response time measures the duration from sending a request instantiating the process until receiving the corresponding response, whereas the update response time measures the duration from correlating a sent request with an existing process instance until receiving the corresponding response [18].

Throughput is the most commonly used metric for software performance, defining the number of transactions that are completed within a specific time. In the context of BPEL engines, the term transaction can refer either to the completion of requests or process instances.

The utilization reflects the degree to which a capacity is used and refers to many parts of the

---

7In this context, metrics are also referred to as criteria, indicators or targets in literature.
4.2 Secondary Criteria

Beyond the previous primary criteria, there are far more classification possibilities. We chose to further distinguish between three additional criteria. First, the license and the number of the system under test, being either open source or proprietary. Second, whether the setup of the test bed is automated or done manually, as this has an effect on the reproducibility of the experiment, which has previously been identified as one of the key aspects for a successful and widely accepted benchmark [17]. And third, as BPEL engines provide plenty of configuration options, we focus on whether the processes are executed in-memory or not, as this is a configuration option available within most BPEL engines, which we denote as persistence in our framework.
5 Results

The approaches in Section 3 are evaluated according to the criteria of our comparison framework in Section 4. The results are listed in Table 1 and Table 5, in which each approach is detailed in a separate row. The characteristics are grouped by their criteria of our comparison framework and shown in the header of the table in the first row. The cells represent the findings, being either filled or empty. Empty cells denote the absence of a criterion, whereas cells marked n/a denote that the approach did not provide any information regarding these criteria. We state the versions of the BPEL engines used in each approach in the following paragraphs, however, in some cases, the version is unknown.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Test Type</th>
<th>Workload</th>
<th>Workload Injection</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>Load</td>
<td>Stress</td>
<td></td>
</tr>
<tr>
<td>Approaches</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOABench [4]</td>
<td>x x</td>
<td>basic</td>
<td>4 x</td>
<td>x x</td>
</tr>
<tr>
<td>OpenESB [35]</td>
<td>x</td>
<td>toy</td>
<td>1 n/a</td>
<td>x</td>
</tr>
<tr>
<td>ActiveVOS [1]</td>
<td>x</td>
<td>real</td>
<td>2</td>
<td>n/a n/a n/a n/a</td>
</tr>
<tr>
<td>Sliver [12]</td>
<td>x</td>
<td>kernel</td>
<td>12 x</td>
<td>x</td>
</tr>
<tr>
<td>Workload model [12]</td>
<td>x</td>
<td>toy</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Intel &amp; Cape Clear [18]</td>
<td>x</td>
<td>real</td>
<td>2 x</td>
<td>x x</td>
</tr>
<tr>
<td>Roller [30]</td>
<td>x</td>
<td>real</td>
<td>1 x</td>
<td>x</td>
</tr>
<tr>
<td>SWoM [23]</td>
<td>x</td>
<td>basic</td>
<td>2 x</td>
<td>x</td>
</tr>
<tr>
<td>FACTS [22]</td>
<td>x</td>
<td>real</td>
<td>1 x</td>
<td>n/a n/a n/a n/a</td>
</tr>
</tbody>
</table>

Table 1: Literature analysis with primary criteria

The benchmark in SOABench [4] comprises two open source engines, jBPM and Apache ODE in the two versions v.1.3.3 and v.2.0-beta2, as well as the proprietary ActiveVOS v.5.0.2 Server Edition. However, at the time of the benchmark, the open source BPEL engine ActiveBPEL v5.0.2 has been part of the corresponding proprietary product, so the essential part of the software was open source. SOABench performs load tests using four BPEL processes. Two processes are built with the flow activity, the other two use only the sequence and the while activity. In addition, all four processes invoke mocked-up external services, thus, use the invoke activity as well. The performance is measured by means of response-time behavior (latency) and throughput metrics. Being based on the tool Weevil [37], SOABench features the automated generation, execution and analysis of testbeds. Hence, it allows repeating the test and reproduce the test results. Workloads are either injected based on a given arrival rate, or delayed by a thinking time between consecutive invocations.

