Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Analysis and Evaluation of Modeling and Composition Languages for Microservices

Pedram Hamidehkhan

Course of Study:	Computer Science
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	Michael Wurster, M.Sc.
Commenced:	July 16, 2018
Completed:	January 16, 2019

Acknowledgment

I would like to express my sincere gratitude to Professor Leymann for giving me the opportunity to conduct my master thesis at the Institute of Architecture of Application Systems at the University Stuttgart.

I am also very grateful to my supervisor M.Sc. Michael Wurster of the University of Stuttgart for his continuous guidance and support and his valuable inputs during this thesis.

I am profoundly grateful to my parents for their love and their constant encouragement throughout my life.

Abstract

Microservices architecture (MSA) is an approach which fosters using granular services formed around business capabilities. Although this brings many advantages regarding scalability and makes the application more flexible, the orchestration and communication of these services become more challenging. Having granular microservices means that end-to-end business processes generally span several microservices and the logic and the order of execution of these microservices must be specified. The industry has foreseen this challenge and companies such as Uber and Netflix have developed their orchestration engines. Moreover, composition languages have emerged which emphasize the need for new programming languages with special constructs particularly designed for MSA.

In this work, we evaluate and analyze the tools, driven by both industry and academia, which are suitable for microservices composition and orchestration in order to find a reference architecture. To determine the current state of research as well as industry with regards to microservices composition and orchestration, a *systematic literature review* (SLR) is conducted which discovers the existing tools and enable performing an unbiased evaluation. Then the discovered tools are evaluated and analyzed to find a reference architecture. Based on the results of the evaluation, a prototype is implemented which proposes a reference architecture for the orchestration engines.

Table of Contents

List of Figures

List of Tables

List of abbreviations

MSA Microservices Architecture.

SLR Systematic Literature Review.

EIP Enterprise Integration Patterns.

MDD Model-Driven Development.

BDD Behavior-Driven Development.

MDE Model-Driven Engineering.

IoT Internet of Things.

JSON JavaScript Object Notation.

XML Extensible Markup Language.

UML Unified Modeling Language.

BPMN Business Process Model and Notation.

BPEL Business Process Execution Language.

RPC Remote Procedure Call.

API Application Programming Interface.

DSL Domain Specific Language.

REST Representational State Transfer.

EGL Epsilon Generation Language.

SDK Software Development Kit.

1. Introduction

1.1. Background and Motivation

Microservices architecture (MSA) is a modern architectural style for software development which has received much attention recently. In his book [Newm15], Newman defines microservices as independent services communicating with each other to achieve a specific goal. The fact that these services are independent entities brings many improvements and flexibility in testing, deployment, development and error handling [Micr00]. It also enhances scalability and allows each microservice to be independently scaled. However, these benefits introduce some challenges as well. Having these granular independent services means that the execution of multiple services in specific orders can derive business value, and therefore, the way they interact has a pivotal role [InSi18].

In real-world scenarios, end-to-end business processes can be long-running, parallel, and also sometimes might require human intervention [InSi18]. Furthermore, direct communication between microservices couples them together and thereby reduces scalability, especially when the number of microservices is high. To address this challenge, companies like Netflix [Cond16] and Uber [Uber17] have developed their orchestration engines. These orchestration engines separate the orchestration logic of these microservices from their implementation.

Moreover, new approaches to the composition of microservices are also introduced. Although it is possible to develop microservices in any programming language, such as C# and Java, and containerize them afterward [Khan17], there is an alternative way which can be a complement to this approach. Languages such as Jolie [Joli07] and Ballerina [Ball17] have emerged with the goal of addressing the challenges of MSA. These languages aim to solve limitations of general-purpose programming languages like Java by providing support for constructs necessary for MSA [MoGZ14].

In this work, we perform a *systematic literature review* (SLR) to discover and investigate modeling and composition tools which are suitable for microservices orchestration. The SLR process makes it possible to evaluate these tools and approaches in an unbiased manner.

After conducting the SLR and finding the available approaches and tools, the specifications of the discovered tools are briefly discussed, and an evaluation is conducted. The criteria for the evaluation are derived based on the way a microservice must communicate with an orchestration engine and the requirements to which the microservice must conform. Using these criteria, all the discovered tools are evaluated to find out how microservices must be specified with these tools. Furthermore, the results of the evaluation step are utilized to propose a model which abstracts the requirements of each tool from a microservice. This gives the basis for the implementation of a prototype which acts as an abstraction layer between a microservice and an orchestration engine. The prototype defines a set of requirements which are abstracted from all the discovered orchestration tools. This enables a microservice to be used within the orchestration platforms without needing to change its model to each orchestration engine.

1.2. Thesis Structure

This thesis is structured into seven chapters as follows:

Chapter 1 (Introduction): A brief introduction to microservices composition and orchestration engines as well as the motivation and structure of this thesis.

Chapter 2 (Related Work): The related work is presented.

Chapter 3 (Systematic Literature Review): A systematic literature review (SLR) which aims to determine the current state of the research and discover related tools is performed.

Chapter 4 (Modeling and Composition Tools): The specifications and characteristics of the tools discovered in the SLR chapter are discussed.

Chapter 5 (Evaluation): The discovered tools are analyzed and evaluated using a set of criteria.

Chapter 6 (Implementation): A prototype which abstracts the specifications of the evaluated tools is proposed.

Chapter 7 (Discussion): This chapter summarizes this document and discusses the possibilities of future work.

2. Related Work

In this chapter, we outline the related work which focuses on the similar problem to this thesis.

The difficulty of managing microservices increases as the microservice-based application grows. In addition to identifying similar tasks for improving the reusability of services, it is necessary to specify how they communicate [Khan17].

Container orchestration platforms, such as Kubernetes[1] and Docker Swarm[2], are introduced which can manage and orchestrate containerized microservices. Containers are self-contained portable units which contain processes and their required dependencies packaged together [Khan17]. The orchestration platforms provide logging capabilities for the infrastructure as well as activities of containers. Rollback and isolation mechanisms offered by the orchestration engines enable support for continuous deployment pipelines [Khan17]. In [Khan17], Khan presents the essential functionalities of container orchestration platforms. The twelve-factor app methodology is used to determine the necessary functionalities of a container orchestration platform. Given the orchestration of containerized microservices is investigated, the focus is on the deployment aspect of microservices.
[DüHo17], [RSSZ18], and [Petr17] introduce metamodels using model-driven approaches. In [RSSZ18], Rademacher et al. propose a metamodel which is mainly focused on applying *Model-Driven Development* (MDD) concepts to microservices architecture. In [DüHo17], a metamodel is suggested which concerns the deployment environment of microservices. The metamodel concentrates on dependencies and deployment aspect of microservices architecture. It defines the deployment environment as a physical host, virtual machine or a container. To address the agility requirements of an application based on microservices architecture, a configuration component is proposed which describe the state of deployed microservices, and the interaction is performed via REST operations [DüHo17]. A code generation tool is introduced which can generate the code of the microservices using the metamodel. The metamodels introduced in [Petr17] focus on how the boundaries of microservices should be modeled and with regards to the communication aspect of microservices, *Enterprise Integration Patterns* (EIP) are applied.

This work investigates composition languages and orchestration tools for microservices and does not consider the container technologies. The works mentioned above mainly focus on how microservices should be deployed, but the focus of this work is the modeling and composition aspect of microservices. Moreover, the proposed metamodel in this work aims to obtain an abstraction on the requirements of each orchestration tool from a microservice. Because of the existing gap between industry and academia regarding the topic of interest, we conduct an SLR to determine the current state of research. After discovering and introducing the suitable tools, an evaluation is performed. The evaluation focuses on the requirements of the mentioned tools from the microservices. Finally, a prototype implementation is presented which acts as an abstraction layer between the microservice and the platform. The tool enables the use of the

[1] www.kubernetes.io
[2] www.docs.docker.com/swarm/overview

microservice within the mentioned platforms without needing to adapt to the requirements of each platform.

3. Systematic Literature Review

In this chapter, the *Systematic Literature Review* (SLR) performed for this thesis is presented. SLR is a means of identifying, evaluating and interpreting available research in the subject of interest in an unbiased manner [Kitc00a]. Based on existing guidelines by Cochrane as well as Australian National Health and Medical Research Council and CRD, Kitchenham [Kitc00a] has suggested an SLR process adapted for Software Engineering domain, which is the method used in this work.

An SLR is composed of three main phases which are to be completed one after the other. Figure 3.1 illustrates the phases in of an SLR and the order in which they are to be conducted.

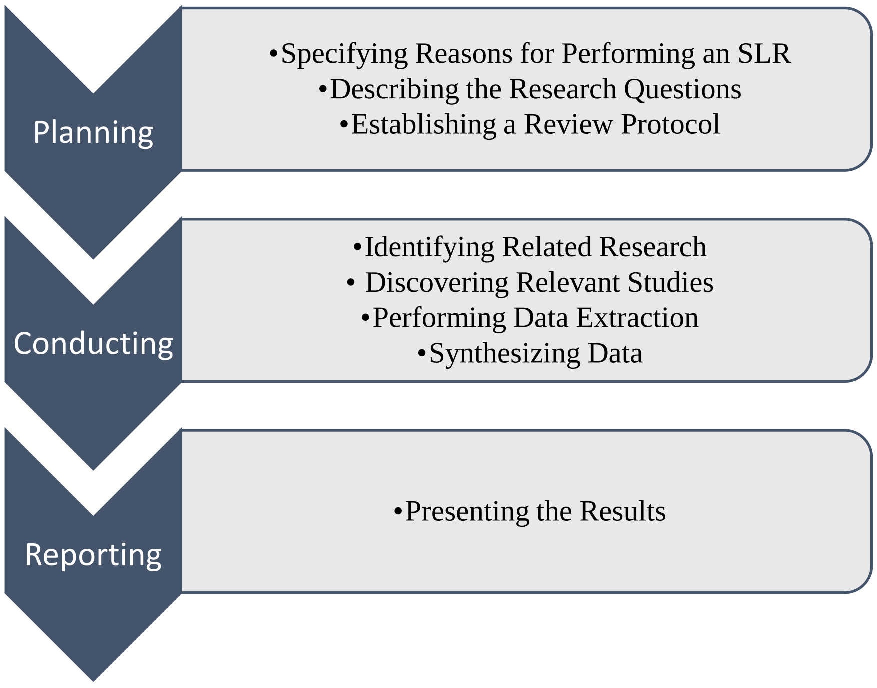
Planning
•Specifying Reasons for Performing an SLR
•Describing the Research Questions
•Establishing a Review Protocol
Conducting
•Identifying Related Research
• Discovering Relevant Studies
•Performing Data Extraction
•Synthesizing Data
Reporting
•Presenting the Results

Figure 3.1: The steps of performing an SLR [Kitc00b]

Planning phase lays the foundation of the SLR. In this phase, first, the reasons for performing an SLR are identified. Afterward, the research questions are defined, and then, a pre-defined protocol is developed in order to avoid bias and also to make the review transparent. Then the conducting phase begins with identifying relevant research. After that, the necessary data to answer the defined research questions are extracted and synthesized. Then, the results of the review are reported [Kitc00a]. Kitchenham also proposes some optional steps in performing SLR [Kitc00a], which are not relevant to this work and hence discarded.

3.1. Planning

Starting with the first phase, the reasons and goals of the SLR must be declared. Then the research questions need to be identified. Moreover, the policy of the review must be protocoled [Kitc00a].

3.1.1. Reasons for Performing the SLR

Identifying the necessity to perform an SLR is an indispensable prerequisite to its further processing [Kitc00a]. The intention behind performing this SLR stems in the gap which exists between industry and academia with respect to composition, modeling and orchestrating microservices. Moreover, companies like Uber and Netflix have developed their orchestration engines [Cond16, Uber17] and they have been probably developed based on different models. Therefore, it is necessary to determine the current state of the domain and thoroughly identify and evaluate such tools in an unbiased manner.

3.1.2. Research Questions

Research questions are the forces giving direction to the SLR and therefore formulating them is of great importance [Kitc00a].

