Universität Stuttgart

Workload Mix Definition for Benchmarking BPMN 2.0 Workflow Management Systems

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Marigianna Skouradaki

aus Heraklion, Griechenland

Hauptberichter: Prof. Dr. Dr. h.c. Frank Leymann

Mitberichter: Prof. Dr. Dimitris Plexousakis

Tag der mündlichen Prüfung: 12.12.2017

Institut für Architektur von Anwendungssystemen der Universität Stuttgart

2017

In Memory Of
Prof. Christos Nikolaou
(1954 – 2015)

Contents

ZUSAMMENFASSUNG

Geschäftsprozessmanagementsysteme (GPM) werden heute in vielen Unternehmen aller Größenordnungen für den Entwurf, die Bereitstellung, die Ausführung, das Monitoring sowie die Analyse von automatisierten Geschäftsprozessen eingesetzt. Über die Jahre hinweg haben sich GPM zu Plattformen für komplexe dienst-orientierte Anwendungen entwickelt. Im Hinblick darauf werden hohe Anforderungen an die Leistungsfähigkeit, wie zum Beispiel Zuverlässigkeit oder Skalierbarkeit, gestellt, die es zu erfüllen gilt. Mit der ständig wachsenden Anzahl der auf dem Markt verfügbaren GPM stehen die Unternehmen vor der Herausforderung ein Produkt auszuwählen, das für ihre Anforderungen und Geschäftsmodelle am besten geeignet ist.

Benchmarking ist eine etablierte Praktik für den Vergleich alternativer Produkte und unterstützt zudem die kontinuierliche Weiterentwicklung von Technologien durch die Fähigkeit zur Definition von klaren Zielen und Zielwerten für das Messen und Bewerten von Leistungsdaten. Obwohl sich im Bereich dienst-orientierter GPM Geschäftsprozessmodellierungssprachen wie Web Services Business Process Execution Language (WS-BPEL) oder Business Process Model and Notation 2.0 (BPMN 2.0) als Standards etabliert haben, gibt es keinen mehrheitlich akzeptierten Standard Benchmark für dienst-orientierte GPM. Eine mögliche Erklärung für dieses Defizit ist die inhärente Komplexität der Architektur von GPM Systemen sowie die hohe Anzahl von Parametern die deren Leistung beeinflussen können. Nichtsdestotrotz wird

die Notwendigkeit eines Standard Benchmarks für GPM in der Literatur vielfach bestätigt.

Das Ziel des BenchFlow Ansatzes ist die Entwicklung eines ersten Standard Benchmarks für die Bewertung und den Vergleich der Leistung von BPMN 2.0 GPM. Zu diesem Zweck adressiert der Ansatz unterschiedliche Herausforderungen, von logistischen Herausforderungen bei der Identifikation repräsentativer Anwendungsfälle, bis hin zu technischen Herausforderungen welche die spezifischen Charakteristiken einzelner GPM Systeme betreffen.

Die vorliegende Arbeit befasst sich mit einer Untermenge der vom BenchFlow Ansatz adressierten Herausforderungen. Der Schwerpunkt liegt dabei auf der Definition einer repräsentativen Menge von Geschäftsprozessen und den entsprechenden Daten, welche dem Benchmark als Eingabe übergeben werden. Diese Menge von repräsentativen Geschäftsprozessen und den dazugehörigen Daten wird auch als Workload Mix des Benchmarks bezeichnet. Im ersten Schritt wird in dieser Arbeit zunächst der theoretische Hintergrund für die Definition eines repräsentativen Workload Mixes vorbereitet. Dies wird erreicht durch *die Identifizierung der Grundkomponenten eines Workload Modells für GPM Benchmarks* sowie durch *die Untersuchung des Einflusses einzelner BPMN 2.0 Sprachstrukturen auf die Leistung eines GPM mit der Einführung des ersten BPMN 2.0 Micro-Benchmarks*. Für die Bestimmung eines repräsentativen Workload Mix werden im nächsten Schritt reale Geschäftsprozesse gesammelt. Diese Menge wird dann hinsichtlich ihrer statistischen Eigenschaften untersucht sowie mit einem *Algorithmus zur Identifikation und Extraktion der häufigsten Strukturmuster analysiert*. Die gefundenen Strukturmuster werden dann zur Generierung von synthetischen Geschäftsprozessen benutzt, welche die Kerncharakteristiken der ursprünglichen Menge widerspiegeln. Die vorgestellten Methoden werden in einer *Tool-Chain zur Unterstützung der Generierung eines Workload Mixes* zusammen gebracht. Abschließend werden sie an einem konkreten Fallbeispiel angewendet, in welchem aus einer Menge von Tausenden realen Prozessen ein repräsentativer Workload Mix für die Verwendung in einem Benchmark erzeugt wird. Es wird gezeigt, dass der erzeugte Workload Mix erfolgreich angewendet werden kann, um das getestete GMP System zu belasten.

Abstract

Nowadays, enterprises broadly use Workflow Management Systems (WfMSs) to design, deploy, execute, monitor and analyse their automated business processes. Through the years, WfMSs evolved into platforms that deliver complex service oriented applications. In this regard, they need to satisfy enterprise-grade performance requirements, such as dependability and scalability. With the ever-growing number of WfMSs that are currently available in the market, companies are called to choose which product is optimal for their requirements and business models.

Benchmarking is an established practice used to compare alternative products and leverages the continuous improvement of technology by setting a clear target in measuring and assessing performance. In particular, for service oriented WfMSs there is not yet a widely accepted standard benchmark available, even if workflow modelling languages such as Web Services Business Process Execution Language (WS-BPEL) and Business Process Model and Notation 2.0 (BPMN 2.0) have been adopted as the de-facto standards. A possible explanation on this deficiency can be given by the inherent architectural complexity of WfMSs and the very large number of parameters affecting their performance. However, the need for a standard benchmark for WfMSs is frequently affirmed by the literature.

The goal of the BenchFlow approach is to propose a framework towards

the first standard benchmark for assessing and comparing the performance of BPMN 2.0 WfMSs. To this end, the approach addresses a set of challenges spanning from logistic challenges, that are related to the collection of a representative set of usage scenarios, to technical challenges, that concern the specific characteristics of a WfMS.

This work focuses on a subset of these challenges dealing with the definition of a representative set of process models and corresponding data that will be given as an input to the benchmark. This set of representative process models and corresponding data are referred to as the *workload mix* of the benchmark. More particularly, we first prepare the theoretical background for defining a representative workload mix. This is accomplished through *identification of the basic components of a workload model for WfMS benchmarks*, as well as the *investigation of the impact of the BPMN 2.0 language constructs to the WfMS's performance, by means of introducing the first BPMN 2.0 micro-benchmark*. We proceed by *collecting real-world process models* for the identification of a representative workload mix. Therefore, the collection is analysed with respect to its statistical characteristics and also with a novel algorithm that *detects and extracts the reoccurring structural patterns of the collection*. The extracted reoccurring structures are then used for *generating synthetic process models* that reflect the essence of the original collection. The introduced methods are brought together in a *tool chain that supports the workload mix generation*. As a final step, we applied the proposed methods on a *real-world case study*, that bases on a collection of thousands of real-world process models and generates a representative workload mix to be used in a benchmark. The results show that the generated workload mix is successful in its application for stressing the WfMSs under test.

Chapter 1

Introduction & Problem Statement

"The beginning is the most important part of the work..."

Plato

The importance of benchmarking is succinctly summarised by Huppler [Hup09] by stating that "the computing industry is so vast and changes so rapidly that new benchmarks are constantly required, just to keep up". Performance benchmarking is an established practice that helps to drive the continuous improvement of technology by setting a clear standard in measuring and assessing performance. For example, transaction processing benchmarks were introduced since a long time [BBC+85] and recognised as a key factor towards an enormous performance improvement of database technology [DL08]. Despite the rapid evolution of benchmarks, a comprehensive standard benchmark targeting Service Oriented Architecture

(SOA) middleware is not yet available[1] and only recently there have been some proposals for benchmarks of SOA middleware tools, as for example SOABench [BBD10]. A possible explanation on this deficiency can be given by the inherent architectural complexity of service oriented middleware systems and the very large number of parameters affecting their performance [PFR+15]. For example, the distributed nature of SOA middleware systems hinders the measurements through instrumentation or monitoring tooling [LGZ07] and introduces cross-cutting challenges concerning the impact elimination of external, interacting components [PFR+15]. In this work, we focus on the service oriented variety of Workflow Management Systems (WfMSs), a type of middleware that enables the business process automation and service composition. Similarly to the broad category of SOA middleware, there is not yet an accepted benchmark for service oriented WfMSs, even if standard business process modelling languages such as Web Services Business Process Execution Language (WS-BPEL) WS-BPEL [Org07] and Business Process Model and Notation 2.0 (BPMN 2.0) (BPMN 2.0) [ISO13] are widely used in academia and industrial practice. To this effect, in previous work [PFR+15] we identified a set of logistic and technical challenges that need to be addressed when benchmarking a service oriented WfMS. More specifically, the main identified challenges are: (i) collecting real world process models, (ii) synthesising representative process models to be used as input in the performance tests (i. e., benchmark workload mix), (iii) defining general or domain-specific workload mix, (iv) investigating the performance impact of workflow language features, (v) designing the benchmark environment, (vi) assessing and selecting the WfMSs to be tested, (vii) defining expressive Key Performance Indicators (KPIs).

The rich expressiveness of the BPMN 2.0 [ISO13] language led to its wide acceptance from the business and Information Technology (IT) as the de-facto standard for the modelling and executing business processes. Thus, the "BenchFlow: A Benchmark for Workflow Management Systems" (BenchFlow)

[1] e. g., SPEC - SOA Benchmark (Subcommittee has been dissolved.)`http://www.spec.org/soa/`

approach[1] aims at tackling the aforementioned challenges and providing a framework towards the first standard benchmark WfMSs that are compliant to BPMN 2.0.

The construction of a robust benchmark lies heavily on the definition of a representative workload model [Fei15]. In other words, the artefacts issued to the System Under Test (SUT) during the performance tests should stand for different sets of characteristics. Only then they can reflect the interests of the users that exploit the benchmark results [vKAH+15]. This work is conducted within the scope of the BenchFlow approach and its principal contribution is to introduce the Workload Mix Generation for Workflow Management Systems (WINE4WfMSs) method that tackles challenges *i, ii, iii, iv,* that related to the definition of the workload model.

The remainder of this section is structured as follows: Section 1.1 introduces some terminology in the fields of Business Process Management (BPM) and benchmarking; Section 1.2 describes the recognised challenges, defines the research questions and maps them to the distinct contributions of this work; Section 1.3 presents the peer-reviewed publications that resulted as an outcome of this work; and Section 1.4 presents the structure of this thesis.

1.1. Terminology and Conventions

This chapter aims at introducing the basic terminology in the fields of Business Process Management (BPM) (cf. Section 1.1.1) and benchmarking (cf. Section 1.1.2).

1.1.1. Business Process Management

The Workflow Management Coalition Specification [Spe99] defines as a workflow the "computerized facilitation or automation of a business process, in whole or part". In other words, a workflow is the automation of a series of business activities that are needed for achieving a goal. Information systems

[1]BenchFlow,
URL: `http://www.iaas.uni-stuttgart.de/forschung/projects/benchflow.php`

that support the definition of workflows and the provision of fast re-design and re-implementation as the business needs change [GHS95] are called Business Process Management Systems (BPMSs) [Kar95]. WfMSs appeared during the '90s as the first generation of BPMSs that were workflow-based technologies with the ability to assign tasks to the right employees at the right time using the right information resources [Kar95]. In current practice, the term BPMS is used to describe an information system that offers a wider variety of management services, while the term WfMS can be considered as more targeted on the re-engineering and automation of business processes. In this work we use the term Workflow Management System (WfMS).

The definition of a workflow with a process modelling and execution language [MTJ+10] is usually referred to in the literature as *workflow model* or *(business) process model*. The definition of a process model is realised through the utilisation of the WfMS's build-time components that are responsible for providing the functions to define the user-specific constructs with respect to a *workflow language metamodel* [LR00; Hol95]. For many years there was no vendor-neutral definition language and each company chose to implement their own form of definition languages on their WfMSs. WS-BPEL [Org07] became an industry-accepted standard for executing processes over Web Services Description Language (WSDL)-based web services. Later, it was followed by the creation and industrial acceptance of the BPMN 2.0 standard [ISO13]. As our data shows a tendency for a broader adoption of the BPMN 2.0 language [SRL+15], this work focuses on benchmarking WfMSs that are compliant with the BPMN 2.0 language.

Another fundamental part of a typical modern WfMS architecture are the runtime components, which are responsible for the process execution. More specifically, they drive the interaction of the WfMS with external systems (e. g., such as end-user clients, web services and other software applications), which are orchestrated within a process model [LR00]. The main runtime components of a WfMS are: a workflow engine, databases, applications and IT tools [Hol95]. As the runtime components are responsible for the process execution, they become the point of interest when investigating WfMS performance. The workflow engine (sometimes also referred to as the workflow enactment services) implements both the runtime control functions and the runtime interactions, in order to handle the execution of the process models [Hol95]. In a SOA environment the workflow engine consists of the following subcomponents: (i) the *process navigator* that is the core of the workflow engine, and is responsible for navigating through the control flow of the process models; (ii) the *request interceptor* acts like a gateway between the clients requesting the execution of a web service and the navigator; (iii) the *service invoker* manages the the invocation of web services that arrive from the navigator; (iv) the *inter-component communication layer* is used by the navigator, the request interceptor and the service invoker for communication; (v) the *transaction manager* is responsible for the complete processing that should happen under transactional control; and (vi) the *persistence manager* that ensures the persistence and recovery of the process execution state [Rol13].

In this work, we discuss the performance of WfMSs with the focus shifted on the performance behaviour of the process navigator. Since BenchFlow is a first effort to develop a comprehensive benchmark for BPMN 2.0 WfMSs, we consider the analysis of the navigator's performance behaviour of major importance before proceeding to the definition of more complex performance measurements.

1.1.2. Benchmarking

Benchmarks are special types of performance tests that are used to compare the performance of diverse software of hardware systems [SEH03]. As very often benchmarks lead to significant improvements on the areas that they are applied to [DL08], they are considered as a vital tool in experimental computer science and research. Benchmarks are usually classified into two categories: micro- and macro-benchmarks [Wal14]. *Micro-benchmarks* (also known as synthetic, narrow spectrum, kernel or simple benchmarks) aim to evaluate the performance of a very specific, fundamental part of a software system. Contrariwise *macro-benchmarks* (also known as natural, application or complex benchmarks) constitute larger, more complex environments that target to evaluate a wide set of performance influencing factors of a software system [Wal14]. To do so, macro-benchmarks usually apply a realistic task-sample for evaluating the systems under test [Wal14]. The performance of a system is also dependent to the *workload model*, which is the set of requests performed in a fixed time of period. Consequently, the definition of reliable performance tests relies heavily on the definition of the workload model they will use [Fei15]. In the case of WfMSs we define as *workload mix* the part of the workload model that describes the process models to instantiate during the performance tests and the intensities with which they should be instantiated. As a first approximation, in the scope of this work, we consider a workload mix as *representative*, when it reflects the essence of the structural characteristics of a process models collection.

1.2. Problem Domain and Contributions

Benchmarking BPMN 2.0 WfMSs is complex as many different performance affecting factors need to be taken into consideration [PFR+15]. Defining a representative workload model is an inseparable part of the benchmarking process. In previous work [PFR+15] we identified a set of logistic and technical challenges that emerge when building a benchmark for BPMN 2.0 WfMSs. Initially, we focus on the challenges (prefixed by CH-) that are related to the construction of a representative and reliable workload model. Afterwards, we discuss the resulting research questions and how they are answered by the contributions of this thesis. An overview of the major research questions (prefixed by RQ-) and contributions (prefixed by C-) of this thesis is provided in Table 1.1, while emerging secondary research questions and contributions are presented in corresponding chapters, throughout this work.

Collecting real-world scenarios (CH-1): In order to come up with a benchmark that correctly reflects the usage of a WfMS in real world practice, we need to collect as many process models representing real world scenarios as possible. In this way, we can have a real representation of the applications that are built on workflow technology. Because “process equals product" [Ley01] most companies and business organisations are not willing to share their process models to protect their intellectual property and competitive advantage. Hence, collecting real world scenarios can be a very challenging task. This brings us to the following research question:

RQ-1: How to overcome obstacles in creating a collection of real-world practice process models?

Towards addressing intellectual property and competitive advantage concerns, we propose confidentiality agreements and a method to anonymise process models, while maintaining their executional semantics. This resulted in the following contribution:

C-1: An anonymisation method for process models ("BPanon") and the obtained collection of process models.

Through this approach we managed to collect 14,167 process models, expressed in diverse modelling languages. Some of the collected process models were anonymised and some were reference process models (i. e., non-executable).

Capturing and determining WfMSs performance factors (CH-2): A performance model is an abstract representation of the system that relates the workload parameters with the system configuration and captures the main factors that determine the system's performance. The results of the performance tests should be analysed or predicted through the correlation of the workload parameters to the system's configuration. Through this approach we can capture and determine the main factors that affect the system's performance [MAD99]. Consequently, in order to reflect the major performance affecting factors the workload model should be designed with regards to the architectural characteristics of the system under test. The related work in benchmarking or performance testing of WfMSs reveals simplified approaches for the definition of utilised workload models [PFR+15]. Thus, the challenge in this case is to identify the key decision points that should be considered during the definition of a workload model for WfMSs, bringing us to the following research question:

RQ-2: What are the basic components of a workload model for WfMSs?

In this respect, we conduct a literature study on the state-of-art of standard middleware and custom WfMSs benchmarks and present the following contribution:

C-2: A metamodel of the basic workload model components for WfMSs.

Identifying workflow language constructs that affect performance (CH-3): The BPMN 2.0 language provides a large set of constructs, that express iteration, parallelism, exception handling, interactions with external entities and others. For defining meaningful process models as part of the workload, we first investigate the performance impact of individual language constructs on the WfMS's performance. In this way, we will be able to define the appropriate workload mix with respect to our research goals. For example, process models with small size may be proven better candidates for throughput experiments, while process models with more complex structures might be more appropriate for response time tests. This raises the following question:

RQ-3: What is the impact of diverse BPMN 2.0 language constructs on the process navigator's performance?

An answer to RQ-3 can be approached through the definition and execution of experiments that target the fundamental components of a WfMS. The goal of a micro-benchmark is to stress fundamental components of complex systems [Wal14]. Hence, RQ-3 is addressed by the following contribution:

C-3: The first micro-benchmark for BPMN 2.0 WfMSs.

A similar approach was followed in the field of databases, when in 1989 Transaction Processing Performance Council (TPC) introduced the TPC A benchmark which consists of a single, simple, update-intensive transaction to the load system under test. The single transaction type provided a simple, repeatable unit of work, and was designed to exercise the key components of an Online Transaction Processing (OLTP) system. Although TPC A micro-benchmark is obsolete since 1995, it set cornerstone knowledge for the more complex, and widely accepted TPC C benchmark [Tra92a; Fei15].

Automating the generation of a realistic workload mix for different use case scenarios (CH-4): In order to keep our measurements accurate we need to create realistic workload scenarios [Fei15]. As workflow-based applications are present in various types of application domains [LR97], it is challenging to select a sufficiently large subset of domains and synthesise a domain-independent workload. This brings us to the following research question:

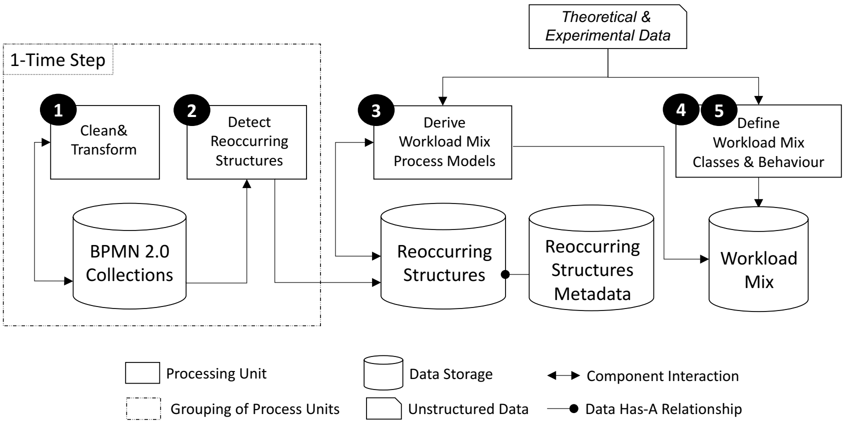
RQ-4: How to derive a representative and meaningful workload mix for both general and domain specific benchmarks?

In order to address this research question we propose the five-phase Workload Mix Generation for Workflow Management Systems (WINE4WfMSs) method for automating the creation of the workload mix with respect to user-defined criteria. The WINE4WfMSs method forms the following contribution:

C-4: The Workload Mix Generation for Workflow Management Systems (WINE4WfMSs) method.

An overview of the WINE4WfMSs method is shown in Figure 1.1. The WINE4WfMSs method starts with a collection of real world BPMN 2.0 process models out of which we derive a representative workload mix. As the process models of the original collection might not follow the BPMN 2.0 standard and/or might be incomplete or invalid, in the first phase (*phase 1*) of the WINE4WfMSs method we clean the original collection. Moreover, in *phase 1* we apply statistical analysis on the collection, in order to extract relevant structural information (i. e., size of process models, number of gateways, number of events, etc.).

The collected process models may not contain textual or behavioural information, as some of the collected process models were provided in an anonymised format, or as reference process models. Thus, in terms of a



Theoretical & Experimental Data
1-Time Step
1
Clean& Transform
2
Detect Reoccurring Structures
3
Derive Workload Mix Process Models
4
5
Define Workload Mix Classes & Behaviour
BPMN 2.0 Collections
Reoccurring Structures
Reoccurring Structures Metadata
Workload Mix
Processing Unit
Grouping of Process Units
Data Storage
Unstructured Data
Component Interaction
Data Has-A Relationship

Figure 1.1.: Overview of the WINE4WfMSs method

similarity analysis we can only detect and extract structural similarities that exist in the original collection (*phase 2*). The structural pattern to search is not known beforehand, thus this challenge can be mapped to the problem of *pattern discovery without candidate generation*, a variance of the subgraph isomorphism [YH02]. This leads to the following research question:

RQ-5: How to detect reoccurring BPMN 2.0 structures in a process models collection?

In order to tackle RQ-5 we introduce the Reoccurring Structures Detection (RoSE) method that detects and extracts reoccurring structures (i. e., structural similarities) in a collection of BPMN 2.0 process models and calculates metadata regarding the frequency of occurrence. With this regards we present the following contribution:

C-5: The Reoccurring Structures Detection (RoSE) method.

The reoccurring structures detected by the RoSE method and corresponding metadata are stored in “Reoccurring Structures” and “Reoccurring Structures Metadata” as shown in Figure 1.1. Having this information stored, we may now proceed to define a method for generating representative process models out of the discovered reoccurring structural patterns (*phase 3*).

RQ-6: How to synthesise a representative BPMN 2.0 process model?

Through the utilisation of the statistical analysis (*phase 1*), theoretical and experimental data derived from C-1 and C-2 and the detected reoccurring structural patterns, we define a set of criteria that are given as input in a method that generates the representative process models.

C-6: Representative BPMN 2.0 process model generation method.

At this point, the generated process models can be used as part of the workload mix. For completing the definition of the workload mix in *phase 4* we define the workload mix classes, i. e., the intensity with which the instances of a process model are instantiated when executing the workload mix. In order to represent real world conditions, for defining the workload mix classes we need to identify the percentage to which a generated process model reflects the original collection. Finally, in *phase 5* we define the behaviour of the workload mix’s process models, e. g., the probability with which the control flow will follow specific execution paths in the process model.

CH-1: Collect real-world scenarios.	RQ-1: Overcome obstacles in real-world process models collection?	C-1: “BPanon” method & Process models collection.
CH-2: Capture WfMS performance factors.	RQ-2: WfMS workload model components?	C-2: Metamodel of WfMS workload model components.
CH-3: Identify language constructs that affect WfMS performance.	RQ-3: BPMN 2.0 language impact on performance?	C-3: Micro-benchmark
CH-4: Automate workload mix generation to satisfy diverse scenarios.	RQ-4: General vs. domain specific workload mix?	C-4: WINE4WfMSs method
	RQ-5: Reoccurring structures in a process model collection?	C-5: RoSE method
	RQ-6: Synthesis of representative process models?	C-6: BPMN 2.0 process model generation method

Table 1.1.: Research questions and contributions overview

1.3. Publications

The contributions answering the research questions introduced in Section 1.2 resulted in peer-reviewed publications, which are presented in Table 1.2 as contribution – publication correspondence and in the following list in inverted chronological order:

1. V. Ferme, M. Skouradaki, C. Pautasso, F. Leymann, and A. Ivanchikj. “Performance Comparison Between BPMN 2.0 Workflow Management Systems Versions.” In: *International Workshop on Business Process Modeling, Development and Support*. BPMDS ’17. Springer, 2017
2. M. Skouradaki, V. Andrikopoulos, O. Kopp, and F. Leymann. “RoSE: Reoccurring Structures Detection in BPMN 2.0 Process Model Collections.” In: *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*. Springer Nature, 2016, pp. 263–281
3. M. Skouradaki, T. Azad, U. Breitenbücher, O. Kopp, and F. Leymann. “A Decision Support System for the Performance Benchmarking of

Workflow Management Systems." In: *Advanced Summer School of Service Oriented Computing*. SummerSOC '16. IBM Research Division, 2016, pp. 41–58

4. M. Skouradaki, V. Andrikopoulos, and F. Leymann. "Representative BPMN 2.0 Process Model Generation from Recurring Structures." In: *International Conference on Web Services*. ICWS '16. Institute of Electrical & Electronics Engineers (IEEE), June 2016, pp. 468–475
5. V. Ferme, A. Ivanchikj, C. Pautasso, M. Skouradaki, and F. Leymann. "A Container-centric Methodology for Benchmarking Workflow Management Systems." In: *International Conference on Cloud Computing and Services Science*. CLOSER '16. Springer, 2016
6. M. Skouradaki, V. Ferme, C. Pautasso, F. Leymann, and A. van Hoorn. "Micro-Benchmarking BPMN 2.0 Workflow Management Systems with Workflow Patterns." In: *Advanced Information Systems Engineering*. Ed. by S. Nurcan, P. Soffer, M. Bajec, and J. Eder. CAISE '16. Springer International Publishing, 2016, pp. 67–82
7. M. Skouradaki and F. Leymann. "Detecting Frequently Recurring Structures in BPMN 2.0 Process Models." In: *Advanced Summer School of Service Oriented Computing*. SummerSOC '15. IBM Research Division, 2015
8. M. Skouradaki, K. Görlach, M. Hahn, and F. Leymann. "Application of Sub-Graph Isomorphism to Extract Reoccurring Structures from BPMN 2.0 Process Models." In: *International Symposium on Service-Oriented System Engineering*. SOSE '15. Institute of Electrical & Electronics Engineers (IEEE), Apr. 2015, pp. 11–20
9. M. Skouradaki, V. Ferme, F. Leymann, C. Pautasso, and D. H. Roller. "BPELanon: Protect Business Processes on the Cloud." In: *International Conference on Cloud Computing and Service Science*. CLOSER '15. Lisbon, Portugal: SciTePress, May 2015, pp. 241–250
10. M. Skouradaki, D. H. Roller, F. Leymann, V. Ferme, and C. Pautasso. "On the Road to Benchmarking BPMN 2.0 Workflow Engines." In:

ACM/SPEC International Conference on Performance Engineering. ICPE '15. Austin, Texas: Association for Computing Machinery ACM, 2015, pp. 301–304

11. M. Skouradaki, D. Roller, C. Pautasso, and F. Leymann. "BPELanon: Anonymizing BPEL Processes." In: *Central European Workshop on Services and their Composition*. ZEUS '14. 2014, pp. 9–15

In order to support the openness of data, the results derived under the scope of this work are provided to the Benchmark for Conformance and Performance of Workflow Engines (PEaCE) interactive dashboard [BMHW16]. PEaCE is a collaborative effort among the Distributed Systems Group at University of Bamberg, the Institute of Architecture of Application Systems at University of Stuttgart, the Faculty of Informatics at Università della Svizzera Italiana, and the Software Engineering Research Group of the University of Karlstad to present data derived by performance and conformance benchmarking on WS-BPEL and BPMN 2.0 WfMSs. Furthermore, it is a common research effort that has resulted in organisation of the "1st International Workshop on Performance and Conformance of Workflow Engines" held in conjunction with the 5th European Conference on Service-Oriented and Cloud Computing (ESOCC '16).

C-1: "BPanon" method & Process models collection.
M. Skouradaki et al. "Application of Sub-Graph Isomorphism to Extract Reoccurring Structures from BPMN 2.0 Process Models." In: *International Symposium on Service-Oriented System Engineering*. SOSE '15. Institute of Electrical & Electronics Engineers (IEEE), Apr. 2015, pp. 11–20
M. Skouradaki et al. "BPELanon: Protect Business Processes on the Cloud." In: *International Conference on Cloud Computing and Service Science*. CLOSER '15. Lisbon, Portugal: SciTePress, May 2015, pp. 241–250
M. Skouradaki et al. "On the Road to Benchmarking BPMN 2.0 Workflow Engines." In: *ACM/SPEC International Conference on Performance Engineering*. ICPE '15. Austin, Texas: Association for Computing Machinery ACM, 2015, pp. 301–304
C-2: A metamodel of basic workload model components for WfMS.
M. Skouradaki et al. "A Decision Support System for the Performance Benchmarking of Workflow Management Systems." In: *Advanced Summer School of Service Oriented Computing*. SummerSOC '16. IBM Research Division, 2016, pp. 41–58
V. Ferme et al. "A Container-centric Methodology for Benchmarking Workflow Management Systems." In: *International Conference on Cloud Computing and Services Science*. CLOSER '16. Springer, 2016
C-3: Micro-benchmark
M. Skouradaki et al. "Micro-Benchmarking BPMN 2.0 Workflow Management Systems with Workflow Patterns." In: *Advanced Information Systems Engineering*. Ed. by S. Nurcan et al. CAISE '16. Springer International Publishing, 2016, pp. 67–82
C-4: WINE4WfMSs method
M. Skouradaki et al. "On the Road to Benchmarking BPMN 2.0 Workflow Engines." In: *ACM/SPEC International Conference on Performance Engineering*. ICPE '15. Austin, Texas: Association for Computing Machinery ACM, 2015, pp. 301–304
V. Ferme et al. "Performance Comparison Between BPMN 2.0 Workflow Management Systems Versions." In: *International Workshop on Business Process Modeling, Development and Support*. BPMDS '17. Springer, 2017
C-5: RoSE method
M. Skouradaki et al. "RoSE: Reoccurring Structures Detection in BPMN 2.0 Process Model Collections." In: *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*. Springer Nature, 2016, pp. 263–281
M. Skouradaki and F. Leymann. "Detecting Frequently Recurring Structures in BPMN 2.0 Process Models." In: *Advanced Summer School of Service Oriented Computing*. SummerSOC '15. IBM Research Division, 2015
M. Skouradaki et al. "Application of Sub-Graph Isomorphism to Extract Reoccurring Structures from BPMN 2.0 Process Models." In: *International Symposium on Service-Oriented System Engineering*. SOSE '15. Institute of Electrical & Electronics Engineers (IEEE), Apr. 2015, pp. 11–20
C-6: BPMN 2.0 process model generation method
M. Skouradaki, V. Andrikopoulos, and F. Leymann. "Representative BPMN 2.0 Process Model Generation from Recurring Structures." In: *International Conference on Web Services*. ICWS '16. Institute of Electrical & Electronics Engineers (IEEE), June 2016, pp. 468–475
V. Ferme et al. "Performance Comparison Between BPMN 2.0 Workflow Management Systems Versions." In: *International Workshop on Business Process Modeling, Development and Support*. BPMDS '17. Springer, 2017

Table 1.2.: Contribution - publication correspondence

1.4. Thesis Structure

The remainder of this work is structured as follows:
Chapter 2: Background & Related Work
This chapter covers fundamental concepts and related work from two major research fields: performance benchmarking and BPM. In the area of benchmarking we overview existing standard benchmarks on diverse middleware applications, state-of-art custom benchmarks on WfMSs and methodologies and approaches on workload modelling. In the area of BPM we review existing work in the diverse research areas that intersect with the contributions of this thesis. More particularly, we focus on the areas of process model anonymisation, process models collections and repositories, methods of process models decomposition, structural similarities, and generation.

Chapter 3: Collecting Real-World Process Models
In this chapter we address the challenge of collecting real world practice process models. That corresponds to the first research question RQ-1: "How to overcome obstacles in creating a collection of real-world practice process models?". We propose and describe a method that anonymises the process models while maintaining their executional semantics. This method is afterwards used to foster the sharing of the process models. We then describe the composition of the obtained process models collection. The anonymisation method as well as the obtained collection of real-world practice process models cover contribution C-1: "An anonymisation method for process models("BPanon") and the obtained collection of process models.".

Chapter 4: Defining Meaningful Workload
This chapter targets to address the second identified research question RQ-2: "What are the basic components of a workload model for WfMSs?". For this we first identify the key components of a WfMS benchmark and proceed by identifying the basic components of the workload model (C-2: "A conceptual metamodel of the basic workload model components for WfMSs.").

Chapter 5: Micro-benchmarking BPMN 2.0 Workflow Management Systems
This chapter addresses research question RQ-3 “What is the impact of the diverse BPMN 2.0 language constructs on the process navigator’s performance?”. To answer this question we define and execute the first micro-benchmark of BPMN 2.0 WfMSs (C-3: “The first micro-benchmark for BPMN 2.0 WfMSs”) and conduct a thorough analysis of the results.

Chapter 6: Reoccurring Structures Detection (RoSE)
This chapter addresses research question RQ-5: “How to detect reoccurring BPMN 2.0 structures in a process models collection?”. More particularly, we define a novel algorithm to detect and extract the reoccurring structures in a collection of process models without the usage of textual semantics. The structural similarities of the collection are identified without knowing a subgraph to search before hand. This challenge belongs to the problem category of *pattern discovery without candidate generation* that is a subclass of the subgraph isomorphism problem. The proposed method forms contribution C-5: “The RoSE method”.

Chapter 7: Generating BPMN 2.0 Process Models
This chapter addresses research question RQ-6: “How to synthesise a representative BPMN 2.0 process model?”. To address this question we introduce a (semi-) automated method that follows user defined constraints and generates a process model out of discovered reoccurring structural patterns (as discussed in the previous chapter). The proposed method is mapped to contribution C-6: “Representative BPMN 2.0 process model generation method”.

Chapter 8: A Process and Toolchain for Workload Mix Generation
This chapter presents the five phase WINE4WfMSs method (C-4: “The WINE4WfMSs method”) and the architecture of a toolchain that supports it method. The architecture is designed with respect to well defined design patterns on various architectural styles (e. g., cloud computing, distributed

applications, etc.). To this effect, we discuss the applied architectural styles and selected design patterns, as well as architectural facts and implementation details for each component separately.

Chapter 9: Deriving a Workload Mix - A Real World Case Study
This chapter brings together the methods and contributions addressed in previous chapters with the goal to answer research question RQ-4: “How to derive a representative and meaningful workload mix for both general and domain specific benchmarks?”. We apply stepwise all presented methods to a real world case study to derive a representative workload mix with regards to a given real world practice process models collection. We then provide this workload mix to execute a macro-benchmark for BPMN 2.0 WfMSs and overview the results.

Chapter 10: Conclusions and Outlook
This chapter summarises this work and discusses answers to research questions, as well as its identified limitations. It also provides an outlook on research opportunities that can be based on this work.

Chapter 2

Background & Related Work

"Those who cannot remember the past are condemned to repeat it."

G. Santayana

This chapter focuses on two major research areas: performance benchmarking (Section 2.1) and Business Process Management (BPM) (Section 2.2). Viera et al. [VMSS12] define as benchmark a tool that contributes towards the evaluation and comparison of competitive systems or components. This is done with respect to well defined objectives, such as for example performance, dependability or security. In addition to completing its functional goal, a benchmark should satisfy the following requirements [vKAH+15]:

Relevance: The benchmark's behaviour and the behaviour that is important to the consumers of the benchmark's results should be closely correlated.

Reproducibility: The benchmark should produce similar and comparable results every time it is executed with the same configuration.

Fairness: The benchmark should allow the different configurations to compete to their full potential without introducing overhead or limitations.

Verifiability: The benchmark results should provide confidence of its accuracy.

Usability: The benchmark should be easily deployable and executable to the users' test environments.

A benchmark can be constructed and published under a standard or custom scope. Typically, *standard benchmarks* are published by established consortia after being verified by academic and industrial partners. In principle, standard benchmarks satisfy the aforementioned requirements, and thus they are considered more trustworthy by practitioners. The two most prominent consortia for performance benchmarking are the Standard Performance Evaluation Corporation (SPEC)[1] and TPC[2]. However, standard performance benchmarks are not available for all middleware systems. For example, in the case of WfMSs, a standard benchmark is not yet available [SRL+15]. Consequently, practitioners apply *custom benchmarks* for evaluating their systems. These custom benchmarks might not necessarily satisfy the aforementioned properties, and therefore the emerging results might be unreliable or biased.

Before proceeding with the development of a standard complex benchmark one needs to understand the individual characteristics of the workload. The *workload* of a system can be described as the set of inputs that arrive to a system from its environment during a specific period of time. It is simulated by a *workload model* that is a representation of the real workload, used for testing purposes [EM06]. Hence, one needs to carefully design the workload model, as it has a decisive role on the emerging results [Fei15]. One important part of the workload model are the *basic workload components*, which are an abstract type of requests that arrive to the system from the

[1]SPEC, URL: `https://www.spec.org`
[2]TPC, URL: `https://www.tpc.org`

environment [EM06]. For example, a transaction or a Hypertext Transfer Protocol (HTTP) request can be seen as a basic workload component. In Section 1.1 we defined as *workload mix* the basic workload component that describes the process models to instantiate during the performance tests and the intensities with which they should be instantiated (see Section 4.2). A way to thoroughly comprehend the impact of the individual workload components to the system's performance is to execute a *micro-benchmark* which targets the specific performance evaluation of atomic operations [WH13]. Due to the lack of a standard benchmark for WfMSs and for the purposes of constructing a robust and reliable benchmark we need to consider all these aspects during the *workload modelling* phase.

For purposes of modelling the workload mix for benchmarking WfMSs we have to look across diverse areas of BPM with the intention to derive the necessary information and construct structurally representative process models to be included in the workload mix. We firstly focus on *business process anonymisation*, a method to obfuscate the critical information of process models, while maintaining their structural and executional semantics (cf. Section 3.1). The method is applied for fostering process model sharing. More particularly, through the application of this method we collect a set of diverse process models which are stored in *process model collections* or in more complex scenarios in *process model repositories*. The collected process models are analysed with regards to diverse research areas for deriving a structurally representative workload mix. *Business process models decomposition* is the process of dividing a complete process model into logically diverse fragments with the goal to redistribute them to different execution and controlling partners [KKL07]. It is accepted to have an important role in the re-usability of process models [STK+10; ELtHF11]. To this effect, we adapt the idea of business process model decomposition to our needs. The objective at this point is to create fragments that originate from detected *process models structural similarities*. The fragments are then re-used for generating representative business process models. The approach of *process model generation* is also a recognised research area with multiple application scenarios and focuses on the generation of process models that satisfy specific declarative constraints [MMB16].

The remainder of this chapter is structured as follows: Section 2.1 provides an overview on related work regarding middleware benchmarking and Section 2.2 focuses on areas of BPM that are related to this work.

2.1. Benchmarks

Section 2.1.1 provides an overview of selected up-to-date standard benchmarks for middleware systems; Section 2.1.2 discusses related work in custom benchmarks for WfMSs performance and Section 2.1.3 reviews existing work in workload modelling.

2.1.1. Standard Middleware Benchmarks

Standard benchmarks for performance, as discussed previously, are mainly published by two vendors: SPEC [Sta95] and TPC [Tra92a]. Both consortia aim at collecting a valuable and straightforward set of metrics for producing reliable tests that are easy to employ and provide insightful and verifiable results to the marketplace. SPEC focuses on diverse systems that play a dominant role in the market, while TPC focuses on defining transaction processing and database benchmarks[1]. The following sections summarise the most relevant, non-obsolete benchmarks published by these vendors.

SPEC ® JMS 2007 [Sta07] provides a benchmark for assessing the performance of Message Oriented Middleware (MOM) servers based on the Java Message Service (JMS). MOM is a technology that finds application in many sectors, such as supply chain management, stock trading, online auctions, etc. Furthermore, MOM uses the point-to-point paradigm that is considered as a core component for state-of-art software architectures and technology, such as SOA [Erl05] and Enterprise Application Integration (EAI) [HW04]. Nevertheless, message-based applications go through issues regarding scalability and performance, while prominent companies usually require a high message throughput. For this reason the underlying

[1]As stated by the vendors themselves

MOM is expected to perform and scale in a steady manner [Sta07]. Thus, the SPEC ® JMS 2007 benchmark aims to provide standard workload and metrics that can be used for an in-depth performance analysis of all the underlying components of JMS-based MOM platforms. This benchmark offers workload scalability by allowing users to increase the number of queues and topics (destinations). The SPEC ® JMS 2007 benchmark is developed with respect to an application scenario which models a supply chain for a supermarket. The supermarket is represented as a distributed company that depends on diverse units with different roles for accomplishing its business goals. For example, points of sales, distribution centres and suppliers are different participants of this scenario. This scenario allows a clear specification of interactions that stress various features of JMS servers, as for example the publish/subscribe vs. peer-to-peer communication or different message types. This application scenario also allows workload with different scalability aspects. For example, one can either increase the number of supermarkets or the number of products offered by the supermarket. The macro-benchmark discussed in Chapter 9 is similar to the macro-benchmark SPEC ® JMS 2007 [SKBB09] produced and maintained by SPEC. The former targets to the evaluation of the performance and scalability of JMS-based messaging middleware. Although this is a macro-benchmark, we consider it similar to our work as we could see messaging middleware an ancestor of WfMS [Obe06]. Moreover, the workload of SPEC ® JMS 2007 maintains the following characteristics: realistic scenarios, exercise all platform features, minimise influence of other components, non-limited scalability and configurability. These characteristics are also set as requirements for our macro-benchmark (cf. Chapter 9).

SPECjbb ® 2015 [Sta15] provides performance measurements for the latest Java application features. It is applicable to all organisations that are interested in measuring JVM-hosting server performance. The SPECjbb ® 2015 also uses as application scenario a supermarket company that relies on IT infrastructure to deal with point-of-sale requests, online purchases and data-mining operations. Also in this application scenario the supermarket depends on diverse units (i. e., headquarters, points of sales, suppliers) for completing various business goals. During the execution of the application scenario a component of the workload model injects requests to the SUT by driving the system's load. The requests can either be synchronous or asynchronous and are executed by another component of the workload model that simulates the different roles of the application scenario. The metrics provided by the benchmark are a pure throughput and a throughput under Service Level Agreement (SLA) metric. The benchmark is configurable, thus it supports the users to stress diverse layers of the SUT (i. e., system stack, operating system and application layers). Since the benchmark is recently updated (2015) it also supports benchmarking systems running on virtualised or cloud environments.

SPEC has also introduced a group that focuses on benchmarking SOA infrastructures [Sta10]. However, publicly available information on their progress is still not available at the time of writing this manuscript.

TPC-C [Tra92b] is a benchmark targeting OLTP systems. Although this is not the first benchmark of TPC that targets OLTP systems ([TPC94a; TPC94b]), it is considered as a more comprehensive and improved version of the previously published benchmarks. A fundamental reason for its dominance among practitioners is that the defined workload model expresses multiple transaction types, executed against a database with a rather complex schema [TPK+13]. Overall, the TPC-C benchmark verifies a SUT as production-ready and with sufficient recovery capabilities, if the database supports the Atomicity, Consistency, Isolation, Durability (ACID) properties.

The application scenario of the TPC-C benchmark models a wholesale parts supplier (company) that operates a number of warehouses and their corresponding sales regions. However, this benchmark is not restricted to a specific business area, but targets different market sectors. The workload model includes five concurrent transactions of different types and complexity, and a database containing nine different types of tables with a wide range of record and population sizes. Four of the tables can *grow and shrink* by insert and delete operations, four of the tables can *scale* with respect to the number of the warehouses, and one table is of fixed size. The defined transactions are based on realistic scenarios such as: entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the corresponding warehouse. The benchmark also covers the aspect of scalability as the company expands and new warehouses are added to the model. The expansion of the company is done in a controlled manner by satisfying certain consistency requirements. The defined metric of the TPC-C benchmark is transactions per minute (tpmC).

TPC-E [TPC15] is the ancestor of TPC-C and is also an OLTP benchmark. The TPC-E benchmark models a brokerage house firm, but can be generalised to current OLTP systems. The main target of the benchmark is a central database that executes transactions associated to the company's customer accounts. It contains a more complex database schema that consists of thirty three tables out of which nine are of fixed size, thirteen are scaling with respect to the customer's size and eight are growing with a greater growth rate than this of TPC-C. The tables are populated with pseudo-realistic looking data and, in contrast to TPC-C, the benchmark starts with an empty table that grows during its execution. Despite the usage of the pseudo-realistic data the TPC-E benchmark has been found to use the same random Input/Output (I/O) access pattern as this of TPC-C [CAA+11]. The workload model of TPC-E introduces a transaction mix of ten workload mix transactions of various types and complexity. Moreover, it contains two additional transactions that are executed at different times additionally to the regular workload mix. To increase complexity of the transactions this benchmark introduces dependencies among each other as well as long-running transactions. The TPC-E benchmark uses the metric of transactions per second (tpsE). The increased complexity of TPC-E has caused a slower adaptation compared to its predecessor (TPC-C) [TPK+13].

2.1.2. Workflow Management Systems Benchmarks

When it comes to evaluating the performance of complex systems, such as modern service oriented middleware [GGKS02], there is a lot of work done that focuses on different architectural layers [HZ06], as for example, the storage layer [Gra92; TZJW08; Cha95] or the middleware layer [BCM+05].

Despite the widely recognised need for introducing a standard benchmark for WfMSs [WLR+09; RvdAH07] that will enable the performance evaluation of different research prototypes and commercial products in meaningful conditions, one is still not available [SRL+15]. To the best of our knowledge, BenchFlow is the first benchmark that explicitly targets evaluating the performance of BPMN 2.0 WfMSs [SRL+15]. The presented work is built under the scope of the BenchFlow approach, and targets to tackle open research challenges introduced by the complex nature of BPMN 2.0 WfMSs [PFR+15]. For the sake of completeness, in the following we are looking at previous efforts at benchmarks for service oriented middleware.

SOABench [BBD10] can be seen as an initial step to provide a performance assessment and comparison framework of service oriented middleware systems. It features the automatic generation and execution of testbeds. The evaluation of the proposed framework is based on testing two open source and one proprietary WfMSs supporting the WS-BPEL language. More specifically, four different simple workloads are defined for experiments. These express basic control flow structures of the WS-BPEL language (i. e., Sequential, Flow without Synchronisation Dependency, Flow with Dependencies, While). Each defined workload is addressed with respect to four different loads that span from low to high system loading. The defined metric is limited to response time for all the executed experiments. Sliver [HHGR06] is a WS-BPEL WfMS for mobile devices. In order to evaluate its performance the authors test the Sliver engine against twelve WS-BPEL control flow patterns. The performance of the Sliver WfMS is measured with respect to three different infrastructures (PC, PDA and phone) and compared to one more WfMS (i. e., ActiveBPEL) on a PC infrastructure. Also in this case, the examined metric is the response time of the WfMS. Both approaches contain simple workload models, thus they may be characterised as micro-benchmarks for WS-BPEL WfMSs.

Informatica [Act11] executed an internal benchmark, in order to evaluate their work and inform the prospective customers. For the performance tests four workload mixes are used, and the load is variable with a request rate of maximum 50 clients. Although the configuration of the infrastructure underlying the WS-BPEL engine is described in detail, results on the performance tests of ActiveVOS are not further discussed. Dit et al. [DES08] define a simple synthetic process as the workload mix for benchmarking WS-BPEL WfMSs. The workload model is based on the simulation of real world traffic conditions in order to better define the end-users that characterise it. For the performance tests a two phase workload mix is defined that implements a WS-BPEL correlation. The SUT is the ActiveBPEL engine[1] and the experiments run for 2 minutes in total during which they simulate 2000 clients. The defined metrics are success/failure rate, response times and round-trip delays.