The performance test of OpenESB [35] does not include other engines and has been executed as a load test focusing only on the throughput. The workload is a single process, using a mixture of all supported BPEL activities, i.e., a toy benchmark, and is injected stepwise.
Table 2: Literature analysis with secondary criteria

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Criteria</th>
<th>SUT</th>
<th>Config</th>
<th>Testbed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOABench [4]</td>
<td></td>
<td>2</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>OpenESB [35]</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ActiveVOS [1]</td>
<td></td>
<td>1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Sliver [13]</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workload model [12]</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel &amp; Cape Clear  [18]</td>
<td></td>
<td>1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Roller [30]</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWoM [23]</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FACTS [22]</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The performance evaluation of ActiveVOS [1] does not include any other engines as well. The load test in this study monitors the engine’s throughput, uses two functional processes, with one process being analyzed with persistence enabled or disabled.

Hackmann et al. [13] have compared Sliver with ActiveBPEL v2.0.1.1, both being open source engines. The baseline test measures the request latency as well as the memory utilization of the engines. The workload consists of twelve of the original workflow patterns [36], i.e., kernels, which have been realized as BPEL processes that utilize external services and are invoked in consecutive requests.

Validating their workload model, Din et al. [12] have performed a load test that focuses only on the response time behavior (latency) of the ActiveBPEL engine, distinguishing between the creation and update response times. The tested process uses correlations but does not call external services. The workload is injected by several virtual users over a total duration of two minutes, injecting 20 new processes per second.

The performance test of Intel and Cape Clear [18] has been executed as a load test and focuses on latency, throughput and utilization measures of the tested Cape Clear 7.0 beta 2 engine. The two tested processes invoke external services and implement a functionality which is typical for industry processes. Additionally, the test also focuses on the effect of persistence on the performance metrics. The workload is continuously injected from a set of distributed clients.

Verifying Roller’s [30] proposals, his performance test has been executed as a load test on the IBM WebSphere 7.0 64bit engine. His approach measures only throughput metrics. The tested workload is a single realistic BPEL process, including the invocation of external services, which has been continuously called by the testing clients.

Benchmarking three proprietary BPEL engines, the Stuttgarter Workflow Maschine (SWoM) 1.0, the IBM WebSphere Process Server 6.0.2.3 and the Oracle BPEL Process Manager, Längerer et
al. [23] have conducted a load test focusing on throughput and latency metrics. The workload, which is continuously injected, consists of two processes. One uses the assign activity, whereas the other one calls an external Web service with the invoke activity.

The load test of Liu et al. [22] tests the response time behavior (latency) of a single realistic process, which includes external services and has been deployed on the ActiveBPEL engine. It is not mentioned how the workload has been injected.


6 Discussion and Further Suggestions

This section analyses the overall findings of our evaluation and discusses which parts of BPEL performance testing should be strengthened in future work to increase the quality of their results. As Table 1 and Table 5 show, the approaches differ in many aspects, thus, they follow no common schema. Because of this, we focus on patterns per criteria, i.e., compare the results column-wise.

The performance test type of the approaches is mainly load testing, whereas, in addition, Bianculli et al. [4] also applies stress testing. Only Hackmann et al. [13] solely performs baseline tests. This shows, that there is room for further evaluations, by means of baseline and stress tests.

The workload categories differ for all approaches, with four using real workloads, one using kernels, and two using basic and toy each. Furthermore, the number of processes for each workload also varies, with four approaches using only a single process, three using only two, and one using four processes. In contrast, the remaining approach uses twelve processes. The majority (six out of nine) of the approaches use processes that invoke external services, thus the test always includes the performance characteristics of the invoke activity. For the remaining three approaches, two do not use external services, while it is unknown for the last one. Because of the known number of processes and the fact that external services are not always included, there seems to be lots of space for further work. For all workload categories, a larger corpus of processes would help to improve the meaningfulness of the test results. Regarding the basic category, we propose to have a process per feature to be able to compare their performance characteristics. This can reveal the strengths and weaknesses of a specific engine as well as make the engines and the features comparable. For kernel processes, we suggest to cover more pattern catalogs, whereas for real processes, we propose to use distinct real processes from various use cases in different domains. As services are a crucial part of BPEL processes, they should not be neglected in further studies as well.

Concerning the workload injection, two approaches did not mention it at all. However, for many approaches, the injection strategy is not explicitly stated, but can be deducted for some, but not all cases. As the workload injection is a crucial part of a performance evaluation, we strongly advise to explicitly state the selected strategy.