The declared research questions regarding the goal of the topic are as follows:

- RQ1: How much activity has there been with regards to modeling and composition languages as well as orchestration tools for microservices?
- RQ2: Who is active in this topic?
- RQ3: What are the available approaches for microservices composition?
    - RQ3.1: Which one is the focus of researchers?
    - RQ3.2: Is there a need for a specific language for microservices composition?
- RQ4: What tools and methods are available for orchestrating microservices?

RQ1 aims to obtain an overview of the current research. RQ2 is defined to recognize the active researchers and organizations in the fields. RQ3 aims to identify the alternative approaches for microservice design and find the tendency of researchers as well as examining the applicability of these approaches. Finally, RQ4 aims to identify and investigate orchestration methods and approaches in the field.

3.1.3. Review Protocol

In order to avoid having bias in the SLR as much as possible, a review protocol is determined [Kitc00a]. In this step, first, a procedure for searching for primary studies is defined. Moreover, criteria and the policy required for selecting relevant studies is determined. After that, an approach for obtaining data from the selected studies must be determined. Moreover, a method for synthesizing the data needs to be established.

In order to discover related studies, search phrases must be formed concerning research questions defined in section 3.1.2. Combination of keywords of the research questions as well as their synonyms, abbreviations, and different spelling form the search strings. These words are then connected with Boolean operands such as AND, OR and NOT. The keywords with respect to the research questions are:

- modeling, composition, language, framework, orchestration, tool, approach, microservice, language-based, choreography

Some of the introduced keywords have the following alternatives or abbreviation which will also be considered:

- Microservice: micro-service, MSA, cloud-native, cloud native
- Modeling: modelling

Additionally, as the tools related to this topic are mostly industry-driven, the following search strings are formed to be queried in the Google search engine[3].

- microservices programming language, microservices orchestration engine, microservices workflow engine

In order to improve the understandability, the mentioned search strings are presented in Figure 3.2 to Figure 3.8.

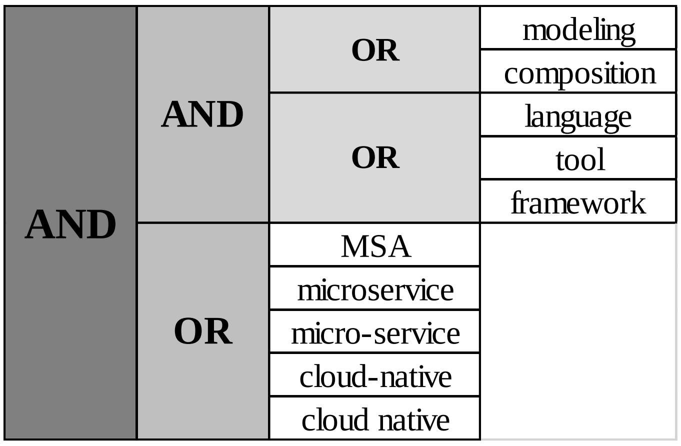

Figure 3.2: Derived Search Phrase 1

[3] www.google.com

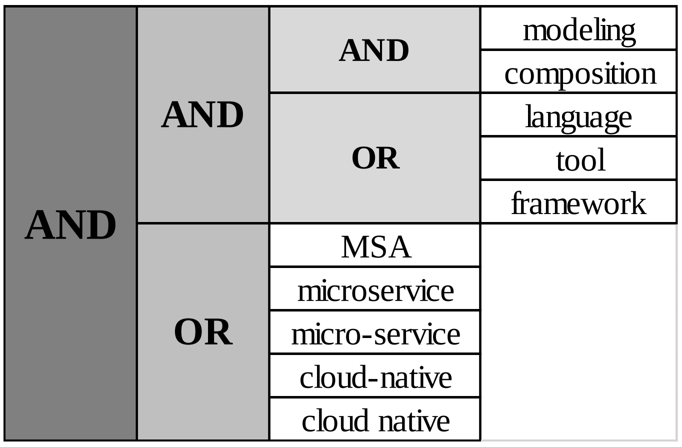

Figure 3.3: Derived Search Phrase 2

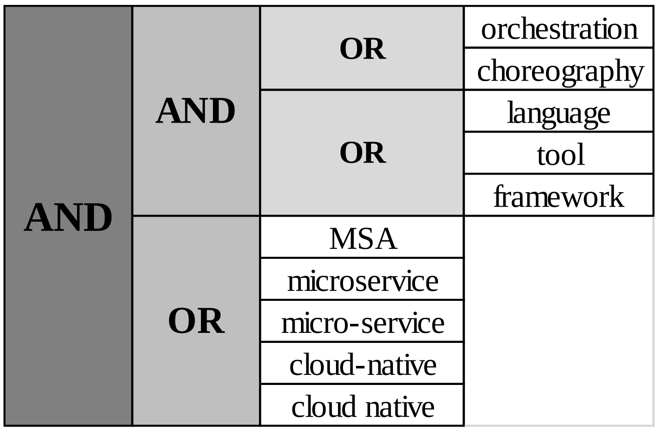

Figure 3.4: Derived Search Phrase 3

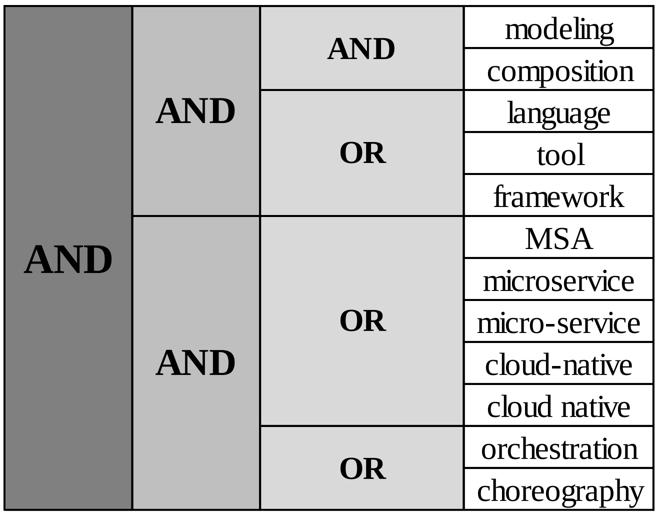

Figure 3.5: Derived Search Phrase 4

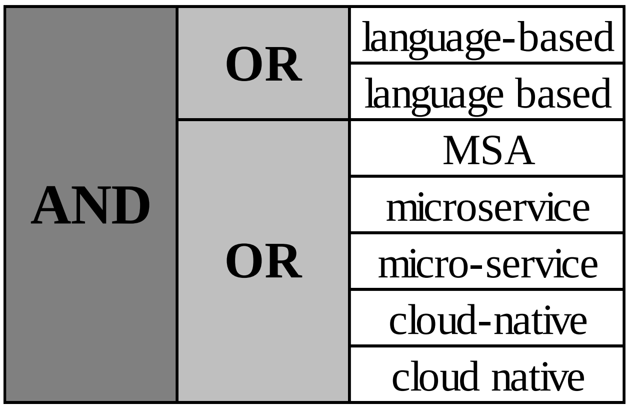

Figure 3.6: Derived Search Phrase 5

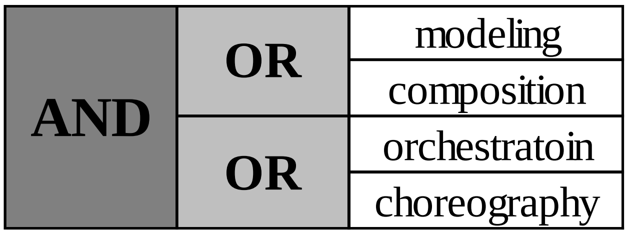

Figure 3.7: Derived Search Phrase 6

OR	"microservices programming language"
	"microservices orchestration engine"
	"microservices workflow engine"

Figure 3.8: Derived Search Phrase 7

Furthermore, the data sources to be scraped also need to be identified. As suggested by [Kitc00a] following data sources are used:

- IEEE Xplore[4]
- Science Direct[5]
- Google Scholar[6]

As searching the databases with the search terms could return a high number of documents, most of which are probably not related to the topic of interest, there is a need for filtering the results of the search [Kitc00a]. For this purpose, the inclusion and exclusion criteria indicate in Table 3.1 and Table 3.2 are defined.

	Inclusion Criteria
1	The study is about modeling or service composition language suitable for MSA.
2	The study is about service orchestration tools suitable for MSA.
3	The study is about different approaches to developing microservice.

Table 3.1: Inclusion Criteria

	Exclusion criteria
1	The study is in a language other than English, German or French: based on [Kitc00a] exclusion based on language should be avoided as much as possible (regarding the industry-driven tools, the tool is excluded if the provided documentation is in a language other than English, German or French).
2	When there are more than one publication of the same data, only one of them (the one which is complete) is included, and the rest are excluded.
3	The full text of the study is not retrievable.

Table 3.2: Exclusion Criteria

After specifying how relevant studies can be identified, the data extraction forms are designed to obtain the data from the selected studies in a systematic manner [Kitc00a]. From each study, one or more attributes must be extracted to answer the research questions. Thus, based on the research questions defined in section 3.1.2, the necessary attributes are specified. RQ1 requires the attributes ‘Title’ and ‘Year’. Attributes ‘Title’, ‘Author(s)’, ‘Organization’ and ‘Country’ are used to answer RQ2. Given RQ3, RQ4 and RQ5 are involve different approaches and concepts regarding modeling and orchestration of microservices, the attributes ‘Modeling and Composition Approach’, as well as ‘Orchestration Approach’, are defined. Following the

[4] www.ieeexplore.ieee.org
[5] www.sciencedirect.com
[6] www.scholar.google.com

suggested format by [Kitc00b], Table 3.3 is designed which contains the attributes required for performing data extraction.

Data Item	Value	Additional notes
Study Title		
Year		
Author(s)		
Institution/Organization		
Country		
Modeling and Composition Approach		
Orchestration Approach		
Main Contribution		

Table 3.3: Data Extraction Form

Moreover, with regards to composition and orchestration tools driven by the industry, Table 3.4 is designed to perform data extraction. The data is extracted from the Github repository as well as the official documentation for each tool.

Data Item	Value	Additional notes
Tool Name		
Year of First Release		
Contributor(s)		
Organization(s)		
Country(s)		
Repository		

Table 3.4: Data Extraction Form for Tools

After defining the data extraction forms, data synthesis forms must be designed with which the gathered data is synthesized to answer the defined research questions. Table 3.5 is designed to answer RQ1.

Year	Studies

Table 3.5: Distribution of Studies by Year

The objective of RQ2 is to gain insight on which authors, organizations and also countries are active in the field. Table 3.6 contains the attributes required by RQ2 which aims to obtain an overview of the activity in the field.

Study	Author	Organization	Country

Table 3.6: Activity in the Field

RQ3 concentrate on different approaches of microservices composition and to find out which approaches have been more investigated. Table 3.7 aims to provide the data to answer this question.

Available Approaches	Supporting Studies

Table 3.7: Available Approaches for Modeling and Composition

Moreover, as RQ4 concerns the orchestration tools, a similar set of attributes are required which are present in Table 3.8.

Orchestration Approach	Supporting Studies

Table 3.8: Available Orchestration Approaches

Additionally, Table 3.9 is designed to present the contribution of each paper. The filled form can be found in section 3.3.2.

Study	Contribution

Table 3.9: Main Contributions

3.2. Conducting

This step is responsible for performing the steps defined above [Kitc00a]. Using the defined protocol in the previous section, the selected databases are queried using the defined search terms. Then, based on the defined criteria the irrelevant are excluded. Afterward, data extraction and synthesizing are performed using the forms defined above.

3.2.1. Related Research

In this step, the defined search queries are run against the selected databases. Table 3.10 contains the initial result of querying the derived search strings in section 3.1.3.