SWoM [Rol13] and FACTS [LLHX10] conduct load performance tests to stress open source WfMSs. SWoM [Rol13] defines a workload mix of four simple WS-BPEL processes. One of the injected WS-BPEL processes contains also the invocation of external services, which are continuously called by the testing clients. Each experiment executed 24,000 instances and lasted in total approximately 40 minutes. The load emulates 30 clients, whose think time for subsequent requests was adjusted to run with respect to the CPU load keeping it in between of 50% and 60%. Similarly, the FACTS framework focuses on building fault handling logic in WS-BPEL and executes load tests for evaluating the proposed approach. The FACTS framework is evaluated with a workload mix of seven fault-tolerant process models invoked 1,000 times. At the end the response time of the invocations is reported. Strauch et el. [SALM12; SASL13] propose the design and realisation to enable multi-tenancy in Enterprise Service Bus (ESB). This approach is later on extended by Hahn et al. [HSA+14] towards the design and realisation of a multi-tenant service composition engine. All of the aforementioned approaches base their experiments on a workload adapted from the Adroit benchmark [Adr16]. More specifically, the generated workload for the evaluation experiments consists of 1 KB SOAP over HTTP messages which are issued to the SUT with a variable load burst rate. The measured metrics are latency and resource utilisation. All the aforementioned approaches

[1]Active endpoints, Active BPEL Server, URL: `http://www.activevos.com/content/developers/education/sample_active_bpel_admin_api/doc/index.html`

utilise external web service invocation through their workload mixes but do not discuss how to ensure reliability of the measurements [PFR+15]. The reliable performance evaluation of web service interactions constitutes an additional challenge [PFR+15] which is outside the scope of this work and the BenchFlow framework [FIP15].

The performance evaluation of WfMSs also finds application on the area of scientific WfMSs which aim at supporting the scientists towards the use of scientific applications [SK10]. For instance, Gómez Sáez et al. [GAH+15] conduct a performance evaluation between different deployments of a long-running scientific workflow being deployed on different cloud providers. The performance evaluation results are analysed with respect to the performance and cost trade-off. A tool for performance analysis of scientific workflows deployed on distributed and grid execution environments is proposed by Prodan et al. [PF08]. The tool is built on a conceptual model that classifies the different types of overhead introduced by a workflow. The classification is derived empirically and is evaluated against two real world workflow applications. This approach can be seen as complementary to the one we propose in Chapter 5 to discover the impact of BPMN 2.0 language constructs to the BPMN 2.0 WfMS navigator's performance.

The need for a common framework for the fair and reliable evaluation of WfMSs has frequently been discussed in the literature [KKL06; RvdAH07; LMJ10]. In previous work [PFR+15] we conducted an extensive analysis to define the open research challenges that should be tackled by such a framework and later on [SRL+15] we proposed an abstract methodology on how to tackle some of these challenges. As discussed in Chapter 1, this work focuses on addressing the workload mix related challenges while Ferme et al. [FIP15] propose the BenchFlow framework for an automatic and reliable calculation of performance metrics for BPMN 2.0 WfMSs. Ferme et al. [FIP15] provide also a proof-of-concept evaluation of the proposed framework by executing a simple workload mix and conducting performance tests on two open source WfMSs. Ferme et al. [FIP+16] describe a benchmarking methodology using a container-centric architecture to realise the BenchFlow framework and tackle challenges regarding the reproducibility of the experiments and the isolation of the performance measurements. Finally, the proposed methodology and framework are enhanced with a set of expressive performance metrics and are utilised for estimating the cost of executing business processes on diverse cloud providers [FIP16]. Focusing also on the cloud, Rosinosky et. al. [RYC16] propose a framework for benchmarking WfMSs hosted in the cloud. In contrast to the BenchFlow framework, the proposed approach does neither include the generation of representative workload mix nor discusses how to tackle fundamental challenges that are introduced during the design of reliable WfMS benchmarking environments [PFR+15].

The framework BPEL Engine Test System (betsy) was initially proposed for investigating the conformance of WfMSs to the WS-BPEL language standards by Harrer et al. [HLW12; HRW14] and was afterwards extended for the BPMN 2.0 language [GHL+15; GHLW16; GHL16]. Research with the betsy system reveals a plethora of inconsistencies between the business process language standards and emerging WfMSs. These results, as well as previous results derived from performance tests, need to be considered during the definition of representative workload mix for benchmarking WfMSs. For this purpose, the results derived from betsy or BenchFlow performance tests can be collected and presented on an interactive dashboard [BMHW16].

Mendes et al. [MBM09] apply several micro-benchmarks on event processing systems to answer fundamental questions on their performance concerning scalability and bottlenecks. Another micro-benchmark is introduced by Angles et al. [APDL13] based on social networks to define the best candidates for macro-benchmarks, and Waller and Hasselbring [WH13] propose a micro-benchmark for measuring the overhead of application-level monitoring. The proposed micro-benchmark identifies three causes of monitoring overhead, and sets the basis for a reliable macro-benchmark. Regarding WfMSs, the already cited works of Biancully et al. [BBD10], Hackmann et al. [HHGR06] and Roller [Rol13] introduce micro-benchmarks for WS-BPEL WfMSs. Röck et al. [RHW14; RH14] conduct a systematic review on approaches that test the performance of WS-BPEL WfMSs, and stress the need for improvement on WfMSs baseline tests. To the extent of our knowledge the micro-benchmark we proposed in previous work [SFP+16] is the first micro-benchmark for BPMN 2.0 WfMS. This work aims at exploiting the micro-benchmark results to derive a complex, representative, synthetic workload mix that will be provided to the BenchFlow approach for a more extensive performance evaluation.

To sum up, the goal of the BenchFlow approach, and consequently this work, is the creation of a benchmark that differs from the related work in terms of: (i) the number of WfMSs to be compared, (ii) the complexity and diversity of the workload mix, (iii) the number of the executed performance tests, and (iv) the number of performance metrics that will be taken into consideration, and their aggregation into a meaningful performance indicator.

2.1.3. Workload Modelling

Feitelson [Fei15] stresses the significance of defining a representative workload model, as a non-representative one may produce misleading results. Various methods and techniques have been proposed in the literature for deriving the workload model [EM06] and most of them usually follow the same general methodology [KHSB12]. The need to use real data for the definition of the workload is also highlighted [Fei15]. The sophisticated architecture of WfMSs [LR00] requires an initial focus shift on the definition of a representative workload mix. By following the literature guidelines [Fei15], we collect real world practice process models (cf. Chapter 3) and utilise graph-based data mining [PJMR14] (cf. Chapter 6), graph generation [VMZ+10] (cf. Chapter 7), descriptive statistics, measuring popularity and clustering [EM06] for defining a representative workload mix (cf. Chapter 9). In the following we discuss how such techniques have been used for workload modelling purposes in the literature.

Traces of real scientific workflow executions as well as performance statistics are publicly available [Pegb]. Using these data as a basis one can derive synthetic scientific workflows based on statistics and use them for benchmarking purposes [Pega]. Ramakrishnan and Gannon [RG08] published scientific workflow structures along with runtime and data statistics for many real workflow applications. As similar data were not publicly available for BPMN 2.0 real world practice business processes these approaches were not applicable for this work. Nevertheless, the need for synthetic realistic process models for WfMSs benchmarking purposes is affirmed by the literature [JCD+13]. Gupta [Gup14] also recognises the need for a benchmark to allow the user to specify different sizes and other structural parameters of the participating components. However, to the extent of our knowledge, this is the first time that synthetic process models generated from reoccurring structural patterns, are proposed as representative workload mix for benchmarking WfMSs.

In the area of big and linked data the generation of realistic graphs for benchmarking purposes is also well established. The graph-based workload has already been used in benchmarking applications that model data as graphs. An IBM's benchmark [DKSU11] proposes the metric of "structuredeness" of real data. The proposed approach combines the metrics of "structuredeness" and size for generating data models of the Resource Description Framework (RDF). The proposed method covers a broader spectrum of the "structuredeness" of the data models and is proved to be more expressive than the already used metrics. Similarly, our method combines structural information of the process models, along with size and other statistical information for deriving synthetic process models, that reflect the structural characteristics of a process model's collection. Vicknair et al. [VMZ+10] compare the performance of a graph to a relational database. The workload of the benchmark is separated to structural and data queries. The structural queries refer to the storage of data provenance information stored as Directed Acyclic Graph (DAG), while the data queries use payload. The used data sets contain artificial provenance information. Our work also focuses on the structural characteristics of process models, while data are generated with normal distribution functions.

Dominiguez-Sal et al. [Dom+10] make a survey on high performance computing scalable graph analysis benchmarks. For this, they utilise a Recursive Matrix (R-MAT) algorithm [CZF04], which generates non-attributed directed graphs. The graphs can represent real networks with respect to specific parameters such as degree distribution and diameter. The XGDBench [DS12] targets stressing exascale clouds, i. e., computing systems capable of at least one billion calculations per second (exaFLOPS). The benchmark is based on the multiplicative Attribute Graph Model (MAG) [KL12] for the generation of synthetic graphs. The generated graphs are compliant with analytics, and thus statistically interesting. Moreover, in contrast to the synthetic graphs of the R-MAT algorithm [CZF04] the generated graphs are attributed. Generally, the MAG algorithm is found to outperform the R-MAT [CZF04] algorithm. Bader and Madduri [BM06] summarise three approaches of graph generators. One of these approaches (SSCA#2 graph generator) aims to produce graphs that are in turn used for benchmarking purposes. Although the application field and method differs, this approach has the same motivation goal as ours. On the whole, the aforementioned works have different application fields from ours, which focuses on the synthetic graph (i. e., BPMN 2.0 business process models) generation from reoccurring structures. The synthetic business process models are then used for benchmarking purposes.

2.2. Business Process Management

In the following, Section 2.2.1 focuses on related work on anonymisation and other approaches regarding the protection of process models privacy; Section 2.2.2 presents related work in process models collections and repositories; Section 2.2.3 introduces techniques on process models decomposition; Section 2.2.4 studies techniques that detect similarities between process models; and Section 2.2.5 addresses existing approaches in process models generation.

2.2.1. Anonymisation of Process Models

Attempts for anonymisation can be found in various fields of computer science such as network security (e.g., filtering, replacement, reduction of accuracy [YWH+07]) and database systems (e.g., data generation, encryption [VP12], k-anonymity, l-diversity, and t-closeness [ARX12]). Data anonymisation is an emerging topic also in cloud environments as “data anonymisation can ease some security concerns, allowing for simpler demilitarised zone and security provisioning and enabling more secure cloud computing" [Sed12]. Likewise, Zhang et. al. [ZLN+14] focus on data analysis and propose a privacy-preserving framework based on MapReduce [DG08]. The proposed framework addresses the challenge of privacy retention of data shared in public cloud infrastructures. Most of the existing approaches discussing data anonymisation cannot be directly applied to the existing process modelling languages as they are tightly coupled to the architecture and principles of different technologies.

When it comes to business processes, privacy protection on the cloud can be described by the following three objectives: know-how preservation, data confidentiality and trust verification [GMG13]. To this effect, Goettelmann et al. [GMG13] propose a novel approach for protecting the corporate assets that are compromised through the deployment of a business process on the cloud. More particularly, the trusted deployment of a business process on the cloud is a three-step approach: requirements definition, business process remodelling and cloud selection. To this effect, Goettelmann et al. [GMG13] propose the anonymisation of business process tasks as an additional step to the trusted-deployment process. Dave et al. [DKP+13] suggest anonymisation techniques as a means to protect data on the cloud and Strauch et al. [SBK+12] describe cloud data patterns to support the data confidentiality on the cloud. However, exact details on how to consistently anonymise the business process while maintaining its executional semantics are not given in any of the aforementioned works. In follow up work, Goettelmann et al. [GAYG15] suggest the fragmentation of a business process and distribution of the individual fragments on multiple clouds. Through this approach a single cloud provider cannot perceive the big picture of the know-how contained in critical business process fragments. To strengthen this approach against a conspiracy of several malicious cloud providers Nacer et al. [NGY+16] propose the obfuscation of the business process through the injection of fake process fragments.

In the field of BPM, and to the extent of our knowledge, at the time of writing this thesis there exists no other approach that describes a comprehensive method to anonymise business processes expressed in the WS-BPEL or BPMN 2.0 languages. Nevertheless, anonymised business processes are already used in existing projects. For example, Kunze et al. [KLWW11] anonymised business processes before publishing them in a large public process model collection. However, the method followed to anonymise the business processes is not discussed, and the business processes used are not in an executable format. Bentounsi et al. [BBDA12] propose a method to publish business processes on the cloud while maintaining privacy. This approach is based on fragmenting the business process and sharing some parts of it. The sensitive data of the client are anonymised but the context of the fragment is preserved. Adopting this approach would not satisfy the functional and non-functional requirements identified in Section 3.1, since we endeavour to encourage the sharing of the complete business process models while completely obfuscating any business information.

2.2.2. Process Models Collections and Repositories

A business process model repository is a location used for storage and retrieval of process knowledge (business rules, relationships, process elements, etc.) [ML08; Ma12; LRvdA+11; Eli15; YDG12]. Consequently, a process model repository can be seen as a successor of a process model collection, which only offers the storage of the process models. Several and diverse architectures are proposed as process model repositories, each one of them offering different features [Eli15; YDG12]. Yan et al. [YDG12] conduct an extensive literature review on existing process model repositories and identify the following driving forces for their usage: (i) storing and retrieving business processes; (ii) storing business process models and their running instances; and (iii) enabling process model re-usability. For this work, we first examine existing process model collections for discovering publicly available process models. The storage, retrieval and re-usability of the collected process models or parts of those (i. e., process fragments) [STK+10; YDG12] comprise fundamental requirements of this work, thus we study existing solutions for process models repositories with respect to these objectives.

Although process model collections constitute an essential artefact for practice and research, only few of them are publicly available [EKMW12]. The SAP reference model collection was published in 1997 and contains 604 Event-Driven Process Chain (EPC) process models focusing on the SAP R/3 system. These process models are broadly used in diverse research works [EKMW12] but are currently outdated and thus no longer available online. As we will discuss further in Chapter 3 the IBM Academic Initiative and the Business Process Management Academic Initiative (BPMAI) constitute two main sources of thousands of publicly available process models. Both collections have been considered in our research along with additional process models derived from private collections. The private collections of process models constitute a corporate asset for the companies [Ley01] and were shared with us under confidentiality agreements. Hence, the full collection of process models used in our research cannot be made publicly available.

There are currently various process model repositories available, each of them providing diverse functionalities [Eli15; YDG12] and/or performance behaviour on information retrieval [Eli15; JWR+13]. Recent research on existing process model repositories identified a set of limitations that hamper the usability of process model repositories [Eli15]. These for example, are the release of repositories under proprietary terms of usage, inefficient process model retrieval mechanisms, the lack of goal orientation on the designed process models and the difficulty to identify relationships and dependencies between the process models. With respect to the identified deficiencies Elias [Eli15] introduced a process model repository that can be used to view, store, search and version process fragment or process model collections. The architecture of the aforementioned repository is different from the traditional three layers model that consists of a presentation layer, back-end (business logic) layer and a data layer. It has five layers, one of which, is the interoperability layer. The latter assists the user to easily exchange the process models among diverse repositories by using a wrapper functionality, that translates the queries with regards to the called repository. The second additional layer is a service layer that exposes the functionality provided by the repository though web services and APIs. Furthermore, the proposed solution allows the manual insertion of metadata information describing the functional semantics of the process models (application area, process type, resources received, resources provided, etc.). The inserted metadata is then used for the retrieval of a process model that can be reused and adapted to fit diverging business needs. Similarly, the prototype proposed in this work exposes its functionalities as services to enable their integration to other existing solutions (see Chapter 8). In our work, the metadata is automatically extracted out of each process model and it describes the structural composition of the process model. Process model re-usability is then enabled through the (semi-) automatic generation of a process model out of individual process fragments with respect to structural criteria (see Chapter 7).

The Advanced Process Model Repository (APROMORE) [LRvdA+11] was originally built as a process model repository and has currently extended and enhanced functionalities as an online business analytics platform. It offers a wide range of features, spanning from basic repository (i. e., filtering process models, clone detection, querying, etc.) to process analytics functionalities (i. e., behaviour based comparison of process models and event logs, visual analysis of process performance, etc.). To this effect, an intermediate canonical format is used that provides diverse possibilities on process models analysis. In a similar manner, the PromniCAT [FHM14] repository uses a uniform format to unite process models originating from diverse process model languages and collections into a unique database. However, performance analysis of WfMSs depends, among other factors, on the underlying process model language that the WfMSs support [PFR+15]. Thus, intermediate formats could cause a loss of information that might be impactful to the WfMS performance. For this reason we decided not to use either PromniCAT or APROMORE for our research. Nevertheless, the developed prototype follows a SOA, hence enabling the integration of developed functionalities with third-party platforms. Signavio [EKMW12] is a tool for business process modelling and analysis. Its core components are built on PromniCAT [FHM14] and Oryx [Ory08], a web-based process model repository developed for browsing, creating, storing and updating process models. It merely focuses on the modelling and correctness analysis of business processes and has been used in our research for modelling purposes only.

Vanhatalo et al. [VKL06] present a repository for process models that may also be conceived as an eXtensible Markup Language (XML) repository as the process models are serialised in XML before being stored in it. The proposed approach takes as input process models serialised in XML and takes care for their representation and retrieval as Eclipse Modeling Framework (EMF) Java data objects. Other important characteristics of the proposed repository are [VKL06]: (i) a data handler which serialises EMF data objects to XML documents and de-serialises the XML documents back to EMF objects; (ii) extensibility functionalities, by allowing the addition of new data type models at a later point in time; and (iii) faster and easier data retrieval through the exploitation of Object Constraint Language (OCL). The proposed approach focuses mostly on retrieving, editing and storing WS-BPEL process models, while our work centers the attention on Business Process Model and Notation (BPMN) process fragments. Moreover, architectural differences occur as Vanhatalo et al. [VKL06] use a file system as a repository, whereas in this work relational databases are used for the storage of the process fragments and its metadata respectively.

Seidita et al. [SCG06] introduce a process fragments repository that emphasises the design of a new process model for multi-agent systems and Schumm et al. [SKLS10] propose a process fragments library called "Frammento", used for storing and managing process fragments. The process fragments can be retrieved from these repositories through exploitation of pre-inserted context-related metadata. Unlike these approaches, our work aims at the automated calculation of structural metadata (cf. Chapter 7). This metadata information, is then used for the synthesis of a new process model. Similarly, a transformation step proposed by Schumm et al. [SKLS10] enables the integration of process fragments into complete process models. The transformation step may take place during design or deployment time and it might cause behavioural or structural changes on the process model. The structure of the composed process model might also slightly change in order to ensure its consistent behaviour (e. g., avoid infinite waits) [SKLS10]. For this, the authors also define the concept of “regions” in a process fragment, which represent placeholders to indicate which places of a process fragment may be customised. In the process models generation approach proposed in this work, we exploit the regions concept introduced by Schumm et al. [SKLS10] and customise it to the needs of our approach (see Chapter 7). Moreover, similar to the transformation step proposed by Schumm et al. [SKLS10] our approach might also lead to small changes on the final process model, for ensuring conceptual and behavioural consistency. The architecture and implementation of Fragmento is tightly coupled to WS-BPEL process modelling language, while this work focuses on the BPMN 2.0 language and aims at different objectives. Namely, these are the automatic calculation of structural metadata and the retrieval of process fragments with respect to structural metadata. Hence, we argue that the development of a new software solution would lead to a more robust prototype.

2.2.3. Decomposition of Process Models

Process decomposition is frequently discussed in the literature [STK+10; ELtHF11] and is accepted to have an important role in the re-usability of process models. The concept of subprocess [RMD10] enables the reuse of process models, by allowing the reuse of semantically complete process model parts [Ma12]. The facilitation of reuse of arbitrary business process model parts that cannot be expressed as self-contained business processes is addressed through the introduction of process fragments [Ma12]. Overall, process fragments are reusable, syntactically or semantically incomplete parts of process models [Ma12]. Typically, process fragments are semantically annotated with some functional semantics (process domain, process type, etc.) or other requirements depending on the application domain. According to Unger et al. [UEKL10] the definition of subprocesses is similar to that of process fragments in terms of re-usability. However, the two concepts reflect a different conceptual context as subprocesses grant their instantiation, execution and handling to the parent process [UEKL10].

The concept of process fragments may also be compared to this of libraries in typical programming languages where one can reuse the provided functionality without starting from scratch [SKK+11]. Eberle et al. [ELS+10] introduce a process fragment modelling language based on the BPMN 2.0 standard, through which different process fragments can be composed and reused at a later point in time. Schumm et al. [SLM+10] proposed two methodologies for the creation of process fragments. The first one is a top-down approach, in which a part of the process model is manually extracted from a bigger process model. This methodology is similar to selecting a sub-graph of a complete graph. The second methodology is bottom-up and a process fragment is built from scratch and designed to fulfil requirements with respect to some particular context. The approach proposed by Schumm et al. [STK+10; SLM+10] considers only functional metadata for semantically annotating the process fragment. Moreover, the semantic annotation as well as the creation of the process fragment are manual processes. In this work, we extend the original definition of process fragments to Relevant Process Fragments (RPFs) (see Chapter 6) by shifting the focus on the re-occurrence of a structural fragment in a collection of process models. Furthermore, the RPFs are automatically extracted from the collection and annotated with regards to its structural metadata.

Refined Process Structure Trees (RPSTs) are an automated technique for fragmenting process models to regions [VVK08]. More specifically, Refined Process Structure Tree (RPST) fragments the process models into Single-Entry-Single-Exit (SESE) regions, out of which a hierarchical tree representation can be constructed [VVK08; PVV11]. The process fragmentation in SESE regions has diverse applications, as for example, the usage of process fragments to enable the refactoring of a process model [WRMR11] or detecting matchings of process models [EDG+12; Dum+13]. As one of this work's objectives is to discover reoccurring structures in a collection of process models (cf. Chapter 6), the RPST technique has been extensively studied for possible suitability. However, in our case the derived fragmentation does not produce all the possible substructures that should be examined as reoccurring. Therefore the RPST technique is not considered by our approach.

Process decomposition serves also to the distributed execution of process parts. For example, Khalaf [Kha08] partitions the original process into process fragments with augmented information. These fragments can then be executed in a distributed manner, while maintaining the execution semantics. Similarly, towards a secure business process execution on (multi-) cloud environments Goettelmann et al. [GAYG15] suggest the distributed execution of individual process fragments on multiple clouds. Security issues may be defeated through obfuscating the business processes by injecting fake process fragments [NGY+16].

2.2.4. Identification of Structural Similarities of Process Models

Process model similarities is a major research stream that branches to three directions: textual, behavioural and structural similarities [Dij+13; Len16]. Textual similarities approaches base their comparisons on the labels of process elements (e. g., task labels, event labels, etc.), behavioural similarities approaches exploit the execution semantics of process models, and approaches on structural similarities compare the textual semantics of process models, as well as their topologies [DDG09; DDvD+11]. This section focuses on work done with regards to the structural similarities of process models as this is the most relevant for this work. The structural approaches presented by Dijkman et al. [DDG09; DDvD+11] focus on the structural similarity metric of Graph Edit Distance (GED). The GED between two graphs is defined as the minimum amount of insertion, substitution or deletion operations that one needs to perform for transforming one graph into the other [DDG09; DDvD+11]. The GED similarity metric focuses on the textual semantics of two process models' nodes for evaluating their similarity. Except for the fact that textual information is required to execute the textual comparison, it was also recognised to under-perform for process models with $size > 20$ [DDG09;

Dum+09]. On the contrary, as discussed in Chapter 9 our approach does not rely on textual semantics and has performed efficiently for application to much bigger models ($size > 100$) where it has detected reoccurring structures with $30 \leq size \leq 51$.

Breuker et al. [BDDS14] conducted an extensive research survey on the performance of publicly available algorithms for frequent pattern matching without candidate generation (i. e., without the usage of a constant structural pattern to be searched for). More particularly, Breuker et al. [BDDS14] executed experiments in a collection of thousands of EPC process models by using the Graph-based Substructure Pattern Mining (gSpan) [YH02] and Graph / Sequence / Tree extractiON (GASTON) [NK05] algorithms to discover frequently occurring structural patterns. The gSpan algorithm introduces a novel labelling method that enables the easier sorting and comparison of graphs. Hence, instead of searching graphs and testing for isomorphism the gSpan algorithm constructs *canonical Depth First Search (DFS) codes* which are equivalent only if the graphs are isomorphic. The GASTON [NK05] algorithm relies on the observation that the most frequent substructures are usually graphs without cycles. The GASTON algorithm discovers all frequent subgraphs by using a level-oriented approach in which, first simple paths are considered, then more complex structures and finally cyclic graphs. The algorithm uses the "quickstart" observation that the various substructures are nested to each other. With respect to this observation the algorithm organises its search space efficiently. The GASTON algorithm is targeted for simple substructures, while for larger substructures more advanced algorithms are suggested [NK05]. Similarly to our approach, the experiments conducted by Breuker et al. [BDDS14] skipped the textual semantics of the process models and executed comparisons only with respect to the structural semantics of the process models. The experiments showed a quick failure of both algorithms [BDDS14]. For our approach we adopted existing techniques suggested initially by Ullmann [Ull76] and later on by Valiente and Martínez [VM97] and customised them for the detection and extraction of RPFs.

Querying of process models repositories for similarities also targets the detection of process models variants in a repository [Ma12; JWR+13]. Jin et al. [JWR+13] propose a method for efficient business process model retrieval from process models repositories. To this effect, a first subset of matching process models is retrieved through the exploitation of the graph database indices. The set is then refined through the usage of an adapted version of Ullmann's algorithm [Ull76], by discarding the models that do not explicitly contain a pre-defined subgraph. Similarly to ours, the proposed method allows the graph matching without the usage of textual semantics. However, in the approach defined by Jin et al. [JWR+13] a pre-defined graph is searched against the collection, while in our case the similarities between the process models are detected without knowing a subgraph a priori. Wang et al. [WJWW13] highlight the importance of querying process models with respect to their graph structure and employ the gSpan algorithm to query business process model repositories. The authors conclude that too many different labels are used between business process models. This fact leverages the necessity of the approach proposed in Chapter 6 as it does not consider any labelling information for conducting a structural matching.

The APROMORE process model repository, discussed in Section 2.2.2, is extensively used by the business process management community for process models comparisons [LRvdA+11]. Among other features APROMORE provides similarity search and pattern-based analysis functionalities that are more relevant to the scope of this work (cf. Chapter 6). However, similarity search indicates the percentage of similarity between two process models, without indicating the exact regions of similarities for the process models. Likewise, pattern based analysis is searching the existence of a certain structural pattern in a collection of process models. In this work, we aim at extracting the structural similarities of a collection of process models without knowing a certain pattern beforehand (cf. Chapter 6). The features offered by the current state of the APROMORE repository do not satisfy the objectives of this work.

Hertis and Juric [HJ14] conduct an analysis similar to ours with a focus on WS-BPEL. In order to detect the reoccurring structures in a collection of thousands of WS-BPEL process models, they transform the process models to process trees. Afterwards, they apply tree mining algorithms to detect and extract the reoccurring structures. Although the ultimate goal of their work is similar to ours, the different nature of the BPMN 2.0 language does not allow to apply the same tree mining techniques for similar structures detection. Moreover, structural similarities are searched with respect to a specific graph. A prototype for the comparison of BPMN 2.0 process models is introduced by Pietsch and Wenzel [PW12]. In their approach they assume that the compared process models are variants of the same process model and heuristics based on textual semantics are used for identifying similarities and differences. Although the approach seems very promising it is argued by the authors that it might not be efficient to large, complex real world process models. Overall, to the extent of our knowledge the approach introduced in this work (Chapter 6) is the first complete approach that detects and extracts structural similarities from a collection of BPMN 2.0 process models by (i) relying solely on the structural information of the process models and (ii) without using an a priori known graph pattern to search for.

2.2.5. Generation of Process Models

The (semi-)automated generation of synthetic process models is necessary in the absence of available large collections of processes from the industry or academia [YDG15]. In this work, we discuss the (semi-) automated workload mix generation as one of the major challenges towards developing a benchmark for WfMS. Generation in this context refers to the process of discovering, extracting, selecting, and synthesising process fragments into *reference process models* that resemble as much as possible realistic models from practice. These non-executable reference process models can be refined by the process modellers with implementations to make them executable [Kha07].

Yan et al. [YDG15] propose a complete method for generating synthetic process models. The method constructs gradually the graph of a synthetic process model by exploiting the statistical properties and similarities of a collection of thousands of real world process models. Moreover, the authors propose the usage of an RPST [VVK08] as a possible improvement of their work. In Section 2.2.3, we argued that the RPST approach cannot be utilised for our purposes and we have therefore introduced the concept of Relevant Process Fragment (RPF)s (see Chapter 6). The method proposed in this work, and the method of Yan et al. [YDG15] can be used complementarily to each other to provide a complete solution of synthetic, representative, executable process model generation.

As discussed in Section 2.2.3, process model decomposition is an approach to easier and faster develop complete process models [SKK+11; SLM+10]. We utilise this concept for storing and querying RPFs, which are an extended type of process fragments. The above approach bases on the textual semantics of the stored process fragments, while our approach focuses on the structural characteristics of the RPFs. Eberle et al. [ELS+10] present a formal model for process fragments and corresponding operations for their composition. In this work, we adopt the suggested methods for composing the process fragments into process models and we extend the composition function in order to stress the representativeness of the generated process model.

Business process consolidation is an approach to construct process models out of process model collections that share common process fragments [LDUD13]. La Rosa et al. [LDUD13] introduce a method to merge variants of the same process model into a unique merged process model that contains all variants. The merging applies a rule that calculates the union of the edges of two graphs and merges these nodes for which the labels match above a threshold. Zemni, Mammar and Hadj-Alouane [ZMH16] also focus on process models consolidation to propose a mechanism that supports the flexible merging of process fragments. The proposed approach is based on the usage of path matrices [Ran91], i. e., matrices that indicate if there is a path from one node to another. In the proposed approach the path matrices are used to reflect the existence of a path between adjacent nodes of a node based graph (i. e., process model). The merging mechanism exploits this information into calculating and providing correct merging paths between pairs of nodes (i. e., activities) of the business process models or fragments. An interesting aspect of this approach is that it is afterwards validated against undesired behaviour that might occur during the merging process. This is done by providing the process model designer with a mechanism to pre-configure behavioural constraints. The merging techniques described above could not be considered by our approach because; (i) we do not consider any textual semantics in any of the methods we propose and (ii) the goal of our method is not merging but combining process fragments to each other to construct a complete, aggregated process model.

Kopp et al. [KLSU11] describe a method to autocomplete the process fragments into complete process models. The background motivation of the proposed approach is to enable the verification on BPMN 2.0 tools that only support complete BPMN 2.0 processes and not process fragments. The approach mainly focuses on the transformation of a process model to Petri-Nets, the consistent addition of start events and gateways, and then the re-transformation of the derived process model to BPMN 2.0. The method relies on existing theorems [KtHvdA03; PGD10] to guarantee the soundness of the derived process model. As already discussed in Section 2.2.2, the performance analysis of WfMSs is also linked to the process modelling language that the WfMSs support [PFR+15]. Therefore, also in this case, the application of intermediate formats was not considered as an appropriate approach for our purposes. The process fragments synthesising method proposed in Chapter 7 follows the ideas presented by Kopp et al. [KLSU11] to add BPMN 2.0 language constructs for a process model's autocompletion. However, the rules we follow are adjusted in such way that the transformation of the synthesized process model to an intermediary format is avoided.

2.3. Chapter Summary

In this chapter we discussed existing related work in research areas that intersect with the application domain of this work. We first presented a set of existing standard benchmarks in technologies that can be considered akin to WfMSs. As a standard benchmark for WfMSs is not yet available, we also showcased existing work on custom benchmarks for WfMSs. This work is part of the BenchFlow approach, that aims at introducing the first standard benchmark for WfMSs. More particularly, we focus on defining a novel method to derive representative workload mixes for benchmarking WfMSs. With respect to this effort, we also studied related work in workload modelling and characterisation techniques. As in the current efforts of benchmarking WfMSs the process models used are arbitrarily defined, we focused on related work for generating synthetic graph-based workloads. Overall, in Section 2.1 we positioned our work at large, by showcasing its necessity in the field of benchmarking WfMSs.

The method we specify for deriving a representative workload mix (WINE4WfMSs) comprises of sub-methods, which can be separately examined as research contributions in the area of BPM. For deriving a representative workload mix one needs to take into account real world cases [Fei15]. As business processes comprise a corporate asset for the companies, and in order to foster the process model sharing we propose an anonymisation method that obfuscated process models, whilst keeping their execution behaviour. With regards to this, we examined related work for business processes anonymisation. The collected process models are stored in process models collections or repositories, that offer the additional advantage of process models querying and efficient retrieval [SKLS10; LRvdA+11]. Our work use existing public collections of process models, but does not use process model repositories as we show that the state-of-art is not directly satisfying the objectives of this work. Nevertheless, as we show in Chapter 8 we implement the defined methods by following a loosely coupled design and allows the integration of developed the components to existing works. Afterwards, the collected process models will be searched for existing structural similarities and decomposed with respect to them (Chapter 6). For this purpose, we also study the respective research areas of process model decomposition and process model similarities. The main difference of our work with respect to these areas, is that our approach detects structural similarities without considering the textual or behavioural semantics of process models. The detected structural similarities are then synthesised back to structurally representative process models (cf. Chapter 7).

Chapter 3

Collecting Real-World Process Models

“In God we trust, all others must bring data.”

W. E. Peming

Given the fact that the usage of a non-representative workload model may lead to misleading results [Fei15], the identification of a representative workload presupposes the extended analysis of a collection of real world practice process models. To this effect, in the context of the BenchFlow project we proceeded in contacting diverse industrial and academic organisations to request real world practice process models. As process models constitute a corporate asset for the companies [Ley01] collecting them is a challenging task. Hence, this chapter introduces research question RQ-1: “How to overcome obstacles in creating a collection of real-world practice process models?”. In order to foster the sharing, while addressing the concerns of the process models owners, we propose: (i) confidential agreements and

(ii) a method to obfuscate process models and their deployment artefacts. Through this method the process models are shared with the data that are suitable for our research without revealing critical company information.

In previous works [SRPL14; SFL+15], we introduced a method to anonymise processes described in the WS-BPEL language. In this work, we extend the method for anonymising both WS-BPEL and BPMN 2.0 processes. More particularly, this chapter approaches contribution C-1: "An anonymisation method for process models ("BPanon") and the obtained collection of process models" by presenting:

(i) a method for anonymising WS-BPEL and/or BPMN 2.0 process models, while maintaining their structural and behavioural semantics, and

(ii) the composition of the obtained process models into a collection.

The remainder of this chapter is structured as follows: Section 3.1 proposes a method for anonymising process models described in the WS-BPEL or BPMN 2.0 language, Section 3.2 describes the composition of the process model collection and Section 3.3 summarises the chapter.

3.1. Anonymisation of Process Models

In this section, we propose a method to obfuscate process models focusing on process modelling languages that allow web services orchestration, namely WS-BPEL or BPMN 2.0. The proposed approach produces an *anonymised* or a *pseudonymised* process model that maintains its executional and timing behaviour. Pseudonymisation is the technique of masking data, while maintaining ways to reach back to its original data, therefore in pseudonymisation the output of the executed process model can also be mapped back to its original, non-anonymised version [Fed90]. This way, it allows process model designers to trace bugs or inconsistencies found in anonymised files and apply changes to the originals. On the contrary, anonymisation edits critical data and makes it impossible to reach back to the original version [SBK+12]. The proposed approach is designed with a focus on the process models, which means that the data and web services that interact with them are simulated using stubs. However, our solution can be extended or combined with already existing solutions for data [Sed12; ZLN+14] and web services [DRS10] anonymisation to protect the company's information to the maximum possible extent.

3.1.1. Process Model Anonymisation Requirements

The design of the anonymisation method must address the following initial list of requirements identified during our work in various research projects, and especially during our collaboration with industry partners. The main requirement and purpose of the method is to provide:

R-1: PSEUDONYMISATION/ANONYMISATION:
Produce a process model that contains obfuscated textual semantics, while it maintains an equivalent timing and executional behaviour. Moreover, the end user of the application can select if the process model will be pseudonymised or anonymised.

In order to fulfil the [R-1: PSEUDONYMISATION/ANONYMISATION] requirement a set of subsequent requirements emerge. These can be divided into groups with respect to their root cause requirements specific to the: (i) use of XML language, (ii) business process modelling and (iii) anonymisation methods.

More particularly, we recognised the following list of XML language requirements:

R-2: NO INFO IN NAMES:
Sensitive information may appear in activity names, variable names, XML Schema Definition (XSD) element names, etc. The name choice for these attributes is usually descriptive and reflects the actual actions to which they correspond, thus they reveal the process model's semantics.

R-3: NO INFO IN NAMESPACES:
The anonymised process model must not include namespace information

with links to external web sites that reveal business information (backlinks).

R-4: NO INFO IN BACKLINKS:
The anonymised process model should not contain any backlink information. Namely, it must not contain names (such as activity names, variable names, message names, operation names, role names or XSD element names) with backlinks to business information.

R-5: NO INFO IN XPATHS:
The anonymised process model should not contain sensitive information in XPaths. Namely, it must not contain XPath expressions with backlinks to business information. If no custom XPath functions are used, [R-5: NO XPATH INFO] is a consequence of requirement [R-4: NO BACKLINK INFO].

R-6: NO INFO IN DOCUMENTATION:
The resulting anonymised process model must not contain description containers (comments and documentation), that reveal critical information and semantics.

The process modelling nature-specific requirements are recognised as follows:

R-7: KEEP STRUCTURE & EXECUTABILITY
The anonymised process model should maintain its structural information and executability.

R-8: KEEP RUNTIME
The anonymised process model must maintain an equivalent runtime.

Finally, the following requirements are related to the used anonymisation method, especially focusing on the renaming of elements during the anonymisation process:

R-9: PREVENT REVERSE ENGINEERING
The obfuscated names of the anonymised process model should prevent

the reverse engineering to retrieve the original names. For example, if the data are obfuscated with anagrams of the words, it is easy to retrieve the original word.

R-10: AVOID NAME CONFLICTS
The names must be chosen in a way that conflicts will be avoided between the original and resulting file. Namely, we should ensure that the names we choose for the anonymisation are not already existent as names in the original file. Otherwise, we will have a sequence of conflicts when trying to separate the original and anonymised elements.

R-11: HUMAN READABLE NAMES
Human readable names must be chosen for the anonymised process model. For example, let us assume that we apply Universally Unique Identifier (UUID) as name choice for the anonymisation process. This action would produce artifact names such as `f81d4fae-7dec-11d0-a765` and lead to a process model that is not human readable.

3.1.2. Method of Process Models Anonymisation

As business process models are typically represented as a collection of diverse artifacts, designing an anonymisation method that consistently fulfils the aforementioned requirements is challenging. The artifact collection of a WS-BPEL process should contain at the minimum a deployment descriptor, one or more process definitions in WS-BPEL, WSDL files and XSD files [Apa13a]. Provided the assumption that all parts of a process model are specified in one single file, a BPMN 2.0 process is contained in a simpler package, as the definition of a deployment descriptor is optional in the state-of-art WfMSs [Apa13b] and WSDL files are included only if required by the underlying process model. Nevertheless, for the purposes of defining the anonymisation method consistently we consider the most complicated case, i. e., the existence of all possible artifacts in the package. Figure 3.1 shows the considered artifacts of a business process package and the ways a change during the anonymisation of one artifact influences the

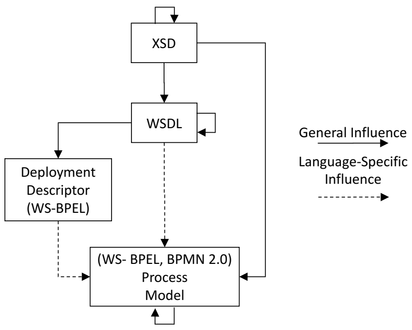

XSD
WSDL
Deployment Descriptor (WS-BPEL)
(WS- BPEL, BPMN 2.0) Process Model
General Influence
Language-Specific Influence

Figure 3.1.: Influence relations among the artifacts of a WS-BPEL or BPMN 2.0 business process model for the anonymisation process.

surrounding ones. This means that the target-artifact of the arrow (e. g., process model) is influenced by the origin-artifact (e. g., XSD). Therefore, when the origin-artifact is changed the interconnected artifacts should be accessed and changed as well. Consequently, the influence relations among files increase the complexity of anonymisation, as small changes in a file may lead to numerous subsequent changes to other business process artifacts.

The Business Process Anonymisation (BPanon) method [SRPL14; SFL+15] starts by anonymising the surrounding artifacts and finally applies the subsequent changes to the process model. More specifically, as shown in Figure 3.1, the XSD file can be seen as an "initial" artifact in the anonymisation process. This is because it is not influenced by any other artifact and it contains definitions referenced by the rest of business process artifacts. By initially anonymising the definitions and then updating their references we achieve a consistent anonymisation. Thus, in our process we start with anonymising the XSD, continue with the WSDL and deployment descriptor, and we lastly anonymise the process model which basically is a combination of references to the rest of artifacts. The elements of the business process artifact files can be divided into three groups:

Free Elements Group: elements that need to be anonymised, but are not bound to subsequent changes that occur in other files.

Externally Bounded Group: elements that are bound by the surrounding artifacts and need to be updated when such an artifact is anonymised.

Internally Bounded Group: elements that need to be changed because they are bound to other changed elements within the same file.

The anonymisation of the *Free Elements Group* is trivial, as it can be reduced to string replacement. For example, documentation elements belong to this group. However, the anonymisation of *Externally Bounded Group* and *Internally Bounded Group* are more complex tasks. For its implementation we need a *Registry of Alterations* (referred to as $registryOfAlterations$), i. e., a registry of metadata that is created during the anonymisation of an artifact and which logs all conducted changes. The $registryOfAlterations$ contains at least information about the element's type and the corresponding attributes of the original and anonymised data. The original and anonymised data are marked with the tags "old" and "new" to the $registryOfAlterations$ respectively.

The main idea of anonymisation is to scan the surrounding artifacts of the business process package (i. e., XSD, Deployment Descriptor, WSDL, etc.) to detect element attributes that might contain semantics that need to be obfuscated, thereafter referred to as *critical attributes*. The *critical attributes* are pre-stored as metadata information for both process modelling languages. For example, in the case of WS-BPEL keywords such as "partnerLinkType" or "portType" are added in the set of critical attributes. As soon as a critical attribute is anonymised a pair of ("old","new") values is entered to the $registryOfAlterations$.

As a next step we scan the business process model itself to detect references of the obfuscated elements and update their value. In the following we discuss the anonymisation method, part of which is presented in Algorithm 3.1. For simplicity reasons the Algorithm 3.1 focuses on the anonymisation of WSDL files and underlying process model. Thus, the demonstrated algorithm shows a part of the *Externally Bounded Group* anonymisation. For the complete anonymisation of the business process artifacts (cf. Figure 3.1) a similar process needs to be followed.

The anonymisation method (see Algorithm 3.1) starts with the creation of a metadata set that describes the interconnections of the business process artifacts (Figure 3.1). From a technical point of view, this can be done by parsing the `<import>` annotations at the beginning of the process model file. The created metadata set, referred to as $tableOfReferences$ in Algorithm 3.1, shows the links between the process model and its WSDL files. For the discovered WSDL files we edit the definitions of artifacts referenced by the process model. To this effect, we parse each of the WSDL files registered in $tableOfReferences$ and gradually anonymise the *critical attributes* of their elements. In order to satisfy requirements regarding the human readability the anonymisation procedure picks randomly a word from an English Dictionary [Win00]. A word of a well known human language will lead to more readable results instead of using random strings as Universally Unique Identifiers (UUIDs). We then remove the chosen word from the dictionary, in order to eliminate the name conflicts.

By maintaining a $registryOfAlterations$ we apply the subsequent changes to the process model file. The changes of the WSDL file are reflected to the process model file by initially calling the findElementOfInterconnection() function. This function returns the WSDL amended element ($changedElement$) that has caused a subsequent change to the process model file. For robust functioning the findElementOfInterconnection() pre-supposes unique identifiers for the *e.id* elements. Then, the value of the current element (c) is changed accordingly and we update the $registryOfAlterations$ for consistency. Finally, the $tableOfReferences$ and $registryOfAlterations$ are destroyed if the method is set to anonymise. If pseudonymisation is chosen instead, then these records are preserved and persisted in order to allow the de-anonymisation of the model if necessary.

The complete BPanon method is realised as a component of the tool chain shown in Chapter 8. In previous work [SFL+15] we conducted an evaluation of the BPanon process and verified that we satisfy the specified requirements.

Algorithm 3.1 Anonymisation process for an externally bounded group (case of WSDL)

```
Input: $m$ : the process model file to anonymise. The changes will be reflected
  directly on it.
Input: $anonymisation$ : a boolean variable. If set to true it indicates that
  we are applying anonymisation otherwise pseudonymisation is applied.
Output: $BP_{anon}$ : a set that contains the anonymised process model ($m$) and
  a set of anonymised WSDL files ($W'$)
  function BPANONWSDL($m$)
      Create $tableOfReferences$ by parsing <import> elements of $m$
      $W' \leftarrow \emptyset$
      $W' \leftarrow \{w : w \in tableOfReferences \wedge type(w) = WSDL\}$
      for each ($w \in W'$) do
          for each (element $e \in w$ ) do
              $C \leftarrow$ GETCRITICALATTRIBUTES($e$)
              for each ($c \in C$) do
                  $registryOfAlterations \leftarrow$ UPDATEREGISTRYOFALTERATIONS(
                      $e.id$, $e.type$, $c.data$, $c.type$, "$old$")
                  c.data $\leftarrow$ APPLYANONYMISATION($c.data$)
                  $registryOfAlterations \leftarrow$ UPDATEREGISTRYOFALTERATIONS(
                      $e.id$, $e.type$, $c.data$, $c.type$, "$new$")
              end for
              $W' \leftarrow W' \cup \{w\}$
          end for
      end for
      // Reflect the changes on the process model file
      for each (element $e \in m$) do
          $C \leftarrow$ GETCRITICALATTRIBUTES($e$)
          for each ($c \in C$) do
              $changedElement \leftarrow$ FINDELEMENTOFINTERCONNECTION(
                  $e.id$, $e.type$, $c.type$)
              $registryOfAlterations \leftarrow$ UPDATEREGISTRYOFALTERATIONS(
                  $e.id$, $e.type$, $c.data$, $c.type$, "$old$")
              $c.data \leftarrow$ GETNEWVALUEOFATTRIBUTE(
                  $changedElement$, $c.data$, $registryOfAlterations$)
              $registryOfAlterations \leftarrow$ UPDATEREGISTRYOFALTERATIONS(
                  $e.id$, $e.type$, $c.data$, $c.type$, "$new$")
          end for
      end for
      if ($anonymisation$) then
          delete $tableOfReferences$
          delete $registryOfAlterations$
      end if
      $BP_{anon} \leftarrow m \cup W'$
      return $BP_{anon}$
  end function
```

3.2. Real-World Process Models Collection

As process models constitute a corporate asset for most companies [Ley01] the industrial or academic organisations are reluctant to share them. To foster the sharing of the process models we signed confidential agreements with several of the partners, created tools that obfuscate the textual semantics of the process models as discussed in the previous section, and executed extensive searches on the web to discover openly available process model collections. Without focusing on a specific modelling language we collected 14,167 process models.

More specifically, our collection, as of April 2016, contains: 1% WS-BPEL, 2% EPC, 4% Yet Another Workflow Language (YAWL), 14% Petri Net, and 79% BPMN models, out of which two thirds are expressed in the BPMN 2.0 language. The BPMN 2.0 models form the vast majority of the collection. As we endeavour towards a workload model that represents the real world practice, we consider the large set of collected BPMN 2.0 process model as the most suitable for our purposes. Therefore, for this work we *focus solely on the BPMN 2.0 language*.

The targeting subset of the real world process models contains *12,624* BPMN 2.0 process models originating from the:

(i) IBM Industry Process and Service Models[1],

(ii) the BPMN 2.0 standard,

(iii) the research by Pietsch et al. [PY14],

(iv) the Business Process Management Academic Initiative[2] (BPMAI), and

(v) other research and industrial partners.

BPMAI requires a set of user-defined criteria for provisioning a collection of process models. Thus, the following selection filters were used for deriving the BPMAI collection: (i) BPMN or BPMN 2.0 processes, (ii) all languages

[1]IBM, Industry Models,
URL: `http://www-01.ibm.com/software/data/industry-models/`

[2]BPMAI, URL: `http://bpmai.org/`

available (English, German, etc.), (iii) 100% connectedness, (iv) any size available, (v) any date available.