Regarding the metrics, latency and throughput are used by six approaches each, whereas only two measure utilization. In this context, Intel and Cape Clear [15] provide the most complete approach as they measure all three metrics, whereas one and two metrics are measured by five and three approaches, respectively. Therefore, we propose to focus more on the utilization as it has been neglected by many approaches, but can reveal interesting characteristics of the engines as well as ensure that there are no system overloads falsifying any results.

The number of engines under test range from one up to at most three per approach, limiting the meaningfulness of their results for buying decisions. Only three approaches [4, 13, 23] really compare the performance of multiple engines, while the remaining six evaluate the performance of a single engine. Furthermore, there exists only one comparison between the performance of open source and proprietary engines. However, Bianculli et al. [4] benchmark ActiveVOS
v5.0.2 which incorporates the open source ActiveBPEL v5.0.2 engine. Therefore, this is not a comparison of open source with proprietary engines as basically only open source ones are compared, leaving room for further work in this area.

Regarding the configuration opportunities of the BPEL engines, only two approaches \cite{1,18} tested their engines in different configurations, namely either execute their processes in-memory or not. As engines have more than this single configuration option, it shows that this has been neglected in research, despite its importance. The two approaches that test this option solely evaluate a single engine. Therefore, when comparing more than one engine, it has to be ensured that all engines support this capability.

With only Bianculli et al. \cite{4} allowing to automatically setup the testbed and trigger the test execution, it is very hard and tedious to redo the experiment for all other approaches. Moreover, only \cite{4,30,23,18} even published their detailed test setup, processes and tools, which are essential for the repeatability of these tests.

None of the nine approaches allow analyzing the influence of environmental aspects, for instance the system’s hardware, the database or influences of long-running transactions on the engines’ performance. However, modern multi-core systems and solid-state drives provide new challenges and opportunities for differentiation among different middleware products.

One additional problem is that none of the approaches take into account that BPEL engines greatly vary in their degree of support of the BPEL specification \cite{14,15}, i.e., they implement different subsets of the BPEL features. We propose to take these results into account, creating and selecting appropriate workloads for the engines to be compared.
7 Conclusion and Future Work

In our work, we created a comparison framework with which existing performance benchmarking approaches of process engines, and BPEL engines in particular, can be classified. We applied our comparison framework to nine approaches, revealing their differences and similarities. Based on the findings, we derived guidance for further research in the areas which have been neglected so far.

In future work, we want to apply our comparison framework and method onto other process languages. For instance, the nowadays very popular Business Process Modeling and Notation (BPMN) \cite{BPMN} 2.0, which includes execution semantics \cite[pp. 425-444]{BPMN}, thus, is similar to BPEL in this regard. However, it is also interesting how to take human based tasks for performance approaches into account.
References


REFERENCES


8 List of previous University of Bamberg reports

<table>
<thead>
<tr>
<th>Nr.</th>
<th>(Year)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nr. 23 (1994)</td>
<td>Ferstl O.K., Hagemann U.: Simulation hierarchischer objekt- und transaktionsorientierter Modelle</td>
<td></td>
</tr>
<tr>
<td>Nr. 31 (1995)</td>
<td>Ferstl O.K., Mannmeusel T.: Gestaltung industrieller Geschäftsprozesse</td>
<td></td>
</tr>
<tr>
<td>Nr. 34 (1995)</td>
<td>Sinz E.J.: Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme - Entwicklung, aktueller Stand und Trends -</td>
<td></td>
</tr>
</tbody>
</table>


Applications. (DEXA 98), Vienna, Austria, Aug. 98, pp. 240-249, Springer, Lecture Notes in Computer Sciences, No. 1460


Änderung des Titels der Schriftenreihe Bamberger Beiträge zur Wirtschaftsinformatik in Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik ab Nr. 61

Note: The title of our technical report series has been changed from Bamberger Beiträge zur Wirtschaftsinformatik to Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik starting with TR No. 61
Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik


Nr. 65 (2006) Schönberger, Andreas: Modelling and Validating Business Collaborations: A Case Study on RosettaNet


Nr. 73 (2007) Martin Schissler: out of print