Database	Number of Documents
IEEE Xplore	189
Science Direct	190
Google Scholar	186502
Google (industry-driven tools)	93

Table 3.10: Search Result 1

3.2.2. Relevant Studies

Given the high number of irrelevant studies found in the previous step, it is necessary to distinguish the relevant research and remove the irrelevant ones. To that end, the inclusion and exclusion criteria defined in section 3.1.3 are applied to the results. Given Google Scholar returns considerably higher results compared to other search engines, an additional step is performed to reduce the number of irrelevant results. The returned file type is set to PDF (‘filetype: pdf’), and language restriction is applied.

After applying the inclusion and exclusion criteria, the number of documents is shown in Table 3.11.

Database	Studies
IEEE Xplore	[Fran17, HaAB17, KWGJ17, MaCL17, MMPP16, Petr17, RaGa15, RaSS18, RaSZ17, SaSR17, SeGM18, SMMR16, TaLP18, ThVS18]
Science Direct	[Arel18, GFZV16, KoSi17]
Google Scholar	[BRBC16, GLMM17, GuLJ18, InSi18, KoDK18, LTOG18, MFCL18, MGMR18, Sorg17, StOb17, VCTG17]
Google (industry-driven tools)	[Awss00, Ball17, Cond16, Joli07, Toki17, Bake17, Rock17, Uber17, Zeeb17]

Table 3.11: Search Result 2

3.2.3. Data Extraction

In this stage, the data extraction forms defined in section 3.1.3 are filled with the information of the identified relevant studies. The tabulated studies using the extraction forms can be found in Appendix A. The filled data extraction forms of the industry-driven tools can be found in Appendix B.

3.2.4. Data Synthesis

This step is responsible for summarizing the data with the help of the defined synthesis forms. These forms will be used to answer the RQs which can be found in section 3.3.1.

3.3. Reporting

This step assures the efficient conveying of the result discovered during the previous steps. Here the synthesized data is reported, and the research questions are answered with respect to the findings.

3.3.1. Presenting the Results

RQ1: How much activity has there been with regards to modeling and composition languages as well as orchestration tools?

Year	Studies
2015	[RaGa15]
2016	[BRBC16, GFZV16, MMPP16, SMMR16]
2017	[Fran17, GLMM17, HaAB17, KoSi17, KWGJ17, MaCL17, Petr17, RaSZ17, SaSR17, Sorg17, StOb17, VCTG17]
2018	[Arel18, GuLJ18, InSi18, KoDK18, LTOG18, MFCL18, MGMR18, RaSS18, SeGM18, TaLP18, ThVS18]
Total Number of Studies	28

Table 3.12: Distribution of Studies by Year

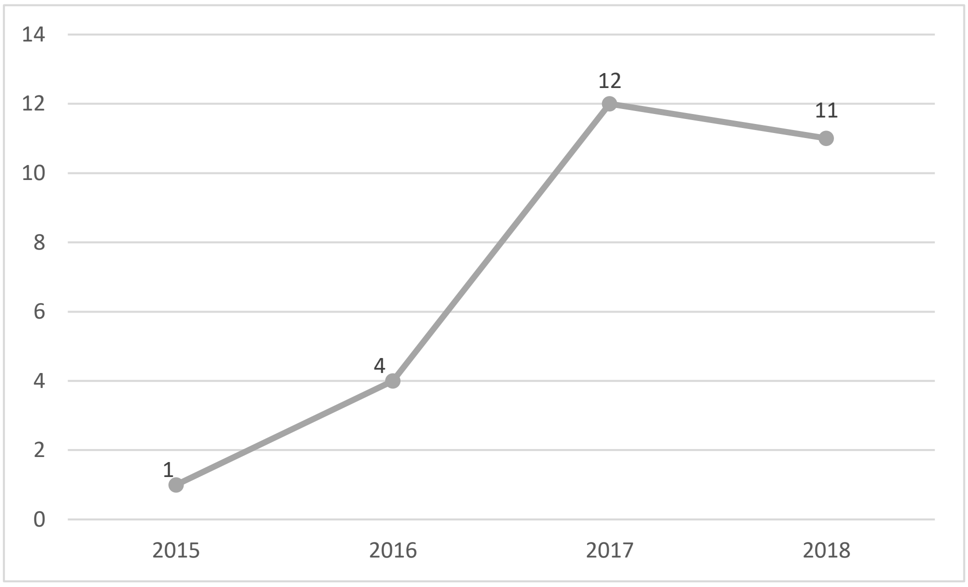

Figure 3.9: Distribution of Studies by Year

As can be seen in Figure 3.9 and Table 3.12, there is an upward trend in the number of studies in the field each year with the exception of a slight decrease in the last year.

RQ2: Who is active in this field?

Country	Studies
Germany	[KoDK18, RaSS18, RaSZ17, Sorg17, StOb17]
Italy	[Fran17, GLMM17, MMPP16, TaLP18, VCTG17]
UK	[Arel18, HaAB17, SeGM18]
Greece	[GFZV16, ThVS18]
Denmark	[GLMM17, SMMR16]
India	[SaSR17, SeGM18]
Russia	[GLMM17, SMMR16]
France	[BRBC16, LTOG18]
Australia	[KoSi17]
Brazil	[MGMR18]
Finland	[TaLP18]
Spain	[GuLJ18]
Kazakhstan	[KoSi17]
Sri Lanka	[InSi18]
Taiwan	[MFCL18]
Thailand	[Petr17]
Netherlands	[KWGJ17]
Switzerland	[MaCL17]
USA	[RaGa15]

Table 3.13: Distribution of Studies by Country

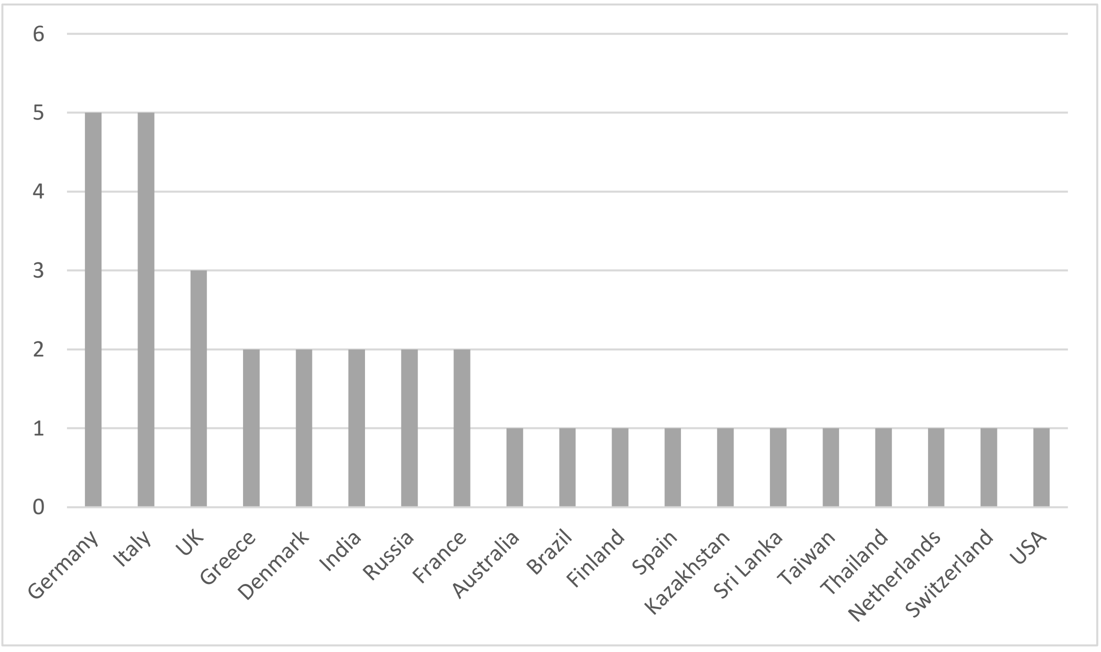

Figure 3.10: Distribution of Studies by Country

Table 3.13 gives an overview of the active countries. It can be observed that Germany and Italy are the most active countries in the field. The complete data synthesis form as defined in Table 3.6 can be found in Appendix C.

RQ3: What are the available approaches for microservices design?

RQ3.1: Which one is the focus of researchers?

RQ3.2: Is there a need for a specific language for developing microservices?

Available Approaches	Supporting Studies
Container Technologies	[GuLJ18, KoSi17, LTOG18, SeGM18, TaLP18]
Behavior-Driven Development	[RaGa15]
Language-based Approach	[BRBC16, InSi18, MMPP16]
Model-Driven Approach	[Fran17, RaSZ17, Sorg17]
Aspect-oriented Metamodeling	[HaAB17]

Table 3.14: Filled Form for Available Approaches for Modeling and Composition

Table 3.14 indicates the available approaches for modeling and composing microservices and gives an overview of the activity with regards to each approach.

RQ4: What tools and methods are available for orchestrating of microservices?

Orchestration Approach	Supporting Studies
Choreography	[MFCL18, RaSZ17]
Container Orchestration	[GuLJ18, LTOG18, KoDK18]
Service Orchestration	[BRBC16, MMPP16, SMMR16, GFZV16, SaSR17, InSi18]

Table 3.15: Filled Form for Available Orchestration Approaches

Table 3.15 shows the approaches used for orchestrating microservices.

With regards to the industry-driven tools, the distribution of activity in the development of such tools by country and by year is shown in Figure 3.11 and Figure 3.12 respectively.

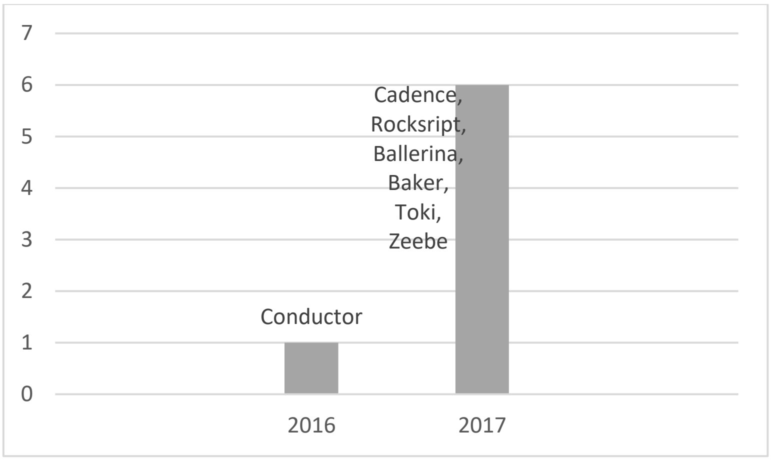

Figure 3.11: Distribution of Developed Tools by Year

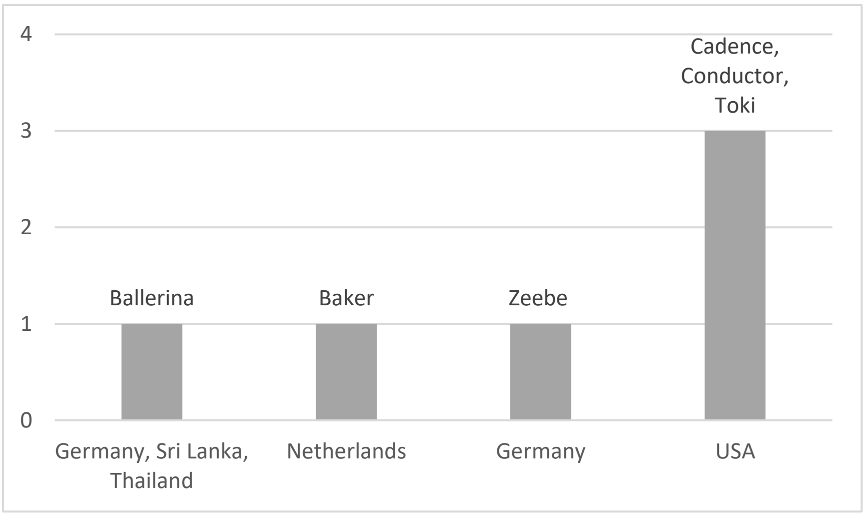

Figure 3.12: Distribution of Developed Tools by Country

3.3.2. Main Contributions

Table 3.16 gives a summary of the contributions of each paper related found in the SLR.