The serialisation format of both the IBM Process and Service Models, as well as the BPMAI process models were not compliant with the BPMN 2.0 standard [ISO13]. Thus, extra editing was required in order to normalise the models to be compliant with the BPMN 2.0 standard serialisation. The IBM process models included custom IBM namespaces, which needed to be replaced with standard namespaces. Similarly, the elements that addressed these namespaces needed to be edited. To this end, we realised a normaliser that transforms the custom serialised process models to their standard equivalents. This step is also represented as a component of the toolchain presented in Chapter 8. The resulting model collection, including the application of the anonymization process to it as discussed in the previous section, is presented in more detail in Chapter 9 as part of the case study used for the evaluation of this work.

3.3. Chapter Summary

The definition of a representative workload requires the analysis of multiple, diverse, real word practice process models. Nevertheless, the confidential nature of the real world process models makes their collection a challenging endeavour. In this chapter, we discussed the BPanon method that targets the anonymisation of WS-BPEL and BPMN 2.0 business processes. The anonymisation of a business process can be complex due to the numerous artefacts that comprise it and their underlying dependencies. For conducting a consistent anonymisation of a business process one needs to know the critical elements to anonymise, and the dependencies between the included artefacts, in order to track down the sequences of changes that need to be applied.

The proposed method is applied for the collection of real world process models. This attempt resulted in the collection of 14,167 process models described in diverse process modelling languages (see C-1). An analysis on the collection indicates that BPMN 2.0 is the dominant process modelling language of the collection (79%), thus the rest of this work shifts the focus on BPMN 2.0 language.

Chapter 4

Defining Meaningful Workload

Lies, Damned Lies and Benchmarks

D. R. Moscato

Among other factors, a benchmark's reliability depends heavily on the definition of its workload model, as failing to derive a representative one might produce misleading results [VMSS12; Fei15]. The complexity of deriving a representative workload model for software systems is frequently discussed in the literature [CMT00; Alm02; EM06; KHSB12].

In order to support practitioners towards the definition of a robust workload model, Kounev et al. [KHSB12] summarise an abstract methodology. However, despite the existence of a general methodology, the parameters of the workload model (workload mix, load functions, and test data) vary depending on the objective of the test type (e.g., load test, stress test, soak test) and the System Under Test (SUT) [Mol14]. In order to increase the quality of our workload model's design (see research question RQ-2: "What are the basic components of a workload model for WfMSs?"), we primarily need to get an in-depth knowledge of related existing benchmarks. To this effect, in Chapter 2 we studied standardised benchmarks, that were published by industry-accepted consortia such as SPEC [Sta95] and TPC [Tra92a], as well as state-of-art custom benchmarks that target to measure the performance of WfMSs. This study leads to the identification of components that play a key role when designing a new WfMSs benchmark.

Having identified the components of a WfMSs benchmark, we afterwards delve into the definition of a metamodel of the basic workload model components for WfMSs (see contribution C-2: "A metamodel of the basic workload model components for WfMSs.")). To sum up, the original scientific contributions of this chapter are to:

(i) identify the components that are relevant for the design of new WfMS benchmarks and their underlying dependencies;

(ii) identify the basic components for a workload model for WfMSs benchmarking and their interactions.

The results of the aforementioned contributions are published in [FIP+16; SAB+16].

The remainder of this chapter is structured as follows: Section 4.1 describes the components that need to be defined when designing a new WfMSs benchmark and their dependencies; Section 4.2 identifies the basic workload components for WfMS benchmarking and their underlying interactions and Section 4.3 summarises the chapter.

4.1. Key Components of a Workflow Management System Benchmark

In the following subsections we identify the key components of a WfMS benchmark (see Section 4.1.1) and discuss existing dependencies on the design phase (Section 4.1.2).

4.1.1. Identifying the Key Components of a Workflow Management System Benchmark

Before proceeding to the design of a benchmark one should first identify the individual underlying artefacts. After studying the design of relevant standard middleware benchmarks [Tra92b; TPC15; Sta15; Sta07], as well as custom benchmarks for WfMSs [SFP+16; BBD10; DES08; Act11; Rol13; IC07] we extracted the key components of a benchmark. Due to the fact that all studied benchmarks have similar domains of application, their structure is common in many aspects. Thus, the identified components are derived from common information, while concepts that were not common in all studied benchmarks are not included. The identified components and their underlying relations are organised as a conceptual model shown in Figure 4.1.

The *experiment* can be considered as the orchestrator of the overall benchmark, as it is responsible for stressing the benchmarked system (*SUT*) with respect to the definition of a meaningful *workload model*. As *SUT* we refer to the individual system that is deployed and benchmarked during an experiment, and its benchmark-related attributes are the hardware configuration of its comprising artefacts, the product's name (i. e., Camunda[1], Apache Active MQ[2], etc.), as well as its software license, and the version of the benchmarked SUT. Keeping information on the systems license, is important to ensure the legality of results publication. As a widely accepted guideline, a representative benchmark should follow hardware configurations similar to the consumer environment [Hup09]. The *workload model* of an experiment can be described as the group of components that are used for stressing the SUT. The components of a workload model for WfMSs are extensively discussed in Section 4.2.

Each experiment is repeated for a predefined number of rounds and lasts for a specific time (i. e., duration). If for some reason the SUT stops responding, then the experiment times out after a predefined amount of time (i. e., time-out duration). The warm-up time takes place at the beginning of

[1]Camunda, URL: `https://camunda.org`
[2]Apache, Apache Active MQ, URL: `http://activemq.apache.org`

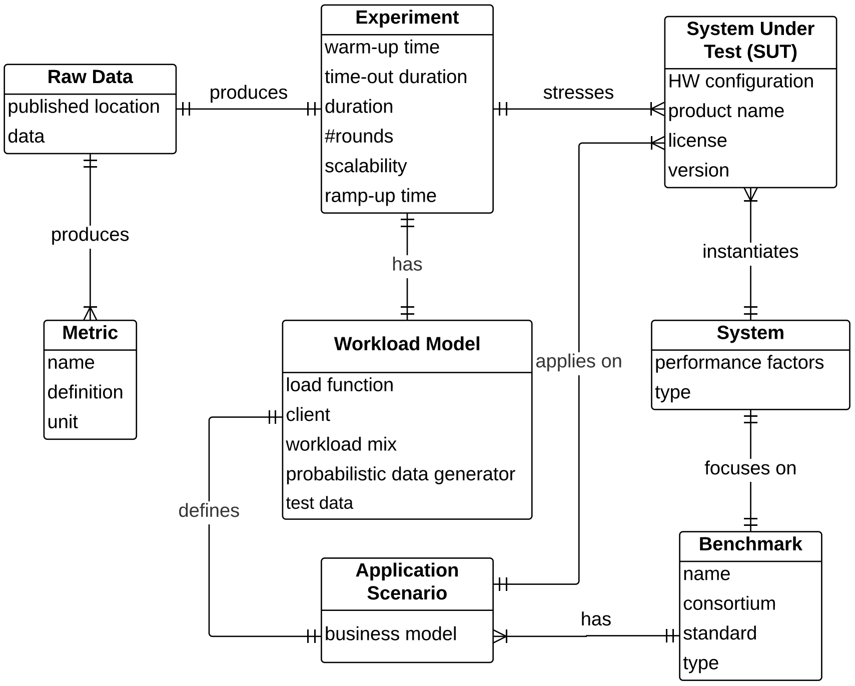


Figure 4.1.: Metamodel of a benchmark's components

the experiment and refers to the time that the system needs for initialisation before reaching a stable running state. Likewise, ramp-up time is called the time that the experiment driver need to reach the maximum level of workload. The durations of the warm up and the ramp-up times are usually excluded from the measurements of the experiment. Experiments might also stress the SUT's ability to scale the workload (i. e., scalability). Hence, an experiment also contains information on the scalability property that is being stressed (e. g., horizontal or vertical scalability [Sta07]).

At the end of the benchmarking procedure the experiments produce *raw data*. As suggested by research guidelines the raw data should be published online in order to foster reproducibility. Thus, the raw data are linked to their location of publication. Finally, the raw data are analysed in order to derive meaningful metrics. One of the characteristics of a good benchmark is the usage of meaningful and understandable *metrics* [Hup09]. Therefore, the selection of the metrics plays an important role in the design of the benchmark.

Each *benchmark* is characterised by its name, the consortium that proposed it, an indicator that shows if it is widely accepted and adopted as a standard benchmark and a type. Based on their nature, benchmarks can be categorised into different types (e. g., synthetic benchmark, application benchmark, micro/toy benchmarks, etc.). Various benchmarks focus on addressing the performance of different types of *systems*. Each system has a specific type, as for example a Java Server, Workflow Management System, etc. Each system has a set of factors or components whose performance affect the performance of the overall system (i. e., performance factors).

The *application scenario* of a benchmark describes a specific business model along with tasks that should be fulfilled (realised in the workload model). Thus, its design is dependent to the type of benchmark that we construct or apply. As the benchmark should be reachable and suitable for a large number of clients [Hup09], its application scenario should define concepts with respect to the area in which the benchmark will be applied. Moreover, it must contain and utilise the most widely used components and configurations of the SUT [MA01]. Finally, the application scenario should stress different functionalities that are offered by the underlying technology [SKCB07].

4.1.2. Model of Dependencies

The identified design dependencies between the aforementioned benchmark components are shown in Table 4.1. When a component of a row influences the design of a component in a column, then the corresponding table cell is marked with an “X”. Components that do not affect the design of any other component (e. g., workload model) or are not affected by any other component (e. g., system) are omitted from the rows or columns of Table 4.1

	Application Scenario	Benchmark	Experiment	Metric	System Under Test (SUT)	Workload Model
Application Scenario	■	–	–	–	–	X
Benchmark	X	■	X	–	–	–
Experiment	–	–	■	–	–	X
Metric	–	–	X	■	–	X
System	X	–	X	X	X	X
System Under Test (SUT)	–	–	–	–	■	X

Table 4.1.: Components dependencies of a WfMS benchmark design

respectively.

The system component affects decisions regarding the application scenario, the experiment, the metric, the SUT and the workload model. More particularly, the application scenario is strongly dependent to the system as it needs to demonstrate a representative use case of the system's usage. The design of the experiments is also inseparable from the system, as the benchmarking infrastructure that executes them, and the workload model they execute should be compliant to the system. The system plays also a big role in the definition of the metrics to be computed, as these should be interesting, relevant and representative of the system's performance [Hup09]. Lastly, the SUT must be compliant with the defined system type (e. g., WfMS).

The application scenario affects the design of the workload model, as the workload model needs to define tasks that are relevant to the application scenario. Likewise, the metric's design will affect the design of the workload model in order to be representative and drive to relevant raw data. Moreover, the implementation and behaviour of the workload model components are influenced by the experiment (e. g., load test, stress test, soak test) [Mol14]. Finally, the workload model is also affected by the system and the SUT, as its definition, design and behaviour should be compliant to the SUT's type.

The benchmark's goal as well as historical data on existing benchmarks affect the design of the application scenario and experiments. This is because data derived from applied practices provide information on best practices or detected pitfalls [FLH+17]. For instance, it is likely that a benchmark applied on a system for the very first time will not identify all the performance affecting factors in its initial design, or might design experiments that are not considering all the possible pitfalls for reliable measurements. For this reason TPC [Tra92a] and SPEC [Sta95] propose iterative processes for benchmark submission and maintenance [Poe12]. Moreover, the design of an experiment should follow the definition of the metrics and target to the execution of performance tests that will produce meaningful raw data as well as the characteristics of the system that is being benchmarked.

In general, in Table 4.1 we observe that the system has a core role as it represents the overall goal of the benchmark and therefore the benchmark's design is centred on it. On the contrary, the workload model seems to be the artifact whose design is affected the most by the surrounding components. What is more, the workload model plays an important role to the benchmark's execution and its design affects heavily the derived results and quality of the benchmark. Hence, designing a meaningful workload model is a critical task during a benchmark's design. In the following section we discuss the components of a workload model for WfMSs and their underlying interactions.

4.2. Identifying the Basic Workload Components

The workload model components for WfMSs and their interactions are presented in Figure 4.2. As we showed in Figure 4.1 (Section 4.1.1) the workload model of WfMSs consists of load function, clients, the workload mix, probabilistic data generators and test data. The *load function* describes

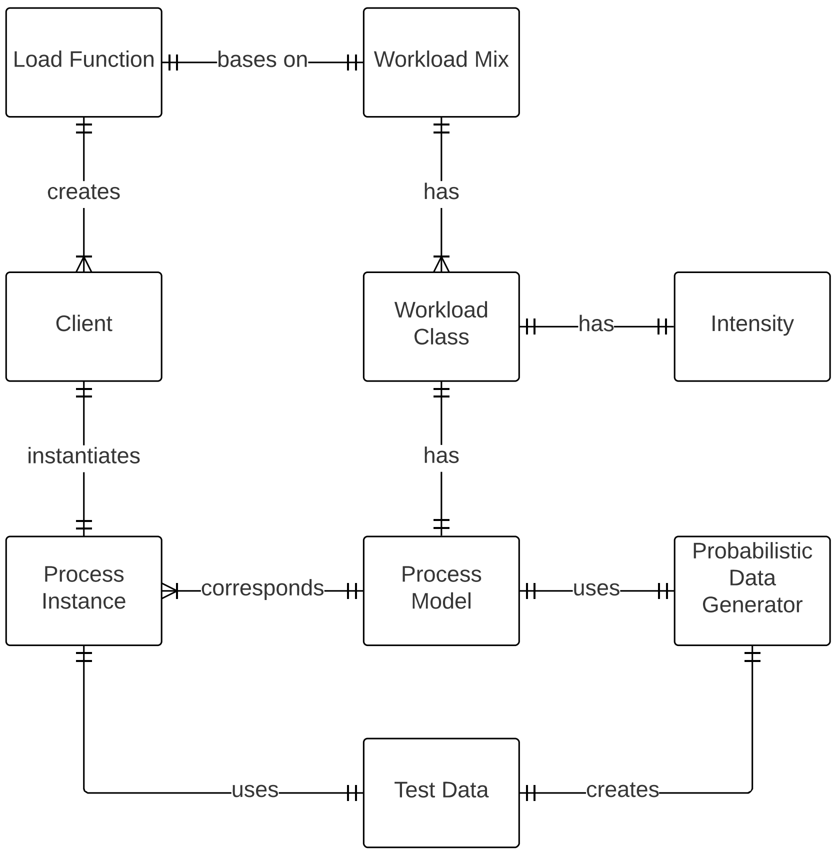


Figure 4.2.: Basic components of workload model for WfMSs and interactions

how the workload is issued to the SUT, i. e., the number of requests that will be executed against the SUT per time unit. The requests are made in the form of *clients* which are created by the load function and in their turn instantiate requests to the SUT [FIP+16]. In the case of WfMSs, these requests are *process instances* that correspond to the specification of a *process model*.

For their instantiation and execution process instances require and produce *test data*, respectively. The test data may relate, for example, to the evaluation of conditions or persistent data required for completing a task. Namely, they determine the execution behaviour of the workload mix (cf. Figure 4.2). In this regards, *probabilistic data generators* are responsible for generating the test data that are needed as input for the process instance's execution (cf. Chapter 7).

The *workload mix* of WfMSs is comprised of *process instances* and *test data* that determine the behaviour of each process instance. Moreover, we define as *workload class* the pair of process model and corresponding intensity, with which each process model participates in a workload mix (i. e., the number of instances of a process model created in a workload mix). In this regard the workload mix is the set of workload classes of a performance test and the execution behaviour of the underlying process models. The underlying process models reflect a diverse applications, ranging from simple to complex processes. To keep the benchmark manageable it is essentially impractical to construct a benchmark that includes any possible scenario. Hence, the workload mix should comprise of a small but meaningful number of representative process models [PFR+15]. Hence, in order to ensure a well-defined and representative workload mix [Fei15], the process models that comprise it should comply with the following characteristics: (i) stress the performance of the WfMS, (ii) contain language features that are supported by the tested WfMS [GHL+15], (iii) be representative of process models used in real world practice and (iv) be in-line with the targeted performance metrics and tests. Therefore, in this work we propose the WINE4WfMSs method to parametrically define the workload mix (cf. Chapter 8). To this end, the user of the benchmark can define different workload mixes for diverse application scenarios, e. g., domain-specific or customer-specific.

To summarise, the load functions are used to create clients that in turn instantiate process models defined in the workload mix. Before executing the benchmark on the SUT we should deploy all the process models of the workload mix on the tested WfMS and make them ready for execution. The process instances are instantiated and executed using the test data which are produced by probabilistic data generators.

4.3. Chapter Summary

In this chapter we focused on the conceptual aspects of a WfMS benchmark. Through a literature study on state-of-art standard middleware and custom WfMS benchmarks we recognised the basic components of a WfMS benchmark and their underlying interactions (cf. Section 4.1). In order to develop a new benchmark or to apply an existing benchmark on a WfMS, practitioners need to comprehend and analyse the design of existing benchmarks. Through this analysis, they are able to learn from historical data and adopt successful practices. As this is a time consuming task, we have gathered the information derived by our analysis into a knowledge base, and offer it as a *Decision Support System for WfMSs Benchmarking*, that is implemented as part of the tool chain presented in Chapter 8. Other historical data and results derived by conformance and performance benchmarking on WfMSs can be found on the “PEaCE” Interactive Dashboard [BMHW16]. The aim of these initiatives is to support practitioners with future decisions concerning the construction of WfMS benchmarks.

In Section 4.2 we identified the basic workload components for benchmarking WfMSs and their underlying interactions. The sophisticated nature of WfMSs requires a flexible definition of the workload model, as the performance testing goals might differentiate from general to domain or customer specific [PFR+15]. Thus, the remainder of this work focuses on defining the methods for deriving a parametric and structurally representative workload mix for WfMSs.

Chapter 5

Micro-benchmarking BPMN 2.0 Workflow Management Systems

"Everything must be made as simple as possible. But not simpler."

A. Einstein

Before defining a complex workload mix for BPMN 2.0 WfMSs information regarding the individual impact of the BPMN 2.0 language constructs on the WfMS performance is required (see RQ-3). As discussed in Chapter 2, the goal of micro-benchmarks is to stress fundamental components of a system, as for example, single operations or narrow aspects of more complex systems [WH13]. For the purposes of our work, we consider a micro-benchmark as suitable means to investigate the performance evaluation of the commonly used BPMN 2.0 modelling constructs. In general, the workflow patterns defined by van der Aalst et al. [VTKB03] can be seen as the minimum set of concepts and constructs that should be implemented by any workflow language [VTKB03]. In our context and given the complexity of the BPMN 2.0 language, we focus on the basic control-flow workflow patterns that apply onto the core of the BPMN 2.0 language. Hence, by using the basic control-flow workflow patterns we implicitly assume that these are the simplest and most frequent atomic operations that a WfMS executes. This chapter is based on the work presented in [SFP+16] and its main contribution is to *introduce the first micro-benchmark for BPMN 2.0 WfMSs process navigator*. The micro-benchmark is based on the sequence flow, exclusive choice and simple merge, explicit termination, parallel split and synchronization, and arbitrary cycle control-flow workflow patterns.

Similar efforts for different systems [MBM09; APDL13; BBD10] have revealed fundamental bottlenecks and have therefore been proven beneficiary to improve the tested systems. The main goal of this work is to enable further research in the performance engineering of BPMN 2.0 WfMSs by examining three state-of-the-art open-source WfMSs and providing a first insight on those BPMN 2.0 language factors that affect the WfMSs performance. More particularly, we aim at answering the research question RQ-3: "What is the impact of the diverse BPMN 2.0 language constructs on the process navigator's performance?".

For this purpose, we implement the selected workflow patterns through specifying a corresponding BPMN 2.0 process model for each one of them. We then enact two sets of experiments to three open-source BPMN 2.0 WfMSs. The workload model of the experiments builds on the conceptual metamodel presented in Section 4.2. The first set of experiments aims to execute a large load of process instances for each workflow pattern and to investigate the behaviour of the WfMSs. The workload mix of each experiment consists of process instances executing only one workflow pattern. The second set of experiments aims to stress the WfMSs during the execution of a uniformly distributed workload mix of all workflow patterns. Thus, the workload mix of these experiments contains process instances of all workflow patterns. We use the benchmarking environment BenchFlow [FIP15] to obtain *raw data* and calculate the *throughput*, *process execution time* and *resource utilisation metrics*. The results show bottlenecks on architectural design decisions, wasteful resource utilisation, and load limits for specific workflow patterns.

To summarise, the main contribution of this chapter is to provide *the first micro benchmark for BPMN 2.0 WfMSs* (see contribution C-3 “The first micro-benchmark for BPMN 2.0 WfMSs”):

(i) analyse the effect of selected core BPMN 2.0 language constructs on the WfMS performance;

(ii) define meaningful candidate constructs for BPMN 2.0 complex benchmarks and

(iii) present findings of the emerging results for the selected WfMSs.

The remainder of this chapter is structured as follows: Section 5.1 presents the workflow patterns that participate in the workload mixes of the experiments and Section 5.2 describes the micro-benchmark experiments. A discussion of the results and most important findings is provided in Section 5.3, in Section 5.4 discusses possible threats to validity and Section 5.5 concludes this chapter.

Additional supplementary material of the raw data and aggregated metrics can be found at: `http://benchflow.inf.usi.ch/results/2015/caise-microbenchmark.tgz`, as well as, at the PEaCE interactive dashboard: `https://peace-project.github.io`.

5.1. BPMN 2.0 Specification of Workflow Patterns

Before proceeding to more complex performance measurements that will also include external interactions, we consider it important to understand the performance behaviour of the WfMS fundamental components. As discussed in Chapter 1, the process navigator of a WfMS is responsible for driving the execution of the tasks of each process instance with respect to the semantics of the underlying process model language [LR00]. Therefore, when external interactions are not taken into account, we consider the process navigator to be mostly accountable for the performance of a WfMS. To this effect and in order to stress the process navigator's performance, we design processes which only make use of script tasks. These processes are fully automated and only use embedded application logic that is co-located with the engine. Thus, the process navigator is the responsible component for their execution. In this regard, we design the process models of the workload mixes and their corresponding behaviours with respect to the following constraints:

(i) we maximise the simplicity of the process model implementing the workflow pattern;

(ii) we drop the interactions with external systems. All tasks are implemented as script tasks, while human tasks and Web service invocations are excluded. This way we stress mainly the process navigator, since script tasks are fully automated and only use embedded application logic that is co-located with the engine;

(iii) we assume that most script tasks are empty. Only the ones required to implement the workflow pattern semantics contain the minimal amount of code and produce the minimum amount of data to do so;

(iv) we assume equal probability of passing the control flow to any outgoing branch of the gateways;

(v) as per the BPMN 2.0 standard [ISO13, p. 434 – 435], the exclusive choice is combined with the simple merge workflow pattern (EXC shown in Figure 5.2) and the parallel split is combined with the synchronisation workflow pattern (PAR shown in Figure 5.3).

For defining the process models of the workload mixes we focus on the basic control flow and structural workflow patterns [WvdAD+06]. The process models designed for our experiments are shown in Figures 5.1 to 5.5. In the rest of this section we shortly present the process models of the micro-benchmark's workload mixes and their behaviour, and define hypotheses concerning their expected performance.

Sequence Flow (SEQ) – This process model executes two consecutive empty script tasks representing the simplest structure with which a process

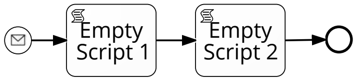


Figure 5.1.: SEQ – Sequence flow workflow pattern [SFP+16]

model may express a sequence of tasks. Due to the structural simplicity of this workflow pattern, we assume that each WfMS will preserve similar execution times throughout the experiments (HYP$_1$).

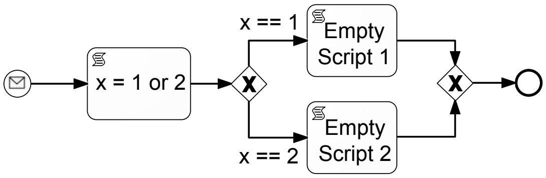


Figure 5.2.: EXC – Exclusive choice and simple merge workflow patterns [SFP+16]

Exclusive Choice and Simple Merge (EXC) – The Exclusive Choice and Simple Merge workflow patterns are implemented in BPMN 2.0 using exclusive gateways and conditional sequence flows (e. g., $x == 1$ and $x == 2$ in Figure 5.2) [ISO13, p. 435]. In the defined process model the first script task randomly generates the number 1 or 2 following a uniform probability, and assigns it to variable x. The generated numbers define if the upper or the lower branch is chosen by the execution flow based on the specified conditions. In both cases an empty script task is executed. The generation of the random number and/or the evaluation of the conditions emerging from the exclusive choice are expected to have an impact on the performance (HYP$_2$).

Parallel Split and Synchronisation (PAR) – The Parallel Split and Synchronisation workflow pattern uses the BPMN 2.0 parallel gateway [ISO13, p. 434]. This process model executes in parallel two empty script tasks. As parallelism is generally demanding in processing power we expect this to reflect on the performance measurements (HYP$_3$).

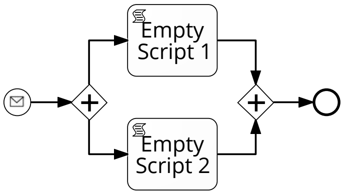


Figure 5.3.: PAR – Parallel split and synchronisation workflow patterns [SFP+16]

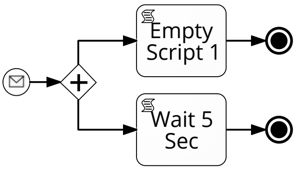


Figure 5.4.: EXT – Explicit termination workflow pattern [SFP+16]

Explicit Termination (EXT) – This process model executes two branches concurrently. According to the BPMN 2.0 language semantics when one of these branches ends it will also terminate the rest of the executing branches and the overall execution of the process instance will be interrupted. Then, the process instance execution is marked as completed successfully. The "Empty Script 1" is an empty script task, while the "Wait 5 Sec" task waits for five seconds. The value of five seconds was chosen to guarantee that the lower branch will be slower than the upper one. As the "Empty Script 1" task is the fastest, we expect that the process instance will be completed with the completion of the path containing "Empty Script 1", and then the terminate event will interrupt the "Wait 5 Sec" task. The structures of the PAR and EXT process models are alike, although they represent different concepts and workflow patterns. We assume that the concurrent execution of tasks will impact the performance in a similar manner (HYP_4).

Arbitrary Cycle (CYC) – Arbitrary cycles are not expressed through any specific, individual BPMN 2.0 construct but through a combination of ex-

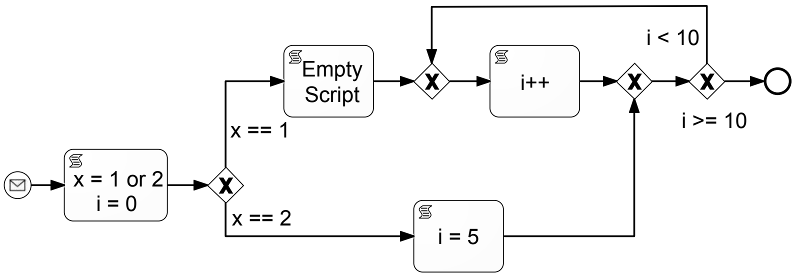


Figure 5.5.: CYC – Arbitrary cycle workflow pattern [SFP+16]

clusive gateways that form a cyclic structure that has at least two entries or two exits [VTKB03]. The CYC workflow pattern is implemented with two entry points at the second exclusive gateway, and starts by a script task that randomly generates the integer number $x = 1$ or $x = 2$ following a uniform probability and initializes a variable $i = 0$. With respect to the value of x variable the upper or the lower branches are followed (like the EXC workflow pattern shown in Figure 5.2). The lower branch executes a script task that assigns value 5 to variable i (script task"$i = 5$" in Figure 5.5), while the upper branch executes an empty script task (script task "Empty Script" task in Figure 5.5) and then increases the variable i (script task "$i++$" in Figure 5.5). This path will be followed until variable $i == 10$. To have a different but deterministic behaviour of the branches we implemented the script task of the lower branch to set $i = 5$ (as shown in Figure 5.5). In this case the loop (cycle) will be repeated fewer times, until $i == 10$. The CYC workflow pattern represents a slightly more complex structure than the aforementioned process models. Therefore, this workflow pattern might be more appropriate for revealing performance bottlenecks due to the usage of sequential or nested exclusive gateways (HYP_5).

5.2. Experiments

For the micro-benchmark we define the following six workload mixes:

Workload Mixes 1-5 issue a large load to the WfMSs and investigate their behaviour for individual workflow patterns (SEQ, EXC, EXT, PAR, CYC). Thus, each workload mix consists of workload classes that correspond to exactly one of the aforementioned workflow patterns. Consequently, each class constitutes of a workflow pattern participating with 100% intensity in the corresponding workload mix.

Workload Mix 6: emulates the performance behaviour during the concurrent execution of diverse workflow patterns (MIX). More particularly, we test if the performance of a workflow pattern is affected when it is executed concurrently with other types of workflow patterns. For this purpose, we inject a mix of five workload classes, each one containing instances of a workflow pattern distributed with a uniform intensity of 20% for each.

All workload mixes are executed three times for each WfMS to verify that there is a comparable behaviour between the executions. In some cases a WfMS did not sustain the predefined load. Then we re-execute the experiments with a lower load and observe the WfMS behaviour for this execution.

Following the work of Ferme et al. [FIP15; FIP16] who define meaningful metrics for benchmarking WfMSs, for each workload mix we calculate the following metrics:

Process Instance Duration: Time difference between the start and the completion of a process instance. In our experiments the duration is calculated in seconds (sec).

Resource Utilisation: CPU utilisation (calculated in percentage (%)) and memory utilisation calculated in (megabyte (MB)).

# Process Instances: The absolute number of executed process instances by the WfMS per benchmark run. Process instances are also abbreviated as pi.

Throughput: The number of executed process instances per time unit. Throughput is calculated with Equation (5.1):

$$Throughput = \frac{\#ProcessInstances}{Time} \tag{5.1}$$

The resulting throughput unit is $\#pi/s$.

Details about the experiments set up as well as the identified load functions are discussed in detail in [SFP+16].

5.3. Evaluation

The experiments described in Section 5.2 are executed against three WfMSs. Since, out of the three benchmarked WfMSs only *Camunda* [Cam13] consented to the publication of the results, for the rest of this chapter the other two WfMSs are referred to as *WfMS A* and *WfMS B*. In the following we summarise the results for each workload mix and discuss the most important findings. A more detailed analysis of the results is available in [SFP+16].

For each workload mix and each WfMS, we present the duration in milliseconds (ms, Figures 5.6 and 5.9), the CPU utilisation in percentage (%, Figure 5.7), and the mean amount of RAM that was allocated by the WfMS in megabytes (MB, Figure 5.8). For the duration of the experiments for each workload mix and for each WfMS we calculate also relevant statistics [MR03] shown in Table 5.1. More specifically, Table 5.1 presents the mean with Confidence Interval (CI) 95%, mode, min, max, Standard Deviation (Sd) and quartiles (Q1, Q2, Q3) of the average data collected during the maximum load ($\#clients$) each WfMS could sustain. Sometimes the WfMS failed to handle the maximum load (i. e., $1,500$ concurrent clients), leading to the execution of new experiments with reduced load. These cases are also presented in Table 5.1. Table 5.1 also provides the aggregated metrics (see Section 5.2) regarding (i) the total number of completed process instances for each WfMS, (ii) the total duration (seconds) for each experiment, and (iii) the average throughput expressed in completed process instances per second ($\#pi/s$).

Workload Mix 1 (Sequence Flow Workflow Pattern, SEQ) – As seen in Figure 5.6 this workload mix has resulted in the shortest duration times for all WfMSs. The short duration of this workload mix is also followed by low average CPU usage for all three WfMSs (Figure 5.7). Moreover, this

		Process Instance Execution Duration Statistics (ms)								Experiment Execution Statistics			
		Mean & 95% CI	Mode	Min	Max	Sd	Q1	Q2	Q3	Max Load (# clients)	#Process Instance (# pi)	Experiment Duration (sec)	Throughput (# pi/s)
SEQ	*A.*	0.39 ± 0.01	0	0	561	1.70	0	0	1	1,500	781,736	540	1,447.66
	B.	6.39 ± 0.43	6	4	82	1.21	6	6	7	1,500	35,516	561	63.31
	C.	0.74 ± 0.01	1	0	682	2.29	0	1	1	1,500	786,664	540	1,456.79
EXC	*A.*	0.48 ± 0.01	0	0	485	2.07	0	0	1	1,500	775,455	540	1,436.03
	B.	9.30 ± 0.05	9	6	131	2.11	9	9	10	1,500	27,805	567	49.04
	C.	0.85 ± 0.01	1	0	627	2.51	0	1	1	1,500	765,274	540	1,417.17
EXT	*A.*	14.10 ± 0.06	10	5	858	13.45	10	11	14	1,500	770,229	540	1,426.35
	B.	2,622.00± 237.68	11	8	5,047	2,500.44	13	5,012	5,016	1,500	1,703	4,498	0.38
	C.	0.40 ± 0.01	0	0	74	1.03	0	0	1	1,500	784,614	539	1,455.68
PAR	*A.*	13.29 ± 0.06	8	4	456	11.99	9	10	13	1,500	772,013	540	1,429.65
	B.	10.06 ± 0.06	10	7	145	2.22	9	10	10	1,500	27,718	567	48.89
	C.	0.70 ± 0.01	1	0	691	2.10	0	1	1	1,500	773,883	540	1,433.12
CYC	*A.*	6.23 ± 0.13	2	0	478	18.68	1	2	3	800	347,770	540	644.02
	B.	39.36 ± 0.40	50	25	146	9.52	30	43	47	1,500	8,695	646	13.46
	C.	3.06 ± 0.04	2	0	353	4.43	2	2	3	600	177,770	542	327.99
MIX	*A.*	8.16 ± 0.07	0	0	663	14.65	1	2	12	1,500	758,659	541	1,402.33
	B.	540.02 ± 122.3	11	6	5,195	1,525.27	10	12	38	1,500	2,392	1,343	1.78
	C.	1.22 ± 0.02	0	0	434	4.21	0	1	1	1,500	575,210	542	1,061.27

WfMS A: *A.*,WfMS B: *B.*,Camunda: *C.*

Table 5.1.: Process instance duration and experiment execution statistics [SFP+16]

workload mix resulted in the highest throughput for all WfMSs under test (see Table 5.1).

Workload Mix 2 (Exclusive Choice & Simple Merge Workflow Patterns, EXC) – The random number generation and the evaluation of conditions implemented by the process models of this workload mix, seem to have no important impact on duration, as Figure 5.6 shows that the resulting duration values are similar to those of the SEQ workflow pattern. Concerning the CPU (Figure 5.7) and RAM (Figure 5.8) utilisation, we observe a slight increase when compared to the SEQ workflow pattern.

Workload Mix 3 (Explicit Termination Workflow Pattern, EXT) – As discussed in Section 5.1, the EXT workflow pattern executes concurrently an

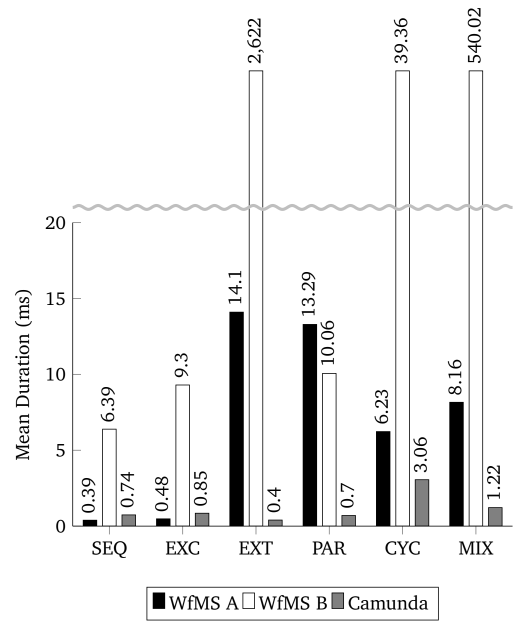


Figure 5.6.: Mean duration (ms) per workflow pattern [SFP+16]

empty script and a script that implements a five seconds wait. According to the BPMN 2.0 execution semantics, the branch of EXT that finishes first terminates the rest of process instance's running branches. Considering this, we implemented the EXT workflow pattern so that the fastest branch (empty script) will complete first, interrupt the slower script on the other branch and terminate the process instance execution. The execution behaviours of WfMS A and Camunda were compliant with the aforementioned design. The resource utilisation of these two WfMSs increased for this workflow pattern (cf. Figures 5.7 and 5.8). At this point, we also observe a notable difference on the performance of the two WfMSs as EXT constitutes the slowest workflow pattern for WfMS A and the fastest for Camunda.

The duration results of WfMS B for the EXT workflow pattern are disproportionally high when compared to those of WfMS A and Camunda. We

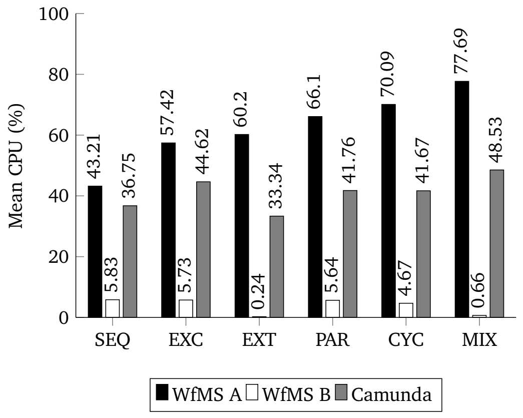


Figure 5.7.: Mean CPU (%) usage per workflow pattern [SFP+16]

thoroughly examined this situation and observed that during the process instance executions WfMS B follows a pseudo-parallel (sequential) execution of each path, with an average percentage of 52.23% of executing the waiting script first and 47.77% for executing the empty script first. Since the completion of the waiting script lasts for five seconds, every time it is chosen for execution it adds a five seconds overhead and leads to a very high average duration time. Later in this section, we endeavour to explain in more detail this behaviour of WfMS B. With regards to the resource utilisation we notice a very low average usage of CPU (cf. Figure 5.7) and a mean RAM usage (cf. Figure 5.8) similar to the rest of the workflow mixes.

Workload Mix 4 (Parallel Split & Synchronization Workflow Patterns, PAR) – There is an increase in the duration times for WfMS A and WfMS B, while Camunda handles parallelism much faster (cf. Figure 5.6). Although WfMS B seems faster by looking the duration results, we should take into consideration that it has a total execution of 27, 718 process instances in 567 seconds, while WfMS A executed 772, 013 process instances in 540 seconds. For both WfMS B and Camunda keep the resource utilisation values in the same range to this resulting from the execution of the rest of workload mixes (cf. Figures 5.7 and 5.8). For WfMS A the values of utilised resources are

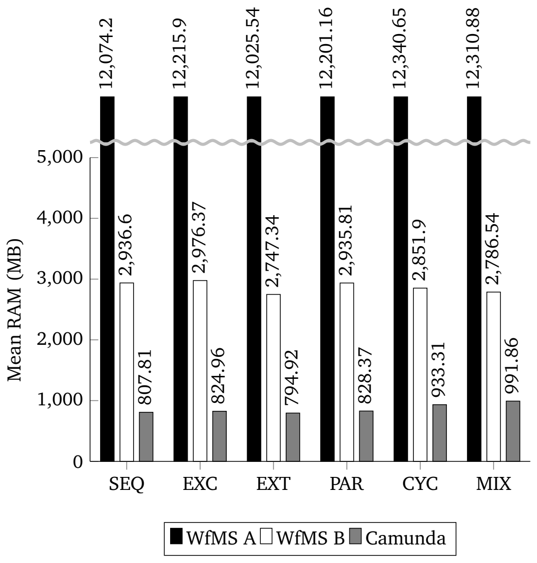


Figure 5.8.: Mean RAM (MB) usage per workflow pattern [SFP+16]

relatively higher than these obtained from the rest of workflow patterns (cf. Figures 5.7 and 5.8).

Workload Mix 5 (Arbitrary Cycle Workflow Pattern, CYC) – The performance of this workload mix cannot be directly compared to workload mixes 1 – 4, as it implements a higher number of language constructs and composes a more complex process model structure. The number generation and the script to increase the variable value (see Figure 5.5) implemented by the CYC workflow patterns are expected to introduce an additional overhead on the performance. Moreover, the duration of this workflow pattern is dependent on the generated number as it determines whether 10 or 5 cycles will be executed. During the execution of this workload mix, Camunda showed connection timeout errors for a load greater than 600 clients, with a connection timeout limit set on 20 seconds. Hence, we reduced the load to 600 clients and repeated the experiments for the other two WfMSs. Thus, the load for the results shown in Figure 5.6, Figure 5.7 and Figure 5.8 for this workload

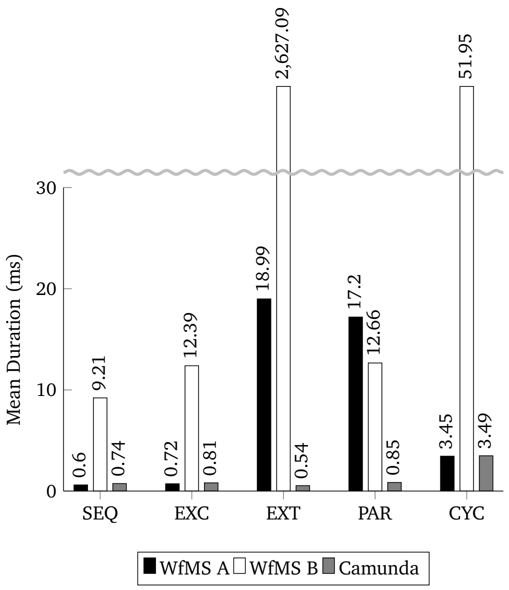


Figure 5.9.: Mean duration (ms) per workflow pattern in MIX [SFP+16]

mix correspond to 600 clients. On the contrary, Table 5.1 shows the results for the maximum load each WfMS could sustain, i. e., 800 clients for WfMS A, 1500 for WfMS B and 600 for Camunda. The mean CYC execution duration is higher than the rest of the workload mixes (cf. Figure 5.6). Concerning resource utilisation, WfMS B and Camunda remain stable and comparable to the rest of the workload mixes (cf. Figures 5.7 and 5.8). WfMS B remains also on the same range of mean RAM usage, while we observe an increase for Camunda (cf. Figure 5.8). In this workload mix WfMS A also demonstrated an increased resource utilisation (cf. Figures 5.7 and 5.8).

Workload Mix 6 (MIX) – This workload mix instantiates simultaneously all workflow patterns (see Figures 5.1 to 5.5) with an intensity of 20% for each participating workflow pattern. Figure 5.9 presents the individual duration times of the workflow patterns during their concurrent, uniform execution on the MIX workload mix. As seen in Figure 5.9, all workflow patterns have little increase in their duration times when compared to their execution in the individual workload mixes. Nevertheless, a quick overview of the MIX statistics indicates that they are the aggregated mean duration times of the individual workflow patterns (cf. Figure 5.9). The throughput of the MIX workload mix is slightly lower for all WfMSs, although WfMS A maintained it on the same range as its previous values (cf. Table 5.1).

In the rest of this section we discuss the most important findings emerging from the executed experiments. As already discussed, WfMS B showcased some peculiarity in its behaviour. This was also noticed by Bianculli et al. [BBD10] who executed performance experiments on a WS-BPEL version of WfMS B. According to the documentation of WfMS B calls to the execution server through its Representational State Transfer (REST) Application Programming Interface (API) are blocked until the process instance completes its execution. In our experiment the clients request the instantiation of a process model, and wait for the completion of the instantiation before creating a new one. However, in the case of WfMS B the clients wait for the entire execution of a process instance to finish, before requesting the instantiation of a new one. This fact introduces a high overhead on the performance, and causes delays that deteriorate the performance.

In order to further investigate this, we conducted a scalability test to analyse the WfMS behaviour under diverse load intensity levels. As discussed in [SFP+16] the throughput remained stable even if the clients increased. This fact indicates that: (i) it is pointless to increase the number of clients and target to the execution of more process instances and (ii) the fact that WfMS B is the only WfMS under test using a synchronous REST API does not impact the comparability of the results.

Another emerging issue for discussion concerning WfMS B is its inconsistent behaviour during the execution of workload mix 3 (EXT). According to the expected execution of the EXT workflow pattern the path with the empty script interrupts the execution of the path with 5 seconds wait and the process instance execution completes gracefully. However, according to its documentation, WfMS B dedicates a single thread to the parallel execution of the scripts, leading to a pseudo-parallel (serial) execution of the parallel paths. Consequently, data evidence that in about 50% of the cases the fast execution path (empty script) is chosen to be executed first, while in the remaining of the executions the longer execution path (5 seconds waiting script) was executed first. In the cases where the longer execution path is chosen the execution needs to wait for 5 seconds until this branch is completed. This explains the very high duration of the EXT as half of the executions lasted for 5 seconds.

We proceed by drawing some conclusions regarding the overall performance of all WfMSs under test. The fact that WfMS B executed sequentially all the initiated process instances leads to a higher response time and a lower actual executed load with respect to the other two WfMSs. This behaviour is also directly reflected by the results in which WfMS B demonstrates in higher duration values for all workflow patterns and much lower CPU and memory utilisation, which is an immediate consequence of the low throughput. Camunda resembles WfMS A architecturally as Camunda was originally a fork of WfMS A. However, as evidenced by the executed experiments the behaviour of the two WfMSs is not identical and leads to some interesting conclusions. Although WfMS A executed slightly better than Camunda for the SEQ and EXC workflow patterns, Camunda also kept the duration values low for all tested workload mixes. WfMS A seems to be impacted by parallelism and resulted in high duration values and resource utilisation for all workload mixes that included parallelism (i. e., EXT, PAR, and MIX). On the contrary, Camunda performed stable for the workload mixes with parallelism, thus we may safely conclude that the performance of Camunda is not affected by parallelism. During the execution of the MIX workload mix WfMS A kept the same throughput and executed a rather constant number of process instances as those executed for each individual workflow pattern workload mix. Concerning the derived results on the resource utilisation WfMS B and Camunda demonstrated a more stable behaviour, while WfMS A showed an increase when it was more stressed. In general, we may conclude that Camunda performed better and more stable for all metrics when compared to WfMS A and WfMS B.

Last but not least, regarding the formulated hypotheses points (see Section 5.1), all WfMSs performed similarly good and stable in the execution of the SEQ workload mix (cf. HYP_1). SEQ had also the highest throughput for all tested WfMSs. As speculated in HYP_2, the generation of a random number and/or the condition evaluation seem to have an impact on the performance (see EXC in Section 5.1). Regarding parallelism, the hypothesis that the PAR and EXT workflow patterns will have a similar impact on the performance (HYP_3, HYP_4) hold primarily for WfMS A and Camunda. Our HYP_4 and HYP_5 for parallelism and complex structures having an impact on the performance seems to hold for WfMS A, while for Camunda no conclusions can be drawn with respect to this point.

Overall, the results designate that sequential workflows (i. e., SEQ) may be used for discovering the maximum throughput of the WfMSs. Process models using parallelism (i. e., PAR and EXT) are more suitable for drawing conclusions regarding the throughput and resource utilisation, while more complex structures (i. e., CYC) are better candidates for stressing the WfMSs in terms of resource utilisation. Finally, the concurrent execution of different process models does not seem to have an important impact on the WfMS performance.

These conclusions are considered in Chapter 9, in which we design a more complex and representative workload mix for a WfMS macro-benchmark.

5.4. Threats to Validity

External validity refers to the extend to which the experimental results can be generalised to other situations [WR+00]. The small number of tested WfMSs leads to an uncertainty degree on the drawn conclusions. For this reason, it constitutes a thread to external validity. For safer deductions, the evaluation of more WfMSs is needed.

Construct validity describes the extend to which the experiments measure what they claim to or should be measuring [WR+00]. The fact that the conducted performance tests utilise simple workload models constitutes a threat to the construct validity. Notwithstanding the micro-benchmark by construction does not correspond to real world practice conditions, we consider it fundamental for the purposes of our work. By exploiting the derived knowledge and using it as a basis we may afterwards proceed to the definition of more complex workload mixes (cf. Chapter 9).

Another threat to the external and construct validity is that our load driver could not generate more than 1,500 concurrent clients. This number does not correspond to real world practice situations, where more clients appear in practice. Nevertheless, the issued load can be compared to this of previously applied custom WfMS benchmarks (e. g., [DES08]).

The BenchFlow environment [FIP15; FIP+16] does not model the network latency, and this parameter is ignored during our tests. It also uses different dedicated networks for the interactions of the WfMS with the DBMS and of the clients with the WfMS [FIP15; FIP+16]. This is a potential threat to external and construct validity. To ensure conclusion validity, we used multiple statistical metrics like absolute counts, means, medians, and standard deviations (Table 5.1).