Study	Contribution
[BRBC16]	Introduces a service composition platform called Medley and demonstrates its performance.
[GLMM17]	Presents the need for specific programming languages aimed towards microservices development and introduces Jolie for such cases.
[KoSi17]	Advocates using containers for hosting and deployment of microservices.

Study	Contribution
[RaGa15]	Utilizes *Behavior Driven Development* (BDD) to apply automated acceptance testing to microservices.
[Sorg17]	Introduces a graphical modeling language for microservices by utilizing *Model Driven Development* (MDD).
[SeGM18]	It promotes the use of containers for microservices and introduces a tool for checking the availability of containers.
[Arel18]	It demonstrates the use of multi-level exogenous communication by performing an evaluation on microservices orchestration in IoT environments.
[Fran17]	Conducts a systematic mapping on architecting microservices. Utilizes MDD to introduce and implement MicroART which architecture recovery approach for microservices. It proposes API Gateway for the orchestration layer.
[TaLP18]	Discovers and analyses a number of widely implemented microservices architecture patterns. Presents the analysis and categorizes the architectural patterns and their proper use cases.
[LTOG18]	Utilizes containers as well as 12-factor principles for microservices development. Compares the performance of container technologies like Docker and Kubernetes and Docker Swarm.
[SMMR16]	Aims to add support for data-driven workflows to Jolie which will, in turn, shift the computation from process to data.
[VCTG17]	Introduces an orchestration broker, which receives a service manifest as input, and extracts information regarding the deployment of microservices in federated cloud environments. It also allows users to use geolocation constraints to determine the regions where the microservices are deployed.
[RaSZ17]	Considers implications of *Model-driven development* (MDD) for microservices development.
[StOb17]	It introduces a lightweight orchestration approach called Microflows. It uses agent-based clients and graph-based methods to extracts information from semantically-annotated microservices. It uses BPMN.
[HaAB17]	It extends aspect-oriented architecture for modeling microservices granularity. It introduces a modeling concept where microservices boundaries are the most important.
[KoDK18]	Introduces an orchestration framework which handles cross-ecosystem dependencies for cloud-based microservices.

Study	Contribution
[GFZV16]	Proposes a framework for policy management regarding distributed applications running on a programmable infrastructure. It advocates that distributed applications should be reconfigurable by design, hence increasing the usability in different contexts.
[GuLJ18]	Introduces an approach to optimize the orchestration of containers and reduce network latency. Provides a genetic approach which can make container orchestration more efficient.
[SaSR17]	Considers differences of SOAP and REST in microservices architecture. It proposes an implementation for dynamic services composition aiming to ameliorate QoS. It also evaluates the performance of two databases using RESTful and SOAP services.
[MFCL18]	Introduces an approach, called GMAT, which improves analyzability of microservices dependency by visualizing dependencies between microservices which ultimately leads to a reduction in risky service invocations.
[KWGJ17]	Introduces a tool, called MicADO, aiming to improve the efficiency of microservices architecture. It leverages a genetic algorithm to recommend an optimized deployment model based on workload.
[MGMR18]	Introduction and implementation of an event-driven orchestration platform for microservices which uses a DSL for specifying orchestration.
[RaSS18]	Utilizes model-driven development to address issues of DDD and microservices. Performs analysis on how to derive microservices and infrastructure from domain model and proposes MDD tools for such use cases.
[ThVS18]	Proposes a framework based on *Model-Driven Engineering* (MDE) to incorporate microservices architecture in cyber-physical manufacturing.
[MaCL17]	Introduces a graph-based model which can break down a monolith to microservices.
[InSi18]	Analyzes difficulties linked with integration and orchestration of microservices. Introduces Ballerina to cope with these challenges.
[Petr17]	Introduces a static model to enable model-to-model transformation and code generation for microservices by utilizing UML. Formalizes microservices modeling and their communications.
[MMPP16]	It utilizes Jolie as an orchestration platform for *Mobility as a Service* (Maas).

Table 3.16: Contributions of Each Study

4. Available Composition and Orchestration Tools

In this chapter, the tools discovered in the SLR chapter are briefly introduced. The aim is to extract some information on the tools to prepare for the upcoming evaluation. In order to be included in the evaluation, the tools must have an open-source implementation to permit investigating different aspects. Therefore the tools which do not comply with this requirement, like Medley [BRBC16] and AWS Step Functions [Awss00], are excluded from this evaluation. Also, the orchestration tools for container technologies are excluded, as this is not the focus of this work.

The tools discussed in this chapter are suitable for microservices composition, modeling, and orchestration. Ballerina and Jolie promote the idea of using a new programming language for applications based on microservices architecture [InSi18, MoGZ14]. The need for agile methods for modern integration scenarios renders languages and platforms like Java, Go and BPEL and BPMN not desirable [WEPL18]. The reason for this stems from the fact that these languages lack native constructs for workflows which brings the burden of dealing with middleware systems such as workflow management systems, enterprise service bus and API gateways [DGLM16, WEPL18]. This adds to the complexity and makes the solution less agile especially in modern cloud applications where agility in deployment is a key. Jolie and Ballerina natively support necessary constructs for microservices-based applications. Also, the runtimes of Ballerina and Jolie support persisting the state of running workflows which enables executing workflows in a resilient manner [Joli07, WEPL18].

Tools like Cadence, Conductor, and Zeebe aim to address issues regarding orchestration aspect for microservices [Cond16, Uber17, Zeeb17]. Such tools enable execution of end-to-end business processes which require the execution of multiple microservices to be complete. Most of these tools have features which enable scalable and resilient execution of such workflows.

4.1. Jolie

Jolie is a fully-fledged programming language designed for (micro)service-oriented architecture which enables the user to translate the design into code without changing the domain model [DGLM16, MoGZ14]. In a Jolie program, the implementation of an application is separated from how the communications are established [GLMM17]. The implementation of the application is contained in the "behavior part" of a Jolie program [MoGZ14], and the required information for communication is specified in the "deployment part" [GLMM17]. This is similar to the idea behind WS-BPEL. However, most of the implementations WS-BPEL are not optimal for microservices architecture [GLMM17]. Similar concurrent programming languages, like Erlang[7] and Go[8], do not efficiently support the separation of behavioral and architectural aspect [GLMM17].

[7] www.erlang.org
[8] www.golang.org

In the microservices environment, the way each service interacts with other services is crucial. In order to specify how communication has to be established, communication ports are introduced where interface, communication technology (e.g., TCP/IP), as well as data protocol (e.g., HTTP), are specified [DGLM16]. Hence, communication ports act as a compatibility layer for the deployment and behavior aspect. The behavior part works on the assumption that the deployment part is appropriately defined. Therefore they refer to communication ports in an abstract manner. The communication is either one-way or request-response. Interfaces and communication ports increase separation of concerns in deployment and behavior allowing the user to specify how microservices interact without changing the implementation [DGLM16].

Jolie has architectural primitives which are suitable for MSA. For example, the embedding construct can create a hierarchical order in which a service can be embedded in another service. The service and sub-service interact via communication ports, meaning the embedded service can be changed with another type, e.g., external service, by only changing the deployment part of the application. The native support for sequential as well as parallel constructs facilitates programming fork-join patterns. The workflow construct facilitates the orchestration of multiple services which are already defined. The private state of the instance is isolated which eliminates the occurrence of race conditions. Given the fact that the data transfer in MSA is commonly done in the tree structure such as JSON[9] or XML[10], Jolie organizes the state of variables as a data tree, which will facilitate marshaling [GLMM17].

4.2. Ballerina

Ballerina is a fully-fledged, general-purpose programming language which gathers the fundamentals of distributed programming [InSi18]. It has support for textual and graphical syntax and has adapted UML sequence diagrams [WEPL18], which enables different stakeholders with different technical levels to collaborate more efficiently. There are optimization features in Ballerina which is a result of having complete control over the program and its execution [WEPL18]. This is a clear advantage because in general-purpose programming languages and middleware systems such possibilities are not available [WEPL18].

Ballerina supports interruptibility, recoverability, transactions, and parallelism which is indispensable for integration scenarios [WEPL18]. Some features of Ballerina can help developers to cope with the non-deterministic nature of distributed environments. [InSi18, WEPL18]. By separating network calls from function calls, implications like latency or security risks can be detected, which from a developer's point of view are useful. Circuit breaking enables the developer to configure connectors to stop sending messages to unresponsive endpoints. The connectors also can interact with other entities like HTTP endpoints. Each endpoint can follow the polling pattern as well as only being exposed via a specific port and waiting for requests [Ball17]. Ballerina also has constructs for control flow and data flow which are necessary for parallel execution or fork-join [WEPL18]. The concept

[9] www.json.org
[10] www.w3schools.com/xml

of workers in Ballerina refers to the concurrent blocks of code. Essentially each function in Ballerina is either a worker or network endpoint with which workers communicate [Ball17].

Moreover, the developer must either handle the possible error which can be caused by calling a function or network call or explicitly mention that the error is not crucial to be handled. This means that in Ballerina, the execution flow can run without failures, because, in case there are errors in lower levels, upper levels can handle them. Compared to general-purpose programming languages such as Java and C#, this is a much more optimal manner of handling errors, given the high cost of processing exceptions and undoing the logic in general-purpose programming language [Ball17]. Ballerina also can improve concurrency by finding the minimal shared scope and locks data structures at that level. This enables parallel programs to deal with issues related to shared data handling [WEPL18].

Ballerina Virtual Machine enables persisting the state of a program. This concept is also utilized by workflow management systems which allow for interruptibility and recovery [WEPL18]. This is very useful when having long-running workflows, where in case of a failure of a task during the workflow, repeating the whole workflow is not desired. Additionally, long-running tasks in workflows make distributed ACID transactions impractical. Using the concepts introduced by BPMN and BPEL, Ballerina supports compensation-based interactions which avoid blocking. Each task has a compensation action attached to it, and in case of failure, the instructions inside the compensation action take place.

Furthermore, a Ballerina program can receive intermediate inputs from external programs by utilizing the concept of correlation variables which have been used by workflow management systems [WEPL18]. This is helpful in scenarios where a program instance needs additional input after starting. This brings flexibility regarding communication with external systems given a Ballerina program can receive messages from any sources and trigger another Ballerina program which needs intermediate inputs.

4.3. Cadence

Cadence is an orchestration engine developed by Uber Engineering. It can perform long-running tasks in a distributed and scalable manner [Uber17].

Cadence separates orchestration and implementation logic through the concept of activities and workflows. Each activity has an interface and can have a synchronous or asynchronous implementation. The interfaces must return serializable value and implementations must be thread-safe. The workflow determines the order of execution of the activities. Workflows are fault tolerant meaning that the errors in activities do not interrupt the execution of the workflow. In case of occurrence of errors in the activities, the workflow is informed and decides how to handle the error. The activities which need to be performed are put into a list, and the workers will handle their execution. Each workflow execution will be assigned a unique workflow ID which can be used for querying purposes [Uber17].

Workers poll the Cadence server for tasks. By using stateless workers, the user can benefit from horizontal scalability features of Cadence. Persisting the state of the workflow is done using event-sourcing, meaning all the events are persisted in an append-only manner and

Cassandra[11] is used as a database. Communications are established with TChannel[12], which is an RPC-based communication protocol developed by Uber Engineering [Netw18, Uber17].

4.4. Zeebe

Zeebe is a horizontally scalable and fault tolerant workflow engine developed by Camunda [Zeeb17]. The use of BPMN 2.0 notions to specify orchestration logic in Zeebe simplifies collaboration within the teams in organizations. Zeebe also supports using structured texts like YAML[13] to specify the orchestration logic; however, under the hood, it converts to BPMN [Zeeb17].

Contrary to Cadence, Zeebe uses the file system for persisting the state of workflows and other logging data. It uses event sourcing to store the history of a workflow. As a result of having a configurable replication mechanism, Zeebe guarantees progress in the workflow and offers flexible error handling [Zeeb17]. The visibility of the state of the workflow offered by Zeebe is also an advantageous feature because the user can have knowledge of the current state and, for example, see if there was a failure in a specific part [Zeeb17].

The workflow-related tasks are managed on the server side, and the actual performing of tasks or business logic takes place on the client side. Servers can be horizontally scaled by adding more servers, thereby forming a cluster, this will allow replication which increases fault tolerance. In a cluster, Zeebe uses Gossip protocol [Jela11] in order to identify nodes connected via a peer-to-peer protocol. For replication, Raft14 protocol is applied which marks one broker as a leader and the rest of the nodes as followers. If a node becomes unavailable, follower elects a new leader in case the current leader fails [Zeeb17].

The job workers are invoked to perform tasks as soon as the workflow execution reaches the respective task. The job worker polls the server and picks the type of the job it can handle. Whenever there is no worker to perform a specific job, then a queueing mechanism is used [Zeeb17].

Zeebe applies at-least-once semantics to perform the processing of the jobs. After connecting the workers and submitting the workflow, the workflow can be performed. Different types of workflows such as sequences, fork-join are supported. The payload of a workflow can be read and modified during the execution of a workflow by workers. Zeebe uses JSONPath to determine the tasks to be executed based on the payload [Zeeb17]. Zeebe uses command and subscription protocol which are non-blocking TCP/IP based protocol and enable the interaction between clients and server. In command protocol, client initiates an interaction by sending a command and waiting for a response. Subscription protocol uses a backpressure mechanism where the clients indicate how much work they can handle so that the server does not over utilize them when data is streamed from the server to clients [Zeeb17].

[11] www.cassandra.apache.org
[12] www.tchannel.readthedocs.io
[13] www.yaml.org
[14] www.raft.github.io

The historical data on the execution of workflows and jobs can be extracted to an external data warehouse, or it can be used with a visualization tool like zeebe-simple monitor. Zeebe deletes historical data when they are not needed. Therefore, in case it is necessary to access the historical data and perform analysis, the data must be exported. To have a better performance, Zeebe batches I/O operations and employs linear memory access [Zeeb17].

4.5. Conductor

Conductor is an orchestration engine developed by Netflix [Cond16]. It provides interruptibility, visibility, and traceability. Horizontal scalability can be achieved easily as the servers are stateless. It abstracts the queuing service from clients and is able to work with HTTP and other transport protocols like gRPC[15]. The user has the possibility of using different technologies in API and storage layer. Additionally, the storage and queueing components can be scaled independently. The communication between client and server is established using an HTTP load balancer or discovery pattern [Cond16].

Conductor has an API layer, a service layer, and a storage layer. The API layer has three components where the workflow management and task definition and execution takes place. Conductor applies the concept of worker and server where the two run on different machines. Workers poll the server for tasks. Dyno-queues[16] implements a queuing mechanism where tasks are scheduled for workers. In the storage layer, Dynomite[17] and Elasticsearch[18] are used for persisting the state and metadata and as well as indexing workflow and task executions [Cond16].

Conductor uses a domain-specific language in the form of JSON for workflow definition. The workflow consists of tasks that are related to control flow, e.g., fork-join. Also, tasks which are related to the application, like sending an email, are executed by the workflows on client machines. Several workflows can re-use both system and worker tasks.

The concept of system task in conductor refers to the tasks which are related to the scalability and execution aspect of Conductor server. The worker tasks contain the logic of the application. Workers are responsible for polling the server and performing the tasks and set the execution status. Before a workflow can carry out the tasks which it orchestrates, all tasks must be registered [Cond16].

4.6. Baker

Baker is a library for microservices orchestration based on Petri Net [Bake17, Orch18]. Its runtime is based on Scala[19] and Akka[20]. Baker is suitable for scenarios where similar functionalities in the system exist as it promotes the re-use of these existing functionalities.

[15] www.grpc.io
[16] www.github.com/Netflix/dyno-queues
[17] www.github.com/Netflix/dynomite
[18] www.elastic.co
[19] www.scala-lang.org
[20] www.akka.io

In Baker, orchestration logic consists of system calls, data, and events. It uses a DSL to specify orchestration logic. It allows for the visualization of the orchestration logic, which provides different stakeholder with different technical levels with efficient communication [Bake17].

A Baker instance can be instantiated by creating the orchestration logic and implementing the system calls which are ideally idempotent. Afterward, the orchestration logic is converted to a Petri net, where the order of system calls is determined. The state is persisted using Cassandra with event-sourcing which enables automatic state recovery. Baker supports parallel execution of instructions and performs automatic retries in case of failure [Orch18].

4.7. Toki

Toki is an orchestration platform written in JavaScript [Toki17]. Toki follows a request-response paradigm. The state of the workflow is held in memory and is not persisted on disk, meaning it is not resilient [Toki17].

Toki uses a JSON configuration to specify the orchestration logic which will be validated using joi schema [Obje18]. It can execute a chain of actions sequentially or in parallel. The payload of each action is exposed to its successor. The failure logic is separated, where some instruction is specified and in case of failure in the flow, they will execute [Toki17].

4.8. RockScript

RockScirpt is an orchestration and integration tool for microservices based on JavaScript [Rock17]. The orchestration logic can be specified in a JavaScript file which can make calls to REST APIs and return responses to the server.

Similar to other orchestration engines explained above, RockScript uses event-sourcing for persisting the state of execution. Scalability can be improved with clustering. Performing long-running tasks is enabled as a result of storing the state of execution. This reduces memory consumption when performing long-running tasks. The business logic is separated from the way it is operated as a result of using activities. In case of failure, retries can be performed. Rockscript has HTTP services which carry out HTTP requests, and it facilitates calling other services which are exposed over HTTP. The HTTP service is fed with an HTTP endpoint, and then it performs a POST request to the endpoint. When the request is done, the function being called can reply in a synchronous or asynchronous manner. RockScript provides an HTTP API which has RPC commands and supports queries. The commands can be performed using POST verb and queries with GET verb. Rockscript also allows the user to inspect execution result by inspecting the server [Rock17].

4.9. Beethoven

Beethoven is an orchestration engine for microservices based on Spring Cloud Netflix[21] and Akka [MGMR18]. It has a DSL based on Xtext[22] where the orchestration logic is defined. Beethoven uses service discovery pattern to address the issues with respect to the dynamic location of microservices [MGMR18].

Beethoven has several layers each of which has different responsibilities. It has a REST API which exposes a given functionality, one layer where the functionality is implemented, another layer which persists data related to workflows and tasks, and a core layer which executes workflows [MGMR18]. A microservice must be created using Spring Cloud Netflix so that it can be orchestrated with Beethoven. This means a microservice must implement service discovery pattern. Beethoven is based on Actor Model and allows the parallel execution of several workflows [MGMR18].

[21] www.spring.io/projects/spring-cloud-netflix
[22] www.eclipse.org/Xtext

5. Evaluation

In the previous chapter, some of the attributes of the discovered tools were presented. Most of these tools enable executing workflows in a distributed manner and comply with the features discussed in [LeRo00].

In this chapter, the tools presented in the previous section are evaluated. The goal of this evaluation is to draw a comparison among the implementations to ultimately identify how a microservice must be specified with these tools. We aimed to find the common points of all the orchestration engines to achieve an abstraction over these tools. Each microservice must fulfill certain specifications to be used within a given orchestration engine. Table 5.1 shows the categories of these specifications which will be used as criteria for this evaluation.

Criteria	
API type	Architectural Style
	Communication protocol
	Data format
Message Exchanging Pattern	
Client SDK	

Table 5.1: Evaluation Criteria

5.1. Evaluation Based on Criteria

In this section, an evaluation of the tools is presented. The evaluation is based on the criteria introduced in Table 5.1 and the result is tabulated using the format presented in Table 5.2.

Criteria	Platform

Table 5.2: Evaluation Form

5.1.1. API

We consider the communication protocol, API style, and data format collectively because the result of individual criterion is not conclusive. Table 5.3 shows the API type required by each of the orchestration engines.

API			Platform
Architectural Style	Communication Protocol	Data format	
REST	HTTP	JSON	Conductor, Toki, Rockscript, Ballerina, Jolie, Beethoven
RPC	HTTP/2	Protocol Buffers (+JSON)	Conductor, Zeebe, Ballerina
RPC	HTTP	JSON, XML	Baker
RPC	WebSockets	JSON, XML	Ballerina
RPC	TCP Sockets	JSON, XML	Jolie
RPC	Tchannel	Thrift (+JSON)	Cadence

Table 5.3: API type

The architectural styles used by the tools are REST and RPC. The tools which employ REST communicate over HTTP using JSON as the wire format and exchange messages using request-response pattern. The RPC-based communication using gRPC is performed over HTTP/2 with protocol buffers[23] as wire format. JSON can also be used given it can be converted to protocol buffers. Cadence employs RPC style communication over Tchannel using Thrift[24] as wire format. TChannel is an RPC protocol developed by Uber to establish communication between client and server. In Ballerina and Jolie, RPC communications using SOAP are performed over WebSockets and TCP sockets respectively. Jolie additionally supports SODEP over TCP sockets. SODEP is a protocol specific to Jolie developed for efficient communications [Joli07].

5.1.2. Message Exchanging Pattern

Each orchestration engine requires a microservice to implement a specific message exchanging pattern so that it can be used within that platform. The message exchanging patterns used by the platforms are:

- Polling
- Publish-Subscribe
- Request-Response

[23] www.github.com/protocolbuffers/protobuf
[24] www.thrift.apache.org

Table 5.4 shows the messing pattern required by each of these platforms. Polling pattern means a consumer of gets connected to a producer, in our case an orchestration engine, and repeatedly asks for messages [HoWo03]. The publish-subscribe pattern is also about having the producer and the consumer of messages agree on a specific channel. In this pattern, the consumer subscribes to a specific topic, and when there is a message on that specific topic, it gets notified to perform some action [HoWo03]. Moreover, in the request-response pattern, the sender sends a request and waits for a response [HoWo03].

Conductor and Zeebe use publish-subscribe internally [Cond16, Zeeb17]. However, from the client application perspective, they apply polling pattern to implement workers. That means, the workers connect to the server and poll the server to get tasks in configurable polling periods. Then they perform the task and send back the response to the server. Conductor supports request-response in addition to polling. The workers are stateless and do not take error handling into account. This means that the possible errors are supposed to be handled in the workflow.

Ballerina and Jolie program can communicate via publish-subscribe or polling or request-response [Joli07, WEPL18]. Other platforms, like Toki and Baker and Beethoven, follow a request-response pattern [MGMR18, Toki17].

Message Exchanging Pattern	**Platform**
Request-Response	Conductor, Toki, Baker, Rockscript, Ballerina, Jolie, Beethoven
Polling	Conductor, Cadence, Zeebe, Ballerina, Jolie
Publish-Subscribe	Ballerina, Jolie

Table 5.4: Message Exchanging Pattern

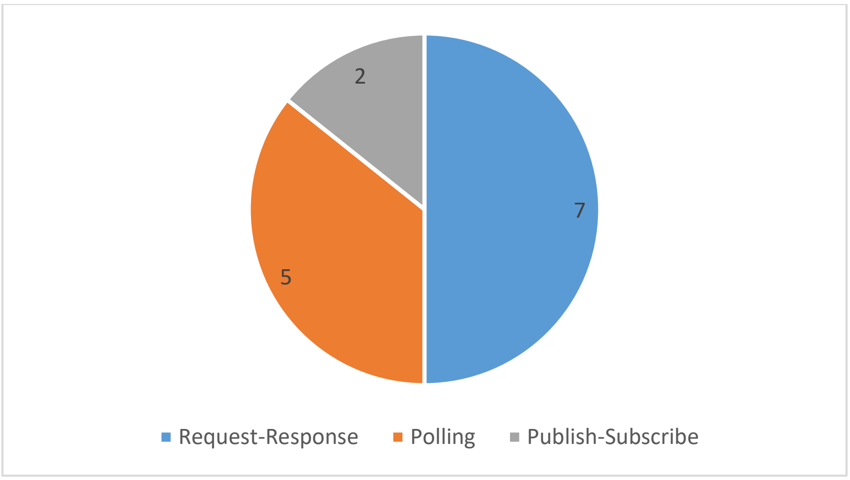

Figure 5.1: Message Exchanging Pattern

5.1.3. Client Environment

In order to establish the connection between the client and server, some of the orchestration engines require the microservice to include a client code. This inherently restricts the language with which a microservice can be developed given the piece of code which must be included in the microservice. Cadence provides client SDKs in Java and Go. Zeebe and Conductor also have out-of-the-box clients in Java, Python and Go. Nonetheless, the use of gRPC in Zeebe and Conductor enables generating client code in the programming languages supported by gRPC. Table 5.5 shows the platforms which require the mentioned client code and the languages in which the code is provided.