5.5. Chapter Summary

In this chapter we showcased a micro-benchmark for BPMN 2.0 WfMSs. The micro-benchmark is based on the BPMN 2.0 representation of basic control flow patterns, which are reoccurring concepts and structures shared by any workflow language. To the extent of our knowledge this is the first attempt to investigate the impact of BPMN 2.0 language constructs on the WfMSs performance. For the execution of the benchmark we used the BenchFlow benchmarking environment [FIP15], on which we issued a load of more than one thousand clients initiating process instance executions. Among other interesting observations, our results revealed important bottlenecks due to architectural design decisions for WfMS B, that resource utilisation can be a potential issue for WfMS A and load bottlenecks during the execution of the arbitrary cycle workload mix for Camunda. Consequently, despite the simplicity of the micro-benchmark we argue that it is a potentially suitable choice for benchmarking fundamental behaviour of complex WfMS executing in real world practice.

Regarding individual workflow patterns we observed that the sequential workflow pattern revealed the maximum throughput for all of the examined WfMSs. Parallelism (i. e., explicit termination and parallel pattern) affected two of the three WfMSs in terms of throughput and resource utilisation. More complex structures, such as arbitrary cycles, also seem to impact the resource utilisation, thus they can be better candidates for stressing the WfMS. Finally, the mix execution helped us to conclude that there are no adverse performance effects when executing different workflow patterns concurrently. However, we did observe a slight increase on individual performance metrics when compared to the homogeneous experiments with individual patterns. The above results provide the first insights on which constructs constitute meaningful candidates for building more complex benchmarks. For instance, a test aiming to measure the throughput or resource utilisation of the WfMS should preferably choose complex, parallel structures. In Chapter 9 we exploit the conclusions of this work to define a more complex and representative workload mix that is afterwards used for macro-benchmarking BPMN 2.0 WfMS.

Chapter 6

Reoccurring Structures Detection (RoSE)

"A problem well put is half-solved."

J. Dewey

Business process model similarities is a topic frequently discussed in the literature [Dij+13; BDDS14] (see also Section 2.2.4 for a more extensive discussion on the subject). It finds application in multiple scenarios, as for example to facilitate the comparison or integration of business processes, to inspect the validity of process models, to adjust the process models to different target groups [BDDS14]; to detect and refactor duplicates (clones) in process model repositories [Dum+13; EDG+12]; to detect different versions of the same process model [PW12] or to generate process model collections [YDG15]. In Chapter 3, we described a collection of *12,624* BPMN 2.0 business process models used in real world practice. These can be analysed for deriving representative process models that will then be used for the workload mix. Given the existence of anonymised and reference models in the original collection, our analysis must therefore *exploit solely the structural information of the process models* (Objective 1). Hence, we need to apply a structural analysis on the collection and determine reoccurring structures. This chapter aims to address research question RQ-5: “How to detect reoccurring BPMN 2.0 structures in a process models collection?”. Objective 1 can generally be reduced to the very well known challenge of subgraph isomorphism [Ull76], which is discussed as NP-complete in the literature [GN95]. However, as well-structured BPMN 2.0 process models are special types of graphs, i. e., Series Parallel Graphs [GR15], the challenge of subgraph isomorphism can be solved in polynomial time [GN95]. The reduction to polynomial complexity indicates the *feasibility* of developing and applying a technique that targets BPMN 2.0 process models.

Although many existing approaches are solving the subgraph isomorphism problem [Ull76; CFSV04; GN95; VM97], they assume a given graph whose occurrence is searched in a larger graph. In our case, however, *there is no known subgraph to be searched for. Instead, we need to identify and extract it based on the case of structural similarity between a set of business process models* (Objective 2). To this end, and in order to target both of the defined objectives, we are reaching the challenge of *frequent pattern discovery without candidate generation* which can be seen as a variant of the subgraph isomorphism problem [YH02]. Breuker et al. [BDDS14] have applied an extensive research on the performance of publicly available algorithms for frequent pattern matching without candidate generation. More particularly, Breuker et al. [BDDS14] conducted experiments for the gSPAN [YH02] and Gaston [NK05] algorithms to discover frequently occurring patterns in a collection of thousands of EPC process models. In the experiments, the textual semantics were omitted and the comparison is applied only with respect to the structural semantics of the process models (cf. Objective 1). In this case, the authors reported quick failure of both algorithms when applied to a collection of 2,200 process models.

We exploit, adapt and combine well established techniques to propose a novel approach for the efficient detection of reoccurring structural patterns in a process models collection. The proposed approach focuses on

BPMN 2.0, however it may be generalized to any process modelling language, by adapting the proposed formal model. More particularly, the contributions of this chapter are to bring together and extend previous work presented in [SGHL15; SL15; SAKL16] and to introduce:

(i) a formal model of our approach;

(ii) a novel algorithm for detecting and extracting reoccurring structures (i. e., structural similarities) from a collection of BPMN 2.0 process models and calculating metadata regarding their frequency of appearance; and

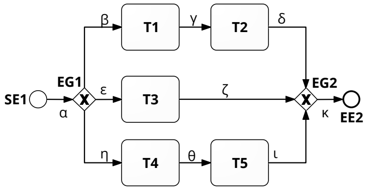
(iii) the proof of completeness of the aforementioned algorithm.

The above contributions form contribution C-5: "The Reoccurring Structures Detection (RoSE) method".

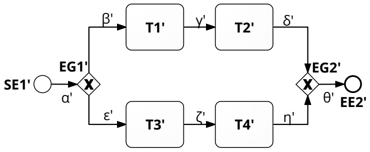
The remainder of this chapter is structured as follows: Section 6.1 provides a representative example of reoccurring structures detection between two process models; Section 6.2 presents the formal model of our approach; Section 6.3 explains the algorithm that detects and extracts the reoccurring structures from the process models collection and presents the proof of completeness of the aforementioned algorithm; and Section 6.4 summarises the contributions of the chapter and presents an outlook for future work.

6.1. Representative Example

For detecting structural similarities, our approach focuses on *acyclic* BPMN 2.0 process models and handles them as special types of graphs. Textual or behavioural semantics of the process models are not used in our approach. Figure 6.1 shows two examples of BPMN 2.0 process models for which we detect the reoccurring structures. The presented process models are very similar to each other and a quick observation leads to the detection of many reoccurring structures. Figure 6.2 shows some of such reoccurring structures in the process models of Figures 6.1a and 6.1b. The demonstrated structures in Figure 6.2 are only some of the possible results and not the

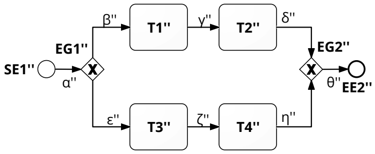


(a) Model A

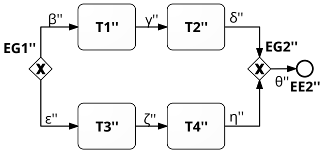


(b) Model B

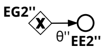
Figure 6.1.: BPMN 2.0 example process models [SAKL16]



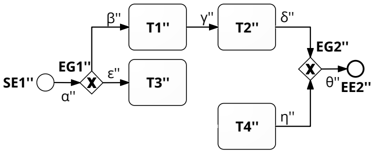
(a) Reoccurring Structure 1



(b) Reoccurring Structure 2



(c) Reoccurring Structure 3



(d) Reoccurring Structure 4

Figure 6.2.: Reoccurring structures in the process models of Figure 6.1 [SAKL16]

complete set of reoccurring structures between these two process models. Focusing on the reoccurring structures shown in Figure 6.2a and Figure 6.2b we observe that the only difference between these two structures is their starting point. Namely, the structure of Figure 6.2a begins with a start event, while the structure of Figure 6.2b with an exclusive gateway. As seen in Figure 6.2a and Figure 6.2b, both of these structures constitute maximal common reoccurring structures between the compared models that stem from the corresponding participating starting points (i. e., *SE1* and *SE1*$'$ matched to *SE1*$''$ and *EG1*, and *EG1*$'$ matched to *EG1*$''$ respectively). In other words, there are no larger common structures between models A and B in Figure 6.1 stemming from these starting points. Moreover, the elements of the structures in Figure 6.2a and Figure 6.2b can be reached through the starting points *SE1″* and *EG1″* by any common graph traversal algorithm (e. g., DFS). For the discovered structures we also calculate statistics metadata regarding their frequency of appearance. In particular, we calculate: (i) how many times a structure has been detected in a collection and (ii) in how many models the reoccurring structure appears. With respect to these metadata both the structures shown in Figure 6.2a and Figure 6.2b appear one time in the collection of Figure 6.1 and in both (i. e., two) process models.

In order to satisfy the purposes of our approach not all possible reoccurring structures are to be considered. For example, the structure of Figure 6.2c is comprised of two BPMN 2.0 elements connected with each other, and appears in both models of Figure 6.1. Since this structure is a minimalistic building block for any BPMN 2.0 model we consider it of no particular structural interest and exclude it from our results. Another more complex structure that is not considered by our approach is shown in Figure 6.2d. In this case, all the BPMN 2.0 elements of this structure can be mapped to exactly one BPMN 2.0 element of either Figure 6.2a or Figure 6.2b. However, the structure cannot exactly be characterized as reoccurring, because not every element of the structure in Figure 6.2d is connected to a starting point. For the purposes of this work we focus only on identifying the *non-trivial, reoccurring structures of maximum size* which contain *exactly one starting point*, out of which we can reach *any other element* of the process model by any common graph traversal algorithm. An indirect consequence of this design decision is that BPMN 2.0 collaboration elements are not supported by our approach, since this would demand more than one starting points.

6.2. Formal Model

A BPMN 2.0 model can be seen as a directed, attributed graph $G = (\boldsymbol{V}, \boldsymbol{E})$ [GR15] with $n = |V|$ vertices (denoting the activities) and $m = |\boldsymbol{E}|$ edges (denoting the sequence flow connectors). For this work, we assume the sets $\boldsymbol{V}$ and $\boldsymbol{E}$ to be partially ordered sets ($(\boldsymbol{V}, \leq), (\boldsymbol{E}, \leq)$), where the partial order is defined by the order in which the vertices and edges would be visited by a DFS traversal. The set $\boldsymbol{L_a}$ contains element types from the BPMN 2.0 set as described by the BPMN 2.0 ISO standard [ISO13]. *Type* of a vertex v_i is called the function *type* : $\boldsymbol{V} \rightarrow \boldsymbol{L_a}$ that assigns an element from a set $\boldsymbol{L_a}$ to each $v_i \in \boldsymbol{V}$.

Zur Muehlen and Recker [MR08] showed that most of the BPMN 2.0 elements are not actually applied in practice. Throughout the years BPM has matured, leading to the adoption of more features by programmers and IT specialists. Nevertheless, business process modellers still seem reluctant to apply more complex constructs and apply only a subset of the language [All16]. As we will show in Chapter 9 our collection (Chapter 3) also reflects a subset of the BPMN 2.0 language. Moreover, a preliminary statistical analysis of our collection has indicated that approximately a 12% of the contained process models contain cyclic structures. Hence, since our approach targets the detection of *the most commonly reoccurring structures* of a BPMN 2.0 collection, the proposed approach supports the most commonly discovered BPMN 2.0 elements [MR08], while it does not support cyclic graphs. More particularly, we consider the elements of the set $\boldsymbol{L_a} = \boldsymbol{Evt} \cup \boldsymbol{Gt} \cup \boldsymbol{Act}$ only, where:

Evt: is the set of all event types as defined in the BPMN 2.0 standard (e. g., start event, end event, etc.)

Gt: is the set of all gateways types in the BPMN 2.0 standard (e. g., parallel gateway, exclusive gateway, complex gateway, etc.)

Act: is the set of all task types (e. g., manual task, script task, service task, etc.) in the BPMN 2.0 standard, call activities and subprocesses.

The concept of checkpoints introduces *areas for investigation* in a process model, as the $\boldsymbol{Evt}$ and $\boldsymbol{Gt}$ elements of BPMN 2.0 are those differentiating the models structurally. In other words, without event ($\boldsymbol{Evt}$) and gateway ($\boldsymbol{Gt}$) elements, process models consist only of activity ($\boldsymbol{Act}$) elements and this is not of any particular structural interest.

Definition 1 (Checkpoints)
Let $\boldsymbol{L_{ch}} = \boldsymbol{L_a} \setminus \boldsymbol{Act}$ be the set of all BPMN 2.0 element types excluding all activity types. We define as checkpoints *the set of vertices of graph G, $\boldsymbol{V}_{ch}^{G} \subset \boldsymbol{V}$ for which it holds $\boldsymbol{V}_{ch}^{G} = \{v | type(v) \in \boldsymbol{L_{ch}}\}$.*

For example, the checkpoints of the models presented in Figure 6.1 are vertices $\boldsymbol{V}_{ch}^{ModelA} = \{SE1, EG1, EG2, EE2\}$ and $\boldsymbol{V}_{ch}^{ModelB} = \{SE1', EG1', EG2', EE2'\}$.

Following on, and for any directed edge $e = (u, v) \in \boldsymbol{E}$, we say that e is *outgoing* from u and *incoming* to v. Likewise, we call u the *source vertex* of e and v the *target vertex* of e. The functions $incoming : \boldsymbol{V} \rightarrow \wp(\boldsymbol{E})$ and $outgoing : \boldsymbol{V} \rightarrow \wp(\boldsymbol{E})$ are defined accordingly. A vertex $v \in \boldsymbol{V}$ is called a *source* when $incoming(v) = \emptyset$ and a *sink* when $outgoing(v) = \emptyset$. In our case we assume that any graph has a unique source, i. e., $|\{v \in \boldsymbol{V} \,|\, incoming(v) = \emptyset\}| = 1$. In the following, we denote the source vertex of a graph G as v_{src}^{G}.

Definition 2 (Path)
A path of length $k, k \in \mathbb{N}^{+}$ *from a vertex u to a vertex w, denoted as $u \mapsto w$, is a sequence of edges $e_1, ..., e_k \in \boldsymbol{E}$ for which it holds:*

1. *The target vertex of each edge is the source of its succeeding edge.*
2. *The edges within a path are pairwise distinct ($e_i \neq e_i$ for $i \neq j$).*
3. *The source vertex of the first edge of the path (e_1) is the startpoint of the path (u), and the target vertex of the last edge of the path (e_k) is the endpoint of the path (w).*

For example, the sequence of edges $(\alpha, \epsilon, \zeta, \kappa)$ is a path of length 4, from vertex *SE1* to vertex *EE2* in the process model of Figure 6.1a. The existence of a path $u \mapsto w$ denotes *reachability* from u to w with any common graph traversal algorithm. Hence, through determining the reachability of a process model's paths, we are able to identify if a process model is source connected.

Definition 3 (Source connectivity)
A graph G is called source connected *if there exists a path from its source v^G_{src} to any other vertex $v \in \boldsymbol{V}$.*

For example, the graph presented in Figure 6.2d is not source connected, because the vertex *T4''* cannot be reached from the source vertex *SE1''*.

Definition 4 (Checkpoint-subgraph)
We call a source connected subgraph H of G a checkpoint-subgraph *iff $v^H_{src} \in \boldsymbol{V}^G_{ch}$. The function $\mu : \boldsymbol{V}^G_{ch} \rightarrow \boldsymbol{C}^G_{ch}$ maps a checkpoint of a graph G to the* set *$\boldsymbol{C}^G_{ch}$ of* all *checkpoint-subgraphs of G starting at that checkpoint.*

It can be seen that all subgraphs shown in Figure 6.2, except for Figure 6.2d, are a subset of the checkpoint-subgraphs of the models shown in Figure 6.1.

Moving now to the definitions of *graph and subgraph isomorphism* for our formal model we are using these provided by Valiente [Val02].

Definition 5 (Graph isomorphism [Val02])
Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic *if there is a bijection $M \subseteq \boldsymbol{V}_1 \times \boldsymbol{V}_2$ such that, for every pair of vertices $(v_i, v_j) \in \boldsymbol{V}_1$ and $\{w_i, w_j\} \in \boldsymbol{V}_2$ with $(v_i, w_i) \in M$ and $(v_j, w_j) \in M$, $(v_i, v_j) \in \boldsymbol{E}_1 \iff (w_i, w_j) \in \boldsymbol{E}_2$. In such a case, M is a graph isomorphism of G_1 to G_2 and we denote it as $G_1 \sim G_2$.*

Definition 6 (Subgraph isomorphism [Val02])
A subgraph isomorphism *of the graph $G_1 = (V_1, E_1)$ into a graph $G_2 = (V_2, E_2)$ is an injection $M \subseteq \boldsymbol{V}_1 \times \boldsymbol{V}_2$ such that, for every pair of vertices $(v_i, v_j) \in \boldsymbol{V}_1$ and $\{w_i, w_j\} \in \boldsymbol{V}_2$ with $(v_i, w_i) \in M$ and $(v_j, w_j) \in M$, $(w_i, w_j) \in \boldsymbol{E}_2$ if $(v_i, v_j) \in \boldsymbol{E}_1$. In such a case, M is a subgraph isomorphism of G_1 into G_2 and we denote $G_1 \cong G_2$.*

In the following we adjust the definitions of *common subgraph isomorphism* and *maximum common subgraph isomorphism* provided by Fernández et al. [FV01] to be consistent with the formal model of our approach.

Definition 7 (Common Subgraph Isomorphism (CSI))
A graph C is a common subgraph isomorphism of two graphs G_1 and G_2 if there exist subgraphs $C_1 \subset G_1$ and $C_2 \subset G_2$ such that $C \cong C_1 \cong C_2$.

In other words, a subgraph isomorphism of two graphs is a CSI when its structure has an one-to-one mapping in both graphs. For example, the graphs in Figures 6.2a to 6.2d constitute different common subgraph isomorphisms of the two graphs G_1 (Model A) and G_2 (Model B) presented in Figure 6.1. In principle, we are interested in the largest possible graph that stems from a specific starting point, so for this purpose we define:

Definition 8 (Maximum Common Subgraph Isomorphism (MCSI))
A CSI $C = (\boldsymbol{V_C}, \boldsymbol{E_C})$ of two graphs G_1 and G_2 is a maximum common subgraph isomorphism *iff $\nexists\, C' = (\boldsymbol{V_{C'}}, \boldsymbol{E_{C'}}) \triangleq (G_1, G_2)$ such that $|\boldsymbol{V_{C'}}| > |\boldsymbol{V_C}|$.*

It therefore follows that the graph shown in Figure 6.2a is the MCSI of the models shown in Figure 6.1. Bringing now the above definitions together, we have:

Definition 9 (Common checkpoint-subgraph)
If two graphs $G_{ch1} \in \boldsymbol{C}_{ch}^{G_1}$ and $G_{ch2} \in \boldsymbol{C}_{ch}^{G_2}$ are checkpoint-subgraphs of graphs G_1 and G_2, respectively, then any $C \triangleq (G_{ch1}, G_{ch2})$, is called common checkpoint-subgraph. *We then define the set $\mathscr{M}(\boldsymbol{C}_{ch}^{G_1}, \boldsymbol{C}_{ch}^{G_2})$ as the set containing all MCSIs of the common checkpoint-subgraphs of two graphs G_1, G_2.*

We are now ready to define the concept of *Relevant Process Fragment (RPF)* as the output of our proposed approach.

Definition 10 (Relevant Process Fragment (RPF))
A graph C is called a Relevant Process Fragment (RPF) of a process model collection $\boldsymbol{G_{coll}} = \{G_1, \dots, G_n\}$ when it holds:

1. $C = (\boldsymbol{V_C}, \boldsymbol{E_C}) \in \mathscr{M}(\boldsymbol{C}_{ch}^{G_i}, \boldsymbol{C}_{ch}^{G_j})$, $G_i, G_j \in \boldsymbol{G_{coll}}$, $1 \le i, j \le n$ *(p_1)*
2. $|\boldsymbol{V_C}| \ge v_{min}$, *where* $v_{min} \ge 5$ *(p_2)*
3. $\exists v \in \boldsymbol{V_C} : type(v) \in \boldsymbol{Act}$ *(p_3)*

In other words, as RPF we define a maximum common checkpoint-subgraph between any two graphs G_1 and G_2 of a graph collection $\boldsymbol{G_{coll}}$ (p_1). It also follows from Definitions 4 and 10 that an RPF will always have exactly one source that is a checkpoint according to Definition 1. As seen in Definition 10 an RPF also has two extra properties. The first extra property is that an RPF must have a minimum number of vertices v_{min} (p_2). For our approach we considered $v_{min} = 5$ vertices, as this is the minimum threshold from which a BPMN 2.0 subgraph has any structural interest. Moreover, activities have a key role to the substance of any BPMN 2.0 process model. Thus, we are interested into detecting the structures that contain at least one element of activity (***Act***) type (p_3).

Figures 6.2a and 6.2b show RPFs as they are: (i) both MCSI and checkpoint-subgraphs starting from *SE1″* and *EG1″* of Model A (cf. Figure 6.1a) and Model B (cf. Figure 6.1b) respectively (p_1); (ii) they have at least 5 vertices (p_2); and (iii) contain at least one activity (***Act***) element (p_3). In contrast, Figure 6.2c is not an RPF because although it is a common checkpoint-subgraph of Model A and Model B it does not satisfy the following requirements: (i) it does not have 5 vertices and (ii) it does not contain an activity (***Act***) element. Finally, the subgraph of Figure 6.2d is not an RPF because it is not source connected and it does not have a unique source, i. e., it is not a maximum common checkpoint-subgraph of the two models. In the following, we discuss an algorithm that identifies RPFs in a collection of process models.

6.3. Algorithms

6.3.1. Cycles Detection

The formal model presented in Section 6.2 does not consider BPMN 2.0 process models that contain cycles. For this purpose, we need to apply an approach that detects BPMN 2.0 process models with cycles in a collection of process models in order to remove them. Before proceeding to the explanation of the defined approach, in the following we explain what we consider

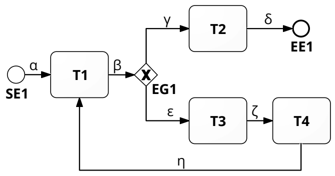


Figure 6.3.: BPMN 2.0 process model with cycle

as a cycle in the graph of a process model.

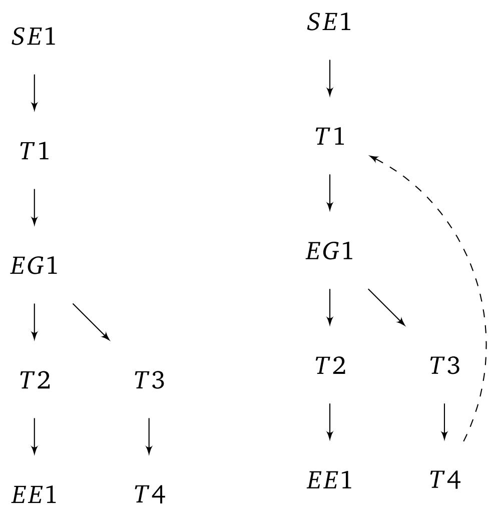
Definition 11 (Cycle)
A graph G contains a cycle, if there exists a path of length $k, k \in \mathbb{N}^+$, $u \mapsto w$ such that $u = w$.

In order to detect cycles in our process models collection we apply the algorithm proposed by Tarjan [Tar72]. Since we could not find an implementation of the algorithm for BPMN 2.0 language, for purposes of completeness, in the following we provide the implementation we applied. The algorithm focuses on detecting the *strongly connected components* of a graph, which are defined as follows:

Definition 12 (Strongly Connected Components (SCC))
Let $G = (\boldsymbol{V}, \boldsymbol{E})$ be a directed graph. Strongly connected components of G is a subset of vertices $\boldsymbol{M} \subseteq \boldsymbol{V}$, such that any two vertices $\{v_i, v_j\} \in \boldsymbol{M}$ are reachable from each other, i. e., $v_i \mapsto v_j, v_j \mapsto v_i$.

For example, in the process model shown in Figure 6.3 the vertices $\boldsymbol{M} = \{T1, T4\}$ comprise strongly connected components because $T1 \mapsto T4$ through the edges $\{\beta, \epsilon, \zeta\}$ and $T4 \mapsto T1$ through the edge $\{\eta\}$.

For the detection of a graph's strongly connected components Tarjan's algorithm [Tar72] is based on a DFS traversal. The DFS algorithm traverses a graph by creating a DFS tree. This ensures that each vertex is not encountered more than once throughout the graph traversal. For example, by applying the DFS algorithm on the process model of Figure 6.3 we construct the DFS tree shown in Figure 6.4a. Starting from vertex "SE1", there is only one



(a) DFS traversal tree (b) Tarjan's algorithm traversal tree

Figure 6.4.: DFS and Tarjan's traversals trees for the process model shown in Figure 6.3

way to move forward and that is to go to "T1". This builds vertices "SE1" and "T1" to the DFS tree as shown in Figure 6.4a. From vertex "EG1" of the process model we then have two choices in the traversal. We may either proceed to "T2" or to "T3". This is shown in the DFS tree of Figure 6.4a as vertex "EG1" has two children ("T2", "T3"). We can pick any of the two vertices to proceed with our traversal, if we proceed to "T2" then we discover vertex "EE1" and add it as a child to the vertex "T2" of the tree. From vertex "EE1" we cannot continue the traversal, thus we move back to the parent vertex "EG1" and choose the rest of the undiscovered vertices, in this case "T3". Similarly, from vertex "T3" we can discover vertex "T4". After "T4" there is no edge that leads to a not already traversed vertex. Therefore, "T4" constitutes a leaf of the DFS tree.

Vertex "T4" has an edge to a vertex that has already been traversed by the DFS algorithm (i. e., edge "η" in Figure 6.3). This edge cannot be identified by the DFS algorithm as it only moves to undiscovered vertices. Tarjan's algorithm [Tar72] basically extends the DFS traversal to also keep the information of such edges, thereafter named as *back link* edges. A back link edge is an edge that allows the backward linking of a descendant vertex to its ancestor. Consequently, their existence indicates the existence of a cycle in a graph, and the algorithm will return true. Thus, the goal of Tarjan's algorithm (Algorithm 6.1) is to *detect back link edges in a graph and consequently, determine the strongly connected components*.

Algorithm 6.1 Initiates the variables and calls the function StrongConnect

```
Input: $G = (V, E)$: The graph $G$ for which we search the cycles.
Output: true, if the graph contains strongly connected components, false
    otherwise.
 1: function TarjansAlgorithm($G$)
 2:     $index \leftarrow 0$
 3:     $stack \leftarrow \emptyset$
 4:     $scc \leftarrow false$
 5:     $hasCycle \leftarrow false$
 6:     for each $v_i \in V$ do
 7:         if ($v_i.index =$ NULL) then
 8:             $hasCycle \leftarrow$ StrongConnect($G, v_i, index, stack, scc$)
 9:         end if
10:     end for
11:
12:     return $hasCycle$
13: end function
```

The Tarjan's algorithm executes the DFS algorithm that recursively explores a vertex v_i (line 5, Algorithm 6.2) and its ancestors. For the algorithm's implementation we consider each vertex $v \in V$ of a graph $G = (V, E)$ as a data structure with following information:

v.index: is the depth first traversal vertex number counter. The counter increases for each newly visited vertex;

v.lowLink: is the smallest index of any vertex known to be reachable from

Algorithm 6.2 Discovers the Strongly Connected Components starting from a vertex v_i

```
Input: $G = (V, E)$: The graph $G$ for which we search the cycles.
Input: $v_i$: the vertex from which we start the graph traversal.
Input: $index$: variable contains the smallest unused index.
Input: $stack$: the stack that holds information for the graph traversal.
Output: $true$ if the vertex is connected to back links, $false$ otherwise.
 1: function StrongConnect($G, v_i, index, stack, scc$)
    // Set the depth of $v_i$ to the smallest unused index
 2:     $v_i.index \leftarrow index$
 3:     $v_i.lowLink \leftarrow index$
 4:     $index \leftarrow index + 1$
 5:     $stack.push(v_i)$
 6:     $v_i.onStack \leftarrow true$
        // Handle the ancestors of $v_i$
 7:     for each $((v_i, v_j) \in E)$ do
        // If the ancestors is not yet visited apply a recursion
 8:         if $(v_j.index = \varnothing)$ then
 9:             $scc \leftarrow scc \cup$ StrongConnect($G, v_j, index, stack, scc$)
10:             $v_i.lowLink \leftarrow min(v_i.lowLink, v_j.lowLink)$
                // If $v_j$ is $onStack$ this means that it is the current SCC
11:         else if $(v_j.onStack)$ then
12:             $v_i.lowLink \leftarrow min(v_i.lowLink, v_j.index)$
13:         end if
14:     end for
        // If $v_i.lowLink$ equals to the smallest unused index then $v_i$ is the
          root vertex. We then pop the stack
15:     if $(v_i.lowLink = v_i.index)$ then
16:         $scc \leftarrow \emptyset$
17:         do
18:             $v_j \leftarrow stack.pop()$
19:             $v_j.onStack \leftarrow false$
20:             $scc \leftarrow scc \cup \{v_j\}$
21:         while $stack \neq \emptyset$
22:     end if
23:     if $(v_i.lowLink = v_i.index)$ then
24:
25:         return $true$
26:     end if
27:
28:     return $false$
29: end function
```

a vertex v, including v itself;

v.onStack: is a boolean value that shows if the vertex is contained in the stack.

As the vertices are discovered they are placed in a stack (line 5, Algorithm 6.2). The vertices are not necessarily popped from the stack before the recursive call returns. A vertex v_i remains in a stack if and only if it has a path to some vertex v_j that has been discovered earlier on the same path (i. e., v_j is an ancestor of v_i). Thus, at the end of the recursive call that explores a vertex v_i and its ancestors, we can conclude if v_i has a path or not to one of its ancestors.

Every ancestor v_j of a vertex may be already visited by the traversal. If the ancestor is not already visited we apply again the recursion for vertex v_j. In this case, we also update $v_j.lowLink$ to the smallest index of any vertex known to be reachable from v_j including v_j. If v_j has already be discovered by the traversal, then v_j is already on stack (line 11, Algorithm 6.2). The existence of v_j on stack means that v_j is a strongly connected component. Hence, we need to update $v_i.lowLink$ to the smallest index of any vertex known to be reachable from v_i including v_i. At the end we must check if v_i is the root of a strongly connected component (line 15, Algorithm 6.2). If it is, then we remove it from the stack and create a new strongly connected component. At the end the set of strongly connected components starting from vertex v_i is printed as output.

The complexity of Tarjan's algorithm is linear to the number of edges and vertices of the graph G, i. e., $O(|V|+|E|)$ in the worst case [Tar72]. Based on it we can detect and remove process models with cycles from the collection. The following algorithms assume that such a process has already taken place in order to ensure their absence.

6.3.2. Detection of Frequently Reoccurring Structures

Let $\boldsymbol{G_{coll}}$ be a collection of BPMN 2.0 process models and $\boldsymbol{G^*_{coll}}$ the set of pairs of process models to check for RPFs. $\boldsymbol{G^*_{coll}}$ is constructed to be irreflexive and asymmetric to avoid two runs for the same pairs of graphs. This means that either (G_1, G_2) or (G_2, G_1), but not both, will be contained in $\boldsymbol{G}^*_{coll}$. $\boldsymbol{G}^*_{coll}$ is the maximum set such that:

(i) $\boldsymbol{G}^*_{coll} \subset \boldsymbol{G}_{coll} \times \boldsymbol{G}_{coll}$,

(ii) $x \in \boldsymbol{G}_{coll} \implies (x, x) \notin \boldsymbol{G}^*_{coll}$ and

(iii) $(x, y) \in \boldsymbol{G}_{coll}, x \neq y \implies (x, y) \in \boldsymbol{G}^*_{coll} \veebar (y, x) \in \boldsymbol{G}^*_{coll}$, where $\veebar$ denotes the exclusive disjunction.

For every pair $(G_1, G_2) \in \boldsymbol{G}^*_{coll}$ we execute the RoSE algorithm (Algorithm 6.3) to construct the sets $\mathscr{M}(\boldsymbol{C}^{G_1}_{ch}, \boldsymbol{C}^{G_2}_{ch})$ (cf. Definition 9) of all possible pairs of checkpoints of the process models. Then $\forall\, G \in \mathscr{M}(\boldsymbol{C}^{G_1}_{ch}, \boldsymbol{C}^{G_2}_{ch})$ we need to detect RPFs (according to Definition 10).

RoSE algorithm takes as input two process models G_1 and G_2 for which we need to detect RPFs. As a pre-condition the model with the biggest number of edges always corresponds to G_1 and the set of edges $(\boldsymbol{E}_1, \boldsymbol{E}_2)$ are considered as partially ordered sets, where the partial order of the elements denotes the sequence with which the edges are traversed by DFS algorithm. The sets of process model checkpoints $\boldsymbol{V}^{G_1}_{ch}$ and $\boldsymbol{V}^{G_2}_{ch}$ are also given as input to RoSE algorithm. As it is trivial to obtain a process model's checkpoints (Definition 1) we omit an algorithm that describes this process. Upon termination the RoSE algorithm outputs the collection of the detected RPFs of the two process models (G_1 and G_2). For returning the RPFs of a complete process models collection, RoSE needs to be applied to all pairs in $\boldsymbol{G}^*_{coll}$.

Before proceeding to a detailed description of the algorithm we need to explain the involved variables. The variable $\boldsymbol{RPF}_{coll}$ (line 2, Algorithm 6.3) stores the collection of the detected RPFs, which will be returned at the end of the execution. The variable *matrix* (line 5, Algorithm 6.3) is an $|\boldsymbol{E}_1| \times |\boldsymbol{E}_2|$ *incidence matrix* in which each cell corresponds to a pair of edges $(edge_1, edge_2)$ where $edge_1 \in \boldsymbol{E}_1$ and $edge_2 \in \boldsymbol{E}_2$. An incidence matrix (see for example Table 6.1) is essentially a sparse matrix with ones in the cells where the two edges "match", or zeroes otherwise. Since our approach focuses

Algorithm 6.3 Compares two graphs to discover and extract the existing RPFs. It inserts the extracted RPFs in the RPFs collection $\boldsymbol{RPF}_{coll}$ and keeps metadata on the RPF frequency of appearance in the $\boldsymbol{RPF}_{coll}$.

```
Input: $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$: The models to compare.
Input: $\boldsymbol{V}_{ch}^{G_1}$, $\boldsymbol{V}_{ch}^{G_2}$: the sets of checkpoints of the two models $G_1$, $G_2$
Output: $\boldsymbol{RPF}_{coll}$: A multi-set with the discovered Relevant Process Fragments
    and corresponding intensities
Require: It holds that $((|E_1| \geq |E_2|)$ and $(\{E_1, \leq\}), (\{E_2, \leq\}))$ are partially
    ordered sets, where the partial order of the elements is defined by the
    sequence with which an element (edge) is accessed by the DFS traversal
 1: function RoSE($G_1, G_2, \boldsymbol{V}_{ch}^{G_1}, \boldsymbol{V}_{ch}^{G_1}$)
 2:     $\boldsymbol{RPF}_{coll} \leftarrow \emptyset$
 3:     for each ($outgoing(v_1) \in \boldsymbol{V}_{ch}^{G_1}$ $as\ edge_1$) do
 4:         $\boldsymbol{matrices} \leftarrow \emptyset$
 5:         $matrix \leftarrow$ InitialiseMatrixWithZero($matrix$)
 6:         for each ($outgoing(v_2) \in \boldsymbol{V}_{ch}^{G_2}$ $as\ edge_2$) do
 7:             $matrix \leftarrow$ CreateMatrix($edge_1, edge_2, matrix$)
 8:             $\boldsymbol{matrices} \leftarrow \boldsymbol{matrices} \cup \{matrix\}$
 9:         end for
            // Comparison of this checkpoint finished.
10:         $\boldsymbol{RPFs} \leftarrow$ MatricesToRPFs($\boldsymbol{matrices}$)
            // The edges of a checkpoint might return many subgraphs that
              were isomorphic. We only need to keep the largest RPF starting
              from two specific checkpoints.
11:         $\boldsymbol{RPFs} \leftarrow$ FindMaxRPF($\boldsymbol{RPFs}$)
12:         for each ($rpf_{new} \in \boldsymbol{RPFs}$) do
13:             $\boldsymbol{RPF}_{coll} \leftarrow \boldsymbol{RPF}_{coll} \cup$ FixDuplicates($G_1, G_2, rpf_{new}, \boldsymbol{RPF}_{coll}$)
14:         end for
15:     end for
16:     return $\boldsymbol{RPF}_{coll}$
17: end function
```

only on the structural characteristics of the process models, two edges $edge_1 = (u_1, v_1)$ and $edge_2 = (u_2, v_2)$ are considered to "match" when their incident vertices are of the same type, i. e., $e_1 \simeq e_2 \Leftrightarrow type(u_1) = type(u_2) \wedge type(v_1) = type(v_2)$. For example, in Figure 6.1 it holds that $\alpha \simeq \alpha'$ (start events connected to exclusive gateways) and $\beta \simeq \beta'$ (exclusive gateways

$G_1 \backslash G_2$	α'	β'	γ'	δ'	ϵ'	ζ'	η'	θ'	Σ_R
α	1	0	0	0	0	0	0	0	*1*
β	0	1	0	0	1	0	0	0	*2*
γ	0	0	1	0	0	1	0	0	*2*
δ	0	0	0	1	0	0	1	0	*2*
ϵ	0	1	0	0	1	0	0	0	*2*
ζ	0	0	0	1	0	0	1	0	*2*
η	0	1	0	0	1	0	0	0	*2*
θ	0	0	1	0	0	1	0	0	*2*
ι	0	0	0	1	0	0	1	0	*2*
κ	0	0	0	0	0	0	0	1	*1*
Σ_C	*1*	*3*	*2*	*3*	*3*	*2*	*3*	*1*	

Table 6.1.: Incidence matrix for the process models of Figure 6.1

connected to tasks) match. The initial incidence matrix of the process models of Figure 6.1 is shown in Table 6.1 and it contains all possible edge matchings between the two models. For example, $\alpha \simeq \alpha'$, $\beta \simeq \alpha'$, $\beta \simeq \zeta'$, etc.

As seen in Table 6.1 the matrix may contain more than one ones per row (R) and per column (C) (i. e., $\Sigma_R \geq 1$ and $\Sigma_C \geq 1$). However, RPFs contain only one-to-one matchings of edges as the definition of isomorphism they rely on (Definition 6) is an injective function. Therefore, our goal is to reduce the extra ones per row and column in a consistent way and derive the resulting RPF. By a closer look at the incidence matrix we observe that multiple ones on the same row indicate alternative matchings of the corresponding edge, e. g., $\beta \simeq \beta'$ and $\beta \simeq \epsilon'$. For obtaining the one to one isomorphism we need to choose either $\beta \simeq \beta'$ or $\beta \simeq \epsilon'$. Let us assume for example that we choose the first matching (i. e., $\beta \simeq \beta'$). In this case we need to eliminate the $\beta \simeq \epsilon'$

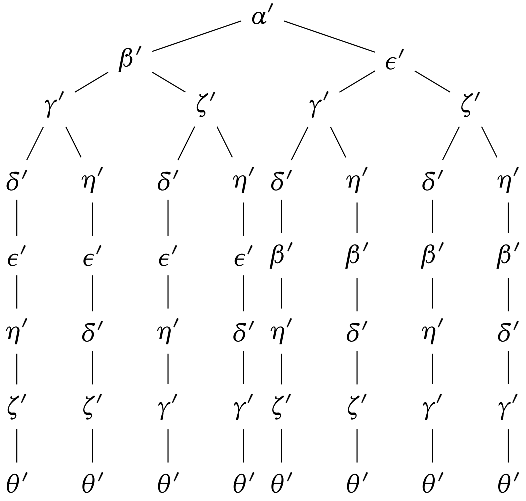


Figure 6.5.: Decision tree for RPFs detection

from the incidence matrix. We also need to eliminate the rest of the ones on the β row so that it will not be chosen for another edge (i. e., column). In a similar manner it can be concluded that the incidence matrix should contain at most one one in the Σ_R and Σ_C for any detected RPF. In other words, for the detection of an RPF it should hold that for each row $\Sigma_R \leq 1$ and for each column $\Sigma_C \leq 1$.

The challenge at this point is to eliminate the redundant ones of the incidence matrix in a consistent way. Reducing the graph isomorphisms to tree searches is a well established technique [Ull76]. To this effect, we are introducing a tree that represents all the possible isomorphic candidates indicated by the incidence matrix. By traversing each row of the matrix, we are gradually building the tree that represents all the possible choices (i. e., RPFs candidates).

The tree that maps to the incidence matrix of Table 6.1 is shown in Figure 6.5 and it represents all possible isomorphic choices that can be produced by the incidence matrix, when we are gradually eliminating the ones of each row. The root of the tree is the source node of G_2 and each child of the root indicates the possible choices of ones of the next row. Each tree path from the root to the leaves indicates the set of edges that build a candidate RPF. For example, for the incidence matrix in Table 6.1 and corresponding decision tree of Figure 6.5 we start traversing the matrix from the cell (α, α') which is the root. The next row (β) has ones in the positions (β, β') and (β, ϵ'). Hence, we can choose either of one of these two. Thus, the root node α' on the tree has children β' and ϵ' as they constitute alternative choices of the RPF.

Proceeding now to the next row of the incidence matrix, γ is mapped to γ' or ζ'. This choice is represented in the tree by putting γ' and ζ' as children of β' and ϵ', i. e., the alternatives produced by the edge β of the previous step. Similarly, the ones of row δ will produce the children of the leaves γ' and ζ'. We are now on row ϵ where the ones are on β' and ϵ'. The ones of these positions have already been used on the first step (edge α) of our procedure and are already placed as the children of the root. In this case, we cannot add both as children because each tree path must contain each edge only once. Therefore, for the subtree of the decision tree that starts from β' we will only add as leaf-child ϵ', while for the subtree of the tree that starts from ϵ' we will add β' as leaf-child. We proceed likewise to build the rest of the decision tree.

With respect to the method described above, for the representative example (Figure 6.1), the resulting RPF starting from the checkpoints (*SE1* and *SE1'*) is the model of Figure 6.2a which is isomorphic to the complete model B (Figure 6.1b). In other words, the resulting RPF is a subgraph of model A (cf. Figure 6.1a) and the complete model B (cf. Figure 6.1b). In this case the tree is presenting different orderings of the same set of edges $(\alpha', \beta', \gamma', \delta', \epsilon', \eta', \zeta', \theta')$.

In the following we explain in more detail the set of algorithms that are used to detect the RPFs, by exploiting the concepts of the incidence matrices and decision tree. For detecting the RPFs of two process models the RoSE algorithm (Algorithm 6.3) exploits the aforementioned constructs of incidence matrix and its corresponding decision tree. The CreateMatrix

Algorithm 6.4 Runs DFS on checkpoint-subgraph of G_1 and G_2 starting from a specific checkpoint to compare all the visited checkpoints with each other and create the incidence matrix (*matrix*) accordingly

```
Input: $edge_1, edge_2$: the outgoing edges of a checkpoint of graphs $G_1$ and $G_2$
    respectively.
Input: $matrix$: the sparse incidence matrix, initialised with zeroes ("0").
Output: $matrix$: the sparse incidence matrix that contains ones ("1") on the
    cells for which the edges matched.
 1: function CreateMatrix($edge_1, edge_2, matrix$)
    // Let $edge_1 = (u_1, v_1)$ and $edge_2 = (u_2, v_2)$
 2:     if ($edge_1 \simeq edge_2$) then          // If the two edges are of the same type
 3:         $\boldsymbol{O}_{u_1} \leftarrow outgoing(u_1)$
 4:         $\boldsymbol{O}_{u_2} \leftarrow outgoint(u_2)$
 5:         $matrix[edge_1][edge_2] \leftarrow 1$
 6:         for each ($o_1 \in \boldsymbol{O}_{u_1}$) do
 7:             for each ($o_2 \in \boldsymbol{O}_{u_2}$) do
 8:                 $matrix \leftarrow$ CreateMatrix($o_1, o_2, matrix$)
 9:             end for
10:         end for
11:     end if
12:     return $matrix$
13: end function
```

function (Algorithm 6.4) is called by the RoSE algorithm to compare all possible combinations of the checkpoints of G_1 to the checkpoints of G_2 (lines 3 and 6, Algorithm 6.3). The concept of checkpoints reduces the emerging comparisons (line 2, Algorithm 6.4) as when the type of two checkpoints do not match (e. g., comparing start event to an exclusive gateway) the comparison is terminated immediately and an empty incidence matrix is returned by the CreateMatrix function (line 12, Algorithm 6.4). If the two checkpoints match, we start traversing all possible combinations of the edges of the two process models by using a DFS algorithm. During the DFS traversals we compare all edges of the process models to each other and we gradually build the incidence matrix which is stored in the *matrix* variable (line 7, Algorithm 6.3). If there is a mismatch between an edge, the DFS

Algorithm 6.5 Function that uses ***matrices*** to detect and synthesize the set of discovered RPFs (***RPFs***)

```
Input: $\boldsymbol{matrices}$ a set of $matrix$ variables of incidence matrices describing
    the edge matches discovered between two checkpoint subgraphs
Output: $\boldsymbol{RPFs}$ the set of detected RPFs, derived from the individual $matrix$
    variables
 1: function MatricesToRPFs($\boldsymbol{matrices}$)
 2:     $\boldsymbol{RPFs} \leftarrow \emptyset$
 3:     for each ($matrix \in \boldsymbol{matrices}$) do
 4:         $column \leftarrow$ FindFirstOneInRow()
 5:         $root \leftarrow$ NewTreeNode(
                $0, column, matrix, map.rowSize, map.columnSize$)
 6:         $tree \leftarrow$ FixDecisionTree($root, matrix$)
 7:         for each ($branch \in tree$) do
 8:             $\boldsymbol{V}_{new} \leftarrow$ DiscoverVerticeFromEdges($\boldsymbol{E}_{new}$)
 9:             $rpf_{new} \leftarrow G(\boldsymbol{V}_{new}, \boldsymbol{E}_{new})$
10:             if ($rpf_{new}.isValid()$) then
11:                 $\boldsymbol{RPFs} \leftarrow \{rpf_{new}\} \cup \boldsymbol{RPFs}$
12:                 $break$
13:             end if
14:         end for
15:     end for
16:     return $\boldsymbol{RPFs}$
17: end function
```

traversal stops for the examined path and the *matrix* will contain zeroes to all the forthcoming edges.

When an incidence matrix of a comparison is constructed, we insert it to a set of incidence matrices variables (line 8, Algorithm 6.3). The edges of a checkpoint might return many subgraphs that were isomorphic, therefore at this point we have different versions of incidence matrices variables stored. Then through the function MatricesToRPFs(***matrices***) (Algorithm 6.5) we detect all RPFs that occur from each $matrix \in \boldsymbol{matrices}$ based on Definition 10, by applying the tree-based approach described above. The purpose of the MatricesToRPFs function is to detect and synthesise the set of RPFs that are described by the incidence matrices given as input to the function. The

Algorithm 6.6 Function for building the decision tree (*tree*) for RPF detection with respect to a matrix (*matrix*)

```
 1: global variables
    // Global variable used as memory for defining the edges which are still
      not added in the tree. It is implemented as a LIFO Queue. This variable
      is shared with the AddChildren function (cf. Algorithm A.2).
 2:     childrenQueue ← ∅
 3: end global variables
Input: e the edge that is currently handled for the tree construction
Input: matrix the matrix for which we build the decision tree
Output: tree the decision tree that corresponds to matrix
 4: function FixDecisionTree(parent, matrix)
 5:     children ← AddChildren(parent, matrix)
 6:     while (childrenQueue ≠ ∅) do
 7:         child ← childrenQueue.poll()
 8:         if (child ≠ ∅) then
 9:             children ← AddChildren(child, matrix)
10:         end if
11:     end while
12:     return tree
13: end function
```

MatricesToRPFs function handles each incidence matrix separately. For this it constructs a decision tree by calling the FixDecisionTree(*parent*, *matrix*) function (Algorithm 6.6). The purpose of this function is to build a decision tree that reflects the corresponding incidence matrix. For implementation purposes each branch of a decision tree is considered a data structure that contains the following information:

Matrix State: the values of the updated *matrix* variable when we choose to add the specific node in the decision tree;

row: the row index of the cell in the *matrix* in which we found the one ("1") that led to the creation of this node in the decision tree;

column: the column index of the cell in the *matrix* in which we found the one ("1") that led to the creation of this node in the decision tree;

Matrix Row Size: constant variable that contains the total number of rows of the *matrix* variable;

Matrix Column Size: constant variable that contains the total number of columns of the *matrix* variable.

The construction of a tree node is executed by calling the NewTreeNode function (line 5, Algorithm 6.5). The implementation of this function is considered trivial and it is omitted. Overall, the FixDecisionTree function recursively calls the assistive function AddChildren(*parent*,*matrix*) (Algorithm A.2 in the Appendix). The AddChildren function calculates the children of each tree node with respect to the *matrix* state of the *parent* node, and applies the needed updates to the *matrix* variable for the newly added node (line 11, Algorithm A.2). The FixDecisionTree and AddChildren functions share a global queue variable that implements a Last-In-First-Out (LIFO) queue (*childrenQueue*) as memory and stores the edges of the *matrix* variable for which we have still not calculated their children in the decision tree. The FixDecisionTree function terminates when the *childrenQueue* is empty and returns a decision tree.