Client SDK	**Platform**
Java	Conductor, Cadence, Zeebe, Baker
Go	Cadence, Zeebe,
Python	Conductor
Scala	Baker

Table 5.5: Client SDK

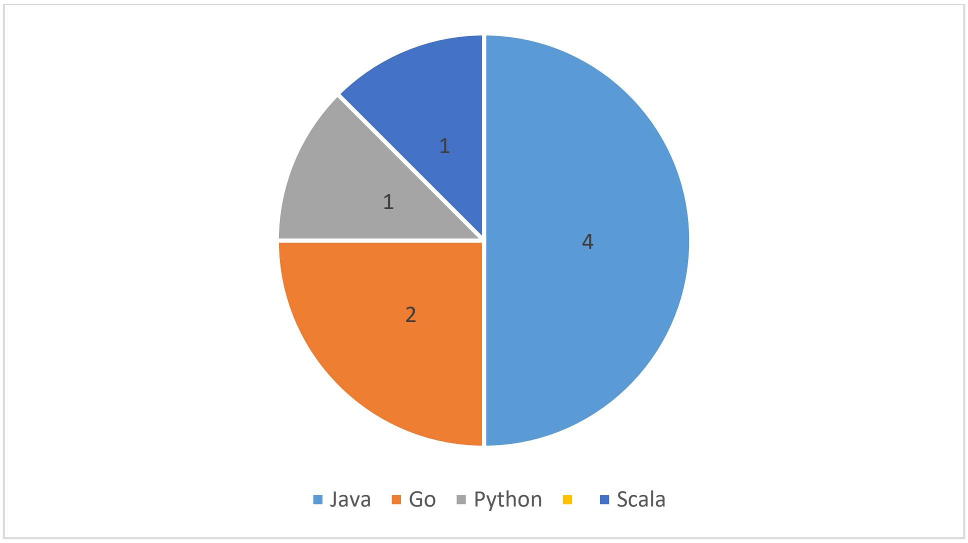

Figure 5.2: Client SDK

5.2. Result

Figure 5.3 illustrates the mentioned specifications. A microservice can communicate with an orchestration engine using the mentioned API types. The API should follow the mentioned communication protocol and data format and message exchanging pattern.

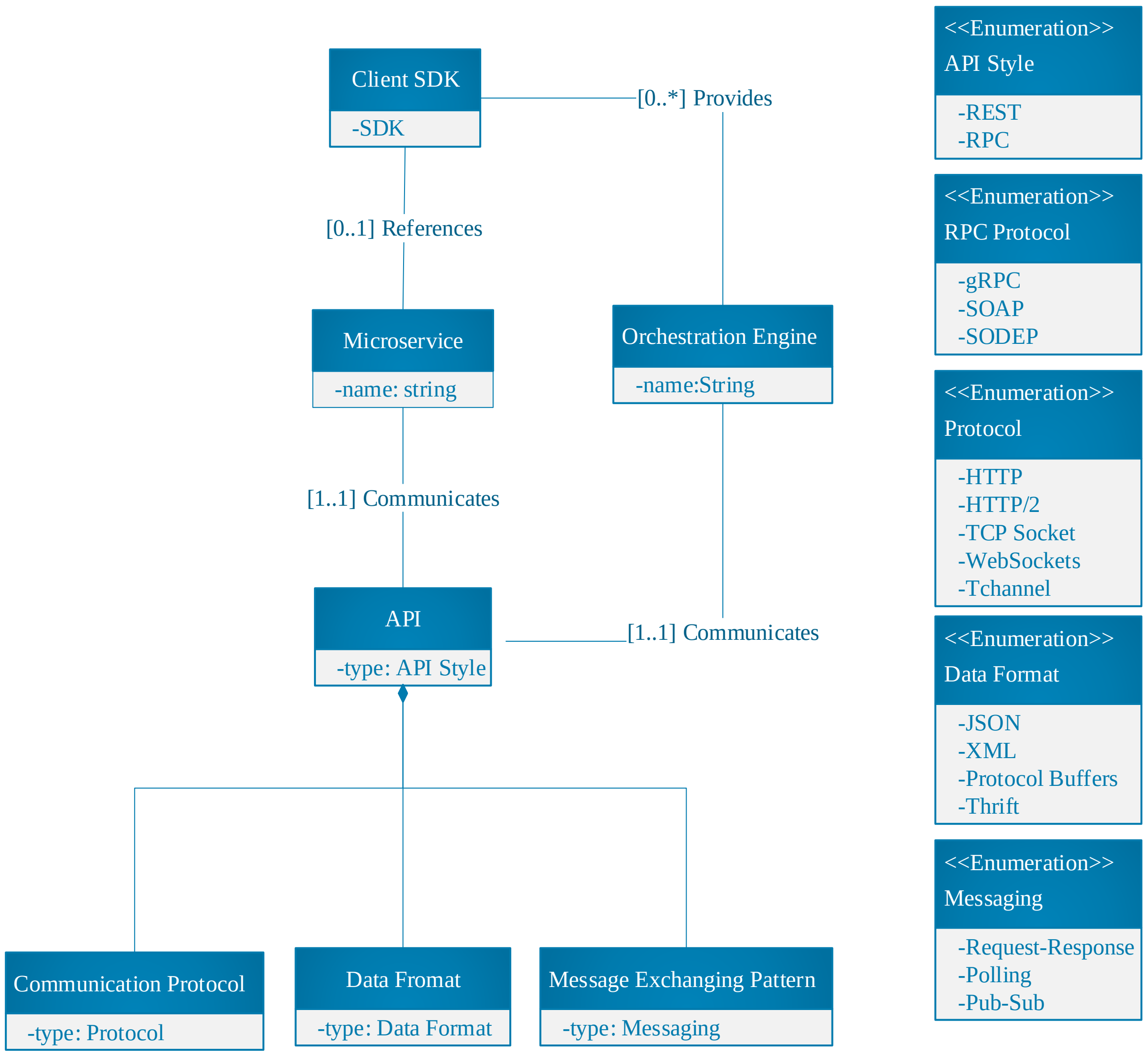

Figure 5.3: Specifications of Microservices with the Orchestration Tools

In case the communication is performed using HTTP and REST, the microservice can be implemented in any language. Exceptionally, Beethoven relies on Spring Framework and requires the microservice to be implemented using Spring Framework. Also, Zeebe, Cadence, and Conductor which communicate using an RPC-based protocol require a piece of code to be included in the client application. Zeebe and Conductor employ gRPC protocol over HTTP/2 and Cadence uses RPC over TChannel.

In the next chapter, a method is proposed to abstract the different implementation specifications for each platform.

6. Prototype Implementation

As can be seen in the previous chapter, each of these orchestration engines has some requirements which must be satisfied when applying to a microservice-based application. This is with regards to the type of the communication protocol, the message exchanging pattern, and the data format. In this chapter, an effort is made to implement a prototype which creates an abstraction of these specifications to enable a microservice to be used with these tools without needing to adapt to the requirement of each tool. The source code can be found at this address[25].

6.1. Approach

In this section, the similarities and common points of the orchestration tools have been analyzed to discover how the specifications of these tools can be abstracted. As discussed in chapter 4, it can be observed that the orchestration engine utilize the concept of workflow and workers to separate the orchestration logic from implementation logic. Therefore, it is necessary to establish a common definition of their properties.

A workflow is a blueprint containing a set of tasks and their order whose collective execution accomplishes a business goal [LeRo00]. Some of the presented orchestration engines, like Conductor, require the workflow to be defined in a JSON file, whereas some of them like Zeebe provide a visual solution using BPMN in addition to the textual form of JSON and YAML. Persisting the state of flows allows for having long-running tasks, as the workflow can release resources when a task requires a considerable amount of time to complete. Then as soon as the task is completed, the workflow can resume without needing to start from the beginning. This is also valid for failures in tasks. A failure in a task is caught in workflow and handled respectively. Most of the orchestration engines mentioned have been made based on this principle.

The worker is where the implementation of the tasks should be defined. In tools like Cadence, Conductor, and Zeebe they communicate with the server by polling to receive the tasks they can execute [Cond16, Uber17, Zeeb17]. They can receive the data they require from the server and return the payload to the server after completing the execution. Generally, these workers do not deal with error handling, and in case of failure, they report the error to the server, where it will be managed.

The results of the evaluation chapter have been utilized to propose a reference architecture which can be used with the mentioned orchestration engines. To enable this architecture to work with the orchestration engines which require other specifications, like Cadence and Zeebe, a prototype has been in introduced in section 6.2. The prototype communicates with microservices over HTTP and uses JSON data format. Request-Response is used for the message exchanging pattern.

[25] www.github.com/pedramha/wfms_abstractionV1

As mentioned earlier, platforms like Zeebe and Cadence, require the workers to perform the tasks specified in the orchestration logic. A generic worker will allow services to be used with these engines. This method eliminates the need of changing the microservice to the specific requirements of the orchestration engine, e.g., including the client code in the microservice, as well as other specifications. The proposed prototype is implemented as a generic worker which abstracts the communication aspect with the server as well as messaging protocol and data format. It acts as an abstraction layer between server and microservices.

Figure 6.1 shows how the interactions between the orchestration engines, the prototype, and the microservices are established. Initially, the generic worker registers itself with the orchestration engine and then it receives tasks to perform. It delegates the actual performing of tasks to the microservices defined externally. After receiving the response from the microservice, it returns the payload to the orchestration engine.

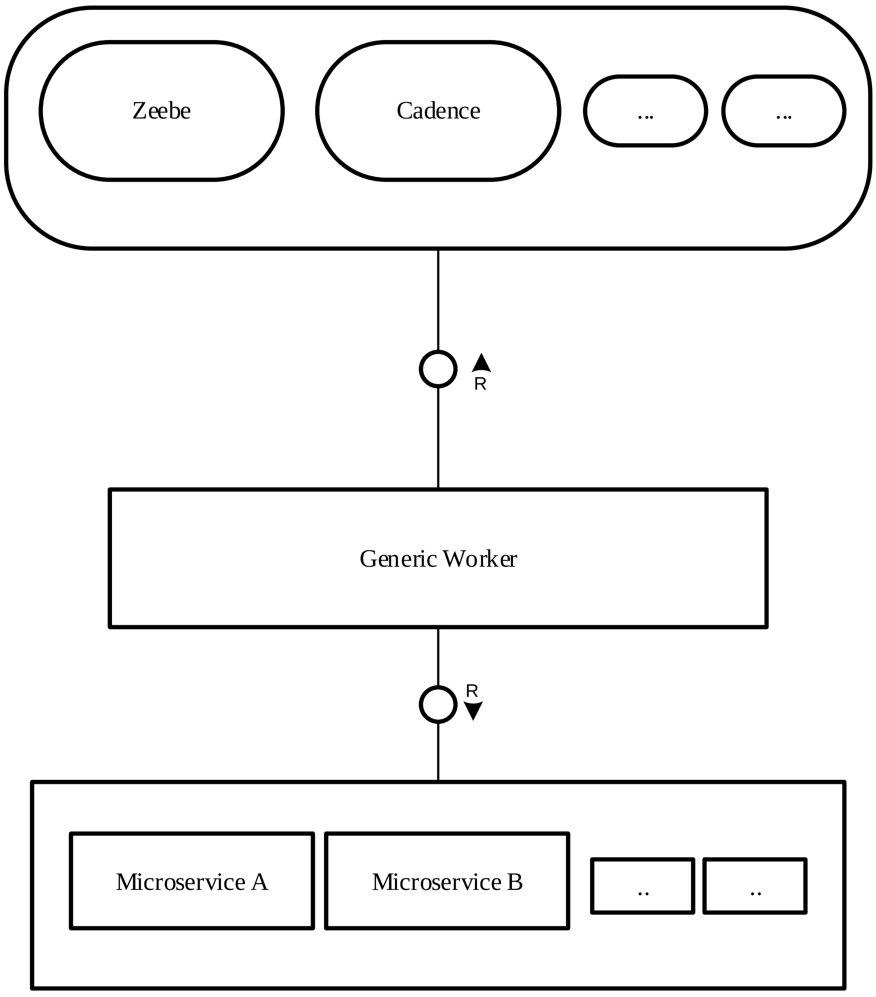

Figure 6.1: Prototype Model

6.2. Implementation

The proposed prototype aims to abstract how the workers communicate with the server and does not consider the orchestration aspect. The orchestration logic is to be submitted as required by the tool.

The prototype uses the Java SDKs provided by the orchestration engines to start a worker which gets connected to the server and polls for tasks. It requires the IP and the port on which the orchestration engine is running for establishing the connection. The current prototype performs the abstraction for a subset of orchestration engines, namely Zeebe, Cadence. However, it can be easily extended for other platforms. Netflix Conductor has support for HTTP workers, which is very similar to the proposed concept in this chapter.

When the prototype starts, it gets connected to the server using the given IP and starts polling for tasks. After deploying a workflow to the server and starting a workflow instance, the server sends a task of HTTP type to the prototype to be handled. The prototype receives the payload of the workflow, the URL of the service to be called as well as the HTTP verb to make the call. Then the worker attempts to make a call to the service indicated in the URL. If the attempt is successful, then the serialized response in the form of JSON is sent back to the workflow to proceed with the appropriate action. If the attempt was unsuccessful, by default the prototype makes two more attempts, and in case the call still does not succeed, an error is reported to a workflow which must be handled there. Also, if the call lasts more than a specific duration, 10 seconds by default, an error is reported to the workflow. Figure 6.2 shows how successfully a workflow can be executed using the proposed prototype.

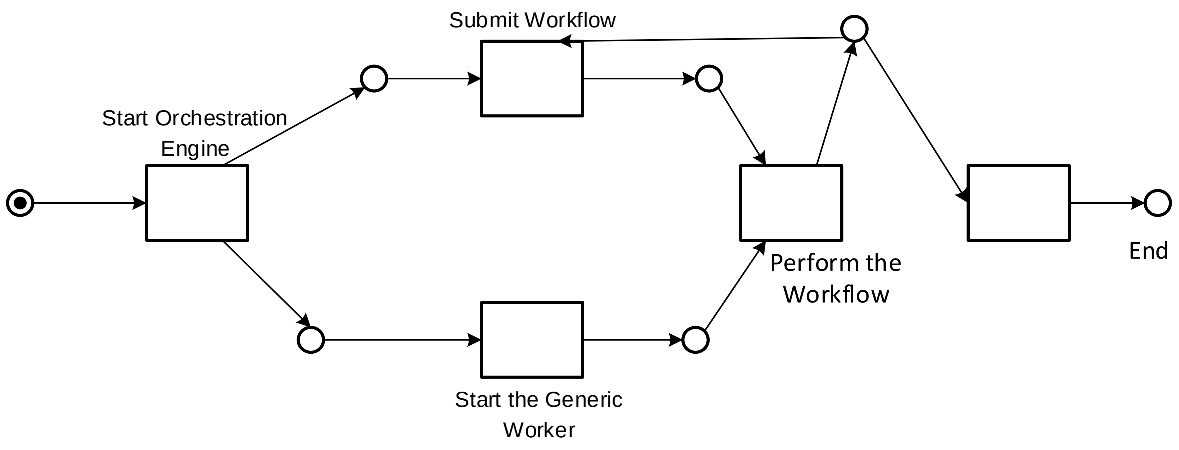

Figure 6.2: The Process of Performing a Workflow Using the Proposed Prototype

The prototype enables using microservices with the introduced orchestration engines. It removes the requirement of the orchestration engine to include the client SDK and thereby enables writing microservices in any language as long as they are accessed over HTTP with JSON.

In case the prototype runs on a different (virtual) machine as the orchestration engine, then there would be the performance would be reduced as a result of having one network call to our tool and another one to the defined microservice.

7. Conclusion and Future Work

In this chapter, a summary of this work is presented, and the improvement possibilities to the implemented prototype and the possibility of future research are discussed.

Microservices-based applications benefit from the advantages that the architecture has to offer. Nonetheless, the complexity of applications increases as end-to-end business processes require the execution of multiple microservice. As the microservice-based application grows so does its complexity and direct communication between microservices couples them together and thereby reduces scalability. To address this issue, orchestration tools such as Cadence and Conductor have been developed. In addition, composition languages such as Jolie and Ballerina have also emerged. These tools are developed based on different models and have different specifications.

In this work, we aimed to discover suitable tools for modeling and composition of microservices and find a glue which fills the existing gap between them. In order to do this, we first conducted a systematic literature review to discover and examine the suitable tools and approaches for modeling and orchestrating microservices in chapter 3. Then, in chapter 4, the tools discovered in the SLR chapter were briefly presented, and some key specifications of these tools were discussed. Also, some information was extracted to prepare for evaluating the discovered tools. In chapter 5, an evaluation was performed on the tools in order to determine how microservices can be used within the mentioned tools. The criteria defined for the evaluation were based on the specifications these tools require from microservices. In chapter 6, the results of the evaluation were used to propose a reference architecture and conceptualize and implement a prototype for creating an abstraction for the mentioned orchestration tools. This prototype aims to abstract different aspects such as message exchanging pattern and communication protocol which are specific to a given tool. Therefore, the similar approaches for each criterion were analyzed to create this abstraction layer. This prototype enables using the mentioned orchestration tools in a microservices-based application without having the need to adapt the microservices to the requirements of a specific orchestration tool.

With regards to future work, in this thesis, we did not consider container technologies and their orchestration. So this work can be further extended by including these technologies and also drawing a comparison between container orchestration such as Kubernetes[26] and the presented microservices orchestration engines in this thesis. The result of the evaluation of the container orchestration technologies can help to enrich the current prototype to create an abstraction for both container orchestration and the microservices orchestration engines mentioned in this thesis. Furthermore, authentication and authorization concerns have also not been taken into account. The prototype can be enhanced by taking these security concerns into account. Additionally, the specification of the orchestration logic was not considered in the prototype.

[26] https://kubernetes.io

Each orchestration engine uses a different approach to specify the orchestration logic, and therefore an abstraction with this regard can be added to the current prototype.

Moreover, each of the tools has many internal differences. For instance, some of the tools make use of queues and some only databases. This can be investigated in depth to evaluate the efficiency of the underlying technologies and improve the performance. Also, the orchestration engines which gather the data related to the execution, offer different possibilities. For instance, some allow the extraction of this data which enables analysis and offers potential improvements. Using this type of data, some of the orchestration engines enable monitoring the workflow execution. Each tool possibly uses a different way of visualizing and monitoring workflows which can be standardized.

Appendices

Appendix A

Data Item	Value	Additional notes
Reference	[GLMM17]	
Study	Microservices: A Language-Based Approach	
Year	2017	
Author	Claudio Guidi, Ivan Lanese, Manuel Mazzara, and Fabrizio Montesi	
Institution	italianaSoftware, Focus Team, University of Bologna/INRIA, Innopolis University, University of Southern Denmark	
Country	Italy, Russia, Denmark	
Modeling and Composition approach	Jolie	
Orchestration approach	–	
Main Contribution	Discusses the need for specific programming languages aimed towards microservices development and introduces Jolie for such cases.	
Source	Google Scholar	

Table A.1. Filled Data Extraction 1

Data Item	Value	Additional notes
Reference	[MGMR18]	
Study	Beethoven: An Event-Driven Lightweight Platform for Microservice Orchestration	
Year	2018	
Author	Davi Monteiro, Rômulo Gadelha, Paulo Henrique M. Maia, Lincoln S. Rocha, and Nabor C. Mendonça	
Institution	State University of Ceará, Federal University of Ceará, University of Fortaleza	
Country	Brazil	
Modeling and Composition approach	–	
Orchestration approach	Event-driven service orchestration	Service discovery is used
Main Contribution	Introduction and implementation of an event-driven orchestration platform for microservices which uses a DSL for specifying orchestration.	
Source	Google Scholar	

Table A.2. Filled Data Extraction 2

Data Item	Value	Additional notes
Reference	[MMPP16]	
Study	A Microservice-Based Architecture for the Development of Accessible, Crowdsensing-Based Mobility Platforms	
Year	2016	
Author	Andrea Melis, Silvia Mirri, Catia Prandi, Marco Prandini, Paola Salomoni	
Institution	Department of Electrical and Information Engineering, Department of Computer Science and Engineering-University of Bologna	
Country	Italy	
Modeling and Composition approach	–	
Orchestration approach	Data Driven Workflow	Uses Jolie
Main Contribution	It utilizes Jolie as an orchestration platform for *Mobility as a Service* (Maas).	
Source	IEEE	

Table A.3. Filled Data Extraction 3

Data Item	Value	Additional notes
Reference	[BRBC16]	
Study	Medley: An event-driven lightweight platform for service composition	
Year	2016	
Author	Elyas Ben Hadj Yahia , Laurent Reveillere1, Yerom-David Bromberg, Raphael Chevalier, Alain Cadot	
Institution	LaBRI, Universite de Bordeaux, IRISA, Universite de Rennes	
Country	France	
Modeling and Composition approach	Medley	
Orchestration approach	–	
Main Contribution	Introduces a service composition platform called Medley and demonstrates its performance.	
Source	Google Scholar	

Table A.4. Filled Data Extraction 4

Data Item	Value	Additional notes
Reference	[SMMR16]	
Study	Data-Driven Workflows for Microservices: Genericity in Jolie	
Year	2016	
Author	Larisa Safina, Manuel Mazzara, Fabrizio Montesi, Victor Rivera	
Institution	Innopolis University, University of Southern Denmark	
Country	Denmark, Russia	
Modeling and Composition approach	–	
Orchestration approach	Service orchestration using Jolie	
Main Contribution	Aims to add support for data-driven workflows to Jolie which will, in turn, shift the computation from process to data.	
Source	IEEE	

Table A.5. Filled Data Extraction 5

Data Item	Value	Additional notes
Reference	[Fran17]	
Study	Architecting Microservices	
Year	2017	
Author	Paolo Di Francesco	
Institution	Gran Sasso Science Institute	
Country	Italy	
Modeling and Composition approach	Model-Driven Approach	
Orchestration approach	–	
Main Contribution	Conducts a systematic mapping on architecting microservices. Utilizes MDD to introduce and implement MicroART which architecture recovery approach for microservices. It proposes API Gateway for the orchestration layer.	
Source	IEEE	

Table A.6. Filled Data Extraction 6

Data Item	**Value**	**Additional notes**
Reference	[HaAB17]	
Study	Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity	
Year	2017	
Author	Sara Hassan, Nour Ali, Rami Bahsoon	
Institution	School of Computer Science University of Birmingham School of Computing, Engineering and Mathematics University of Brighton	
Country	UK	
Modeling and Composition approach	Aspect-oriented architectural metamodelling	
Orchestration approach	–	
Main Contribution	It extends aspect-oriented architecture for modeling microservices granularity. It introduces a modeling concept where microservices boundaries are the most important.	
Source	IEEE	