The decision tree of each *matrix* is returned to the MatricesToRPFs function. Each branch in a decision tree is essentially an alternative set of matched edges (cf. Figure 6.5). Consequently, through the DiscoverVerticeFromEdges function (line 8, Algorithm 6.5) we can discover the set of vertices contained in a branch. The DiscoverVerticeFromEdges function is trivial and thus its detailed description is omitted. After the completion of the DiscoverVerticeFromEdges function the vertices and edges are known to us. Hence, we can define a new checkpoint subgraph (see Definition 4) which is the newly discovered RPF (line 9, Algorithm 6.5). If the newly discovered structure is an RPF we can: (i) add it to the discovered RPFs set of the MatricesToRPFs function (***RPFs***) and (ii) stop the RPF search for this branch.

As we will prove in Section 6.3.3, all branches of the RPF detection decision tree are of the same size. Thus, they reveal the maximum set of edges the graphs ($G1, G_2$) have in common. To this effect extracting one RPF from a tree is enough and we can stop the search to reduce complexity. Hence, we need to choose the first set of edges (branch) that satisfies the RPF properties (see Definition 10). Due to the fact that the ***matrices*** variable might contain subsets of an RPF, we need to detect and keep only the largest RPF starting from a specific checkpoint. This is done with function FindMaxRPF (line 11, Algorithm 6.3). The implementation of FindMaxRPF is considered trivial as it contains size and source checkpoint comparisons, thus its omitted.

Each RPF detected for a distinct model combination is compared against the $\boldsymbol{RPF_{coll}}$ for possible existing duplicates (clones) of it. This is done through the FixDuplicates function (cf. Algorithm A.1). The FixDuplicates function takes as input an RPF (rpf_1) and the RPFs collection ($\boldsymbol{RPF_{coll}}$) against which we search duplicates of rpf_1. To check if two RPFs are isomorphic to each other (Definition 5) we first check the number of their edges (line 3, Algorithm A.1). If the two RPFs have equal number of edges then we proceed to check if there exists a real isomorphism between the RPFs. For this we re-apply the CreateMatrix function and derive the RPF from *matrix* by using the MatricesToRPFs({*matrix*}) function (Algorithm 6.5) as described above. The CreateMatrix function will return the maximum match between the two graphs (i. e., rpf_1 and rpf_2). To this effect, at this point we only need to check that the returned maximum common subgraph between the RPFs is the RPF itself (i. e., rpf_2). This is done by checking the set-theoretic difference between the edge sets of the resulting RPF (i. e., rpf_{result}) and rpf_2. If the set-theoretic difference of the two sets returns an empty set then the two sets are equal (i. e., the two RPFs have exactly the same edges) and all the three RPFs are isomorphic to each other ($rpf_1 \sim rpf_2 \sim rpf_{result}$).

For each RPF in the $\boldsymbol{RPF_{coll}}$ we store two types of metadata: (i) the occurrences of an RPF in the $\boldsymbol{RPF_{coll}}$ and (ii) the distinct process models in which an RPF was discovered. If a newly discovered RPF (rpf_1) has a duplicate (rpf_2) in the $\boldsymbol{RPF_{coll}}$ then it will not be inserted again in it. Instead we update the metadata of rpf_2 regarding the frequency of occurrence of this specific RPF (line 12, Algorithm A.1) and the number of graphs that contain this RPF (line 13, Algorithm A.1). If an RPF rpf_1 does not have duplicates in the $\boldsymbol{RPF_{coll}}$, we insert it in it and update its metadata (line 16, Algorithm A.1).

These metadata are used after the execution of the RoSE algorithm for deriving the statistics on frequency of occurrence of an RPF. In terms of process models synthesising for workload mix generation, these metadata are also used to calculate the intensity with which a process model will participate in the workload mix (see Chapter 9).

Finally, the detected $\boldsymbol{RPF}_{coll}$ is returned by the RoSE algorithm (Algorithm 6.3). The RoSE algorithm is iteratively called externally for all the possible distinct pairs of model combinations in the collection.

6.3.3. Completeness and Complexity

The RoSE algorithm (cf. Algorithm 6.3) uses DFS traversal on two models for discovering all possible isomorphisms between their edges and insert them into an incidence matrix (Table 6.1). By construction the incidence matrix will contain all possible subgraph isomorphisms between the edges of the two models. We are then applying a tree search [Ull76] to consistently produce all possible isomorphisms between the edges of the two process models. Although the set of edges represented by tree branches may vary, in the following we prove through proof by contradiction that the branches of the tree will always be of same length:

Theorem 1 (Branches are of equal length)
All branches (paths from root to leaves) of the decision tree for RPF detection are of the same length.

Proof
We make the following assumption:

$\mathcal{P}$: There exists a branch in the tree that is longer than the rest of the branches (Assumption 1).

Then, for every matching of edges (e_i, e_j) that exist in a branch we define the following sentences:

$\mathcal{Q}$: The matching of edges had an alternative in the incidence matrix.

$\neg\mathcal{Q}$: The matching of edges did not have alternative in the incidence matrix.

$\mathcal{W}$: The matching of edges belongs to the branch of the tree.

$\neg\mathcal{W}$: The matching of edges does not belong to the branch of the tree.

Then, from the tree construction process the following premises hold for any matching of edges that belongs to a branch of the tree:

$\mathcal{Q} \wedge \mathcal{W}$: The matching of edges has alternatives and it belongs to a branch (Premise 1).

$\mathcal{Q} \wedge \neg\mathcal{W}$: The matching of edges has alternatives and it is not part of a branch (Premise 2).

$\neg\mathcal{Q} \rightarrow \mathcal{W}$: The matching of edges does not have alternatives, therefore it belongs to a branch (Premise 3).

Moreover, from construction the following premise is an antinomy (i. e., always *false*):

$\neg\mathcal{Q} \wedge \neg\mathcal{W}$: The matching of edges does not have alternatives and it does not belong to a branch (Premise 4).

For $\mathcal{P}$ to hold, i. e., for a branch to be longer than the rest of the branches, it means that there exists at least one edge in the branch for which Premises

1–3 do not hold. More particularly, the following must hold:

$$\begin{aligned}
\neg\Big\{\big((\mathcal{Q} \wedge \mathcal{W}) \vee (\mathcal{Q} \wedge \neg\mathcal{W}) \vee (\neg\mathcal{Q} \rightarrow \mathcal{W})\big)\Big\} &\equiv \\
\neg\Big\{\big(\mathcal{Q} \wedge (\mathcal{W} \vee \neg\mathcal{W})\big) \vee (\neg\mathcal{Q} \rightarrow \mathcal{W})\Big\} &\equiv \\
\neg\Big\{\big(\mathcal{Q} \wedge (\boldsymbol{T})\big) \vee (\neg\mathcal{Q} \rightarrow \mathcal{W})\Big\} &\equiv \\
\neg\big(\mathcal{Q} \vee (\neg\mathcal{Q} \rightarrow \mathcal{W})\big) &\equiv \\
\neg\big(\mathcal{Q} \vee (\neg\neg\mathcal{Q} \vee \mathcal{W})\big) &\equiv \\
\neg\big(\mathcal{Q} \vee (\mathcal{Q} \vee \mathcal{W})\big) &\equiv \\
\neg(\mathcal{Q} \vee \mathcal{Q} \vee \mathcal{W}) &\equiv \\
\neg(\mathcal{Q} \vee \mathcal{W}) &\equiv \\
\neg\mathcal{Q} \wedge \neg\mathcal{W} &\equiv \\
\boldsymbol{F}.
\end{aligned}$$

The above set of statement is an antinomy, thus we may conclude that our assumption is not true (i. e., $\neg\mathcal{P}$ holds), i. e., there is not a branch in the tree that is longer than the rest. Similarly, we can prove that there is not a branch of the tree that is shorter than the rest. Thus, we may conclude that the branches of the tree will always be of equal size.□

If there is a solution our methodology will detect it, as it exhausts the search space for the detection of RPFs between two process models. *Consequently, we can argue that the proposed methodology is complete*. In terms of complexity, due to the naïve nature of the decision tree construction in the worst case we are in the area of $O(m^n)$ where $m = |\boldsymbol{E}_1|$ and $n = |\boldsymbol{E}_2|$. This also holds for the FixDuplicates function, which basically re-applies the RoSE algorithm. Furthermore, we need to compare all possible pairs of graphs in the collection, denoting $O(k^2)$ comparisons, where k is the size of business process model collection.

In real life practice process models are smaller graphs with low structural complexity. Thus, in combination to the applied heuristics (i. e., checkpoints, decision tree, incidence matrix) the observed performance can be achieved in realistic times (cf. Chapter 9). As we will show in Chapter 9, unlike to similar efforts [BDDS14] the algorithm completed successfully in realistic times and discovered 143 RPFs.

6.4. Chapter Summary

In this chapter we introduced the RoSE algorithm that detects frequently reoccurring structures in a collection of BPMN 2.0 process models. Structural similarities of process models is accepted to have application in different scenarios of BPM, such as detection of differences in various versions of the same process models [PW12], clone detection in process model repositories [EDG+12; Dum+13] or generation of synthetic process models with respect to reoccurring fragments [YDG15].

In contrast to most of the approaches of subgraph isomorphisms that are based on a known subgraph that is searched against a bigger graph, our approach starts by comparing the bigger graphs (i. e., process models) to extract similarities. The goal is to detect and extract the reoccurring subgraphs (defined as Relevant Process Fragments (RPFs)), which are basically the reoccurring structures (or subgraphs) of the compared process models. Thus, in the scope of this work we first defined the underlying formal model of our approach. We described a group of algorithms that detect the RPFs of a BPMN 2.0 process models collection and calculate metadata regarding the frequency of their appearance. Our approach utilises and customises well established techniques of subgraph isomorphism as for example Ullmann's algorithm [Ull76] and we argued that our approach is complete as it exhausts the search space. The presented approach currently does not support cyclic structures, therefore we presented algorithms that detect cyclic process models in the original process models collection, in order to be able to remove them. Later, in Chapter 9 we apply the RoSE algorithm on the real world process models collection described in Chapter 3 to derive RPFs that are then used towards the generation of representative workload mix.

Chapter 7

Generating BPMN 2.0 Process Models

"Great wisdom not applied to action and behaviour is meaningless data."

R. Drucker

In Section 1.2 we described (semi-) automated workload mix generation as one of the most challenging tasks towards the development of a benchmark for WfMSs. In the context of WfMSs, workload mix generation refers primarily to the constituent process models and their execution behaviour. Having already discussed the discovery and extraction of RPFs (Chapter 6), in this chapter we shift the focus on research question RQ-6: "How to synthesise a representative BPMN 2.0 process model?", by proposing a method to support the (semi-) automatic process model generation (see contribution C-6: "Representative BPMN 2.0 process model generation method"). Our proposed approach is agnostic with respect to its application. This means that we essentially use the workload development problem as a use case for our proposal that, due to its generality, can also be applied for other purposes, as for example, the evaluation of refactoring techniques requiring large repositories of process models [DRR12; YDG15].

In order to provide a generic solution, we use an initial set of structural criteria to drive the process model generation. These criteria are considered reusable across diverse use cases. Some examples are model size, structural criteria like specific events, e. g., start or end events in the model, number of control/activity nodes etc. [Car08]. As we will show, RPFs also play a key role in the generation of the process models, and they provide useful information for measuring the *representativeness* of a generated process model with respect to a process model collection and define the *intensity* of a process model in the workload mix (Chapter 9). More particularly, this chapter extends previous work [SAL16], by presenting in a more comprehensive way a method for generating *automated* process models out of RPFs with respect to given structural criteria.

The rest of this chapter is structured as follows: Section 7.1 describes a high level overview of our approach that is split in a group of phases. These phases are discussed more extensively in Section 7.2. Finally, Section 7.3 provides a short summary of the chapter.

7.1. Method Overview

The overall goal of the process model generation method is to construct a synthetic, executable process model that follows specific structural criteria defined by the application user. As the generated process model will be used for benchmarking purposes, its execution behaviour is fully automated, i. e., it does not pause its execution to require input from the user. Our proposed method for process model generation is divided into three main phases (i. e., *"Characterisation", "Synthesis" and "Executable Refinement"*), that are presented in Figure 7.1. The application user ("Application User") acts as the orchestrator of process model generation, by invoking it and providing required data in each phase. Hence, the process model generation

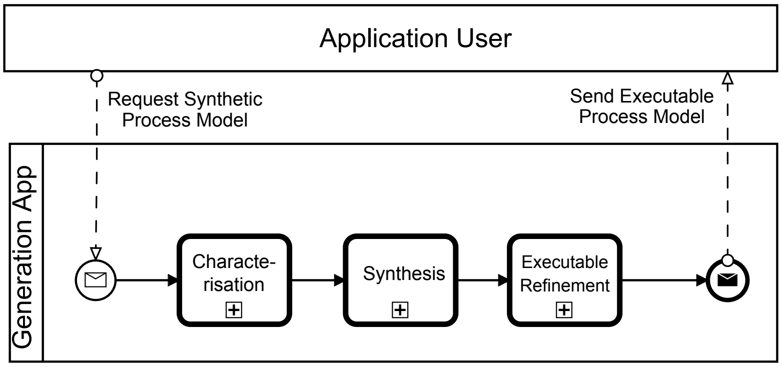


Figure 7.1.: Process model generation method overview

application ("Generation App" in Figure 7.1) is initially triggered with a request made by the application user ("Request Synthetic Process Model" in Figure 7.1). Figure 7.1 presents the sequence with which each phase needs to be invoked for an end-to-end generation of a synthetic process model. However, provided the existence of the required data, the user may also invoke any of the phases individually. Therefore, the phases are modelled as call activities ([ISO13, [p. 183]]) to allow also an independent instantiation. At the end, the process model generation returns the generated process model to the user ("Send Executable Process Model" in Figure 7.1).

The first phase of the process generation method is *Characterisation* (Section 7.2.1) and its expanded view is shown in Figure 7.2. This phase needs a reference to a data storage that contains a collection of RPFs ("RPFs Collection" data storage in Figure 7.2) extracted by any means (e. g., manually, through the execution of the RoSE algorithm (Chapter 6) etc.). The RPFs are parsed ("Parse Metadata" script task in Figure 7.2) in order to create structural metadata, which are persisted in a separate data storage ("RPFs Structural Metadata" data storage in Figure 7.2). This phase may be executed only once for a specific RPF collection. Hence, it is not needed to recreate the metadata data storage each time the user needs to construct a synthetic process model.

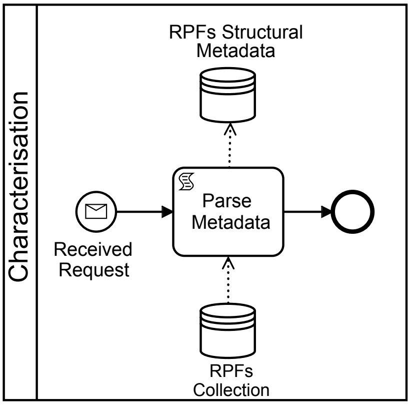


Figure 7.2.: Characterisation Phase

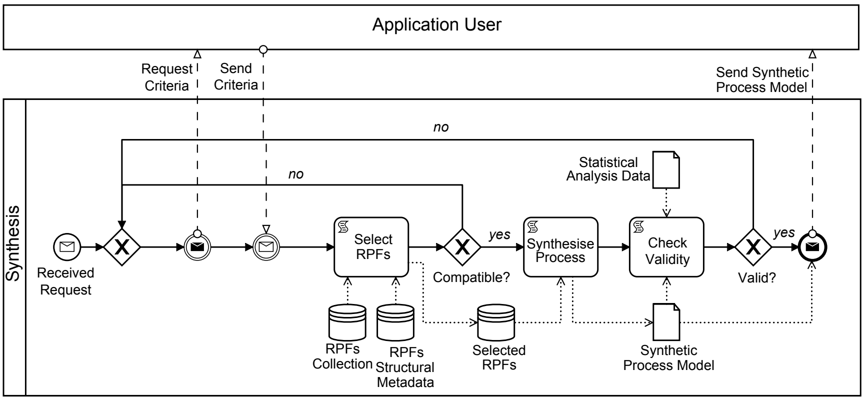


Figure 7.3.: Synthesis Phase

The *characterisation* phase is followed by the *synthesis* phase (Section 7.2.2) and its expansion is shown in Figure 7.3. In this phase structural criteria are required as input from the application user (“Request Criteria”, “Send Criteria” in Figure 7.3). These criteria contain information about the structural characteristics of the RPFs to be selected. For example, some structural criteria might be the total number of exclusive gateways or script tasks an

RPF must have. The criteria are afterwards used for selecting appropriate RPFs (“Select RPFs” script task in Figure 7.3) for generating a synthetic process model. For this, the “Select RPFs” script task uses the data created in the previous phase, stored in the “RPFs Collection” and “RPFs Structural Metadata” data stores. The selection of appropriate RPFs does not only consider the given structural criteria, but also applies compatibility checks for ensuring that the selected RPFs can be linked to each other. If the RPFs are compatible with each other (“Compatible?” exclusive gateway in Figure 7.3) we proceed to their synthesis (“Synthesise Process” script task in Figure 7.3). If we fail to select any compatible RPFs for the given criteria, we ask the user to refine the structural criteria.

In “Check Validity” script task (Figure 7.3), the synthesised process model is validated at a first stage against the BPMN 2.0 standard [ISO13]. Although the new synthetic business process model might be valid BPMN 2.0, it might not necessarily be representative of the process model collection that is derived from. For this reason the synthetic process model is also validated against data gathered from applying statistical analysis to the original process model collection (“Statistical Analysis Data” data object in Figure 7.3). If the validation is not successful (“Valid?” exclusive gateway in Figure 7.3) the complete phase must be repeated with different structural criteria. The purpose of this iterative process is to create a valid and representative process model.

As soon as a process model is validated successfully, we proceed to the *Executable Refinement* phase (Section 7.2.3) to convert it to an automated process model. The expanded view of this phase is shown in Figure 7.4. This phase, is a manual step, therefore it is expressed as user tasks in the process shown in Figure 7.4. In particular, in this phase we give guidelines on how to produce data generators and test data, so that a process model will be fully automated. It essentially realises the components “Probabilistic Data Generator’ and “Test Data”’ introduced in Figure 4.2 of Chapter 4. These data for example might be specific web services or scripts with which the process model will interact, or probability assignments for each control flow of an exclusive gateway.

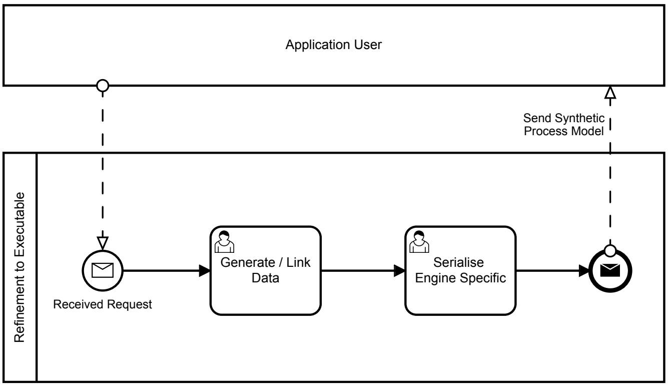


Figure 7.4.: Executable Refinement Phase

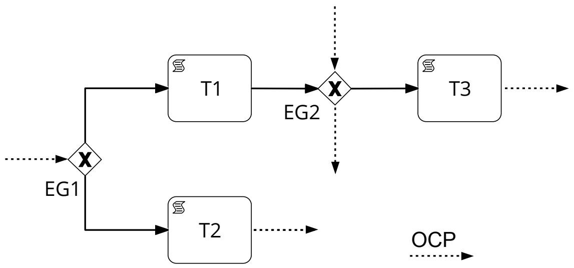
In the “Serialise Engine Specific” user task (Figure 7.4) the user needs to handle the fact that various WfMSs demand different serialisation formats in order to execute a process model [GW13]. Therefore, the user must also provide the specific WfMS on which the model will be executed. Finally, the user can get the executable process model (“Send Synthetic Process Model” in Figure 7.4).

7.2. Synthetic, Executable Process Models Generation

This section discusses in more detail the phases of the synthetic process model generation method. Section 7.2.1 describes the *characterisation* phase, Section 7.2.2 presents the *synthesis* phase and the *executable refinement* phase is discussed in Section 7.2.3.

7.2.1. Characterisation

For the process model generation method we rely on the concept of process fragment *placeholders* [SKK+11]. While in the original definition [SKK+11] placeholders can be anywhere in the process fragment, in our case they can only exist in the beginning and/or at the end of an RPF. More particularly, as



(a) Example RPF and emerging OCPs

Node Metadata					
	`isFlexible`	`generalType`	`specificType`	`getIncomingOCPs()`	`getOutgoingOCPs()`
EG1	false	gateway	exclusive gateway	1	0
T1	false	task	script task	0	0
T2	false	task	script task	0	1
EG2	true	gateway	exclusive gateway	1	1
T3	false	task	script task	0	1

(b) Node structural metadata

RPF Metadata			
`hasStartEvent()`	`hasEndEvent()`	`getTotalIncomingOCPs()`	`getTotalOutgoingOCPs()`
false	false	4	3

(c) RPF structural metadata

Figure 7.5.: Example of RPF, emerging OCPs and calculated metadata

placeholders of an RPF we consider the set of RPF nodes that miss incoming and/or outgoing sequence flows. These can act as potential connection points during the *synthesis*. Therefore, each missing sequence flow in an RPF is hereafter referred to as an *incoming or outgoing Open Connection Point (OCP)* of an RPF.

For example, consider the RPF shown in Figure 7.5a. This RPF has an exclusive gateway (*EG1*) as a source node. In its current state and according to the definition of the source node (see Section 6.2), *EG1* does not have any incoming sequence flows and has two outgoing sequence flows. According to the BPMN 2.0 standard [ISO13] a gateway must be diverging (i. e., one to many sequence flows) or converging (many to one sequence flows). Hence, *EG1* needs *at least* one incoming sequence flow to be a valid diverging gateway. This missing incoming sequence flow is an incoming OCP for *EG1*. Likewise, for this work we assume that the tasks of a valid process model should have at least one incoming and one outgoing sequence flow. Therefore, the script tasks (*T2*, *T3* in Figure 7.5a) miss their outgoing sequence flows and have one outgoing OCP each, causing two outgoing OCPs on the depicted RPF. On the contrary script task *T1* is already connected with incoming and outgoing sequence flows, thus it does not have any OCP. Finally, *EG2* is also an exclusive gateway with one incoming and one outgoing OCP, as it needs either one incoming or one outgoing sequence flow to be a converging or diverging exclusive gateway respectively.

With respect to this concept, the goal of this phase is twofold: we firstly need to parse a given collection of RPFs and obtain their structural metadata, and secondly to determine the existing OCPs of each node and RPF. The calculated metadata are afterwards exploited in the *synthesis* phase to select these RPFs that comply with the structural criteria and can be linked to each other towards the synthesis of a valid process model. As discussed in the previous section, the characterisation of the RPF collection needs to be executed only once for an RPFs collection. The metadata are then stored in a persistent data storage, out of which they can be retracted for any future request.

Figure 7.6 shows the conceptual model of an RPF's structural metadata that were calculated for the purposes of this work. More specifically, an RPF is characterised by the following metadata ("`RPF Metadata`" in Figure 7.6):

`hasStartEvent()`: returns a boolean, indicating if the RPF has a start event,

`hasEndEvent()`: returns a boolean, indicating if the RPF has an end event,

`getTotalIncomingOCPs()`: returns the total number of incoming OCPs

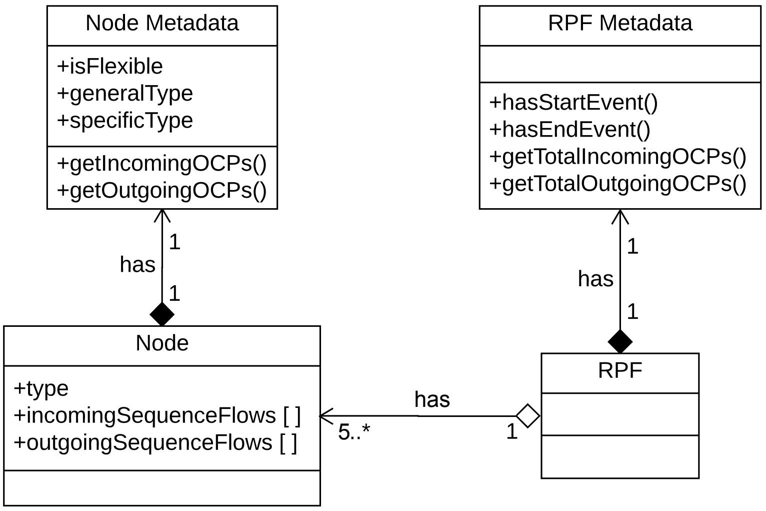


Figure 7.6.: Conceptual model of RPF structural metadata

that exist in the RPF, i. e., the sum of incoming OCPs of the RPF's nodes, and

`getTotalOutgoingOCPs()`: returns the total number of outgoing OCPs that exist in the RPF, i. e., the sum of outgoing OCPs of the RPF's nodes.

Each node of an RPF contains by default the following information ("`Node`" in Figure 7.6):

`type`: the type of the node (see Section 6.2),

`incomingSequenceFlows`: information for the set of incoming sequence flows (i. e., id, name, etc.) of this node,

`outgoingSequenceFlows`: information for the set of outgoing sequence flows (i. e., id, name, etc.) of this node.

In addition to the OCPs, the following metadata are calculated for each RPF node ("`Node Metadata`" in Figure 7.6):

`isFlexible:` it can participate in a synthesis by either linking an incoming or outgoing OCP. For example, *EG2* in Figure 7.5a has one incoming and one outgoing OCP, as it needs either one incoming or one outgoing sequence flow to be a converging or diverging exclusive gateway respectively. This makes *EG2* a flexible node, as it can be used flexibly per case for linking incoming or outgoing sequence flows. We discuss flexible nodes in more detail later in this section.

`generalType:` the node's general type (i. e., task, gateway, event, etc.),

`specificType:` the node's specific type (i. e., script task, exclusive gateway, start event, etc.),

`getIncomingOCPs():` calculates the total number of incoming OCP's for this node, and

`getOutgoingOCPs():` calculates the total number of outgoing OCP's for this node.

A summary of the calculated `Node Metadata` and `RPF Metadata` for the RPF of Figure 7.5a are shown in Figure 7.5b and Figure 7.5c respectively.

7.2.2. Synthesis

The synthesis phase starts with a set of user-defined criteria, that are used to select a set of RPFs that satisfy it. For example, the user may either query the RPF collection by providing general node types in the criteria (e. g., $l_1 = \{\text{2 tasks AND 3 gateways}\}$) or by requesting specific node types (e. g., $l_2 = \{\text{4 service tasks AND 3 parallel gateways}\}$) or a mix of general and specific types (e. g., $l_3 = \{\text{2 tasks AND 3 exclusive gateways}\}$). Overall, we target to select RPFs that: (i) satisfy the given user-defined criteria, and (ii) can be successfully linked to a complete process model.

The selection of RPFs that match the structural criteria is trivial, as each criterion can be translated into simple queries against the "RPFs Structural Metadata" data storage (Figure 7.3). Let $\boldsymbol{L}$ be a set of all criteria used for selecting RPFs from the data storage. Based on the criterion $l_i \in \boldsymbol{L}, 1 \leq i \leq |\boldsymbol{L}|$, let $\boldsymbol{C}_i$ be the set of selected RPFs that resulted from l_i.

As start or end events in the middle of a synthetic process model are not permitted, we apply some additional rules during the RPFs selection. More specifically, let $\varkappa$ and λ be the predicate functions that indicate if an RPF element contains a start event or an end event respectively. Then for the intermediary sets of RPFs it also holds that
$\forall(\{x \in \boldsymbol{C}_i, (2 \leq i \leq |\boldsymbol{L}| - 1) \wedge (|\boldsymbol{L}| \geq 3) : \varkappa(x) = false \wedge \lambda(x) = false\})$.

Finally, the family of sets $\boldsymbol{C}$ of all $\boldsymbol{C}_i$, contains one set of RPFs for each provided criterion.

The next step is to check the Open Connection Points (OCPs) between the sets of selected RPFs. In order to enable the linking between RPFs we need to verify that there is an appropriate number of OCPs between them (i. e., compatibility check). Hence, for every consecutive sets $(\boldsymbol{C}_i, \boldsymbol{C}_{i+1})$ we need to find sequences of pairs (x, y) where:
$\{\exists\, x \in \boldsymbol{C}_i \wedge \exists\, y \in \boldsymbol{C}_{i+1}, 1 \leq i \leq |\boldsymbol{L}| - 1 \,:\, |x_{outgoingOCPs}| \geq |y_{incomingOCPs}|\}$. This condition indicates that the left-hand RPF should have more outgoing OCPs than the right-hand RPF. Despite being more straightforward, a condition of equality would have been too restrictive in our selection process. Nevertheless, as we will discuss in the following, the extra outgoing OCPs introduced by this condition can be handled by heuristics.

Determining RPFs that can be linked (i. e., they are compatible) to each other is handled by Algorithm 7.1. Before proceeding to the explanation of the algorithm's functionality, in the following we explain the variables that the algorithm uses as input. Algorithm 7.1 takes as input the aforementioned collection $\boldsymbol{C}$ with the sets of RPFs that were selected for each criterion. It also uses the variable i to point to the set $(\boldsymbol{C}_i)$ of collection $\boldsymbol{C}$ that we examine at each iteration. Algorithm 7.1 goes over the sets of RPFs $(\boldsymbol{C}_i)$ until it discovers a sequence of RPFs that can be linked. In order to satisfy this purpose Algorithm 7.1 uses backtracking techniques (line 24, Algorithm 7.1). The resulting RPFs set of the algorithm is stored and returned through the ***RPFs*** set. The $(\boldsymbol{RPFs}, \leq)$ set is a partially ordered set were the partial order indicates the criterion to which an RPF corresponds. In other words, the element $rpf_i \in \boldsymbol{RPFs}$ corresponds to the RPF that was chosen by the algorithm for the i^{th} criterion. Consequently, it holds that $rpf_i \in \boldsymbol{C}_i$. When calling

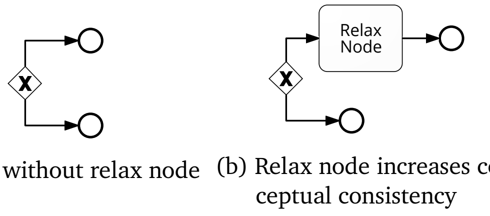
Algorithm 7.1 Detects a set of RPFs that can be linked with each other to synthesise a process model

```
Input: $\boldsymbol{C}$: a family of sets of RPFs, where the set $\boldsymbol{C}_i \in \boldsymbol{C}$ corresponds to the
    $i^{th}$ criterion.
Input: $i$: the index that points to the set $\boldsymbol{C}_i \in \boldsymbol{C}$ that we examine at the
    specific iteration.
Input: $\boldsymbol{RPFs}$: the set of RPFs that are compatible to each other. The $rpf_i \in$
    $\boldsymbol{RPFs}$ is the selected RPF for the $i$ criterion. Consequently, $rpf_i \in \boldsymbol{C}_i$. In
    the first call $\boldsymbol{RPFs} = \emptyset$.
Input: $memory$: is an array that stores information of the positions of a
    compatible RPFs. Namely, the position $memory[i] = j$ indicates the $j^{th}$
    element of the set $\boldsymbol{C}_i \in \boldsymbol{C}$ denoted as $rpf_j$ in the algorithm. In the first
    call it is initialised with zeroes ("0").
Output: $\boldsymbol{RPFs}$: the set of RPFs that are compatible.
 1: function FindCompatibleRPFs($\boldsymbol{C}$, $i$, $\boldsymbol{RPFs}$, $memory$)
 2:     if $(i + 1 < |\boldsymbol{C}|)$ then
 3:         $j \leftarrow memory[i]$
 4:         $l \leftarrow memory[i + 1]$
 5:         $rpf_j \leftarrow \boldsymbol{C}_i.getElementAt(j)$
 6:         while $(l < |\boldsymbol{C}_{i+1}|)$ do
 7:             $rpf_l \leftarrow \boldsymbol{C}_{i+1}.getElementAt(l)$
 8:             if (isCompatible($(rpf_j, rpf_l)$) $= true$) then
 9:                 $memory[i + 1] \leftarrow l$
10:                 $\boldsymbol{RPFs} \leftarrow \boldsymbol{RPFs} \cup \{rpf_j, rpf_l\}$
11:                 if $(i + 1 = |\boldsymbol{C}|)$ then
12:                     return $\boldsymbol{RPFs}$
13:                 end if
14:                 $\boldsymbol{RPFs} \leftarrow$ FindCompatibleRPFs($i + 1, \boldsymbol{RPFs}, memory$)
15:             end if
16:         end while
17:         if $(i = 0)$ then
18:             $memory[i] \leftarrow j + 1$
19:             $\boldsymbol{RPFs} \leftarrow$ FindCompatibleRPFs($i, \boldsymbol{RPFs}, memory$)
20:         else                                        // Erase and backtrack
21:             $memory[i] \leftarrow j + 1$
22:             $memory \leftarrow$ resetMemory($memory, i + 1$)
23:             $\boldsymbol{RPFs} \leftarrow \boldsymbol{RPFs} \setminus \{rpf_j\}$
24:             $\boldsymbol{RPFs} \leftarrow$ FindCompatibleRPFs($i - 1, \boldsymbol{RPFs}, memory$)
25:         end if
26:     end if
27:     return $\boldsymbol{RPFs}$
28: end function
```

Algorithm 7.1 the ***RPFs*** set is initialised with an empty set. Algorithm 7.1 also uses the *memory* variable, which is an integer array that stores the indices of the successfully linked (compatible) RPFs for each set in $\boldsymbol{C}$. More particularly, the integer $j \in \mathbb{N}$ found in the position $memory[i]$ indicates that the j^{th} element of the set $\boldsymbol{C}_i$ is linked with elements of the sets $\boldsymbol{C}_{i-1}$ and $\boldsymbol{C}_{i+1}$. In the beginning of the algorithm the *memory* array is initialised with zeroes ("0").

Algorithm 7.1 is executed for every set of RPFs ($\boldsymbol{C}_i$) in the family of sets $\boldsymbol{C}$. For every set we use the *memory* variable to discover this RPF of the set that has already been linked with other RPFs (rpf_j, line 5, Algorithm 7.1). We keep the rpf_j fixed and for every element in the successive set $\boldsymbol{C}$ (i. e., for all $rpf_l \in \boldsymbol{C}_{i+1}, l \in \mathbb{N}^+$), we examine if the pair of elements (rpf_j, rpf_l) is compatible. In case we find two compatible RPFs, we store the position of the newly linked element in the *memory* variable (line 9, Algorithm 7.1) and add the rpf_l to the set of results (***RPFs***). If the examined set is the next to last set of $\boldsymbol{C}$ (line 11, Algorithm 7.1), we terminate the algorithm and return the resulting set of RPFs (i. e., ***RPFs***). This is because an RPF of the last set has already been paired to this of the next to last set as we always iterate the pair $(i, i+1)$ of sets. In case there are more sets to examine, we recursively call the algorithm for the next sets of the collection.

When Algorithm 7.1 reaches the end of the loop (line 16, Algorithm 7.1) it means that it iterated a full set without finding a match. In this case, if we are at the first set $\boldsymbol{C}_1 \in \boldsymbol{C}$ we can simply recursively call the algorithm for the next RPF of the first set. However, in a different case we need to apply backtracking. In the case of backtracking, we first set the memory variable to the next RPF of the current set ($rpf_{j+1} \in \boldsymbol{C}_i$), then we remove rpf_j from the sets of results, as we could not find an RPF in the successive set that is compatible to it, and lastly we reset the memory to zero ("0") values for all the positions that follow i. Then we recursively call the algorithm for the predecessor set (i. e., $\boldsymbol{C}_{i-1}$) and continue the search of compatible RPFs in a similar manner. If the algorithm is successful the size of the resulting set ($|\boldsymbol{RPFs}|$) must be equal with the number of the criteria. Namely, we should result to one RPF for each user-defined criterion. As the goal of this phase



(a) RPF without relax node (b) Relax node increases conceptual consistency

Figure 7.7.: Heuristics example of adding a relax node

is to create a synthesised process model for benchmarking purposes, the presented method executes non-deterministically. Namely, two executions of this phase may result in diverse synthetic process models. If for the returned set of RPFs holds that $|\boldsymbol{RPFs}| < |\boldsymbol{L}|$ it means that we exhausted all possible combinations of RPFs without finding a sequence of RPFs that can be linked based on the given criteria.

After the selection of compatible RPFs we proceed to their synthesis into a process model. This is done by linking outgoing OCPs with incoming OCPs of the selected RPFs. In order to assist the synthesis of a valid process model we apply the following set of heuristics:

(i) parallel gateways are connected first, exclusive and inclusive gateways are connected next, and activity nodes are connected at the end. This way we eliminate the risk to connect a diverging exclusive and/or inclusive gateway to a converging parallel gateway and produce deadlocks.

(ii) flexible nodes (see Figure 7.5) are handled as outgoing OCPs if they are at the left-hand RPF or as incoming OCPs otherwise. This way we manage to synthesise the process model in a gradual manner, and handle as many as possible those OCPs that are positioned in the middle of the process model.

(iii) at the end of the synthesis process we add start events to the incoming OCPs and end events to the outgoing OCPs. This way we produce a complete process model.

Table 7.1.: Criteria for selecting RPFs and synthesising the example process model of Figure 7.8

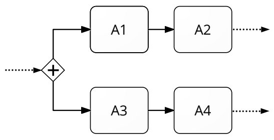
	Exclusive Gateways	Parallel Gateways	Tasks
Crit. 1	2	-	3
Crit. 2	-	1	4

(iv) if a gateway at the end of the synthesis has two outgoing OCPs, then we add an extra empty script task (“relax node”) to one of the branches before the addition of the end event. In this way, we avoid producing meaningless process models (see Figure 7.7) without adding extra overhead in its execution performance (see Chapter 5).

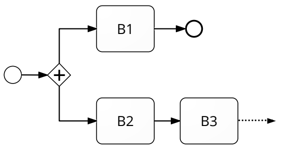
Before returning the result of this phase, the synthetic process model will be additionally checked against its structural validity and its validity against the BPMN 2.0 standard. The structural validity of the synthetic process model is checked against statistics derived from the initial process models collection. The statistics can describe simple structural metrics such as for example, model size, and structural criteria like specific events, e. g., start or end events in the model, number of control/activity nodes, etc. [Car08]. To this effect, we ensure that the synthetic process model reflects the original collection. The validity of the process model with respect to the BPMN 2.0 standard can be achieved with the utilisation of external, well-known frameworks such as Camunda BPMN model API [1]. If any of the aforementioned checks are not satisfied then a corresponding warning or error message is produced and the user is asked to repeat this phase with different criteria.

In Figure 7.8 we present the selected RPFs for the criteria shown in Table 7.1. For the first criterion we select one RPF as shown in Figure 7.8a. For the second criterion we assume that the RPFs shown in Figures 7.8b and 7.8c were selected from the RPF collection. The OCPs of each RPF are marked with dotted arrows. Namely, the RPF of Figure 7.8a has one

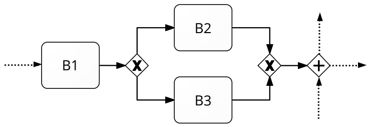
[1]Camunda, Camunda BPMN Model API,
URL: `https://github.com/camunda/camunda-bpmn-model`



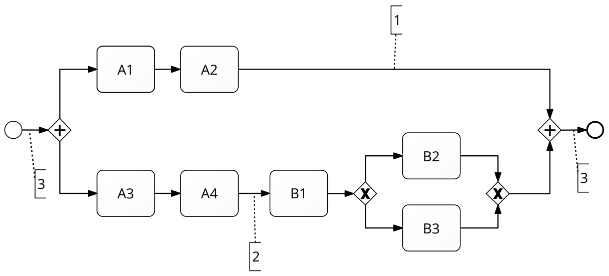
(a) RPF selected for the first criterion of Table 7.1



(b) RPF selected for the second criterion of Table 7.1



(c) RPF selected for the second criterion of Table 7.1



(d) Synthesised process model for the criteria of Table 7.1

Figure 7.8.: Example of synthesising a process model out of RPFs

incoming and two outgoing OCPs, the RPF of Figure 7.8b has one outgoing OCP and the RPF of Figure 7.8c has either two incoming and one outgoing, or one incoming and two outgoing OCPs. This is because the parallel gateway may be used flexibly. Choosing the OCP of Figure 7.8a for the first criterion, we proceed in finding a compatible RPF from the second criterion. The RPF of Figure 7.8b is incompatible, as it contains a start event and its number of incoming OCPs (0) is lower than the number of the first RPF's outgoing OCPs (2). On the contrary, the compatibility check will evaluate to true for RPF of Figure 7.8c. This RPF does not contain a start event and it can be used with two incoming OCPs (RPF of Figure 7.8a and RPF of Figure 7.8b have equal number of outgoing and incoming OCPs respectively).

The RPF of Figure 7.8a and RPF of Figure 7.8b are linked to each other following the aforementioned heuristics. Namely, the incoming OCP of the parallel gateway (Figure 7.8b) will be connected first (see sequence flow marked with "1" in Figure 7.8d) and OCPs of task nodes are connected afterwards (see sequence flow marked with "2' in Figure 7.8d). Finally, start and end events are added to the remaining unconnected OCPs (see sequence flows marked with "3' in Figure 7.8d).

7.2.3. Executable Refinement

The process model generated during the previous phase comprises a reference, non-executable model. As the synthetic process models are targeted for performance benchmarking of the WfMS's process navigator the application user needs to refine it into a process model for automated execution, i. e., a process model that does not require input data during its execution. In this section, we provide guidelines on how to refine the reference process model into a process model with automated execution.

In principle, for deriving the executable process model the application user should implement all the external interactions with placeholder activities. Since our process models are generated for benchmarking purposes we need to eliminate additional overhead of the external interactions and/or redundant code in the process model. This way we manage to isolate the behaviour of the WfMS to the maximum possible extend and obtain clean measurements.

For this we suggest the following changes:

Script Task: the application user should implement it as an no-operational script task. The only case to edit the script task is when it precedes an exclusive gateway. In this case, the script task should implement random number generators, for evaluating the exclusive gateway condition that follows. Defining the probability distribution with which the number generators will produce the numbers is left to the user.

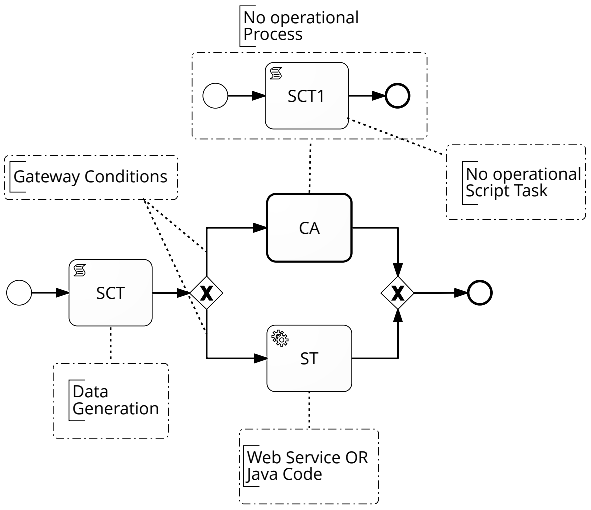


Figure 7.9.: Example of completing a synthetic process model to an executable

Call Activity: the application user should implement it as a non-operational activity (i. e., start event - no operational script - end event). When a call activity of a synthetic process model precedes an exclusive gateway, it should be converted to a script task that generates the required numbers. This is because the needed underlying variable sharing between the parent and the child process models is not supported by all WfMSs [GHLW16].

Service Task: the application user should link it to an no operational web service or Java application. In case a service task precedes an exclusive gateway, the linked web service or Java application should implement data generation respectively.

Exclusive Gateway: the application user should add conditions and variables on the respective outgoing sequence flows to take the control of execution. This should be done in accordance to the preceding script task that implements the data generation. In case there is not any preceding script task to the exclusive gateway, the user should manually add one. In order to eliminate additional overhead, the user should find an optimal way to add a minimum number of script tasks to the process model.

Inclusive Gateway: the application user should explicitly define the conditions with which the outgoing sequence flows of the gateway are activated.

Figure 7.9 shows a process model created for demonstration purposes. The process model starts with a script task (“SCT” script task). As this precedes an exclusive gateway, this needs to implement a data generation script. The script task is followed by an exclusive gateway. For each of the outgoing sequence flows of the gateway, we need to add conditions to take over the control flow. For example, if the “SCT” is set to randomly generate numbers $1 \leq x \leq 10$, then the upper sequence flow could be assigned with the condition $x \leq 5$ and the lower sequence flow with the condition $x > 5$. The upper sequence flow is connected to a call activity (“CA”). This should be linked to a non operational process, as shown in Figure 7.9. Likewise, the lower sequence flow is connected to a service task (“ST”) that should be connected to a running web service or to local Java code.

Finally, different WfMSs may demand different serialisation of the BPMN 2.0 file in order to execute it [GW13]. Therefore, the user needs to apply appropriate changes on the serialisation of the process model.

7.3. Chapter Summary

In this chapter, we introduced a method to generate synthetic BPMN 2.0 process models out of an RPF collection. The synthesis is driven from user-defined structural criteria, as for example, the number of exclusive gateways or script tasks the participating RPFs should have. The selection of the required RPFs is achieved by applying backtracking techniques on sets of RPFs, while the synthesis of the RPFs to a complete process model is based on heuristics. The synthetic process model is validated against structural representativeness with respect to statistics from a real world practice process models collection and against validity to the BPMN 2.0 standard [ISO13]. We then suggest guidelines on how to convert a valid synthetic process model to an executable, fully automated one. The execution of the synthetic process models should be fully automated as they are targeted for performance tests that stress the WfMS's process navigator. The proposed method can be utilised for the creation of single process models or even collections of thousands of synthetic process models. This fact, constitutes our method useful in diverse use cases that require the generation of synthetic, executable process models.

In Chapter 9 we apply the process models generation method for the RPFs detected on the original collection (cf. Chapter 3). We then use the generated process models as part of a workload model that is used for performance benchmarking.

Chapter 8

A Process and Toolchain for Workload Mix Generation

“Software is a great combination between artistry and engineering.”

Bill Gates

The goal of this chapter is to answer research question RQ-4: “How to derive a representative and meaningful workload mix for both general and domain specific benchmarks?”. Hence, we present contribution C-4: “The Workload Mix Generation for Workflow Management Systems (WINE4WfMSs) process”, which builds on concepts and components introduced in Chapters 3, 4, 6 and 7. We also present the architecture and implementation details of a toolchain that supports the WINE4WfMSs process.

As discussed in previous chapters, each sub-method participating in the WINE4WfMSs process may also be used to serve diverse application scenarios in the BPM domain. Thus, as a guiding principle on the toolchain's design we follow the principle of loose coupling to enable the future integration of the components to third-party systems. The movement of cloud technology to the mainstream advocates towards the design of cloud-aware applications [Sen15]. Therefore, we also focus on diverse architectural paradigms and recognised design patterns to facilitate a cloud aware application.

The remainder of this chapter is structured as follows: Section 8.1 describes the WINE4WfMSs process; Section 8.2 discusses related architectural principles and presents the architecture of the toolchain; Section 8.3 shows architectural and implementation details for each component separately; Section 8.4 presents other software that is related to BenchFlow project and Section 8.5 summarises the chapter.

8.1. Workload Mix Definition Method

The definition of the workload model constitutes a critical task in the design of a benchmark (see Chapter 4). In order to support the practitioners towards the definition of a robust workload model, Kounev et al. [KHSB12] summarise an abstract methodology to be followed during the performance modelling process. The process can be described in the following five steps [KHSB12]:

Step 1: identify the basic components of the workload.

Step 2: partition the basic components into classes.

Step 3: identify the resources used by each workload class.

Step 4: describe the inter-component interactions.

Step 5: quantify the workload demands and intensities.

In Chapter 4 we identified the basic workload components for BPMN 2.0 WfMSs (step 1) and described the inter-components interactions (step 4), while the identification of resources used by the workload classes (step 3) is extensively presented by Ferme et al. [FIP15; FIP+16]. The remaining steps (i. e., to partition the basic components into classes (step 2) and to quantify the workload demands and intensities (step 5)) are addressed during the application of the WINE4WfMSs method.

More specifically, the *workload mix* consists of the executable process models that are given as input to a performance test, their execution behaviour, and the *intensity* with which each process model participates in the workload mix. In the scope of this work we defined as *workload class* of the workload mix each pair of process model and corresponding intensity. Consequently, the workload mix is comprised of different workload classes and the execution behaviour of the underlying process models (see Section 4.2). In the case of WfMSs additional attention should be paid to the structure of the workload mix's process models, as this dictates the sequence with which the WfMS will execute a set of activities. Therefore, in order to reflect a diverse set of use cases the structure of the process models participating in the workload mix should be representative of the whole range of a process models collection. To derive a structurally representative workload mix of a given collection we propose the WINE4WfMSs method, which is based on the components and concepts discussed in Chapters 3, 4, 6 and 7. More specifically, the WINE4WfMSs method consists of the following phases (Figure 1.1):

Phase 1: Cleaning and analysing the initial process models collection.

Phase 2: Discovery of the existing RPFs.

Phase 3: Derivation of the process models for the workload mix.

Phase 4: Partitioning the process models into workload mix classes.

Phase 5: Defining the execution behaviour of the workload mix.

In phase 1 we transform custom serialised process models (cf. Section 3.2) or remove invalid and incomplete process models, as none of them can be part of the workload mix. Then we apply a statistical analysis on the collection, in order to obtain an insight into its structural composition. In phase 2 we proceed to the detection and extraction of the RPFs, by applying the RoSE algorithm on the process models collection (Chapter 6). The detected RPFs are extracted and semantically annotated with their frequency of appearance in the original collection, as well as other metadata regarding their structure. In phase 3 we exploit the results acquired by the previous analyses for recognising or synthesising (Chapter 7) distinct, structurally representative process models that will participate in the workload classes. In phase 4, we partition the derived process models into workload classes.

Each class participates in the workload mix with a different intensity, which corresponds to the degree of the process model's representativeness to the collection. Let $C = \{c_1, c_2, ..., c_k, ..., c_m\}, k, m \in \mathbb{N}, 1 \leq k \leq m$ be the set of process models that we include in the classes of the workload mix. For calculating the representativeness *repr* of a process model ($c_k \in C$) for the collection we use the following formula [FSP+17]:

$$repr(c_k) = \frac{1}{2\,|S_k|} \sum_{rpf_i \in S_k} \left(\frac{t(rpf_i)}{|Sc|} + \frac{m(rpf_i)}{|M|} \right) \tag{8.1}$$

where:

$M = \{m_1, m_2, ..., m_j\}$ is the set of process models in the original collection.

$Sc = \{rpf_1, rpf_2, ..., rpf_n\}$ is the multiset of all the RPFs rpf_i detected in the original process models collection M. A given RPF rpf_i can reoccur multiple times within the same $m_i \in M$, and/or in different models in M, and thus multiple times in the multiset Sc.

$S_k \subset Sc$ is the set of RPFs participating in the process model c_k.

$t: \; Sc \rightarrow \mathbb{N}$ is a function counting how many times an RPF $rpf_i \in Sc$ is present in $m_i \in M$, counting each time rpf_i is found in the same m_i.

$m: \; Sc \rightarrow \mathbb{N}$ is a function returning the number of process models in the set M, in which the RPF $rpf_i \in Sc$ is present at least once.

For deriving the intensity (*inte*) of a class ($c_k \in C$) we normalise its representativeness ($repr(c_k)$) to 100 and round to the closest integer. This is described by the following formula:

$$inte(c_k) = \left\lfloor \frac{100 \cdot repr(c_k)}{\sum_{i=0}^{|C|} repr(c_i)} \right\rceil \tag{8.2}$$

The metric of *intensity* is therefore a proportional number of the times and process models for which an RPF is detected in the collection. Since a process model of the workload mix can be synthesised by many RPFs, the *representativeness* and *intensity* are cumulative metrics for each RPF in a process model. Finally, in phase 5 we define the execution behaviour of the workload mix. More specifically, we need to link each process model of the workload classes to artifacts that are required for its execution (e. g., script tasks or web services) and define probabilities with which a process model will take over the control-flow. Essentially, this phase corresponds to the executable refinement phase presented in Section 7.2.3.

The application of WINE4WfMSs method is demonstrated in Chapter 9, where we apply it in a real world case study to derive a workload mix.

8.2. Toolchain Architecture

In this section we present the architecture of a toolchain that supports the WINE4WfMSs method. Section 8.2.1 presents architectural principles to be followed by cloud-aware applications, Section 8.2.2 discusses some of the cloud patterns that were considered in our design and Section 8.2.3 presents an overview of our toolchain's architecture.

8.2.1. Cloud-Aware Architecture Principles

In order to benefit from cloud environments a cloud-aware application should follow specific design principles [Feh15]. To this effect, Fehling et al. [FLR+14] recognise the Isolation, Distribution, Elasticity, Automated Management, Loose Coupling (IDEAL) set of properties, that a paradigmatic cloud-aware application should satisfy. Namely, a cloud application should fulfil to the maximum possible extend the following properties:

Isolated: the majority of application components should be stateless;

Distributed: the application's functionality should be distributed across multiple components;

Elasticity: it should be possible to scale the cloud application based on changing workload (i. e., to add or remove cloud resources in a flexible way);

Automated Management: there should be a way to independently handle runtime management (e. g., elastic scaling or failure resiliency) and

Loose Coupling: the application should maintain the platform, reference, time and format autonomies as defined by the loose coupling principle.

Hence, cloud applications should follow the architectural principles of loose coupling and distributed applications that simplify the fulfilment of the aforementioned properties [Feh15]. As we will show in Section 8.2.3 our architecture was designed with these principles in mind. Overall, our application is composed out of distributed components that interact with each other towards the completion of diverse actions. The architecture of the business logic components follows the REST architectural style. Namely, the business logic of the WINE4WfMSs process is exposed as RESTful services that comply with the following REST properties identified by [HKLS14]:

Layered Client Server: the existence of a client on an upper layer that uses the functionality of the underlying (lower) server layer;

Cache: a server response should be marked as "cacheable" or "non-cacheable". In the first case the response may be saved locally in order to be faster accessible for future requests;

Stateless Server: each subsequent request to the server should be independent from other requests. Namely, sessions should not be used;

Uniform Interface: the client and the server interact to each other through a well-defined set of Create, Read (Retrieve), Update (Modify), Delete (Destroy) (CRUD) methods. Especially for the HTTP interaction protocol is used, these methods map to POST, GET, PUT and DELETE [FGM+99];

Identification: the location of the resources should be easily identifiable and addressable in the distributed environment. This is achieved through Uniform Resource Identifier (URI);

Manipulation through Representations: there is a distinction between a resource and each representation corresponding to the client that processes it;

Self-Descriptive Messages: the structure of a request should provide all information that is needed to understand and process it and

Hypertext as the Engine of Application State (HATEOAS): the state of a client is driven by the resources it can access.

In the following section we present the architecture of the tool chain that is designed with respect to the aforementioned properties and architectural styles.

8.2.2. Cloud Design Patterns Selection

For designing the toolchain architecture we considered the architectural principles described in Section 8.2.1. As the design of a cloud application constitutes a complex task we also follow cloud application design patterns identified by Fehling [Feh15]. Fehling [Feh15] divides these patterns into five major categories: (i) cloud computing fundamentals; (ii) cloud offerings; (iii) cloud application architecture; (iv) cloud application management; and (v) composite cloud application.

Each of the categories need to be individually assessed by the developer of a cloud application before proceeding to the design of the architecture. In the rest of this subsection we discuss our decision for selecting a subset of the recognised design patterns as guidelines towards the design of the

Category	Design Pattern	Explanation
Cloud Computing Fundamentals	Once-in-a-lifetime Workload (p. 33)	As this is a proof-of-concept prototype, we expect low workload during a certain event.
	PaaS (p. 49)	We utilise the PaaS service model for hosting application components
Cloud Offerings	Node-based Availability (p. 95)	We need the node-based availability provided by the cloud provider for hosting our application components.
	Hypervisor (p. 101)	We need this functionality for being able to provision and decommission virtualised components.
	Relational Database (p. 115)	We use relational database components for storing data needed for the functionality of our application.
	Key-Value Storage (p. 107)	We use key-value storage for storing unstructured data needed for the functionality of our application.
Cloud Application Architecture	Loose Coupling (p. 156)	We apply the principles of loose coupling in our architecture.
	Distributed Application (p. 160)	We follow the principles of distributed application design in our architecture.
	Stateless Components (p. 171)	We design the components implementing interface, or business logic to not handle the state of the application.
	User Interface Components (p. 175)	We design some of the application components to implement user interface.
	Processing Components (p. 180)	We design some of the application components to implement processing.
	Data Access Components (p. 188)	We design some of the application components to be used for data access.
Composite Cloud Application	Three-Tier Architecture (p. 290)	The application components are categorised in three layers (presentation, business logic and infrastructure) for enabling the scaling with regards to individual needs.

Table 8.1.: Selected cloud design patterns [FLR+14] for our architectural design

architecture. The applied design patterns and the reasoning for selecting them are summarised in Table 8.1. The selected design patterns are marked with the corresponding page on which they appear in Fehling et al. [FLR+14] for quicker access. The first two categories refer to properties that are usually provided by the cloud provider and thus should be considered when defining hosting options for the cloud application. With respect to the *cloud computing fundamentals* category we first need to think of the workload that will arrive to our application. The proposed cloud application is implemented as a prototypical proof-of-concept of the workload generation method. Thus, it is expected to receive low workload over a certain period of time. In this regards, we match to the *once-in-a-lifetime workload* pattern were the application will only be used during a certain event. Afterwards, we need to define the service model on which we will build and deploy our application. In this case, a cloud provider that offers a *Platform as a Service (PaaS)* service model are required for deploying middleware components (e. g., databases and WfMSs) as well as functional components of the application.

The functional and non-functional behaviour provided in the cloud runtime environments are described by patterns in the category of *cloud offerings*. We set some of the properties as defined by the design patterns in this category, as the minimum set of requirements to select cloud providers to host our application components. For instance, we base on the design pattern of *node-base availability*, where the provider guarantees availability of individual nodes that host middleware, servers or application components. The cloud offerings category also includes design patterns that describe different behaviours of the cloud provider under different workloads and define the conditions under which these design patterns should be selected. Out of these design patterns we find applicable the *hypervisor* pattern which advises the virtualisation of the components to easily provision and decommission them on the servers. Offerings regarding data storage should at least support the *relational database* and *key-value storage* design patterns as we use both of these options for implementing different databases for our application.

The *cloud application management* category includes application components that execute management plans during the runtime of the application. Hence, in principle our design focuses merely on the design patterns that lay under *cloud application architecture* and *composite cloud application* categories. As discussed in Section 8.2.1 our architecture follows the two fundamental architectures of *loose coupling* and *distributed application*. Moreover, we implement the application through *stateless components,* i. e., components that do not handle the state of the application and ease scaling-out. The diverse components realise different functionalities of our application by implementing the design patterns of *user interface components*, *processing components* and *data access components*. The correspondence of the implemented functional components to these design patterns is discussed in the following section.

Finally, the *composite cloud application* category reveals common design pattern combinations revealed in many real world practice application scenarios. In this case, we focused on the *three-tier cloud application* architecture which separates the presentation, business logic, and data handling components in different architectural layers. Our application also follows this architectural style as it allows separate tiers to scale with regards to individual need. In the next section we show how the aforementioned design patterns are applied to the architecture of the toolchain.

8.2.3. Architecture Overview

An overview of the toolchain's architecture is shown in Figure 8.1. As discussed in Section 8.2.2 we follow a distributed architecture. More particularly, the toolchain components are distributed across three layers (i. e., Presentation, Business Logic as a Service, Middleware as a Service). The separation of the layers provides increased scalability on the cloud application and is preferred when there are more than one data sources [Feh15].

The *Presentation* layer contains the user interface component, i. e., the components with which the user may request the execution of a service and receive the visualised results. With respect to the *cloud application architecture category* this component implements a user interface component (see Table 8.1). More particularly, the presentation layer is realised in the form of a *Mashup (Web App Hybrid)*. The mashup is basically a hybrid web portal application, which brings together content from diverse sources and displays it into a single User Interface (UI). By implementing a mashup one may achieve easy and fast integration, frequent usage of open API's and data

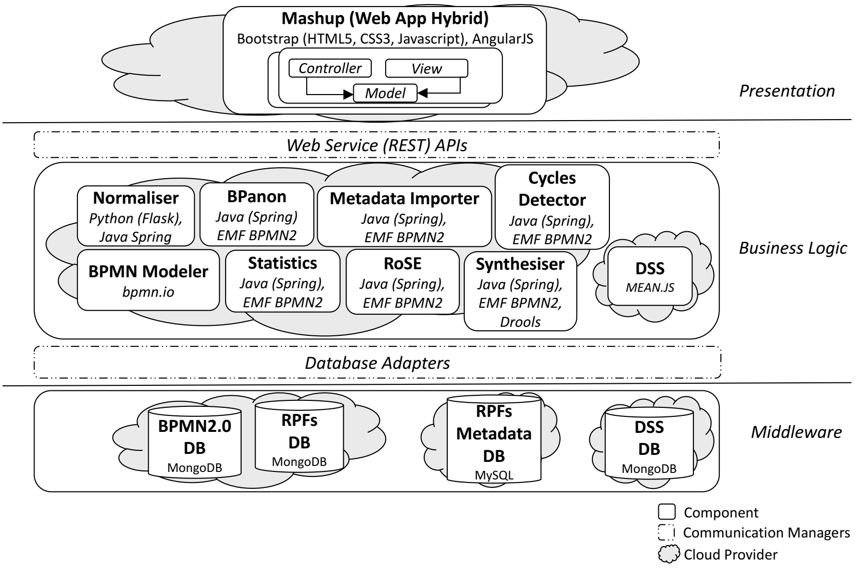


Figure 8.1.: Workload mix generation toolchain architecture overview and implementation details

sources. In more detail, the architecture of the *Mashup (Web App Hybrid)* relies on the Model-View-Controller (MVC) design pattern [BMR+96], which is a broadly applied software design pattern for implementing UIs. The MVC design pattern divides the components into three types: model, view and controller. The model component manages the application's data logic and functional rules, the view component is related to any visual representation of information, while the controller acts as a broker that converts commands to model or view [BMR+96]. A characteristic property of the MVC design pattern is that it decouples the views from the models, thus enforcing the flexibility of the system and its easier adaptability to change. Moreover, it enables the attachments of multiple views to the system to provide different adaptations.

The *Business Logic* layer contains the business logic of the application, which is offered as distributed services. These components offer a REST

API and implement different patterns of the *cloud application architecture category* (Table 8.1). Moreover, these components also implement the Cross-Origin Resource Sharing (CORS) technique, which basically relaxes the security concept of *same-origin policy*[1] implemented by web browsers. This prevents JavaScript code from making requests against a different origin (e. g., different domain) than the one from that it is served. Each component of this layer will be discussed in further detail in the following section.

The *Middleware* layer hosts the middleware components (i. e., databases and WfMS), with which the services of the Business Logic layer interact. The communication among the layers is enabled through *Communication Managers*, which are aware of the individual ways to access each service component. Communication Managers implement messaging design patterns (i. e., Message-oriented Middleware, At-least-once Delivery [Feh15]) to enable consistent communication. Through the implementation of diverse design patterns they provide a seamless way of communication among the components. The communication managers also enable the flexibility of the system, in case a change in the implementation of a component occurs.

8.3. Toolchain Components Description

In this section, we discuss each component shown in Figure 8.1 and analyse its functional role in the toolchain, related facts of architectural importance, the underlying inter-component interactions (see Figure 8.2), and its implementation details. As discussed in Section 8.2.3, the inter-component interactions are enabled through the communication layers that ensure an increased system autonomy. For the sake of simplicity, in the following we omit the intermediate layers and describe only the end-to-end component interaction.

[1]W3, Cross Origin Resource Sharing,
URL: `https://www.w3.org/Security/wiki/Same_Origin_Policy`

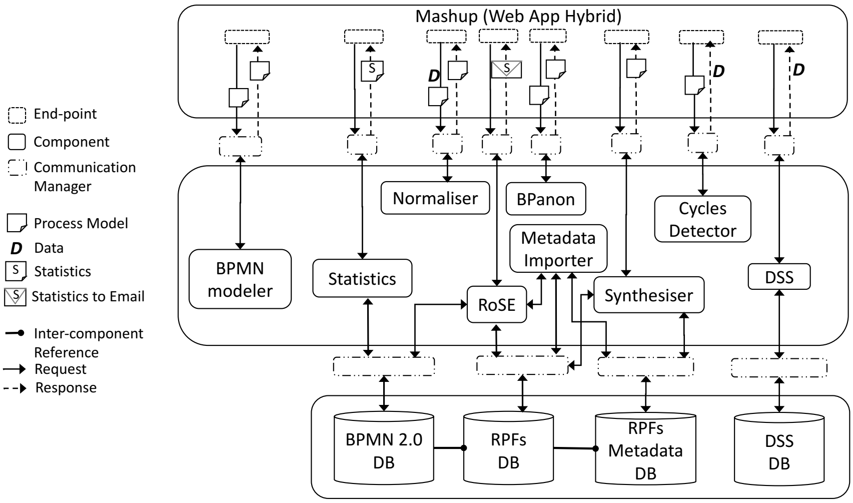


Figure 8.2.: Workload mix generation toolchain components interaction

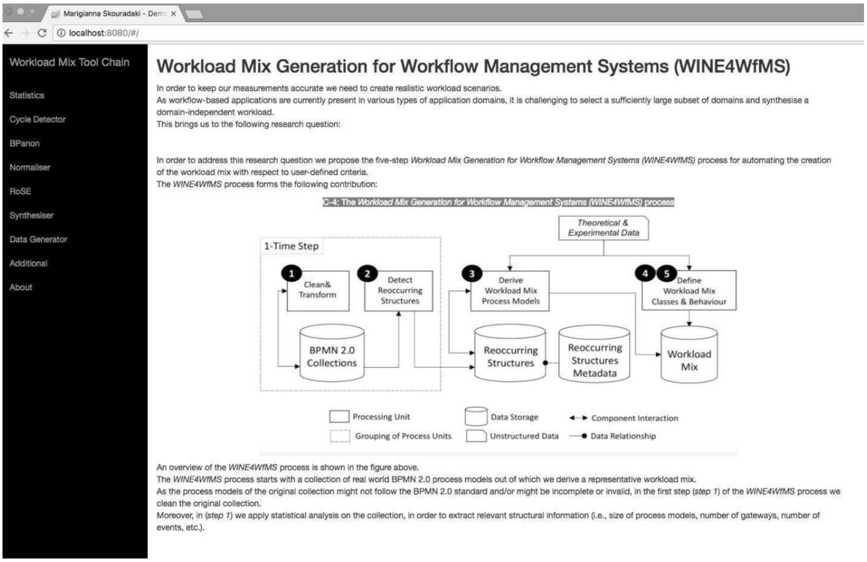
8.3.1. Mashup (Web App Hybrid)

Functionality: Provides a UI with which the user interacts to initialise requests and receive responses of the underlying service components.

Architectural Facts: The UI follows the mashup paradigm to bring the functionality of diverse distributed components together to a consolidated UI. Moreover, we implement the MVC design pattern that decouples the views from the models and fosters the flexibility of the system.

Inter-Component Interactions: The mashup web application acts as a request initiator for any component of the *Business Logic*. The type of requests and responses varies with respect to the service purpose and specification. Namely, a request or response may be synchronous or asynchronous and it may contain a complete process model file or data query. These variations are discussed in detail for each component individually in the following subsections.

Implementation Details: The purpose of the UI interface is to create a



Workload Mix Generation for Workflow Management Systems (WINE4WfMS)

In order to keep our measurements accurate we need to create realistic workload scenarios.
As workflow-based applications are currently present in various types of application domains, it is challenging to select a sufficiently large subset of domains and synthesise a domain-independent workload.
This brings us to the following research question:

In order to address this research question we propose the five-step *Workload Mix Generation for Workflow Management Systems (WINE4WfMS)* process for automating the creation of the workload mix with respect to user-defined criteria.
The *WINE4WfMS* process forms the following contribution:

C-4: The *Workload Mix Generation for Workflow Management Systems (WINE4WfMS)* process

An overview of the *WINE4WfMS* process is shown in the figure above.
The *WINE4WfMS* process starts with a collection of real world BPMN 2.0 process models out of which we derive a representative workload mix.
As the process models of the original collection might not follow the BPMN 2.0 standard and/or might be incomplete or invalid, in the first step (*step 1*) of the *WINE4WfMS* process we clean the original collection.
Moreover, in (*step 1*) we apply statistical analysis on the collection, in order to extract relevant structural information (i.e., size of process models, number of gateways, number of events, etc.).

Figure 8.3.: Screenshot of the toolchain's user interface

user-friendly environment that enables the seamless interaction and easy completion of the user goals. Hence, we choose state-of-art technologies for web application design. Bootstrap[1] is a Hypertext Markup Language (HTML), Cascading Style Sheets (CSS)[2] and JavaScript framework that eases the front-end development and enables the design of responsive web pages. In addition, we use AngularJS[3] for declaring dynamic views in our web application. A screenshot of the toolchain's UI is shown in Figure 8.3.

8.3.2. Normaliser

Functionality: Some of the BPMN 2.0 process models collected for our research were not serialised according to the BPMN 2.0 standard serialisation (see Chapter 3). The purpose of this component is to receive a

[1]Bootstrap, URL: `http://getbootstrap.com`
[2]CSS, URL: `https://www.w3.org/standards/webdesign/htmlcss`
[3]AngularJS, URL: `https://angularjs.org`

process model serialised in the IBM custom serialisation or the BPMAI serialisation (Section 3.2) and convert it to the standard serialisation as defined by the BPMN 2.0 standard [ISO13].

Architectural Facts: The component is a stateless component and implements the processing component design pattern (Table 8.1). Namely, the component takes a file in the original custom serialisation and processes it to transform and return it to the standard format. As there are two different types of serialisation to convert, this service is composed out of two different sub-services one transforming the IBM custom serialisations to standard and one transforming the BPMAI serialisation. Subsequently, the normaliser acts as an orchestrator that calls the appropriate service with respect to the serialisation of the file it must transform.

Inter-Component Interactions: For completing its goal the component needs to receive a request from the client (in our case the UI) that contains at least information regarding the type of the document's custom serialisation, as well as the document itself. The document is processed with respect to its original custom format and the transformed result is returned to the client.

Implementation Details: The orchestrator service of the normaliser component is implemented with the Java Spring Framework[1], the service that transforms the IBM processes to the standard format is implemented in Python and the one that transforms the BPMAI process models is using Java Spring to wrap the BPMAI2BPMN converter proposed by Pietsch and Yazdi [PY14] to a RESTful service.

8.3.3. Statistics

Functionality: The purpose of this component is to calculate statistics of a collection of BPMN 2.0 process models [GL06]. These statistics can afterwards be utilised by the user of the workload generation process for

[1]Java Spring Framework, URL: `https://spring.io`

taking pivotal decisions regarding the synthesis of process models (see Chapter 9).

Architectural Facts: The component is a stateless component and can be seen as a data access and processing component (Table 8.1). For each calculation of size statistics the component iterates and parses the whole collection of process model files. Although this calculation could be accelerated through storing some process model metadata on a new database, the calculation is completed in a few seconds for a collection of thousands process models, while hosting one more database on the cloud would have caused additional costs. Therefore, in the trade-off of calculation time versus database hosting costs we preferred the first approach but in case the size of the process models collection increases significantly then this decision should be reassessed. Moreover, the calculations executed by this component rely on a database with a rather stable state (i. e., it is rarely expected to have process model additions or removals on the database). Therefore, in the future, the client calling this component, may also implement caching for a more efficient result retrieval [FLR+14].

Inter-Component Interactions: For completing its goal the component receives a simple request from the user indicating that a calculation of collection statistics is needed. For this the component accesses the database in which the complete collection of BPMN 2.0 process models is stored (BPMN 2.0 DB in Figure 8.1). Afterwards, the statistics component calculates the size metric for each process models and returns to the client a response containing the statistics in a Comma-Separated Values (CSV) format.

Implementation Details: The component is implemented in Java. It uses the Java EMF[1] of BPMN 2.0 Metamodel[2] for processing the BPMN 2.0 file and is converted to a RESTful service with the Java Spring Framework.

[1]Java EMF, URL: `http://www.eclipse.org/emf`
[2]BPMN 2.0 EMF, URL: `http://www.eclipse.org/modeling/mdt/?project=bpmn2`

8.3.4. BPanon

Functionality: The purpose of this component is to accept a process model and pseudonimise it or anonymise it, while maintaining its executional behaviour. Namely, this component implements the BPanon method presented in Section 3.1.

Architectural Facts: The component is a stateless component and implements the processing component design pattern (Table 8.1). Internally, this component is separated into four different management layers: the *interaction management layer* refers to the part of the implementation that receives and decodes the client request for the business process anonymisation or pseudonymisation; the *anonymise management layer* implements the business process anonymisation; the *rename management layer* provides the new words to be used and finally the *registry management layer* logs the changes to a registry. Each management layer contains more sub-components to complete their functionality. More details on the internal architecture of the BPanon component can be found in previous work [SFL+15].

Inter-Component Interactions: The component receives a request from the client containing the process model to anonymise or pseudonymise, processes it and returns to the client a response containing the anonymised process model.

Implementation Details: The component is implemented as RESTful service with the Java Spring Framework.

8.3.5. Cycles Detector

Functionality: The purpose of this component is to detect if there is a cyclic structure in a given process model. Namely, it implements the algorithm presented in Section 6.3.1.

Architectural Facts: The component is a stateless component and implements the processing component design pattern (Table 8.1).

Inter-Component Interactions: The component receives a request from the client containing the process model to check for cyclic structures and returns a true or false response to the client, expressing if the process model contains a cyclic structure or not respectively.

Implementation Details: The component is implemented as a RESTful service with the Java Spring Framework and uses the Java EMF BPMN 2.0 Metamodel for processing BPMN 2.0 files.

8.3.6. Decision Support System (DSS)

Functionality: The purpose of this component is to provide functionality of a Decision Support System (DSS) for WfMSs benchmarking by allowing user-defined queries against a knowledge base the schema of which is based on the conceptual model presented in Figure 4.1. The data of the DSS knowledge base are derived from standard middleware benchmarks [Tra92b; TPC15; Sta15; Sta07], as well as custom benchmarks for WfMSs [SFP+16; BBD10; DES08; Act11; Rol13].

Architectural Facts: According to Power et. al. [PSB15] the different types of DSS can be summarised as follows: (i) a *data-driven DSS* provides access to large knowledge bases in order to extract information; (ii) a *communication-driven DSS* supports the shared access on a specific task where more than one person is involved in working on it; (iii) a *document-driven DSS* data is retrieved and manipulated in form of a document; (iv) a *knowledge-driven DSS or expert system* provides support to problem-solving in terms of defined rules and procedures and (v) a *model-driven DSS* provides functionality by offering different models for which the data and parameters are provided by the user. The three different components inside a DSS are: (i) the knowledge base, where all relevant information is stored, (ii) the conceptual model that defines different decision criteria and (iii) the user interface that presents the required output. In our case, the provided DSS is offered in the form of a document driven DSS were the data are retrieved and manipulated as unstructured information. More architectural facts regarding the DSS component can be found in previous work [SAB+16].

Inter-Component Interactions: The component receives a request from the client, containing data for querying the knowledge base. In turn the DSS component forms a query and executes it against a knowledge base (see DSS DB in Figure 8.2). The results are formed in a response and returned back to the client.

Implementation Details: For the implementation of the DSS client we used the MongoDB, Express, AngularJS, NodeJS (MEAN) stack[1], which is a full stack JavaScript framework that simplifies web application development.

8.3.7. RoSE

Functionality: The purpose of this component is to detect and extract RPFs contained in a collection of BPMN 2.0 process models. Namely, this component realises the algorithms presented in Section 6.3.2 and Appendix A.

Architectural Facts: The component is a stateless component and implements the data access and processing component design patterns (Table 8.1). It also executes in an asynchronous mode due to the complexity of the RoSE algorithm. As the calculations executed by this component rely on a database with a rather stable state (i. e., it is rarely expected to have process model additions or removals on the database), the client calling this component may also implement caching for a more efficient result retrieval [FLR+14].

Inter-Component Interactions: For the calculation of the reoccurring structures the component first receives a request from the client that initiates it. Then the RoSE component interacts with a database, in which the complete process model collection is stored (cf. BPMN 2.0 DB in Figure 8.2). As soon as the component finds RPFs it stores them into a separate database (RPF DB in Figure 8.2). At the end of the RPFs detection process the

[1]Linnovate, MEAN, URL: `http://mean.io`

RoSE component sends a response to the client containing information for access to the RPF DB, as well as CSV structured metadata information concerning the discovered RPFs files.

Implementation Details: The component is implemented as RESTful service with the Java Spring Framework and uses the Java EMF BPMN 2.0 Metamodel for processing BPMN 2.0 files.

8.3.8. Metadata Importer

Functionality: The purpose of this component is to create the RPFs metadata. Namely, it calculates the metadata to characterise the RPFs, as described in Section 7.2.1.

Architectural Facts: The component is a stateless component and implements the data access and processing component design patterns(Table 8.1). Internally its architecture contains a parser that parses the files of the RPFs, a metadata calculator, that calculates the metadata of the RPFs and an importer that imports the calculated metadata to the database.

Inter-Component Interactions: In contrast to the other components of the business logic layer this one is not initiated by the user-interface. When the RoSE component finishes its execution, it automatically triggers the initiation of this component to calculate and import the RPFs metadata in a new database. The component must respond to the RoSE algorithm regarding failure or success.

Implementation Details: The component is implemented as a RESTful service with the Java Spring Framework and uses the Java EMF BPMN 2.0 Metamodel for processing BPMN 2.0 files.

8.3.9. Synthesiser

Functionality: The purpose of this component is to consistently synthesise RPFs to generate synthetic, non-executable process models with respect to user defined criteria. Namely, it implements the RPFs selection and synthesis presented in Section 7.2.2.

Architectural Facts: The component is a stateless component and implements the data access and processing component design patterns (Table 8.1). For evaluating efficiently if two RPFs can be connected with each other this component exploits a rule engine.

Inter-Component Interactions: Initially, this component receives a request from the client containing information regarding the criteria with respect to which the synthetic process model should be generated. It then uses the criteria to form a query against the RPF metadata DB to retrieve these RPFs that can be used for generating synthetic process models. In case a process model could be successfully generated the component returns the generated process model or false if no combinable RPFs were found.

Implementation Details: The component is implemented as RESTful service with the Java Spring Framework and uses the Java EMF BPMN 2.0 Metamodel for processing BPMN 2.0 files. Moreover, we use the Drools rule engine[1] for rules execution.

8.3.10. BPMN Modeler

Functionality: This component can be used from the user for converting the non-executable BPMN 2.0 process model to an executable equivalent. Namely, this component is used for following the guidelines presented in Section 7.2.3.

Architectural Facts: The component is a stateless component (Table 8.1).

Inter-Component Interactions: For editing a process model the component must be initialised with a request that contains the non-executable process model as well as other relevant data, for example, initialisation values for the variables and probabilities to follow the existing control flow paths.

[1]Red Hat, Drools, URL: `http://www.drools.org`

Then, the component updates the process model and returns it to the client.

Implementation Details: The component is implemented with bpmn.io[1] framework.

8.3.11. Middleware

This section discusses the components of the Infrastructure as a Service (IaaS) layer. Here, we omit the descriptions of the inter-component interactions as they have already been described in the previous subsections.

8.3.11.1. BPMN 2.0 DB and RPF DB

Functionality: The purpose of these databases is to store the complete BPMN 2.0 process models and RPFs collections respectively.

Architectural Facts: The BPMN 2.0 and RPF databases need to store a big amount of data in form of files and allow a seamless access to them from the components of the Business Logic. A document-oriented database seemed the most appropriate for our purposes of usage.

Implementation Details: For the implementation of these databases we use MongoDB [2] and store the process models and RPFs as documents. The database is deployed on the OpenShift [3] PaaS cloud provider.

8.3.11.2. RPF Metadata DB

Functionality: The purpose of this component is to store metadata information regarding the RPFs structural properties. Namely, this component stores information as it is described in Section 7.2.1.

[1]bpmn.io, URL: `http://www.bpmn.io`
[2]Mongo DB, URL: `https://www.mongodb.com`
[3]OpenShift, URL: `https://www.openshift.com`

Architectural Facts: Metadata information comprise structured information and may be stored in relational databases for easy access. Hence, for this purpose we have preferred a relational database.

Implementation Details: A MySQL[1] relational database is used for the realisation of this component.

8.3.11.3. DSS DB

Functionality: The purpose of this component is to store data explained in Sections 2.1.1, 2.1.2 and 4.1 in order to provide a knowledge base for the described DSS.

Architectural Facts: The data stored in this component are derived from different benchmark scenarios and therefore they comprise unstructured information. For these purposes we considered a document database to be more appropriate.

Implementation Details: A MongoDB document database is used for the realisation of this component.

8.4. Related BenchFlow Software

The architecture described in Section 8.2.3 supports the complete end-to-end WINE4WfMSs process. Nevertheless, we also present two additional software solutions that can be used complementary to our toolchain. BP-Meter [IFP15] is an application that applies static analysis to a collection of BPMN 2.0 process models. The analysis includes more than 100 metrics including variability of aggregated size metrics (e. g., number of nodes, number of gateways, number of events, etc.), disaggregated metrics (e. g., number of script tasks, number of exclusive split gateways, etc.) and metrics on the control flow and data flow of the process (e. g., parallel gateway fanin, data input distribution). This application can be used not only for a deeper understanding of the collection's composition, but also for exporting reports

[1]Oracle, MySQL, URL: `https://www.mysql.com`

that can afterwards be utilised by data mining and analysis tools for drawing additional conclusions. For example, in Chapter 9 we showcase how the results of clustering analysis can be combined with the workload mix generation process for deriving a workload mix that is structurally representative of a specific collection.

The Interactive Dashboard for Workflow Engine Benchmarking [BMHW16] is an effort to aggregate the results derived from diverse benchmarks on WfMSs into a common data model and bring them to a common view. Through this application the end-users, developers and researchers are able to analyse and compare the behaviour of the systems in a straight-forward fashion. The experiment configurations, workload mix details and results used in the benchmarks described in Chapters 5 and 9 were also provided for integration to the interactive dashboard.

8.5. Chapter Summary

In this chapter we presented a four phased method for deriving a structurally representative workload mix with respect to a given BPMN 2.0 process models collection. The WINE4WfMSs process employed information and methods presented in Chapters 3, 4, 6 and 7 for the generation of the workload mix. Afterwards we presented an architecture and a prototypical implementation of a cloud aware, web-based toolchain that supports the WINE4WfMSs process. For the design of the demonstrated architecture we studied and applied selected cloud design patterns [Feh15] and followed the fundamental cloud architectures, as for example distributed application and loose coupling. For the prototypical implementation of the application we selected state-of-art technologies and bring them together in a web application mashup. Moreover, we presented additional BenchFlow software that can be used complementary to our toolchain.

Chapter 9

Deriving a Workload Mix - A Real World Case Study

"If the statistics are boring you've got the wrong numbers."

E. Tufte

The purpose of this chapter is to showcase the WINE4WfMSs process by means of an exploratory case study. To this end, we apply the complete WINE4WfMSs process to the process model collection discussed in Section 3.2. After generating the workload mix we use it to execute a benchmark for BPMN 2.0 WfMSs [FSP+17]. The detailed description of the benchmark environment, experiments setup, performance metrics and emerging results [FSP+17] are not a contribution of this work. However, for the sake of completeness we explain them shortly and report the most important findings. We also publish the workload mix and emerging results in the interactive dashboard of the "Performance and Conformance Benchmarking for Workflow Engines" (PeACE) initiative[1] for easy access by third parties.

The remainder of this chapter is structured as follows: Section 9.1 demonstrates the application of the WINE4WfMSs process on a collection of thousands of real world practice process models to result in a workload mix that structurally represents this collection; Section 9.2 describes shortly the benchmark in which we applied the derived workload mix and reports the most important findings; and Section 9.3 summarizes the chapter.

9.1. Deriving a Workload Mix

In the following, we apply each phase of the WINE4WfMSs process as discussed in Section 8.1 to the BPMN 2.0 process models collection discussed in Chapter 3.

9.1.1. Phase 1: Cleaning and Analysis of the Initial Process Models Collection

The collection presented in Section 3.2 contains 12,624 real world BPMN 2.0 process models. Event logs of the process models were not shared with us, since they constitute a valuable corporate asset for the companies. Therefore, a behavioural analysis of the collection was not possible at this point. Before proceeding to the analysis of the process models collection we should ensure that the analysed sample is valid and compliant with the constraints set by the RoSE algorithm presented in Section 6.3 and the BenchFlow framework [FIP15], which will be used for executing the benchmark. To this end, the analysed subset of the original collection should satisfy the following requirements: i) should not contain incomplete or invalid process models; ii) should not contain collaboration elements and iii) should not contain cyclic process models.

As discussed in Section 3.2 the process models collected from the "IBM Banking Process and Service Models", the "IBM Insurance Process and

[1]PeACE, URL: `https://peace-project.github.io/about.html`

Service Models" and the BPMAI process models were not compliant with the BPMN 2.0 standard serialisation. Therefore, we need to convert them into BPMN 2.0 compliant process models. For the IBM process models we used the transformation rules shown in Section 3.2 that are realised by the normaliser component (Section 8.3.2). All IBM process models could be transformed into the standard serialisation without detecting any invalid process models. For the BPMAI process models we succeeded in transforming 7,515 (87.7%) process models of the BPMAI collection to the standard serialisation, while the rest of the models were marked as invalid. In total, after normalisation we discovered 5,491 process models containing collaboration elements and 1,908 incomplete and/or invalid process models. At this point it should be taken into consideration that these statistics overlap for some process models. For instance, a process model may contain collaboration elements and be incomplete or invalid. For the "clean" collection of 3,779 process models (i. e., invalid, incomplete process models and process models containing collaboration elements excluded) in total 532 (14.07%) cyclic process models were detected (Section 8.3.5). After removing the aforementioned process models we resulted in a collection of 3,247 process models that can be analysed statistically (Section 8.3.3) and by the RoSE algorithm (Section 8.3.7).

For a better comprehension of the analysed collection we present its structural composition in Figure 9.1, which shows the number of process models (# Process Models) with a specific size for the clean collection. The size for a process model is defined as the total number of nodes (i. e., FlowNodes [ISO13, p. 97]) per process model [GL06]. As shown in Figure 9.1, the process models have a minimum size of 3 ($size = 3$) while their maximum existent size is 120 ($size = 120$). For values between 3 and 20 ($3 \leq size \leq 20$) we observe a bigger concentration of data, especially in the range of sizes between 5 and 12 ($5 \leq size \leq 12$, first and third quartile respectively) and a mean size of 10.44 ($\overline{size} = 10.44$). For the rest of sizes we observe a sparser distribution [Observation 1]. Following these observations we conclude that a process model with size between 5 and 20 ($5 \leq size \leq 20$) represents the vast majority of the collection's process models, while a process model

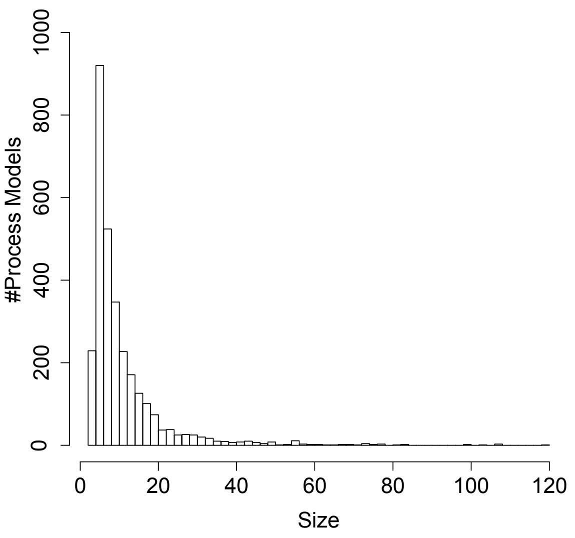


Figure 9.1.: Process model size distribution in the resulting BPMN 2.0 collection

with size between 20 and 40 ($20 < size \leq 40$) represents a smaller subset of the collection [Observation 2]. Process models with sizes greater than 40 ($size > 40$) represent only a minority of the collection's process models. With respect to the above statistics we can see that process models with size between 5 and 32 ($5 \leq size \leq 32$) represent 81.99% of the collection.

As a final step of this phase we apply a clustering analysis [Iva14]. Clustering analysis is a method that assists towards the grouping of a set of objects into clusters, in such way that similar objects will belong to the same cluster. Through the clustering analysis we are able in phase 2 to obtain a more fine grained insight on the actual structural characteristics of the process models. This has a triple utility: i) characterise the discovered RPFs with respect to the obtained clusters; ii) synthesise process models that reflect the characteristics of different clusters; and iii) set the foundation for

Clusters	Representativeness	Description
Cluster 1	35%	1 Start Event, 2 End Events, 4 Tasks/Activities, 1 Exclusive Gateway
Cluster 2	29%	1 Start Event, 2 End Events, 6 Tasks/Activities, 2 Exclusive Gateways
Cluster 3	19%	1 Start Event, 3 End Events, 11 Tasks/Activities, 4 Exclusive Gateways, 1 Parallel Gateway
Cluster 4	11%	1 Start Event, 3 End Events, 16 Tasks/Activities, 5 Exclusive Gateways, 1 Parallel Gateway

Table 9.1.: Clusters of representative process models (adapted from [Iva14])

categorising the process models into workload classes.

A comprehensive clustering analysis of the collection is executed by Ivanchikj [Iva14] who concluded in six clusters each one representing a different group of process models in the collection. We show clusters 1 to 4 in Table 9.1, while we omitted clusters 5 and 6 as they describe a negligible percentage of the collection (4% and 2% respectively). The original work of Ivanchikj [Iva14] describes the clusters in more detail, by giving also information considering the data elements, and the incoming (fan-in) and outgoing (fan-out) values of the participating gateways [GL06]. At this point, we omit these details as the discovered RPFs will already include this information. More sophisticated attributes such as the complexity of the

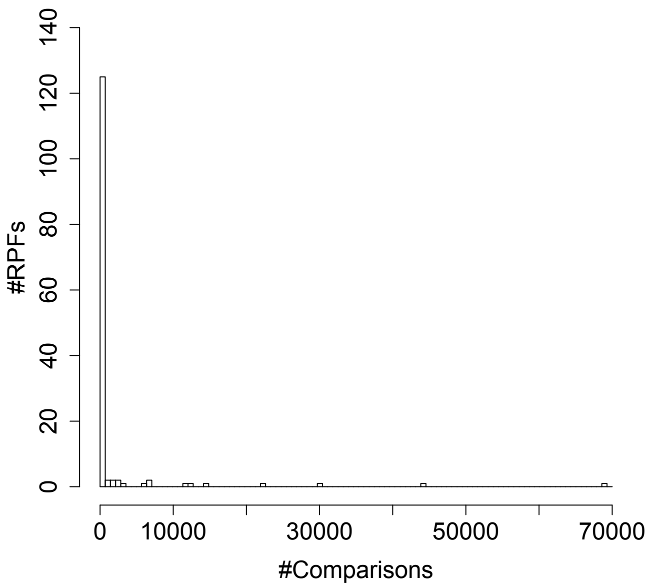
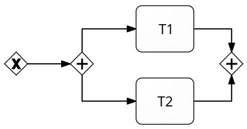


Figure 9.2.: Number of comparisons that extracted an RPF

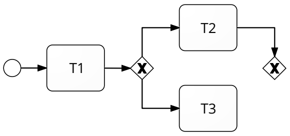
process models of each cluster and levels of parallelism are also not considered in the clustering analysis. However, we also expect the discovered RPFs to contain this information. It is also notable that the detected clusters comply with the observations regarding the sizes of the process models. Smaller process models (i. e., $size = 8$ in cluster 1) represent the 35% of the collection while bigger process models (i. e., $size = 26$ in cluster 4) represent a smaller percentage of the collection (11%).

9.1.2. Phase 2: Discovery of the Existing Relevant Process Fragments

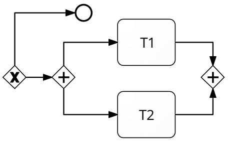
The first step of this phase is to discover the RPFs of the underlying collection. To this effect, we executed the RoSE algorithm (cf. Sections 6.3.2 and 8.3.7) against the collection. For measurement purposes we deployed our implementation on a Virtual Machine (VM) in a private cloud solution. The VM was configured with 8 MB memory and 4 CPUs, using the Ubuntu



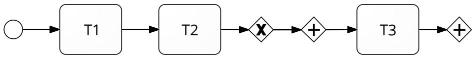
(a) *rpf*$_1$: 69,148 comparisons



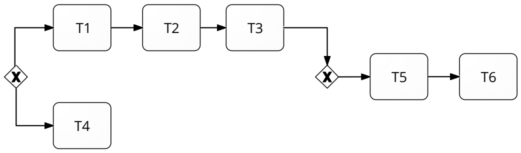
(b) *rpf*$_2$: 44,381 comparisons



(c) *rpf*$_3$: 30,227 comparisons



(d) *rpf*$_4$: 22,593 comparisons



(e) *rpf*$_5$: 14,245 comparisons

Figure 9.3.: Resulting RPFs of the BPMN 2.0 collection

LTS 14.04 operating system. The algorithm executed for 14 hours where it conducted 5,229,769 comparisons and resulted in 143 RPFs. Figure 9.2 shows the number of algorithm comparisons that produced each detected RPF (i. e., the occurrences of each detected RPF). As seen in Figure 9.2, most of the RPFs were detected in one comparison, while only a smaller number of RPFs was detected in more than one thousand comparisons.

The five most frequently detected RPFs are shown in Figure 9.3. The most frequent RPF shown in Figure 9.3a is formed by a simpler structure, while in the rest of the RPFs we can already observe more complex structures. The RPF shown in Figure 9.3a is also a substructure of Figure 9.3c [Observation 3]. Therefore, the count of occurrences of Figure 9.3a contains also the occurrences of Figure 9.3c. The fourth and fifth RPFs (Figures 9.3d

34.59%	20.16%	12.89%	12.20%	9.12%	4.82%	2.97%	1.87%	0.64%	0.34%	0.21%	0.07%	0.05%
Task (Script, Receive, Send, User)	Call Activity	Exclusive Gateway	End Event	Start Event	Parallel Gateway	Sub Process	Intermed. Catch Event	Inclusive Gateway	Intermed. Throw Event	Event Based Gateway	Complex Gateway	Boundary Event

Table 9.2.: Count of appearance of the existing elements in the cleaned BPMN 2.0 collection

and 9.3e) represent more complex structures. Generally, the first three RPFs (Figures 9.3a to 9.3c) confirm the frequent usage of the control flow workflow patterns and reveal combinations of those [VTKB03]. Regarding the clustering analysis we observe mappings of the discovered RPFs to the recognised clusters. More particularly, with a small deviation the Figures 9.3a and 9.3b reflect cluster 1 while Figure 9.3e reflects cluster 2 [Observation 4].

Along with the number of comparisons that extract a specific RPF (rpf_i) we calculate two additional types of metadata. Namely, the total number of occurrences of this RPF in the original collection and the number of process models that contain an RPF. For example, assume two process models A and B and an RPF rpf_1 such that rpf_1 is contained twice in model A and once in model B. Then rpf_1 is extracted by two comparisons (two complete matches from model A to model B), is found in two process models, and has three occurrences in the collection. These metadata are used in phase 4 for calculating the intensity with which an RPF (rpf_i) will participate in the workload mix.

Finally, despite the expressiveness of the BPMN 2.0 language, the detected RPFs contained only a small subset of the BPMN 2.0 constructs as is it is shown in Table 9.2. Although an extensive statistical analysis is beyond the scope of this work, we need to stress out the limited usage of the BPMN 2.0 language constructs, a fact that is already observed in the existing literature [MR08]. With respect to that we may also observe that the recognised clusters also describe process models that contain the mostly used BPMN 2.0 constructs of Table 9.2 (i. e., start event, end event, call activity, exclusive gateway and parallel gateway). Thus, the derivation of the workload mix process models discussed in phase 3 only considers these constructs. However, the proposed method for workload mix generation can be applied on any other process model collection and for deriving workload mixes that use a richer set of the BPMN 2.0 language constructs.

9.1.3. Phase 3: Derivation of the Process Models for the Workload Mix

At this point, the clusters indicate the structural attributes a process model should have, while the detected RPFs reveal the way in which these attributes are connected to each other. A combination of these results helps us towards defining process models that are structurally representative of the full range of process models of the original collection. Thus, by combining information detected by RPFs with respect to size characteristics of the collection with the results of clustering analysis we derive the process models shown in Figures 9.4a to 9.4c (cf. Observation 4). The RPFs synthesis presented in previous chapter (Section 8.3.9), works for synthesising process models out of two or more RPFs. Hence, the transformation of a unique RPF to a complete process model has been done manually.

For deriving the process models shown in Figures 9.4d and 9.4e we use the same information but we additionally synthesise two RPFs for deriving process models that reflect clusters 3 and 4 respectively (Section 8.3.9). The criteria with which we generate the synthetic process models of Figures 9.4d and 9.4e are shown in Table 9.3. Due to space limitations, we use the abbreviation *ST* for script task and *CA* for call activity on the depicted workload mix process models. In the following, we discuss in more detail the generation of the workload mix.

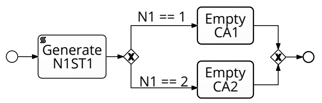
We begin by examining the first discovered RPF (rpf_1 shown in Figure 9.3a). The number of comparisons that extracted the rpf_1 (Figure 9.3a) are much higher than those of the other discovered RPFs (see Figure 9.2). This result indicates an extensive usage of this specific RPF. Moreover, this RPF contains parallelism which has previously indicated interesting performance impacts regarding resource utilisation (cf. Chapter 5). Last but not least, with a small deviation this RPF reflects cluster 1 in Table 9.1. For these reasons we decide to use this RPF in the workload mix and proceed with its transformation to a complete process model (Figure 9.4b). Taking into consideration Observa-

		Script Tasks	Call Activities	Exclusive Gateways	Parallel Gateways
Class 4	Crit. 1	6	-	2	-
	Crit. 2	5	-	2	2
Class 5	Crit. 1	6	-	4	1
	Crit. 2	-	9	1	1

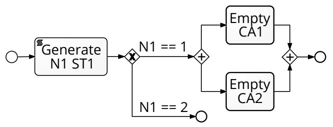
Table 9.3.: Criteria for synthesizing the workload mix process models Figures 9.4d and 9.4e

tion 3 we see that the third discovered RPF (*rpf*$_3$ shown in Figure 9.3c) is an extended structure of the first RPF (*rpf*$_1$ shown in Figure 9.3a). This eases our goal to complete *rpf*$_1$ (Figure 9.3a) into a complete process model. The leftmost exclusive gateway of the *rpf*$_1$ needs data for evaluating its condition and passing the control to a branch. Therefore, we added a script task to precede the exclusive gateway and generate the appropriate data. Then, we link the Open Connection Points (OCPs) (see Chapter 7) of the gateways with start and end events respectively. This results is the process model shown in Figure 9.4b forming class 2 (c_2) of the workload mix because the metadata of this RPF resulted to the second higher intensity for the workload mix class (see also phase 4).

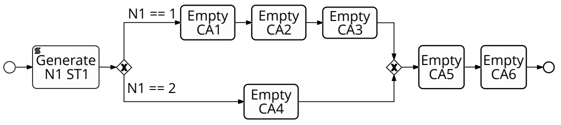
The second and third classes of the workload mix (c_3, c_4 in Figures 9.4c and 9.4d respectively) are in accordance with Observation 4. Namely, cluster 1 is also reflected by the second RPF (*rpf*$_2$, Figure 9.3b), while cluster 2 by the fifth RPF (*rpf*$_5$, Figure 9.3e) with a small deviation on the number of contained BPMN 2.0 elements. Again in this case we need to complete the derived RPFs into complete process models. We follow a naive approach to convert the RPFs to complete process models in order to avoid additional overhead to the new process model. Thus, for *rpf*$_2$ shown in Figure 9.3b we add the missing sequence flow from task “T3” to the merging exclusive gateway and then we add an end event to the exclusive gateway. This generates



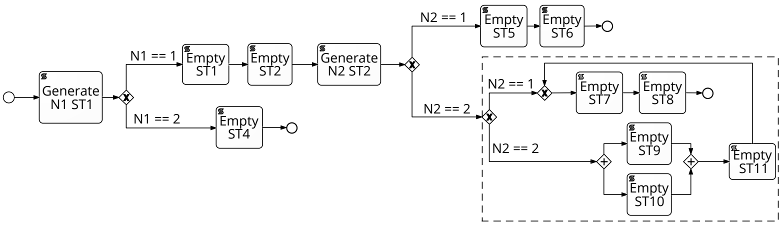
(a) Class 1 (c_1): *rpf₂*



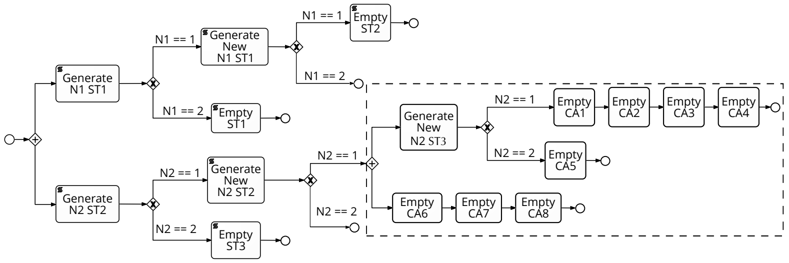
(b) Class 2 (c_2): rpf_1



(c) Class 3 (c_3): rpf_5



(d) Class 4 (c_4): Synthesised from rpf_5 and rpf_{17}



(e) Class 5 (c_5): Synthesised from rpf_{24} and rpf_{47}

Figure 9.4.: Derived process model classes of the workload model [FSP+17]

the process model of class 1 (c_1 shown in Figure 9.4a). Likewise, for the fifth RPF (rpf_5, Figure 9.3e) we add a start event and a script task before the splitting exclusive gateway. Again in this case, the script task needs to be added for providing the relevant data to the exclusive gateway to evaluate its condition. Finally, we add the missing sequence flow from task “T4” to task “T5” and we connect task “T6” with an end event. This results in the process model of class 3 (c_3 shown in Figure 9.4c).

As already discussed by Observation 1 most of the participating models in the workload mix should have size between 5 and 32 ($5 \leq size \leq 32$). For the classes constructed until now, we have process models of sizes between 8 and 10 ($8 \leq size \leq 10$). However, the initial analysis of the collection (Observation 2), as well as the clustering analysis (clusters 3 and 4) denote that also more complex process models should be included in the workload mix. As it is expected, the more complex the process model is, the more difficult it is to have exact recurrences in the original collection. Hence, while a single RPF was sufficient for generating the process models shown in Figures 9.4a to 9.4c, for deriving the bigger and more complex process models shown in Figures 9.4d and 9.4e we need to proceed to the synthesis of individual RPFs. For this we applied the process model generation method as it was introduced in Chapter 7. In order to keep consistency with the characteristics of the collection, we synthesised with respect to the clusters 3 and 4 analysis. The small number of extracted RPFs (143) made it infeasible to synthesise valid process models that were a precise reflection of clusters 3 and 4. Therefore, the derived process models shown in Figures 9.4d and 9.4e for the clusters 3 and 4 respectively are the closest we could get to these clusters with respect to the contents of our database. In particular, the criteria used for the generation of the process models shown in Figures 9.4d and 9.4e are shown in Table 9.3.

The process model shown in Figure 9.4d is synthesised out of two RPFs. The left-handed RPF of the process model in Figure 9.4d consists of the flow nodes not marked by the dashed-line. It is the same as rpf_5 shown in Figure 9.3e but slightly changed, due to the synthesis of different OCPs. The second RPF contained the rest of the elements marked by the dashed line. The process model shown in Figure 9.4e is also synthesised out of two RPFs. The first RPF is the leftmost part of the model up to the second parallel gateway (not marked by the dashed line). The right-handed RPF is marked by the dashed-line and consists of all nodes that exist right to the second parallel gateway, including the parallel gateway. By sorting the

Table 9.4.: Occurrences $t(rpf_i)$ of an RPF rpf_i in the process models collection M, occurrences in process models ($m(rpf_i)$) and its calculated intensity $inte(c_k)$ from Equation (8.1) (adapted from [FSP+17])

	rpf_i	$t(rpf_i)$	$m(rpf_i)$	$inte(c_k)$
Class 1 (c_1)	rpf_2	1,731	1,602	39%
Class 2 (c_2)	rpf_1	2,303	953	27%
Class 3 (c_3)	rpf_5	1,710	640	19%
Class 4 (c_4)	rpf_5	1,710	640	13%
	rpf_{17}	635	309	
Class 5 (c_5)	rpf_{24}	157	130	2%
	rpf_{47}	30	30	

set of resulting RPFs with respect to their number of occurrences, Table 9.4 shows the position of rpf_i in the resulting set (i. e., i indicates the ith element of the set of resulting RPFs). Moreover, on the right part of Table 9.4 we present RPF metadata regarding the number of occurrences (see $t(rpf_i)$ of Equation (8.1) in Section 8.1) and the number of process models (see $m(rpf_i)$ of Equation (8.1) in Section 8.1) in which the RPFs (rpf_i) of the workload mix process models we found.

9.1.4. Phase 4: Partition the Process Models into Workload Mix Classes

In this phase we form the workload classes (Figures 9.4a to 9.4e) of the workload mix. In other words, we calculate the intensity with which a process model participates in the workload mix. For this we apply Equations (8.1) and (8.2) introduced in Section 8.1. In the following we demonstrate how to calculate the representativeness of the process model shown in Figure 9.4d denoted as "Class 4 (c_4)" in Table 9.4. The values needed for calculating the representativeness are shown in Table 9.4. Before proceeding to the calculation of the representativeness we evaluate the participating elements as follows:

$S_k = \{rpf_5, rpf_{17}\}$: where $\{rpf_5, rpf_{17}\}$ are the RPFs comprising the process model shown in Figure 9.4d.

$t(rpf_5) = 1,710$: the RPF rpf_5 has 1,710 occurrences in the collection.

$m(rpf_5) = 640$: the RPF rpf_5 occurs in 640 process models of the collection.

$t(rpf_{17}) = 635$: the RPF rpf_{17} has 635 occurrences in the collection.

$m(rpf_{17}) = 309$: the RPF rpf_{17} occurs in 309 process models of the collection.

$|S_c| = 23,125$: total number of occurrences of all resulting RPFs[1].

$|M| = 3,247$: number of process models in the original collection.

This evaluates the representativeness of process model c_4 of Figure 9.4d as follows:

$$repr(c_4) = \frac{1}{4}\frac{1,710 + 635}{23,125} + \frac{1}{4}\frac{640 + 309}{3,247} = 0.0982$$

For calculating the intensity ($inte(c_4)$) with which process model c_4 participates in the workload mix we multiply the representativeness (c_4) with 100 and divide to the summarised representativeness of all process models comprising the classes of the workload mix (resulting in 0.7295). Hence, we then apply Equation (8.2) as follows:

[1]Note: this number is calculated using raw data

$$inte(c_4) = \left\lfloor \frac{100 \cdot 0.0982}{0.7295} \right\rceil = 13\%$$

We proceed likewise for calculating the representativeness and consequently the intensity of all the workload mix process models. The process models shown in Figures 9.4a to 9.4e are labelled with respect to their calculated intensity, such that a bigger class will map to a bigger intensity. The exact intensity of each class is shown in Table 9.4.

9.1.5. Phase 5: Definition of the Execution Behaviour of the Workload Mix

For defining the execution behaviour of the process models of the workload mix we follow the guidelines provided in Section 7.2.3 as follows:

i) Maximise the simplicity of the process models. A script task implements an no operational script, except for the case in which the script task precedes an exclusive gateway. In that case, generating a number for evaluating an exclusive gateway condition is needed. We keep it to the minimal amount of code that produces the required amount of data. Similarly, the call activities call a no-operative process (start event - no operational script task - end event). By implementing empty script tasks and call activities calling an empty process, we eliminate the time overhead introduced by the executed scripts tasks and call activities and we are able to derive clean measurements;

ii) Keep the semantics of the script task or the call activity for all the derived process models as they were present in the originally detected RPFs. The only cases in which we edited the original RPF is when a call activity precedes an exclusive gateway. In this case, the call activity needs to call another process model that generates a number and returns it to the parent workflow. The required underlying variable sharing between the parent and the child process models is not supported by all WfMSs [GHLW16]. Therefore, we converted such call activities to script tasks.

iii) Drop the interactions with external systems. We implement all tasks as script tasks or call activities and exclude human tasks and web service invocations. We use this approach as our workload mix targets to stress only the process navigator and not components of the WfMS. Generally, the derived process models are fully automated, as the goal is to stress the performance of the process navigator.

iv) Define an equal probability for passing the control flow to any outgoing branch of an exclusive gateway.

9.2. Application of the Workload Mix for Benchmarking Purposes

The derived workload mix distils the most prominent control flow characteristics, sizes (in terms of number of events, activities and gateways) and structures of a large collection of real world process models, and can be used for performance tests. For this purpose, we collaborated with an external partner for running the generated benchmark to assess and compare the performance of BPMN 2.0 WfMSs. The definition of performance metrics, experiments setup, benchmarking and analysis of results are published in detail by Ferme et al. [FSP+17]. For the sake of completeness in the following we also provide a short overview of them as well as the most important findings.

For the experiments we used the same methodology as the one described in Chapter 5. Namely, we use the BenchFlow environment [FIP15; FIP+16], which is an end-to-end Docker-based environment for WfMS performance benchmarking [Mer14]. In the benchmark we include four different versions of two open-source WfMSs. The included versions are the released versions of the last two years (2014 – 2016). Since we did not get the consent of the vendors for publishing their names, the WfMSs that participated in our experiments are thereafter named as WfMS A and WfMS B. Here we should note that there is no mapping between WfMS A and WfMS B of this chapter, to WfMS A and WfMS B of Chapter 5. We stress the tested WfMSs

		Load	WfMS A 7.2.0	7.3.0	7.4.0	7.5.0
Client-side	Mean # Requests per Second $\pm ci$	**50**	49.13±0.04	49.17±0.03	49.16±0.02	49.09±0.01
		500	484.87±0.39	486.44±0.10	484.84±0.82	482.91±2.20
		1,000	890.84±4.82	879.15±9.94	859.81±3.42	763.46±2.17
Server-side	Mean Throughput $\pm ci$ $(\#pi/s)$	**50**	118.23±0.21	119.72±0.17	120.08±0.79	120.05±0.42
		500	1,185.12±0.45	1,185.56±0.33	1,180.80±4.34	1,175.10±0.58
		1,000	2,121.35±6.23	2,130.90±9.50	2,087.26±2.72	1,849.88±5.17
	Weighted Mean Duration (ms)	**50**	1.03	1.08	1.12	1.24
		500	0.87	0.91	1.07	1.14
		1,000	0.93	0.99	1.05	1.15
Resource Consumption	Weighted Mean CPU (%)	**50**	1.66	1.33	1.33	1.35
		500	8.07	6.26	7.03	6.95
		1,000	11.39	8.33	8.81	8.79
	Weighted Mean RAM (MB)	**50**	637.28	590.82	634.79	648.67
		500	885.10	860.70	886.53	866.37
		1,000	970.61	957.47	978.97	971.43
Server-side	Weighted Mean Process Instance Duration (ms)	**Class 1** (c_1)	1.09	1.14	1.24	1.35
		Class 2 (c_2)	1.34	1.41	1.54	1.67
		Class 3 (c_3)	2.31	2.43	2.64	2.88
		Class 4 (c_4)	1.13	1.20	1.30	1.42
		Class 5 (c_5)	1.93	2.03	2.21	2.40

Table 9.5.: Performance and resource consumption metrics for WfMS A [FSP+17]

with a load function of 50, 500, and 1,000 users instantiating business process instances every second. The different numbers of users represent different sized companies in which the WfMS technology was deployed. Ferme et al. [FIP16] identified an expressive set of metrics for the thorough description of the WfMS performance, which are included in this benchmark. The performance metrics cover client-side performance aspects, such as the number of process instantiation requests the WfMSs accept per second, as well as WfMS specific metrics representing the internal behaviour of the system executing the models and resource (i. e., CPU and RAM) consumption. For ensuring reliable results that eliminate the non-determinism in the performance measures we executed three rounds of each experiment. Then we computed the aggregated metrics to analyse performance variability and

			WfMS B			
		Load	**5.18.0**	**5.19.0.2**	**5.20.0**	**5.21.0**
Client-side	Mean # Requests per Second $\pm ci$	**50**	48.84±0.02	48.72±0.05	48.66±0.03	48.65±0.04
		500	488.61±0.12	487.72±0.13	487.34±0.15	487.71±0.55
		1,000	906.10±3.81	900.14±4.09	891.62±1.76	885.80±2.97
Server-side	Mean Throughput $\pm ci$ ($\#pi/s$)	**50**	119.87±0.07	119.82±0.09	119.99±0.09	119.82±0.14
		500	1,161.88±0.98	1,160.46±1.13	1,160.96±1.67	1,132.92±2.43
		1,000	2,182.78±8.25	2,202.37±6.37	2,189.36±4.83	1,974.01±10.59
	Weighted Mean Process Instance Duration (ms)	**50**	6.53	6.75	6.90	7.19
		500	5.08	5.27	5.56	5.65
		1,000	5.18	5.34	5.39	5.43
Resource Consumption	Weighted Mean CPU (%)	**50**	3.01	1.59	1.63	1.66
		500	8.82	7.55	9.59	9.50
		1,000	12.05	12.74	12.78	11.84
	Weighted Mean RAM (MB)	**50**	1,764.83	2,430.69	2,620.57	2,455.11
		500	8,686.66	8,653.84	8,633.66	8,583.11
		1,000	9,549.54	9,749.74	9,509.81	9,350.82
Server-side	Weighted Mean Duration (ms)	**Class 1 (c_1)**	9.28	9.52	9.79	10.02
		Class 2 (c_2)	6.09	6.24	6.42	6.57
		Class 3 (c_3)	16.73	17.16	17.64	18.06
		Class 4 (c_4)	3.53	3.62	3.73	3.81
		Class 5 (c_5)	22.49	23.07	23.72	24.28

Table 9.6.: Performance and resource consumption metrics for WfMS B [FSP+17]

WfMS behaviour across the rounds.

Table 9.5, Table 9.6 and Figure 9.5 show the aggregated results that emerged from our experiments. In the following we summarise the most important findings, while a more detailed analysis of the results if provided in [FSP+17]. As seen in Tables 9.5 and 9.6 WfMS A performed with a significantly lower average single process model instance duration and lower RAM usage. However, WfMS B had the edge in throughput (see Figure 9.5). The obtained results also lead us to interesting conclusions regarding the evolution of the WfMSs per se. As seen in Figure 9.5 each newer version tested on both WfMS A and WfMS B demonstrated a performance decrease with respect to the older versions. This fact indicates that adding new functionalities to the system has an inverse effect on its performance. However,

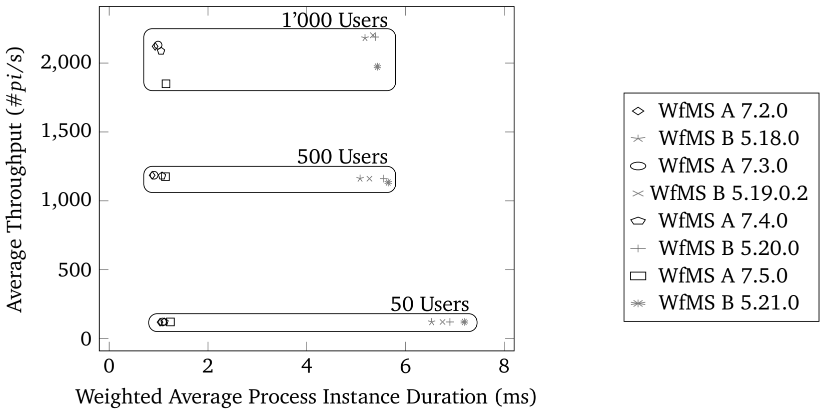


Figure 9.5.: Weighted average process instance duration (ms) vs. throughput ($\# \ pi/s$) [FSP+17]

these results were apparent only after executing multiple experiments with different load functions. This highlights the need for executing diverse, stable, and reliable performance tests before drawing conclusions, and justifies the need for benchmarking complex WfMSs using representative workload mix and varying load functions as part of the workload model.

At the bottom of Tables 9.5 and 9.6 we also show the mean process instance duration for each workload class. The presented mean values for the various workload classes are calculated using diverse data points, as the intensity with which a workload class participated in the workload mix varies (see Table 9.4). The presented results show that the increased average duration is not caused by a particular workload class, as an increase in the performance is evident in each workload class separately. However, the workload classes performed differently for the two WfMSs. Class 1 (see c_1 in Figure 9.4a and Table 9.4) was the fastest for WfMS A across all loads and versions while class 4 (see c_4 in Figure 9.4d and Table 9.4) was the fastest for WfMS B. Similarly, class 3 resulted in the slowest duration results for WfMS A (see c_3 in Figure 9.4c and Table 9.4) and class 5 (see c_5 in Figure 9.4e and Table 9.4) to the slowest duration for WfMS B. The fact that these results are disproportional to the process models sizes and complexities reinforces the conclusions drawn in Chapter 5 that parallelism and conditions also play a role in the performance of WfMSs.

Overall, the applied benchmark detected significant performance differences in the benchmarked systems. We are confident that the overall methodology followed for defining the benchmark can be reliably applied for comparing and evaluating the performance of different WfMSs releases or deployment configurations.

9.3. Chapter Summary

This chapter can be seen as a use case exploratory study of the overall work, as we employed the WINE4WfMSs process for the generation of a workload mix based on a collection of real-world practice process models. Overall, the generation of the workload mix executed smoothly and generated five workload classes of gradual complexity, that distils the prominent structural characteristics of the collection.

The generated workload mix has been given as input to a benchmark for BPMN 2.0 WfMSs executed in the BenchFlow environment [FIP15; FIP+16]. Through the benchmark we tested four different versions of two widely used open source BPMN 2.0 WfMSs. The workload mix was adequate for stressing the WfMSs and revealing interesting performance differences between the different versions of the products or the products per se. Moreover, the results of the generated workload mix and emerging results are openly published in the interactive dashboard of the PeACE initiative for future reference or easy access by other interested parties.

CHAPTER 10

CONCLUSIONS AND OUTLOOK

"Nothing is really over until the moment you stop trying."

B. Dyson

Nowadays, Workflow Management Systems (WfMSs) are broadly used for the design, deployment, monitoring, execution and analysis of automated business process models. Through the years WfMSs have advanced and can now deliver complex, service-oriented applications. The evolution of WfMSs as well as the large number of them available in the market leads companies in an extensive research before choosing which product is optimal for their requirements and business. Benchmarking is an established practice that allows the comparison of alternative products and helps the constant technological evolution, by setting distinct key performance indicators for measuring and assessing performance. For the case of WfMSs there is not yet a standard benchmark available. A possible explanation for this might be the inherent architectural complexity of WfMSs and the very large number of parameters that affect their performance [PFR+15]. The BenchFlow approach targets to propose the first standard benchmark for assessing and comparing the performance of WfMSs. In order to guarantee reliable results the BenchFlow approach targets to fulfil the requirements of a benchmark: portability, scalability, simplicity, vendor neutrality, repeatability, efficiency, representativeness, relevance, accessibility, and affordability. This work was conducted under the scope of the BenchFlow approach and introduced the Workload Mix Generation for Workflow Management Systems (WINE4WfMSs) method that allows the semi-automated generation of a workload mix. The WINE4WfMSs method itself targeted a set of recognised research challenges in the area of WfMS benchmarking and introduced some new research questions that are also addressed by this work.

Initially, in Chapter 2 we covered related work on benchmarking and Business Process Management (BPM). In Chapter 3 we introduced the anonymisation method that we used for collecting real world practice process models and described the composition of the derived process model collection. Afterwards we identified the basic components of a workload model for benchmarking WfMSs (see Chapter 4) and in Chapter 5 we proposed and executed the first micro-benchmark for BPMN 2.0 WfMSs to investigate the impact of BPMN 2.0 language constructs on the WfMS process navigator performance. In Chapter 6 we proposed a method that detects and extracts reoccurring structures (i. e., Relevant Process Fragments (RPFs)) in a collection of BPMN 2.0 process models. The RPFs were then used by a method that follows user-defined constraints for generating complete, executable process models out of RPFs (see Chapter 7). The complete WINE4WfMSs method and its realisation through a toolchain developed with diverse architectural paradigms (e. g., distributed and cloud computing) was discussed extensively in Chapter 8. The WINE4WfMSs method and its realisation were utilised in Chapter 9, where we apply the WINE4WfMSs method to derive a workload mix based on a collection of real world practice process models. The derived workload mix is then provided for executing a benchmark for BPMN 2.0 WfMSs. An overview of the obtained results is also provided in Chapter 9.

In the following sections we discuss the answers to research questions, limitations of this work and emerging open research challenges.

10.1. Answers to Research Questions

The research questions outlined in Chapter 1 have been answered as follows:

RQ-1: How to overcome obstacles in creating a collection of real-world practice process models?

In order to foster the sharing of real-world process models we initially made confidentiality agreements. Moreover, in Chapter 3 we proposed a process model anonymisation method that obfuscates business information of process models, while maintaining the process model's executional semantics. The acquired collection and its composition were also discussed in Chapter 3, forming contribution *C-1: "An anonymisation method for process models("BPanon") and the obtained collection of process models."*

RQ-2: What are the basic components of a workload model for WfMSs?

Through a literature study in Chapter 4 we identified the core components of a WfMS benchmark and their underlying interactions. In order to support future endeavours for designing WfMS benchmarks we built a knowledge base with the collected information and provided it to the users as a *Decision Support System for Benchmarking* (see Chapter 8). Afterwards, we identified the basic components of a workload model for WfMSs and their underlying interactions and introduced a metamodel for WfMSs workload forming contribution *C-2: "A metamodel of the basic workload model components for WfMSs."*.

RQ-3: What is the impact of diverse BPMN 2.0 language constructs on the process navigator's performance?

Before proceeding to the definition of a complex workload mix we needed to comprehend the impact of BPMN 2.0 language constructs on the process navigator's performance. For this purpose, we proceeded to the definition and execution of contribution *C-3: "The first micro-benchmark for BPMN 2.0 WfMSs"*. The micro-benchmark contained six different workload mixes derived from frequently used control flow workflow patterns (i. e., sequence flow, exclusive choice and simple merge, explicit termination, parallel split and synchronisation, and arbitrary cycle) [VTKB03]. The first five workload mixes defined the instantiation and execution of one workflow pattern, while the sixth workload mix included instances of all workflow patterns with equal distribution. The experiments were executed using the BenchFlow environment [FIP15] for measuring response time, throughput and resource utilisation [FIP16]. Among other observations, our results revealed that the sequential workflow pattern makes a better candidate when investigating the maximum throughput of WfMSs, while parallelism may be used for performance tests investigating throughput and resource utilisation. Moreover, more complex structures (e. g., arbitrary cycle) seemed to be better candidates for measuring resource utilisation.

RQ-4: How to derive a representative and meaningful workload mix for both general and domain specific benchmarks?

The complexity of WfMSs architecture and usage needs a flexible approach for defining workload mixes. In this way, one may generate representative workload mixes that match diverse performance testing scenarios and goals. For addressing this challenge we proposed contribution *C-4: The WINE4WfMSs method*. The WINE4WfMSs method was firstly introduced as a vision in Chapter 1 and described in detail in Chapter 8. In summary, WINE4WfMSs takes as input a collection of process models and derives the structures that reoccur in the collection (referred to as Relevant Process Fragments (RPFs)). The RPFs are annotated with related metadata and used in subsequent steps for generating representative process models. The representative process models generation is done with respect to user-defined criteria. These criteria were extracted from the related work presented in Chapters 2 and 4 as well as from the micro-benchmark results (Chapter 5). In future endeavours, the acquisition of existing data on WfMSs benchmarking is facilitated through our toolchain and related BenchFlow software presented in Chapter 8. In Chapter 9 we applied the WINE4WfMSs method to a real world practice process models collection and derive a structurally representative workload mix. The derived workload mix was provided for the execution of a benchmark for BPMN 2.0 WfMSs [FSP+17]. The workload mix was adequate for stressing the WfMSs and evidenced interesting performance behaviour among different versions of the same products or the products per se.

RQ-5: How to detect reoccurring BPMN 2.0 structures in a process models collection?

With respect to the characteristics of our BPMN 2.0 process models collection two objectives were set for the detection of RPFs: the approach should exploit solely the structural information of the process models (i. e., omit textual semantics) and the detection of RPFs should be achieved without knowing the structure to search a priori. For this we defined a formal model and introduced contribution *C-5: "The RoSE method"* that achieves the detection of RPFs while satisfying the aforementioned objectives. The proposed method adapted Ullmann's algorithm [Ull76], which is a broadly accepted approach on subgraph isomorphism. We argued that our approach is complete, as it exhausts the search space. Moreover, in Chapter 9 we showcased that our method can execute in acceptable time, as it completed in 14 hours for comparing 3,247 process models and produced 143 RPFs.

RQ-6: How to synthesise a representative BPMN 2.0 process model?

As part of the WINE4WfMSs process we proposed contribution *C-6: Representative BPMN 2.0 process model generation method*. The method applied structural criteria as constraints for the generation of complete process models out of detected RPFs. The ultimate goal of this method is to produce representative, executable process models that can be provided to the performance tests. To achieve this goal, the method follows an identified set of rules that were afterwards realised through a backtracking algorithm (see Chapter 7). The transformation of a process models to executable is achieved manually, following suggested guidelines.

10.2. Limitations

In this section, we focus on two main limitations that pertain to the complete WINE4WfMSs method. Although we could successfully gather a large and diverse collection of process models, statistical information regarding their execution was not actually shared with us [Limitation 1]. For example, information regarding the ratio of specific process model instantiation per time unit (i. e., load functions), the frequency of usage of each process model, the execution duration of process models, the probability to follow a control flow path or the data that drive the process models execution are unavailable to us. Despite the numerous discussion sessions with industrial and research partners, it slowly became clear that this type of information is highly confidential. Hence, targeting people that are willing to provide this information or log files that contain it is very challenging. In order to overcome this burden we generate synthetic data and probabilistic activation of the process paths. Hence, this type of information is currently user-defined and its representation to real world situations depends on the user's intuition and experience through the provision of criteria that drive the synthesis process.

The workload characterisation process defines performance tests containing the synthetic workload mix to typical cases of workload mixes applied in real world practice. Through collecting and comparing the results of executed performance tests to real world practice, one is then able to evaluate the representativeness of the synthetic workload mix to the real world. Since we could not obtain any information regarding the load of the WfMSs in real world practice and the corresponding behaviour of the process models, it was not possible to proceed to the aforementioned workload characterisation process [Limitation 2] [Fei02]. Thus, in Chapter 9 we could only partially validate the representativeness of the synthetic workload mix with respect to its structural characteristics.

10.3. Future Work

In the following we discuss future research opportunities introduced by this work.

10.3.1. Standardisation of the BenchFlow Benchmark

This work was conducted under the scope of the BenchFlow approach. The BenchFlow approach is built with respect to the requisite properties of a benchmark (portability, scalability, simplicity, vendor neutrality, repeatability, efficiency, representativeness, relevance, accessibility, and affordability). To this effect and in order to address the recognised challenges [PFR+15], the definition of the workload mix as well as the development of the benchmarking environment [FIP15] of the BenchFlow approach can be seen as a first step towards a standard benchmark for BPMN 2.0 WfMSs. For evolving into a standard industry benchmark, the BenchFlow project should make steps towards the inclusion of industrial partners to the performance tests, as well as the approval and support of a benchmarking initiative, as for example Standard Performance Evaluation Corporation (SPEC) [Sta95] or Transaction Processing Performance Council (TPC) [Tra92a].

10.3.2. Extend the RoSE Method

The RoSE method detects frequently reoccurring structures in a collection of BPMN 2.0 process models by relying solely on the structural characteristics of the process models collection. This decision was essentially driven by the need for process model anonymisation and the existence of non-executable process models in our collection [Limitation 1] (see Chapters 3 and 6). Upon availability of the appropriate information, the RoSE method should be extended to a more comprehensive approach that exploits existing techniques of textual and behavioural [Dij+13] semantics in the similarity detection. This will enable the detection of reoccurring structures with similar context. Nevertheless, an improved version of the RoSE method should also support the detection of cyclic structures. Moreover, the RoSE method may be optimised in terms of its performance, through the utilisation of heuristic based methods [VM97].

10.3.3. Extend the Generation of Representative Process Models Method

Currently, the representative process models generation method supports the generation of process models out of RPFs. Its functionality could be extended to cover also generation of representative execution behaviour of the process models (e. g., the number of started processes per second, or the path divergence probability), as well as the probabilistic behaviour of entities (i. e., web services, human actors and interacting applications) interacting with the process instances. This can be achieved through the analysis of real world practice or simulated process model event logs and data ([Limitation 1]). Previous work has been done in the context of process mining [Bur15] and workload characterisation [Fei02]. There exist many techniques to mine process models from event logs when no model is available, as well as techniques to add timing information to the mined models [vdAvD02]. Additionally, many Business Process Intelligence techniques exist to mine time information of known process models [vdAal13]. These techniques may be exploited in order to derive representative synthetic process models that imitate also a realistic behaviour. At the end workload characterisation techniques [Fei02] should be applied to the synthetic process models, to ensure the representativeness of the observed performance ([Limitation 2]).

10.4. Chapter Summary

In this chapter we provided answers to the research questions presented in Chapter 1. The two major limitations of the WINE4WfMSs method have been discussed, emerging essentially as an immediate consequence of the process event logs unavailability. Upon acquisition of event logs, the complete WINE4WfMSs method can be extended to support the generation of workload mixes that imitate realistic behaviour. A possible approach to extrapolate this behaviour would be through the utilisation of well established process minings techniques [Bur15] or through the extension of the RoSE method to support the detection of reoccurring structures considering also textual and behavioural semantics. In order to verify the representativeness of the synthetic workload mix's behaviour, one may proceed to workload characterisation techniques [Fei02]. Through these techniques the synthetic workload mix behaviour is compared to real world process models to derive a ratio of representativeness.

Appendix A

Assistive Functions for RoSE Algorithm

This appendix presents the pseudocode of two assistive functions that are used by the RoSE algorithm (see Algorithm 6.3, Section 6.3.2). Algorithm A.1 shows a function that compares an RPF against a collection of RPFs and detects duplicate occurrences of it (i. e., if there already is an exact isomorphism of the RPF in the collection). In case there are multiple occurrences of the RPF, the function updates the metadata information of the RPF with respect to its frequency of occurrence. The function takes as input the process models ($G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$) that extracted the RPF (*rpf₁*) for which we want to examine the duplicates, the RPF (*rpf₁*) itself, and the complete collection of RPFs ($\boldsymbol{RPF}_{coll}$) which we search for duplicates. The algorithm outputs the updated collection ($\boldsymbol{RPF}_{coll}$). The algorithm needs to compare the targeted RPF with each one of the RPFs that exist in the RPFs collection. As a first condition, we first check if two RPFs are of the same size. If not, we know for sure that these two RPFs cannot be exact isomorphisms to each other and proceed to the next comparison. In case two RPFs are of the same size, we may proceed by re-applying the RoSE algorithm (cf. Section 6.3) on the two RPFs. In case the algorithm returns a resulting RPF that is common between two RPFs, we then proceed by checking the edges of both graphs. If there is an equality among the edges we know that these two RPFs are the same. We then proceed by updating the relevant metadata for the discovered RPF. If we did not find any match (i. e., there was not a duplicate of rpf_1 in the collection), it means that the discovered RPF can be inserted to the collection for first time. Hence, we insert it in the collection. At the end of the algorithm we return the updated collection.

Algorithm A.2 is an assistive function used by the FixDecisionTree algorithm (cf. Algorithm 6.6,Section 6.3.2) to construct the decision tree for RPFs detection construction. The algorithm uses a *childrenQueue* which is a global variable shared with the FixDecisionTree function. This variable is implemented with a LIFO queue that holds information regarding the edges that are still not added in the tree. In addition, the algorithm takes as input the *parent* variable, which is the edge that is currently checked for the tree construction and the *matrix* which is an array for which we build the decision tree. Algorithm A.2 outputs a set of edges that were calculated as children of the edge *parent*. For the calculation of the edge's immediate children the function traverses the array *matrix* row by row. The immediate children can be found in the next row which contains ones ("1"). If we find a child and add it to the tree, then the siblings of this child need to be eliminated from the *matrix*, because they represent alternative edges in the sub-graph isomorphism and cannot be considered again for addition to this version of the tree. At the end of the process we add all discovered children in the *childrenQueue* as they need to be recursively examined for their children.

Algorithm A.1 Function that compares a RPF against the the $\boldsymbol{RPF}_{coll}$ to detect duplicate (clone) RPFs and calculate the RPF metadata regarding its frequency of occurrence

```
Input: $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$: the models Algorithm 6.3 compared to
    extract the $rpf_1$
Input: $rpf_1$ represents the RPF to check against the collection for duplicates
Input: $\boldsymbol{RPF}_{coll}$ represents the collection against which we compare the $rpf_1$
    for duplicates
Output: The updated $\boldsymbol{RPF}_{coll}$
 1: function FixDuplicates($G_1, G_2, rpf_1, \boldsymbol{RPF}_{coll}$)
 2:     for each ($rpf_2 \in \boldsymbol{RPF}_{coll}$) do
 3:         if ($|\boldsymbol{E}_{rpf_1}| = |\boldsymbol{E}_{rpf_2}|$) then
 4:             $E_1 \leftarrow outgoing(v_{src}^{rpf_1})$
 5:             $edge_1 \leftarrow e_1 \in E_1$
 6:             $E_2 \leftarrow outgoing(v_{src}^{rpf_2})$
 7:             $edge_2 \leftarrow e_2 \in E_2$
 8:             $matrix \leftarrow \varnothing$
 9:             $matrix \leftarrow$ CreateMap($edge_1, edge_2, matrix$)
10:             $rpf_{result} \leftarrow$ MapToRPFs($\{matrix\}$)
11:             if ($(\boldsymbol{E}_{rpf_2} - \boldsymbol{E}_{rpf_{result}}) = \emptyset$) then          // It is duplicate
                // The RPF existing already in the $\boldsymbol{RPF}_{coll}$ (i. e., $rpf_2$) will
                  update its metadata
12:                 $rpf_2.count \leftarrow rpf_2.count + 1$
13:                 $rpf_2.processModels \leftarrow \{G_1, G_2\} \cup rpf_2.processModels$
14:             else                                                  // It is not duplicate
15:                 $rpf_1.count \leftarrow 1$
16:                 $\boldsymbol{RPF}_{coll} \leftarrow \boldsymbol{RPF}_{coll} \cup \{rpf_1\}$
17:             end if
18:         end if
19:     end for
20:     return $\boldsymbol{RPF}_{coll}$
21: end function
```

Algorithm A.2 Assistive function for the decision tree for RPFs detection construction. It relies on the *matrix* to detect the immediate children of an edge and add them to the decision tree (*tree*)

```
 1: global variables
 2:     $childrenQueue$: Global variable used as memory for defining the
    edges which are still not added in the tree. It is implemented as a LIFO
    queue. This variable is shared with the FixDecisionTree function (cf.
    Algorithm 6.6).
 3: end global variables
Input: $parent$: the edge that is currently handled for the tree construction
Input: $matrix$: the array for which we build the decision tree
Output: $children$ a set of edges that were calculated as the children of the
    edge $parent$
 4: function AddChildren($parent, matrix$)
    // First we get the immediate children of the edge $parent$
 5:     $children \leftarrow \emptyset$                    // Start searching from the first row
 6:     $i \leftarrow parent.row + 1$
 7:     while ($i < matrix.rowsSize$ ) do
 8:         $j = 0$
 9:         while ($j < matrix.columnsSize$) do
10:             if $matrix[i][j] = 1$ then
11:                 $matrix_{new} \leftarrow$ UpdateMatrix(
                        $matrix, i, j, map.rowSize, map.columnSize$)
12:                 $children \leftarrow$ NewTreeNode(
                        $i, j, matrix_{new}, map.rowSize, map.columnSize$)
13:             end if
14:             $j \leftarrow j + 1$
15:         end while
            // Before going to the next row if found children break
16:         if ($children \neq \emptyset$) then
17:             break
18:         end if
19:         $i \leftarrow i + 1$
20:     end while
        // For the children we just calculated
21:     for each ($child \in children$) do
22:         $tree.add(parent, child)$
23:     end for
24:     $childrenQueue.add(children)$
25:     return $children$
26: end function
```

List of Figures

List of Tables

Table of Definitions

Acronyms

ACID	Atomicity, Consistency, Isolation, Durability.
API	Application Programming Interface.
APROMORE	Advanced Process Model Repository.
BenchFlow	“BenchFlow: A Benchmark for Workflow Management Systems".
betsy	BPEL Engine Test System.
BPanon	Business Process Anonymisation.
BPM	Business Process Management.
BPMAI	Business Process Management Academic Initiative.
BPMN	Business Process Model and Notation.
BPMN 2.0	Business Process Model and Notation 2.0.
BPMS	Business Process Management Systems.
CI	Confidence Interval.
CORS	Cross-Origin Resource Sharing[1].

[1]`https://www.w3.org/TR/cors/`

CPU Central Processing Unit.

CRUD Create, Read (Retrieve), Update (Modify), Delete (Destroy).

CSS Cascading Style Sheets.

CSV Comma-Separated Values.

DAG Directed Acyclic Graph.

DFS Depth First Search.

DSS Decision Support System.

EAI Enterprise Application Integration.

EMF Eclipse Modeling Framework.

EPC Event-Driven Process Chain.

ESB Enterprise Service Bus.

FACTS Framework for Fault-Tolerant Composition of Transactional Web Services.

GASTON Graph / Sequence / Tree extractiON.

GED Graph Edit Distance.

gSpan Graph-based Substructure Pattern Mining.

HATEOAS Hypertext as the Engine of Application State.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

I/O Input/Output.

IaaS Infrastructure as a Service.

IBM International Business Machines Corporation.

IDEAL Isolation, Distribution, Elasticity, Automated Management, Loose Coupling.

IT Information Technology.

JMS Java Message Service.

JVM Java Virtual Machine.

KB KiloByte.

KPIs Key Performance Indicators.

LIFO Last-In-First-Out.

MAG Attribute Graph Model.

MB megabyte.

MCSI Maximum Common Subgraph Isomorphism.

MEAN MongoDB, Express, AngularJS, NodeJS.

MOM Message Oriented Middleware.

MVC Model-View-Controller.

OCL Object Constraint Language.

OCP Open Connection Point.

OLTP Online Transaction Processing.

PaaS Platform as a Service.

PC Personal Computer.

PDA Personal Digital Assistant.

PEaCE Benchmark for Conformance and Performance of Workflow Engines.

REST Representational State Transfer.

R-MAT Recursive Matrix.

RDF Resource Description Framework.

RoSE Reoccurring Structures Detection.

RPF Relevant Process Fragment.

RPST Refined Process Structure Tree.

SAP Systems, Applications & Products in Data Processing SE.

SCC Strongly Connected Components.

Sd Standard Deviation.

SESE Single-Entry-Single-Exit.

SLA Service Level Agreement.

SOA Service Oriented Architecture.

SOAP SOAP.

SPEC Standard Performance Evaluation Corporation.

SUT System Under Test.

SWoM Stuttgarter Workflow Maschine.

TPC Transaction Processing Performance Council.

UI User Interface.

URI Uniform Resource Identifier.

UUID Universally Unique Identifier.

VM Virtual Machine.

WfMS Workflow Management System.

WINE4WfMSs Workload Mix Generation for Workflow Management Systems.

WS-BPEL Web Services Business Process Execution Language.

WSDL Web Services Description Language.

XML eXtensible Markup Language.

XSD XML Schema Definition.

YAWL Yet Another Workflow Language.

Acknowledgements

I would like to express my gratitude to my advisor Prof. Frank Leymann for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. I could not have imagined having a better advisor and mentor for my Ph.D study. Besides my advisor, I would like to thank my second advisor Prof. Dimitris Plexousakis for taking time to review the thesis, but most of all for providing his full support throughout my complete educational journey.

My sincere thanks also goes to Prof. Cesare Pautasso for his insightful comments and feedback, for the hard questions which led me to widen my research from various perspectives, and for the fruitful discussions during my visit in Lugano.

Thanks to all the IAAS team for the stimulating discussions and for all the fun we have had in the last three years. I thank especially Santiago Gómez Sáez and Sebastian Wagner for vitalizing the working routine and Steve Strauch for pushing me to be better. Thank you Dr. Dieter H. Roller, Katharina Görlach, Michael Hahn, Karolina Vukojevic-Haupt, Oliver Kopp and Andrè van Hoorn for sharing your profound knowledge and for participating in the different stages of this research and learning process. I greatly appreciate the support and continuous encouragement received from Dimka Karastoyanova and Vasilios Andrikopoulos. Thank you, for being great friends and mentors.

From Università della Svizzera Italiana, a big thank you to my close colleague Vincenzo Ferme for the productive collaboration and for his dedication and determination to help me meet even the most impossible deadlines and Ana Ivanchikj for the critical comments and the excellent teamwork.

I would like to thank my husband Nikos for standing by my side all these years with infinite patience and love, and my friend Roza for believing in me even when I did not. Finally, I wish to thank my mother Aimilia and my father Vasilis for their unconditional love and amazing opportunities that they have given me, and my brother Giannis for his continuous support and encouragement.

Bibliography

[Act11] Active Endpoints Inc. *Assessing ActiveVOS Performance*. 2011. URL: http://www.activevos.com/content/developers/technical_notes/assessing_activevos_performance.pdf (cit. on pp. 38, 73, 168).

[Adr16] AdroitLogic Private Ltd. *Performance Framework and ESB Performance Benchmarking*. 2016. URL: http://esbperformance.org (cit. on p. 39).

[All16] T. Allweyer. *BPMN 2.0 Introduction to the Standard for Business Process Modeling*. Ed. by B. on Demand. Books on Demand, May 3, 2016 (cit. on p. 106).

[Alm02] V. A. F. Almeida. "Capacity Planning for Web Services Techniques and Methodology." In: *Performance Evaluation of Complex Systems: Techniques and Tools*. Springer Science + Business Media, 2002, pp. 142–157 (cit. on p. 71).

[Apa13a] Apache Software Foundation. *Creating a Process*. 2013. URL: http://ode.apache.org/creating-a-process.html (cit. on p. 63).

[Apa13b] Apache Software Foundation. *Red Hat JBoss BPM Suite 6.1 - Deployment Descriptors*. 2013. URL: https://access.redhat.com/documentation/en-US/Red_Hat_JBoss_BPM_Suite/6.1/html/Administration_And_Configuration_Guide/sect-Deployment_Descriptors.html (cit. on p. 63).

[APDL13] R. Angles, A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. "Benchmarking Database Systems for Social Network Applications." In: *International Workshop on Graph Data Management Experiences and Systems*. GRADES'13. New York, New York: Association for Computing Machinery ACM, 2013, pp. 1–7 (cit. on pp. 41, 82).

[ARX12] ARX. *ARX - Powerful Data Anonymization*. 2012. URL: `http://arx.deidentifier.org/` (cit. on p. 45).

[BBC+85] D. Bitton, M. Brown, R. Catell, S. Ceri, T. Chou, D. DeWitt, D. Gawlick, H. Garcia-Molina, B. Good, J. Gray, et al. "A Measure of Transaction Processing Power." In: *Datamation* 31.7 (1985), pp. 112–118 (cit. on p. 13).

[BBD10] D. Bianculli, W. Binder, and M. L. Drago. "SOABench: Performance Evaluation of Service-oriented Middleware Made Easy." In: *International Conference on Software Engineering*. Vol. 2. ICSE '10. Cape Town, South Africa, 2010, pp. 301–302 (cit. on pp. 14, 38, 41, 73, 82, 95, 168).

[BBDA12] M. Bentounsi, S. Benbernou, C. S. Deme, and M. J. Atallah. "Anonyfrag: An Anonymization-based Approach for Privacy-preserving BPaaS." In: *International Workshop on Cloud Intelligence*. Cloud-I '12. Istanbul, Turkey: Association for Computing Machinery ACM, 2012, pp. 1–8 (cit. on p. 46).

[BCM+05] P. Brebner, E. Cecchet, J. Marguerite, P. Tuma, O. Ciuhandu, B. Dufour, L. Eeckhout, S. Frénot, A. S. Krishna, J. Murphy, et al. "Middleware Benchmarking: Approaches, Results, Experiences." In: *Concurrency and Computation: Practice and Experience* 17.15 (2005), pp. 1799–1805 (cit. on p. 37).

[BDDS14] D. Breuker, P. Delfmann, H.-A. Dietrich, and M. Steinhorst. "Graph Theory and Model Collection Management: Conceptual Framework and Runtime Analysis of Selected Graph Algorithms." In: *Information Systems and e-Business Management* 13.1 (Feb. 2014), pp. 69–106 (cit. on pp. 53, 101, 102, 129).

[BM06] D. A. Bader and K. Madduri. *GTgraph: A Synthetic Graph Generator Suite*. 2006 (cit. on p. 44).

[BMHW16] D. Bimamisa, M. Müller, S. Harrer, and G. Wirtz. “Interactive Dashboard for Workflow Engine Benchmarks.” In: *International Workshop on Performance and Conformance of Workflow Engines*. PEaCE ’16. 2016 (cit. on pp. 26, 41, 80, 174).

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture. A System of Patterns*. Ed. by G. Redvers-Mutton. Wiley John + Sons, 1996 (cit. on p. 161).

[Bur15] A. Burattin. *Process Mining Techniques in Business Environments*. Springer International Publishing, 2015 (cit. on pp. 202, 203).

[CAA+11] S. Chen, A. Ailamaki, M. Athanassoulis, P. B. Gibbons, R. Johnson, I. Pandis, and R. Stoica. “TPC-E vs. TPC-C: Characterizing the New TPC-E Benchmark via an I/O Comparison Study.” In: *ACM SIGMOD Record* 39.3 (Feb. 2011), pp. 5–10 (cit. on p. 37).

[Cam13] Camunda Services GmbH. *Camunda BPM*. Mar. 2013. URL: `https://camunda.org/` (cit. on p. 89).

[Car08] J. Cardoso. “Business Process Control-Flow Complexity: Metric, Evaluation, and Validation.” In: *International Journal of Web Services Research (IJWSR)* 5.2 (2008), pp. 49–76 (cit. on pp. 132, 145).

[CFSV04] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. “A (Sub)graph Isomorphism Algorithm for Matching Large Graphs.” In: *Transactions on Pattern Analysis and Machine Intelligence* 26.10 (2004), pp. 1367–1372 (cit. on p. 102).

[Cha95] A. B. Chaudhri. “An Annotated Bibliography of Benchmarks for Object Databases.” In: *ACM SIGMOD Record* 24.1 (1995), pp. 50–57 (cit. on p. 37).

[CMT00] M. Calzarossa, L. Massari, and D. Tessera. “Workload Characterization Issues and Methodologies.” In: *Performance Evaluation: Origins and Directions*. London, UK, UK: Springer-Verlag, 2000, pp. 459–481 (cit. on p. 71).

[CZF04] D. Chakrabarti, Y. Zhan, and C. Faloutsos. “R-MAT: A Recursive Model for Graph Mining.” In: *International Conference on Data Mining*. ICDM’ 04. Society for Industrial & Applied Mathematics (SIAM), Apr. 2004, pp. 442–446 (cit. on pp. 43, 44).

[DDG09] R. Dijkman, M. Dumas, and L. García-Bañuelos. “Graph Matching Algorithms for Business Process Model Similarity Search.” In: *International Conference on Business Process Management (BPM ’09)*. Ulm, Germany: Springer-Verlag, 2009, pp. 48–63 (cit. on p. 52).

[DDvD+11] R. Dijkman, M. Dumas, B. van Dongen, R. Käärik, and J. Mendling. “Similarity of Business Process Models: Metrics and Evaluation.” In: *Information Systems* 36.2 (Apr. 2011), pp. 498–516 (cit. on p. 52).

[DES08] G. Din, K.-P. Eckert, and I. Schieferdecker. “A Workload Model for Benchmarking BPEL Engines.” In: *International Conference on Software Testing Verification and Validation Workshop*. ICSTW ’08. IEEE. 2008, pp. 356–360 (cit. on pp. 39, 73, 98, 168).

[DG08] J. Dean and S. Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters.” In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113 (cit. on p. 45).

[Dij+13] R. M. Dijkman et al. “A Short Survey on Process Model Similarity.” In: *Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE*. Springer Berlin Heidelberg, 2013, pp. 421–427 (cit. on pp. 52, 101, 202).

[DKP+13] M. Dave, T. Kohlenberg, S. Purcell, A. Ross, and J. Sedayao. *Some Key Cloud Security Considerations*. Service Technology Magazine. Sept. 2013 (cit. on p. 45).

[DKSU11] S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. “Apples and Oranges: a Comparison of RDF Benchmarks and Real RDF Datasets.” In: *International Conference on Management of Data*. SIGMOD’11. Association for Computing Machinery (ACM), 2011 (cit. on p. 43).

[DL08] D. J. DeWitt and C. Levine. “Not Just Correct, but Correct and Fast: A Look at One of Jim Gray’s Contributions to Database System Performance.” In: *SIGMOD Rec.* 37.2 (June 2008), pp. 45–49 (cit. on pp. 13, 17).

[Dom+10] D. Dominguez-Sal et al. “Survey of Graph Database Performance on the HPC Scalable Graph Analysis Benchmark.” In: *Web-Age Information Management*. Springer Science + Business Media, 2010, pp. 37–48 (cit. on p. 43).

[DRR12] R. Dijkman, M. L. Rosa, and H. A. Reijers. “Managing Large Collections of Business Process Models: Current Techniques and Challenges.” In: *Computers in Industry* 63.2 (Feb. 2012), pp. 91–97 (cit. on p. 132).

[DRS10] F. Doelitzscher, C. Reich, and A. Sulistio. “Designing Cloud Services Adhering to Government Privacy Laws.” In: *International Conference on Computer and Information Technology*. CIT ’10. June 2010, pp. 930–935 (cit. on p. 61).

[DS12] M. Dayarathna and T. Suzumura. “XGDBench: A Benchmarking Platform for Graph Stores in Exascale Clouds.” In: *International Conference on Cloud Computing Technology and Science*. CloudCom ’12. Institute of Electrical & Electronics Engineers (IEEE), Dec. 2012 (cit. on p. 44).

[Dum+09] M. Dumas et al. “Similarity Search of Business Process Models.” In: *Institute of Electrical & Electronics Engineers (IEEE) Data Engineering Bulletin* 32.3 (2009), pp. 23–28 (cit. on p. 53).

[Dum+13] M. Dumas et al. “Fast Detection of Exact Clones in Business Process Model Repositories.” In: *Information Systems* 38.4 (June 2013), pp. 619–633 (cit. on pp. 51, 101, 129).

[EDG+12] C. C. Ekanayake, M. Dumas, L. García-Bañuelos, M. La Rosa, and A. H. M. ter Hofstede. “Approximate Clone Detection in Repositories of Business Process Models.” In: *International Conference of Business Process Management*. Ed. by A. Barros, A. Gal, and E. Kindler. BPM ’12. Tallinn, Estonia: Springer Berlin Heidelberg, Sept. 2012, pp. 302–318 (cit. on pp. 51, 101, 129).

[EKMW12] R.-H. Eid-Sabbagh, M. Kunze, A. Meyer, and M. Weske. “A Platform for Research on Process Model Collections.” In: *International Workshop of Business Process Model and Notation (BPMN ’12)*. Ed. by J. Mendling and M. Weidlich. Vienna, Austria: Springer Berlin Heidelberg, Sept. 2012, pp. 8–22 (cit. on pp. 47, 48).

[Eli15] M. Elias. “Design of Business Process Model Repositories.” PhD thesis. Stockholm University, 2015 (cit. on pp. 46, 47).

[ELS+10] H. Eberle, F. Leymann, D. Schleicher, D. Schumm, and T. Unger. “Process Fragment Composition Operations.” In: *Asia-Pacific Services Computing Conference*. APSCC ’10. Institute of Electrical & Electronics Engineers (IEEE) Asia-Pacific, 2010 (cit. on pp. 51, 56).

[ELtHF11] C. C. Ekanayake, M. La Rosa, A. H. M. ter Hofstede, and M.-C. Fauvet. “Fragment-Based Version Management for Repositories of Business Process Models.” In: *On the Move to Meaningful Internet Systems: OTM 2011 Conferences*. Springer, 2011 (cit. on pp. 33, 50).

[EM06] S. Elnaffar and P. Martin. “Techniques and a Framework for Characterizing Computer Systems’ Workloads.” In: *Innovations in Information Technology*. Institute of Electrical & Electronics Engineers (IEEE), Nov. 2006 (cit. on pp. 32, 33, 42, 71).

[Erl05] T. Erl. *Service-oriented Architecture: Concepts, Technology, and Design*. Pearson Education India, 2005 (cit. on p. 34).

[Fed90] Federal Ministry of Justice. *German Federal Data Protection Law*. 1990 (cit. on p. 60).

[Feh15] C. Fehling. “Cloud computing patterns: identification, design, and application.” Dissertation. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Nov. 2015 (cit. on pp. 155–157, 160, 162, 174).

[Fei02] D. G. Feitelson. “Workload Modeling for Performance Evaluation.” In: *Performance Evaluation of Complex Systems: Techniques and Tools*. Springer Science + Business Media, 2002, pp. 114–141 (cit. on pp. 200, 202, 203).

[Fei15] D. G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2015 (cit. on pp. 15, 18, 21, 32, 42, 58, 59, 71, 79).

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol - HTTP/1.1.* W3C RFC 2616. 1999 (cit. on p. 157).

[FHM14] C. Fähnrich, T. Hoppe, and A. Mascher. *PromniCAT-Collecting and Analyzing Heterogenic Business Process Models*. 2014. URL: `http://wiki.promnicat.googlecode.com/git-history/c3eda190b1141b06e131543c38121aa009ca5c56/promnicat.pdf` (cit. on pp. 48, 49).

[FIP+16] V. Ferme, A. Ivanchikj, C. Pautasso, M. Skouradaki, and F. Leymann. "A Container-centric Methodology for Benchmarking Workflow Management Systems." In: *International Conference on Cloud Computing and Services Science*. CLOSER '16. Springer, 2016 (cit. on pp. 25, 27, 40, 72, 78, 98, 152, 190, 194).

[FIP15] V. Ferme, A. Ivanchikj, and C. Pautasso. "A Framework for Benchmarking BPMN 2.0 Workflow Management Systems." In: *International Conference on Business Process Management*. BPM '15. Springer, 2015, pp. 251–259 (cit. on pp. 40, 82, 88, 98, 152, 176, 190, 194, 198, 201).

[FIP16] V. Ferme, A. Ivanchikj, and C. Pautasso. "Estimating the Cost for Executing Business Processes in the Cloud." In: *International Conference on Business Process Management*. BPM '16 (Forum). Springer. 2016, pp. 72–88 (cit. on pp. 41, 88, 191, 198).

[FLH+17] V. Ferme, J. Lenhard, S. Harrer, M. Geiger, and C. Pautasso. "Lessons Learned from Evaluating Workflow Management Systems." In: *International Conference on Software Engineering (Poster Track)*. ICSE'17. ACM. 2017 (cit. on p. 77).

[FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter. *Cloud Computing Patterns*. Springer Wien, Jan. 2014 (cit. on pp. 155, 158, 166, 169).

[FSP+17] V. Ferme, M. Skouradaki, C. Pautasso, F. Leymann, and A. Ivanchikj. "Performance Comparison Between BPMN 2.0 Workflow Management Systems Versions." In: *International Workshop on Business Process Modeling, Development and Support*. BPMDS '17. Springer, 2017 (cit. on pp. 24, 27, 154, 175, 185, 187, 190–193, 199).

[FV01] M.-L. Fernández and G. Valiente. “A graph distance metric combining maximum common subgraph and minimum common supergraph.” In: *Pattern Recognition Letters* 22.6–7 (2001), pp. 753–758 (cit. on p. 108).

[GAH+15] S. Gómez Sáez, V. Andrikopoulos, M. Hahn, D. Karastoyanova, F. Leymann, M. Skouradaki, and K. Vukojevic-Haupt. “Performance and Cost Evaluation for the Migration of a Scientific Workflow Infrastructure to the Cloud.” In: *International Conference on Cloud Computing and Service Science*. CLOSER ’15. SciTePress, May 2015, pp. 1–10 (cit. on p. 40).

[GAYG15] E. Goettelmann, A. Ahmed-Nacer, S. Youcef, and C. Godart. “Paving the Way towards Semi-automatic Design-Time Business Process Model Obfuscation.” In: *International Conference on Web Services (ICWS ’15)*. Institute of Electrical & Electronics Engineers (IEEE), June 2015 (cit. on pp. 45, 52).

[GGKS02] K. Gottschalk, S. Graham, H. Kreger, and J. Snell. “Introduction to Web Services Architecture.” In: *IBM Systems Journal* 41.2 (2002), pp. 170–177 (cit. on p. 37).

[GHL+15] M. Geiger, S. Harrer, J. Lenhard, M. Casar, A. Vorndran, and G. Wirtz. “BPMN Conformance in Open Source Engines.” In: *International Symposium on Service-Oriented System Engineering*. SOSE ’15. Institute of Electrical & Electronics Engineers (IEEE). San Francisco Bay, CA, USA, Mar. 2015 (cit. on pp. 41, 79).

[GHL16] M. Geiger, S. Harrer, and J. Lenhard. “Process Engine Benchmarking with Betsy–Current Status and Future Directions.” In: *Central-European Workshop on Services and their Composition*. ZEUS ’16. CEUR-WS.org, 2016 (cit. on p. 41).

[GHLW16] M. Geiger, S. Harrer, J. Lenhard, and G. Wirtz. “On the Evolution of BPMN 2.0 Support and Implementation.” In: *International Symposium on Service-Oriented System Engineering*. SOSE ’16. Mar. 2016, pp. 101–110 (cit. on pp. 41, 148, 189).

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. “An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure.” In: *Distributed and parallel Databases* 3.2 (1995), pp. 119–153 (cit. on p. 16).

[GL06] V. Gruhn and R. Laue. “Complexity Metrics for Business Process Models.” In: *International Conference on Business Information Systems*. Vol. 85. BIS 2006. 2006, pp. 1–12 (cit. on pp. 165, 177, 179).

[GMG13] E. Goettelmann, N. Mayer, and C. Godart. “A General Approach for a Trusted Deployment of a Business Process in Clouds.” In: *International Conference on Management of Emergent Digital EcoSystems*. MEDES ’13. Luxembourg, Luxembourg: Association for Computing Machinery ACM, 2013, pp. 92–99 (cit. on p. 45).

[GN95] A. Gupta and N. Nishimura. *The Complexity of Subgraph Isomorphism: Duality Results for Graphs of Bounded Path- and Tree-Width*. Tech. rep. University of Waterloo. Computer Science Department., 1995 (cit. on p. 102).

[GR15] W. Grossmann and S. Rinderle-Ma. *Fundamentals of Business Intelligence*. Data-Centric Systems and Applications. Springer-Verlag Berlin Heidelberg, 2015 (cit. on pp. 102, 106).

[Gra92] J. Gray. *The Benchmark Handbook for Database and Transaction Systems*. Ed. by J. Gray. 2nd. Morgan Kaufmann, 1992 (cit. on p. 37).

[Gup14] A. Gupta. “Generating Large-Scale Heterogeneous Graphs for Benchmarking.” In: *Specifying Big Data Benchmarks*. Springer Science + Business Media, 2014, pp. 113–128 (cit. on p. 43).

[GW13] M. Geiger and G. Wirtz. “BPMN 2.0 Serialization - Standard Compliance Issues and Evaluation of Modeling Tools.” In: *International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA ’13)*. Vol. Vol. 2. Lecture Notes in Informatics. St. Gallen, Switzerland, Sept. 2013, pp. 177–190 (cit. on pp. 136, 149).

[HHGR06] G. Hackmann, M. Haitjema, C. Gill, and G.-C. Roman. “Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices.” In: *International Conference of Service Oriented Computing*. ICSOC ’06. Springer, 2006, pp. 503–508 (cit. on pp. 38, 41).

[HJ14] M. Hertis and M. Juric. “An Empirical Analysis of Business Process Execution Language Usage.” In: *IEEE Transactions on Software Engineering* 40.8 (Aug. 2014), pp. 738–757 (cit. on p. 54).

[HKLS14] F. Haupt, D. Karastoyanova, F. Leymann, and B. Schroth. “A Model-Driven Approach for REST Compliant Services.” In: *International Conference on Web Services*. ICWS ’14. Institute of Electrical & Electronics Engineers (IEEE), June 2014 (cit. on p. 156).

[HLW12] S. Harrer, J. Lenhard, and G. Wirtz. “BPEL Conformance in Open Source Engines.” In: *International Conference on Service Oriented Computing & Applications*. SOCA ’12. Institute of Electrical & Electronics Engineers (IEEE) Computer Society, 2012, pp. 1–8 (cit. on p. 41).

[Hol95] D. Hollingsworth. “The Workflow Reference Model.” In: *Workflow Management Coalition* 68 (1995) (cit. on pp. 16, 17).

[HRW14] S. Harrer, C. Rock, and G. Wirtz. “Automated and Isolated Tests for Complex Middleware Products: The Case of BPEL Engines.” In: *International Conference on Software Testing, Verification and Validation Workshops*. ICSTW ’14. Mar. 2014, pp. 390–398 (cit. on p. 41).

[HSA+14] M. Hahn, S. G. Sáez, V. Andrikopoulos, D. Karastoyanova, and F. Leymann. “Development and Evaluation of a Multi-tenant Service Middleware PaaS Solution.” In: *International Conference on Utility and Cloud Computing*. UCC ’14. Institute of Electrical & Electronics Engineers (IEEE) Computer Society, 2014, pp. 278–287 (cit. on p. 39).

[Hup09] K. Huppler. “The Art of Building a Good Benchmark.” In: *Lecture Notes in Computer Science*. Springer Science + Business Media, 2009, pp. 18–30 (cit. on pp. 13, 73, 75, 76).

[HW04] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2004 (cit. on p. 34).

[HZ06] C. Hentrich and U. Zdun. “Patterns for Business Object Model Integration in Process-driven and Service-oriented Architectures.” In: *Conference on Pattern Languages of Programs*. PLoP ’06. Portland, Oregon, 2006, pp. 1–14 (cit. on p. 37).

[IC07] Intel and Cape Clear. *BPEL Scalability and Performance Testing*. White paper. 2007 (cit. on p. 73).

[IFP15] A. Ivanchikj, V. Ferme, and C. Pautasso. “BPMeter: Web Service and Application for Static Analysis of BPMN 2.0 Collections.” In: *International Conference on Business Process Management*. BPM Demo ’15. CEUR-WS.org, 2015, pp. 30–34 (cit. on p. 173).

[ISO13] ISO/IEC. *ISO/IEC 19510:2013 – Information technology - Object Management Group Business Process Model and Notation*. V2.0.2. Nov. 2013 (cit. on pp. 14, 16, 69, 84, 85, 106, 133, 135, 138, 149, 165, 177).

[Iva14] A. Ivanchikj. “Characterising Representative Models for BPMN 2.0 Workflow Engine Performance Evaluation.” MA thesis. Universitá della Svizzera Italiana, Sept. 2014. URL: `https://thesis.bul.sbu.usi.ch/theses/1235-1314Ivanchikj/pdf?1412857872` (cit. on pp. 178, 179).

[JCD+13] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi. “Characterizing and Profiling Scientific Workflows.” In: *Future Generation Computer Systems* 29.3 (Mar. 2013), pp. 682–692 (cit. on p. 43).

[JWR+13] T. Jin, J. Wang, M. L. Rosa, A. ter Hofstede, and L. Wen. “Efficient Querying of Large Process Model Repositories.” In: *Computers in Industry* 64.1 (2013), pp. 41–49 (cit. on pp. 47, 54).

[Kar95] D. Karagiannis. “BPMS: Business Process Management Systems.” In: *SIGOIS Bull.* 16.1 (Aug. 1995), pp. 10–13 (cit. on p. 16).

[Kha07] R. Khalaf. “From RosettaNet PIPs to BPEL Processes: A Three Level Approach for Business Protocols.” In: *Data & Knowledge Engineering* 61.1 (Apr. 2007), pp. 23–38 (cit. on p. 55).

[Kha08] R. Khalaf. “Supporting Business Process Fragmentation While Maintaining Operational Semantics: a BPEL Perspective.” Dissertation. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Mar. 2008 (cit. on p. 52).

[KHSB12] S. Kounev, N. Huber, S. Spinner, and F. Brosig. “Model-Based Techniques for Performance Engineering of Business Information Systems.” In: *Lecture Notes in Business Information Processing*. Springer Science + Business Media, 2012, pp. 19–37 (cit. on pp. 42, 71, 152).

[KKL06] R. Khalaf, A. Keller, and F. Leymann. “Business processes for Web Services: Principles and applications.” In: *IBM Systems Journal* 45.2 (2006), pp. 425–446 (cit. on p. 40).

[KKL07] R. Khalaf, O. Kopp, and F. Leymann. “Maintaining Data Dependencies Across BPEL Process Fragments.” In: *International Conference in Service-Oriented Computing*. ICSOC ’07. Springer Science + Business Media, 2007, pp. 207–219 (cit. on p. 33).

[KL12] M. Kim and J. Leskovec. “Multiplicative Attribute Graph Model of Real-World Networks.” In: *Internet Mathematics* 8.1-2 (Mar. 2012), pp. 113–160 (cit. on p. 44).

[KLSU11] O. Kopp, F. Leymann, D. Schumm, and T. Unger. “On BPMN Process Fragment Auto-Completion.” In: *Central-European Workshop on Services and their Composition*. Ed. by D. Eichborn, A. Koschmider, and H. Zhang. ZEUS ’11. CEUR-WS.org, 2011, pp. 58–64 (cit. on p. 57).

[KLWW11] M. Kunze, A. Luebbe, M. Weidlich, and M. Weske. “Towards Understanding Process Modeling – The Case of the BPM Academic Initiative.” In: *Business Process Model and Notation*. Ed. by R. Dijkman, J. Hofstetter, and J. Koehler. Vol. 95. BPMN ’11. Springer Berlin Heidelberg, 2011, pp. 44–58 (cit. on p. 46).

[KtHvdA03] B. Kiepuszewski, A. ter Hofstede, and W. van der Aalst. “Fundamentals of Control Flow in Workflows.” In: *Acta Informatica* 39.3 (Mar. 2003), pp. 143–209 (cit. on p. 57).

[LDUD13] M. La Rosa, M. Dumas, R. Uba, and R. Dijkman. “Business Process Model Merging: An Approach to Business Process Consolidation.” In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22.2 (Mar. 2013), pp. 1–42 (cit. on p. 56).

[Len16] J. Lenhard. “Portability of Process-Aware and Service-Oriented Software: Evidence and Metrics.” PhD thesis. University of Bamberg, Germany, 2016 (cit. on p. 52).

[Ley01] F. Leymann. “Managing Business Processes via Workflow Technology.” In: *International Conference on Very Large Data Bases*. VLDB ’01. 2001 (cit. on pp. 19, 47, 59, 68).

[LGZ07] J. Liu, I. Gorton, and L. Zhu. “Performance Prediction of Service-Oriented Applications based on an Enterprise Service Bus.” In: *IEEE Annual international Computer Software and Applications Conference*. Ed. by Huimin Lin et al. Beijing, China: IEEE Computer Society, July 2007, pp. 327–334 (cit. on p. 14).

[LLHX10] A. Liu, Q. Li, L. Huang, and M. Xiao. “Facts: A Framework for Fault-Tolerant Composition of Transactional Web Services.” In: *IEEE Transactions on Services Computing* 3.1 (2010), pp. 46–59 (cit. on p. 39).

[LMJ10] G. Li, V. Muthusamy, and H.-A. Jacobsen. “A Distributed Service-Oriented Architecture for Business Process Execution.” In: *ACM Transactions on the Web* 4.1 (Jan. 2010), pp. 1–33 (cit. on p. 40).

[LR00] F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Ed. by C. Little. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000 (cit. on pp. 16, 42, 83).

[LR97] F. Leymann and D. Roller. “Workflow-Based Applications.” In: *IBM Systems Journal* 36.1 (1997), pp. 102–123 (cit. on p. 21).

[LRvdA+11] M. La Rosa, H. A. Reijers, W. M. van der Aalst, R. M. Dijkman, J. Mendling, M. Dumas, and L. García-Bañuelos. “APROMORE: An advanced process model repository.” In: *Expert Systems with Applications* 38.6 (2011), pp. 7029–7040 (cit. on pp. 46, 48, 54, 58).

[MA01] D. A. Menasce and V. Almeida. *Capacity Planning for Web Services: Metrics, Models, and Methods*. 1st. Prentice Hall, 2001 (cit. on p. 75).

[Ma12] Z. Ma. “Process Fragments: Enhancing Reuse of Process Logic in BPEL Process Models.” PhD thesis. Doctoral dissertation, University of Stuttgart, 2012 (cit. on pp. 46, 50, 54).

[MAD99] D. A. Menascé, V. A. Almeida, and L. W. Dowdy. *Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems*. Ed. by D. Mosco. Prentice-Hall, Inc., Mar. 11, 1999 (cit. on p. 19).

[MBM09] M. R. Mendes, P. Bizarro, and P. Marques. *Performance Evaluation and Benchmarking*. Ed. by R. Nambiar and M. Poess. Berlin, Heidelberg: Springer-Verlag, 2009. Chap. A Performance Study of Event Processing Systems (cit. on pp. 41, 82).

[Mer14] D. Merkel. “Docker: Lightweight Linux Containers for Consistent Development and Deployment.” In: *Linux J.* 2014.239 (Mar. 2014) (cit. on p. 190).

[ML08] Z. Ma and F. Leymann. “A Lifecycle Model for Using Process Fragment in Business Process Modeling.” In: *International Workshop on Business Process Modeling, Development and Support*. BPMDS ’08. conjuction with CAiSE, France, 2008, pp. 1–9 (cit. on p. 46).

[MMB16] R. Mrasek, J. Mülle, and K. Böhm. “Process Synthesis with Sequential and Parallel Constraints.” In: *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*. Springer Nature, 2016, pp. 43–60 (cit. on p. 34).

[Mol14] I. Molyneaux. *The Art of Application Performance Testing: From Strategy to Tools*. 2nd. O’Reilly Media, Inc., 2014 (cit. on pp. 71, 77).

[MR03] D. C. Montgomery and G. C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley and Sons, 2003 (cit. on p. 89).

[MR08] M. Muehlen and J. Recker. “How Much Language is Enough? Theoretical and Practical Use of the Business Process Modeling Notation.” In: *International Conference on Advanced Information Systems Engineering*. CAiSE ’08. Springer. Springer Berlin Heidelberg, 2008, pp. 465–479 (cit. on pp. 106, 182).

[MTJ+10] H. Mili, G. Tremblay, G. B. Jaoude, É. Lefebvre, L. Elabed, and G. E. Boussaidi. “Business Process Modeling Languages: Sorting Through the Alphabet Soup.” In: *ACM Computing Surveys* 43.1 (Dec. 2010), pp. 41–456 (cit. on p. 16).

[NGY+16] A. A. Nacer, E. Goettelmann, S. Youcef, A. Tari, and C. Godart. “Obfuscating a Business Process by Splitting Its Logic with Fake Fragments for Securing a Multi-cloud Deployment.” In: *World Congress on Services*. SERVICES ’16. June 2016, pp. 18–25 (cit. on pp. 45, 52).

[NK05] S. Nijssen and J. N. Kok. “The Gaston Tool for Frequent Subgraph Mining.” In: *Electronic Notes in Theoretical Computer Science* 127.1 (Mar. 2005), pp. 77–87 (cit. on pp. 53, 102).

[Obe06] D. Oberle. *Semantic Management of Middleware*. Vol. 1. Semantic Web and Beyond. New York: Springer, 2006 (cit. on p. 35).

[Org07] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*. Apr. 2007. URL: `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html` (cit. on pp. 14, 16).

[Ory08] Oryx. `http://bpt.hpi.uni-potsdam.de/Oryx/WebHome`. 2008 (cit. on p. 49).

[Pega] Pegasus WMS. *Workflow Gallery*. URL: `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator` (cit. on p. 42).

[Pegb] Pegasus WMS. *Workflow Generator*. URL: `https://pegasus.isi.edu/workflow_gallery/index.php` (cit. on p. 42).

[PF08] R. Prodan and T. Fahringer. “Overhead Analysis of Scientific Workflows in Grid Environments.” In: *IEEE Transactions on Parallel and Distributed Systems* 19.3 (Mar. 2008), pp. 378–393 (cit. on p. 40).

[PFR+15] C. Pautasso, V. Ferme, D. Roller, F. Leymann, and M. Skouradaki. "Towards Workflow Benchmarking: Open Research Challenges." In: *Conference on Database Systems for Business, Technology, and Web*. BTW 2015. 2015, pp. 331–350 (cit. on pp. 14, 18, 19, 38, 40, 41, 48, 57, 79, 80, 195, 201).

[PGD10] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas. "Structuring Acyclic Process Models." In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 276–293 (cit. on p. 57).

[PJMR14] A. Petermann, M. Junghanns, R. Müller, and E. Rahm. "Graph-based data integration and business intelligence with BIIIG." In: *Proceedings of the VLDB Endownment* 7.13 (Aug. 2014), pp. 1577–1580 (cit. on p. 42).

[Poe12] M. Poess. "TPC's Benchmark Development Model: Making the First Industry Standard Benchmark on Big Data a Success." In: *WBDB*. Vol. 8163. Lecture Notes in Computer Science. Springer, 2012, pp. 1–10 (cit. on p. 77).

[PSB15] D. J. Power, R. Sharda, and F. Burstein. *Decision Support Systems*. Ed. by DSSResources.COM. John Wiley and Sons, Ltd, 2015. URL: http://dx.doi.org/10.1002/9781118785317.weom070211 (cit. on p. 168).

[PVV11] A. Polyvyanyy, J. Vanhatalo, and H. Völzer. "Simplified Computation and Generalization of the Refined Process Structure Tree." In: *Web Services and Formal Methods*. Ed. by M. Bravetti and T. Bultan. Vol. 6551. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 25–41 (cit. on p. 51).

[PW12] P. Pietsch and S. Wenzel. "Comparison of BPMN2 Diagrams." English. In: *Business Process Model and Notation*. Ed. by J. Mendling and M. Weidlich. Vol. 125. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2012, pp. 83–97 (cit. on pp. 55, 101, 129).

[PY14] P. Pietsch and H. Yazdi. *BPMAI 2 BPMN. Qudimo Project*. 2014. URL: http://pi.informatik.uni-siegen.de/qudimo/bpmn/transformation/ (cit. on pp. 68, 165).

[Ran91] M. Randić. “Generalized molecular descriptors.” In: *Journal of Mathematical Chemistry* 7.1 (Dec. 1991), pp. 155–168 (cit. on p. 56).

[RG08] L. Ramakrishnan and D. Gannon. *A Survey of Distributed Workflow Characteristics and Resource Requirements*. Tech. rep. TR671. Indiana Univeristy, 2008 (cit. on p. 42).

[RH14] C. Röck and S. Harrer. *Literature Survey of Performance Benchmarking Approaches of BPEL Engines*. Tech. rep. Otto-Friedrich University of Bamberg, 2014 (cit. on p. 41).

[RHW14] C. Röck, S. Harrer, and G. Wirtz. “Performance Benchmarking of BPEL Engines: A Comparison Framework, Status Quo Evaluation and Challenges.” In: *International Conference on Software Engineering and Knowledge Engineering*. SEKE ’14. 2014, pp. 31–34 (cit. on p. 41).

[RMD10] H. Reijers, J. Mendling, and R. Dijkman. “On the Usefulness of SubProcesses in Business Process Models.” In: *External Report, BPM center report, No. BPM-10-03*. Eindhoven: BPMcenter.org, 2010 (cit. on p. 50).

[Rol13] D. H. Roller. “Throughput Improvements for BPEL Engines: Implementation Techniques and Measurements applied in SWoM.” PhD thesis. University of Stuttgart, 2013 (cit. on pp. 17, 39, 41, 73, 168).

[RvdAH07] N. Russell, W. M. van der Aalst, and A. Hofstede. “All That Glitters Is Not Gold: Selecting the Right Tool for Your BPM Needs.” In: *Cutter IT Journal* 20.11 (2007), pp. 31–38 (cit. on pp. 38, 40).

[RYC16] G. Rosinosky, S. Youcef, and F. Charoy. “A Framework for BPMS Performance and Cost Evaluation on the Cloud.” In: *Workshop on Business Process Monitoring and Performance Analysis in the Cloud*. CloudCom ’16. Luxembourg, Luxembourg: Institute of Electrical & Electronics Engineers (IEEE), Dec. 2016 (cit. on p. 41).

[SAB+16] M. Skouradaki, T. Azad, U. Breitenbücher, O. Kopp, and F. Leymann. “A Decision Support System for the Performance Benchmarking of Workflow Management Systems.” In: *Advanced Summer School of Service Oriented Computing*. SummerSOC ’16. IBM Research Division, 2016, pp. 41–58 (cit. on pp. 24, 27, 72, 169).

[SAKL16] M. Skouradaki, V. Andrikopoulos, O. Kopp, and F. Leymann. “RoSE: Reoccurring Structures Detection in BPMN 2.0 Process Model Collections.” In: *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*. Springer Nature, 2016, pp. 263–281 (cit. on pp. 24, 27, 103, 104).

[SAL16] M. Skouradaki, V. Andrikopoulos, and F. Leymann. “Representative BPMN 2.0 Process Model Generation from Recurring Structures.” In: *International Conference on Web Services*. ICWS ’16. Institute of Electrical & Electronics Engineers (IEEE), June 2016, pp. 468–475 (cit. on pp. 25, 27, 132).

[SALM12] S. Strauch, V. Andrikopoulos, F. Leymann, and D. Muhler. “ESB^{MT}: Enabling Multi-Tenancy in Enterprise Service Buses.” In: *International Conference on Cloud Computing Technology and Science Proceedings*. CloudCom ’12. Institute of Electrical & Electronics Engineers (IEEE), Dec. 2012, pp. 456–463 (cit. on p. 39).

[SASL13] S. Strauch, V. Andrikopoulos, S. G. Saez, and F. Leymann. “ESB^{MT}: A Multi-tenant Aware Enterprise Service Bus.” In: *IJNGC* 4.3 (2013) (cit. on p. 39).

[SBK+12] S. Strauch, U. Breitenbücher, O. Kopp, F. Leymann, and T. Unger. “Cloud Data Patterns for Confidentiality.” In: *International Conference on Cloud Computing and Service Science*. CLOSER ’12. SciTePress, 2012, pp. 387–394 (cit. on pp. 45, 60).

[SCG06] V. Seidita, M. Cossentino, and S. Gaglio. “A Repository of Fragments for Agent Systems Design.” In: *Workshop on Objects and Agents*. WOA ’06. Catania, Italy, 2006, pp. 130–137 (cit. on p. 49).

[Sed12] J. Sedayao. *Enhancing Cloud Security Using Data Anonymization*. White Paper. Intel, 2012. URL: `http://www.intel.com/content/dam/www/public/us/en/documents/best-practices/enhancing-cloud-security-using-data-anonymization.pdf` (cit. on pp. 45, 61).

[SEH03] S. E. Sim, S. Easterbrook, and R. C. Holt. “Using Benchmarking to Advance Research: A Challenge to Software Engineering.” In: *International Conference on Software Engineering*. ICSE ’03. IEEE Computer Society. 2003, pp. 74–83 (cit. on p. 17).

[Sen15] R. Sen. *Develop Secure Cloud-Aware Application*. Tech. rep. IBM, 2015 (cit. on p. 152).

[SFL+15] M. Skouradaki, V. Ferme, F. Leymann, C. Pautasso, and D. H. Roller. "BPELanon: Protect Business Processes on the Cloud." In: *International Conference on Cloud Computing and Service Science*. CLOSER '15. Lisbon, Portugal: SciTePress, May 2015, pp. 241–250 (cit. on pp. 25, 27, 60, 64, 66, 167).

[SFP+16] M. Skouradaki, V. Ferme, C. Pautasso, F. Leymann, and A. van Hoorn. "Micro-Benchmarking BPMN 2.0 Workflow Management Systems with Workflow Patterns." In: *Advanced Information Systems Engineering*. Ed. by S. Nurcan, P. Soffer, M. Bajec, and J. Eder. CAISE '16. Springer International Publishing, 2016, pp. 67–82 (cit. on pp. 25, 27, 42, 73, 82, 85–87, 89–95, 168).

[SGHL15] M. Skouradaki, K. Görlach, M. Hahn, and F. Leymann. "Application of Sub-Graph Isomorphism to Extract Reoccurring Structures from BPMN 2.0 Process Models." In: *International Symposium on Service-Oriented System Engineering*. SOSE '15. Institute of Electrical & Electronics Engineers (IEEE), Apr. 2015, pp. 11–20 (cit. on pp. 25, 27, 103).

[SK10] M. Sonntag and D. Karastoyanova. "Next Generation Interactive Scientific Experimenting based on the Workflow Technology." In: *IASTED Technology Conferences / 696:MS / 697:CA / 698: WC / 699: EME / 700: SOE*. ACTA Press, 2010 (cit. on p. 40).

[SKBB09] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann. "Performance Evaluation of Message-Oriented Middleware Using the SPECjms2007 Benchmark." In: *Performance Evaluation* 66.8 (2009), pp. 410–434 (cit. on p. 35).

[SKCB07] K. Sachs, S. Kounev, M. Carter, and A. Buchmann. "Designing a Workload Scenario for Benchmarking Message-Oriented Middleware." In: *SPEC Benchmark Workshop*. Austin, Texas, USA: SPEC, Jan. 2007 (cit. on p. 75).

[SKK+11] D. Schumm, D. Karastoyanova, O. Kopp, F. Leymann, M. Sonntag, and S. Strauch. “Process Fragment Libraries for Easier and Faster Development of Process-based Applications.” In: *Journal of Systems Integration* 2.1 (2011), pp. 39–55 (cit. on pp. 51, 56, 136).

[SKLS10] D. Schumm, D. Karastoyanova, F. Leymann, and S. Strauch. “Fragmento: Advanced Process Fragment Library.” In: *19th International Conference on Information Systems Development, ISD*. Prague, Czech Republic, Aug. 2010 (cit. on pp. 49, 50, 58).

[SL15] M. Skouradaki and F. Leymann. “Detecting Frequently Recurring Structures in BPMN 2.0 Process Models.” In: *Advanced Summer School of Service Oriented Computing*. SummerSOC ’15. IBM Research Division, 2015 (cit. on pp. 25, 27, 103).

[SLM+10] D. Schumm, F. Leymann, Z. Ma, T. Scheibler, and S. Strauch. “Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls.” In: *The Multikonferenz Wirtschaftsinformatik, MKWI*. Universitätsverlag Göttingen, 2010 (cit. on pp. 51, 56).

[Spe99] W. M. C. Specification. *Workflow Management Coalition, Terminology & Glossary (Document No. WFMC-TC-1011)*. Workflow Management Coalition Specification, Feb. 1999 (cit. on p. 15).

[SRL+15] M. Skouradaki, D. H. Roller, F. Leymann, V. Ferme, and C. Pautasso. “On the Road to Benchmarking BPMN 2.0 Workflow Engines.” In: *ACM/SPEC International Conference on Performance Engineering*. ICPE ’15. Austin, Texas: Association for Computing Machinery ACM, 2015, pp. 301–304 (cit. on pp. 16, 25, 27, 32, 38, 40).

[SRPL14] M. Skouradaki, D. Roller, C. Pautasso, and F. Leymann. “BPELanon: Anonymizing BPEL Processes.” In: *Central European Workshop on Services and their Composition*. ZEUS ’14. 2014, pp. 9–15 (cit. on pp. 26, 60, 64).

[Sta07] Standard Performance Evaluation Corporation. *SPECjms2007*. 2007. URL: `https://www.spec.org/jms2007/` (cit. on pp. 34, 35, 73, 74, 168).

[Sta10] Standard Performance Evaluation Corporation. *SPEC SOA Subcommittee*. Feb. 2010. URL: `http://www.spec.org/soa/` (cit. on p. 36).

[Sta15] Standard Performance Evaluation Corporation. *SPECjbb2015*. 2015. URL: `https://www.spec.org/jbb2015/` (cit. on pp. 35, 73, 168).

[Sta95] Standard Performance Evaluation Corporation (SPEC). 1995. URL: `https://www.spec.%20org/spec/` (cit. on pp. 34, 72, 77, 201).

[STK+10] D. Schumm, O. Turetken, N. Kokash, A. Elgammal, F. Leymann, and W.-J. Van Den Heuvel. "Business Process Compliance Through Reusable Units of Compliant Processes." In: *International Conference on Current Trends in Web Engineering*. Vol. 6385. ICWE '10. Springer-Verlag, 2010, pp. 325–337 (cit. on pp. 33, 46, 50, 51).

[Tar72] R. Tarjan. "Depth First Search and Linear Graph Algorithms." In: *SIAM Journal on Computing* (1972) (cit. on pp. 111, 113, 115).

[TPC15] TPC. *TPC benchmark E standard specification, version 1.14*. `http://www.tpc.org/tpce/`. 2015 (cit. on pp. 37, 73, 168).

[TPC94a] TPC. *TPC benchmark A standard specification, revision 2.0*. `http://www.tpc.org/tpce/`. 1994 (cit. on p. 36).

[TPC94b] TPC. *TPC benchmark B standard specification, revision 2.0*. `http://www.tpc.org/tpcb`. 1994 (cit. on p. 36).

[TPK+13] P. Tözün, I. Pandis, C. Kaynak, D. Jevdjic, and A. Ailamaki. "From A to E: Analyzing TPC's OLTP Benchmarks: the Obsolete, the Ubiquitous, the Unexplored." In: *International Conference on Extending Database Technology*. EDBT'13. Association for Computing Machinery (ACM), 2013, pp. 17–28 (cit. on pp. 36, 37).

[Tra92a] Transaction Processing Performance Council. *TPC*. `http://www.tpc.org/information/benchmarks.asp`. 1992. URL: `http://www.tpc.org/tpch/` (cit. on pp. 21, 34, 72, 77, 201).

[Tra92b] Transaction Processing Performance Council. *TPC-H*. `http://www.tpc.org/tpcc/`. 1992 (cit. on pp. 36, 73, 168).

[TZJW08] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright. “A Nine Year Study of File System and Storage Benchmarking.” In: *Transaction Storage* 4.2 (2008), pp. 1–56 (cit. on p. 37).

[UEKL10] T. Unger, H. Eberle, O. Kopp, and F. Leymann. “The SubProcess Spectrum.” In: *International Conference on Business Process and Services Computing (BPSC ’10)*. Lecture Notes in Informatics. Gesellschaft für Informatik e.V. (GI), 2010 (cit. on pp. 50, 51).

[Ull76] J. R. Ullmann. “An Algorithm for Subgraph Isomorphism.” In: *ACM* 23.1 (1976), pp. 31–42 (cit. on pp. 53, 54, 102, 119, 126, 129, 199).

[Val02] G. Valiente. *Algorithms on Trees and Graphs*. Berlin; New York: Springer-Verlag Berlin Heidelberg, 2002 (cit. on pp. 108, 213).

[vdAal13] W. M. P. van der Aalst. “Business Process Management: A Comprehensive Survey.” In: *ISRN Software Engineering* 2013 (2013), pp. 1–37 (cit. on p. 202).

[vdAvD02] W. van der Aalst and B. van Dongen. “Discovering Workflow Performance Models from Timed Logs.” In: *Engineering and Deployment of Cooperative Information Systems*. Ed. by Y. Han, S. Tai, and D. Wikarski. Vol. 2480. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 45–63 (cit. on p. 202).

[vKAH+15] J. v. Kistowski, J. A. Arnold, K. Huppler, K.-D. Lange, J. L. Henning, and P. Cao. “How to Build a Benchmark.” In: *ACM/SPEC International Conference on Performance Engineering*. ICPE ’15. Association for Computing Machinery (ACM), 2015 (cit. on pp. 15, 31).

[VKL06] J. Vanhatalo, J. Koehler, and F. Leymann. “Repository for Business Processes and Arbitrary Associated Metadata.” In: *International Conference on Business Process Management*. BPM Demo ’16. Vienna, Austria: CEUR-WS.org, 2006 (cit. on p. 49).

[VM97] G. Valiente and C. Martínez. “An Algorithm for Graph Pattern-Matching.” In: *South American Workshop on String Processing*. Vol. 8. International Informatics Series. Carleton University Press, 1997, pp. 180–197 (cit. on pp. 53, 102, 202).

[VMSS12] M. Vieira, H. Madeira, K. Sachs, and K. S. *Resilience Assessment and Evaluation of Computing Systems*. Ed. by K. Wolter, A. Avritzer, M. Vieira, and A. van Moorsel. Springer Science + Business Media, 2012 (cit. on pp. 31, 71).

[VMZ+10] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins. "A Comparison of a Graph Database and a Relational Database." In: *ACM Southeast Conference*. SE '10. Association for Computing Machinery (ACM), 2010 (cit. on pp. 42, 43).

[VP12] S. Vinogradov and A. Pastsyak. "Evaluation of Data Anonymization Tools." In: *International Conference on Advances in Databases, Knowledge, and Data Applications*. DBKDA '12. 2012, pp. 163–168 (cit. on p. 45).

[VTKB03] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. "Workflow Patterns." In: *Distributed Parallel Databases* 14.1 (July 2003), pp. 5–51 (cit. on pp. 82, 87, 182, 198).

[VVK08] J. Vanhatalo, H. Völzer, and J. Koehler. "The Refined Process Structure Tree." In: *Business Process Management*. Ed. by M. Dumas, M. Reichert, and M.-C. Shan. Vol. 5240. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 100–115 (cit. on pp. 51, 56).

[Wal14] J. Waller. *Benchmark for: Performance Benchmarking of Application Monitoring Frameworks*. Aug. 2014 (cit. on pp. 17, 18, 20).

[WH13] J. Waller and W. Hasselbring. "A Benchmark Engineering Methodology to Measure the Overhead of Application-Level Monitoring." In: *KPDAYS*. Vol. 1083. CEUR Workshop Proceedings. CEUR-WS.org, 2013, pp. 59–68 (cit. on pp. 33, 41, 81).

[Win00] WinEdt. *WinEdt Dictionaries*. Aug. 2000. URL: `http://www.winedt.org/Dict/` (cit. on p. 66).

[WJWW13] J. Wang, T. Jin, R. K. Wong, and L. Wen. "Querying Business Process Model Repositories." In: *World Wide Web* 17.3 (Apr. 2013), pp. 427–454 (cit. on p. 54).

[WLR+09] B. Wetzstein, P. Leitner, F. Rosenberg, I. Brandic, S. Dustdar, and F. Leymann. “Monitoring and Analyzing Influential Factors of Business Process Performance.” In: *Proc. of the IEEE International on Enterprise Distributed Object Computing Conference*. EDOC ’09. 2009, pp. 141–150 (cit. on p. 38).

[WR+00] C. Wohlin, P. Runeson, et al. *Experimentation in Software Engineering: An Introduction*. Kluwer, 2000 (cit. on p. 97).

[WRMR11] B. Weber, M. Reichert, J. Mendling, and H. A. Reijers. “Refactoring large process model repositories.” In: *Computers in Industry* 62.5 (2011), pp. 467–486 (cit. on p. 51).

[WvdAD+06] P. Wohed, W. M. van der Aalst, M. Dumas, A. H. ter Hofstede, and N. Russell. “On the Suitability of BPMN for Business Process Modelling.” In: *Lecture Notes in Computer Science* 4102 (2006), pp. 161–176 (cit. on p. 84).

[YDG12] Z. Yan, R. Dijkman, and P. Grefen. “Business Process Model Repositories - Framework and Survey.” In: *Information and Software Technology* 54.4 (Apr. 2012), pp. 380–395 (cit. on pp. 46, 47).

[YDG15] Z. Yan, R. Dijkman, and P. Grefen. “Generating process model collections.” In: *Software and Systems Modeling (SoSyM)* (Oct. 2015) (cit. on pp. 55, 56, 101, 129, 132).

[YH02] X. Yan and J. Han. “gSpan: Graph-Based Substructure Pattern Mining.” In: *International Conference on Data Mining*. ICDM ’02. Institute of Electrical & Electronics Engineers (IEEE), 2002 (cit. on pp. 22, 53, 102).

[YWH+07] W. Yurcik, C. Woolam, G. Hellings, L. Khan, and B. M. Thuraisingham. “Toward Trusted Sharing of Network Packet Traces Using Anonymization: Single-Field Privacy/Analysis Tradeoffs.” In: *Computer Research Repository (CoRR)* abs/0710.3979 (2007), pp. 1–8 (cit. on p. 44).

[ZLN+14] X. Zhang, C. Liu, S. Nepal, C. Yang, and J. Chen. “Privacy Preservation over Big Data in Cloud Systems.” In: *Security, Privacy and Trust in Cloud Systems*. Ed. by S. Nepal and M. Pathan. Security, Privacy and Trust in Cloud Systems. Springer Berlin Heidelberg, 2014, pp. 239–257 (cit. on pp. 45, 61).

[ZMH16] M. A. Zemni, A. Mammar, and N. B. Hadj-Alouane. “An Automated Approach for Merging Business Process Fragments.” In: *Computers in Industry* 82 (Oct. 2016), pp. 104–118 (cit. on p. 56).