Table A.7. Filled Data Extraction 7

Data Item	Value	Additional notes
Reference	[KWGJ17]	
Study	Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures	
Year	2017	
Author	Sander Klock, Jan Martijn E. M. van der Werf, Jan Pieter Guelen, Slinger Jansen	
Institution	Utrecht University, Princetonplein AFAS Software	
Country	The Netherlands	
Modeling and Composition approach	MicADO (workload-based feature clustering)	
Orchestration approach	–	
Main Contribution	Introduces a tool, called MicADO, aiming to improve the efficiency of microservices architecture. It leverages a genetic algorithm to recommend an optimized deployment model based on workload.	
Source	IEEE	

Table A.8. Filled Data Extraction 8

Data Item	Value	Additional notes
Reference	[KoSi17]	
Study	A performance comparison of container-based technologies for the Cloud	
Year	2017	
Author	Richard O. Sinnott, Zhanibek Kozhirbayev	
Institution	faculty of information technologies, L.N. Gumilyov Eurasian National University, Department of Compuing and Information Systems, The University of Melbourne	
Country	Kazakhstan, Australia	
Modeling and Composition approach	Containerization	
Orchestration approach	–	
Main Contribution	Advocates using containers for hosting and deployment of microservices.	
Source	Science Direct	

Table A.9. Filled Data Extraction 9

Data Item	Value	Additional notes
Reference	[RaSZ17]	
Study	Differences between Model-Driven Development of Service-Oriented and Microservice Architecture	
Year	2017	
Author	Florian Rademacher, Sabine Sachweh Albert Zuendorf	
Institution	Department of Computer Science University of Applied Sciences and Arts Dortmund, Department of Computer Science and Electrical Engineering University of Kassel	
Country	Germany	
Modeling and Composition approach	MDD	
Orchestration approach	API Gateway	
Main Contribution	Considers implications of Model-driven development for microservices development.	
Source	IEEE	

Table A.10. Filled Data Extraction 10

Data Item	Value	Additional notes
Reference	[SaSR17]	
Study	RESTful web services composition and performance evaluation with different databases	
Year	2017	
Author	Neha Singhal, Usha Sakthivel, Pethuru Raj	
Institution	Dept. of ISE Engineering and Dept. of CSE Rajarajeswari, College of Engineering SRE Division Reliance Jio Infocomm.ltd(RJIL)	
Country	India	
Modeling and Composition approach	Using BPEL, JAVA, WSDL	
Orchestration approach	–	dynamic services composition
Main Contribution	Considers differences of SOAP and REST in microservices architecture. It proposes an implementation for dynamic services composition aiming to ameliorate QoS. It also evaluates the performance of two databases using RESTful and SOAP services.	
Source	IEEE	

Table A.11. Filled Data Extraction 11

Data Item	**Value**	**Additional notes**
Reference	[StOb17]	
Study	Microflows: Enabling Agile Business Process Modeling to Orchestrate Semantically-Annotated Microservices:	
Year	2017	
Author	Roy Oberhauser, Sebastian Stigler	
Institution	Computer Science Department, Aalen University, Aalen, Germany	
Country	Germany	
Modeling and Composition approach	Microflows	
Orchestration approach	–	
Main Contribution	It introduces a lightweight orchestration approach called Microflows. It uses agent-based clients and graph-based methods to extracts information from semantically-annotated microservices. It uses BPMN.	
Source	Google Scholar	

Table A.12. Filled Data Extraction 12

Data Item	Value	Additional notes
Reference	[VCTG17]	
Study	Deployment orchestration of microservices with geographical constraints for Edge computing	
Year	2017	
Author	Massimo Villari, Member, Antonio Celesti, Giuseppe Tricomi, Antonino Galletta, Maria Fazio	
Institution	Department of Engineering, University of Messina	
Country	Italy	
Modeling and Composition approach	–	
Orchestration approach	Extended Heat Orchestrator Template	deployment based orchestration
Main Contribution	Introduces an orchestration broker, which receives a service manifest as input, and extracts information regarding the deployment of microservices in federated cloud environments. It also allows users to use geolocation constraints to determine the regions where the microservices are deployed.	
Source	Google Scholar	

Table A.13. Filled Data Extraction 13

Data Item	Value	Additional notes
Reference	[GuLJ18]	
Study	Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications	
Year	2018	
Author	Carlos Guerrero, Isaac Lera, Carlos Juiz	
Institution	Computer Science Department, Balearic Islands University	
Country	Spain	
Modeling and Composition approach	Containerization	
Orchestration approach	Container Orchestration	
Main Contribution	Introduces an approach to optimize the orchestration of containers and reduce network latency. Provides a genetic approach which can make container orchestration more efficient.	
Source	Google Scholar	

Table A.14. Filled Data Extraction 14

Data Item	Value	Additional notes
Reference	[LTOG18]	
Study	Cloudification and Autoscaling Orchestration for Container-Based Mobile Networks toward 5G: Experimentation, Challenges and Perspectives	
Year	2018	
Author	Duc-Hung LUONG, Huu-Trung THIEU, Abdelkader OUTTAGARTS Yacine GHAMRI-DOUDANE	
Institution	Software Defined Mobile Network - SDMN Team Nokia Bell Labs, L3I Laboratory University of La Rochelle	
Country	France	
Modeling and Composition approach	Containerization	
Orchestration approach	Container Orchestration	
Main Contribution	Utilizes containers as well as 12-factor principles for microservices development. Compares the performance of container technologies like Docker and Kubernetes and Docker Swarm.	
Source	Google Scholar	

Table A.15. Filled Data Extraction 15

Data Item	Value	Additional notes
Reference	[MFCL18]	
Study	Using Service Dependency Graph to Analyze and Test Microservices	
Year	2018	
Author	Shang-Pin Ma, Chen-Yuan Fan, Yen Chuang, Wen-Tin Lee, Shin-Jie Lee, Nien-Lin Hsueh	
Institution	Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung Department of Software Engineering and Management, National Kaohsiung Normal University, Kaohsiung Department of Computer Science and Information Engineering, National Cheng Kung University, Department of Computer Science and Information Engineering, Feng Chia University	
Country	Taiwan	
Modeling and Composition approach	GMAT (Graph-based Microservice Analysis and Testing)	
Orchestration approach	Choreography	
Main Contribution	Introduces an approach, called GMAT, which improves analyzability of microservices dependency by visualizing dependencies between microservices which ultimately leads to a reduction in risky service invocations.	
Source	Google Scholar	

Table A.16. Filled Data Extraction 16

Data Item	Value	Additional notes
Reference	[SeGM18]	
Study	An Availability Analysis Approach for Deployment Configurations of Containers	
Year	2018	
Author	Stefano Sebastio, Rahul Ghosh, Tridib Mukherjee	
Institution	London Institute of Mathematical Sciences, Big Data Labs, American Express, Conduent Labs India	
Country	UK, India	
Modeling and Composition approach	Containerization	
Orchestration approach	–	
Main Contribution	It promotes the use of containers for microservices and introduces a tool for checking the availability of containers.	
Source	IEEE	

Table A.17. Filled Data Extraction 17

Data Item	Value	Additional notes
Reference	[TaLP18]	
Study	Architectural Patterns for Microservices: A Systematic Mapping Study:	
Year	2018	
Author	Davide Taibi, Claus Pahl,Valentina Lenarduzzi	
Institution	Tampere University of Technology, Libera Università di Bozen-Bolzano	
Country	Italy, Finland	
Modeling and Composition approach	Containerization	
Orchestration approach	API Gateway	
Main Contribution	Discover and analyses a number of widely implemented microservices architecture patterns. Present the analysis and categorize the architectural patterns and their proper use cases.	
Source	IEEE	

Table A.18. Filled Data Extraction 18

Data Item	Value	Additional notes
Reference	[Arel18]	
Study	Analysis and Classification of Service Interactions for the Scalability of the Internet of Things	
Year	2018	
Author	Damian Arellanes, Kung-Kiu Lau	
Institution	School of Computer Science The University of Manchester	
Country	UK	
Modeling and Composition approach	–	
Orchestration approach	Direct, indirect interactions, event-driven, and exogenous interactions	
Main Contribution	Performs an evaluation on microservices orchestration in IoT environments which determines the suitability of using multi-level exogenous communication.	
Source	Science Direct	

Table A.19. Filled Data Extraction 19

Data Item	Value	Additional notes
Reference	[KoDK18]	
Study	Pishahang: Joint Orchestration of Network Function Chains and Distributed Cloud Applications	
Year	2018	
Author	Hadi Razzaghi Kouchaksaraei, Tobias Dierich, Holger Karl	
Institution	Paderborn University	
Country	Germany	
Modeling and Composition approach	–	
Orchestration approach	Container Orchestration	
Main Contribution	Introduces an orchestration framework which handles cross-ecosystem dependencies for cloud-based microservices.	
Source	Google Scholar	

Table A.20. Filled Data Extraction 20

Data Item	**Value**	**Additional notes**
Reference	[Sorg17]	
Study	AjiL: A Graphical Modeling Language for the Development of Microservice Architectures	
Year	2018	
Author	Jonas Sorgalla	
Institution	University of Applied Sciences and Arts Dortmund, Institute for Digital Transformation of Application and Living Domains	
Country	Germany	
Modeling and Composition approach	Model Driven Engineering	Graphical tool for specifying microservices
Orchestration approach	–	
Main Contribution	Introduces a graphical modeling language for microservices by utilizing Model Driven Development.	
Source	Google Scholar	

Table A.21. Filled Data Extraction 21

Data Item	Value	Additional notes
Reference	[GFZV16]	
Study	A Context Model and Policies Management Framework for Reconfigurable-by-design Distributed Applications	
Year	2016	
Author	Panagiotis Gouvas, Eleni Fotopoulou, Anastasios Zafeiropoulos, ConstantinosVassilakis	
Institution	Ubitech Ltd., R&D Department	
Country	Greece	
Modeling and Composition approach	ARCADIA Component Model to design microservices	
Orchestration approach	–	
Main Contribution	Proposes a framework for policy management regarding distributed applications running on a programmable infrastructure. It advocates that distributed applications should be reconfigurable by design, hence increasing usability in different contexts.	
Source	Science Direct	

Table A.22. Filled Data Extraction 22

Data Item	Value	Additional notes
Reference	[InSi18]	
Study	Integrating Microservices	
Year	2018	
Author	Kasun Indrasiri, Prabath Siriwardena	
Institution	WSO2	
Country	Sri Lanka	
Modeling and Composition approach	Ballerina	
Orchestration approach	–	
Main Contribution	Analyzes difficulties linked with integration and orchestration of microservices. Introduces Ballerina to cope with these challenges.	
Source	Google Scholar	

Table A.23. Filled Data Extraction 23

Data Item	Value	Additional notes
Reference	[ThVS18]	
Study	Cyber-physical microservices: An IoT-based framework for manufacturing systems	
Year	2018	
Author	Kleanthis Thramboulidis, Danai C. Vachtsevanou, Alexandros Solanos	
Institution	Electrical and Computer Engineering, University of Patras	
Country	Greece	
Modeling and Composition approach	Model-Driven Engineering	
Orchestration approach	–	
Main Contribution	Proposes a framework based on Model-Driven Engineering to incorporate microservices architecture in cyber-physical manufacturing.	
Source	IEEE	

Table A.24. Filled Data Extraction 24

Data Item	Value	Additional notes
Reference	[MaCL17]	
Study	Extraction of Microservices from Monolithic Software Architectures	
Year	2017	
Author	Genc Mazlami, Jürgen Cito, Philipp Leitner	
Institution	Software Evolution and Architecture Lab Department of Informatics University of Zurich	
Country	Switzerland	
Modeling and Composition approach	Graph-based model to decompose monoliths	
Orchestration approach	–	
Main Contribution	Introduces a graph-based model which can break down a monolith to microservices.	
Source	IEEE	

Table A.25. Filled Data Extraction 25

Data Item	Value	Additional notes
Reference	[RaSS18]	
Study	Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective	
Year	2018	
Author	Florian Rademacher, Sabine Sachweh, Jonas Sorgalla	
Institution	Dortmund University of Applied Sciences and Arts	
Country	Germany	
Modeling and Composition approach	MDD	
Orchestration approach	–	
Main Contribution	Utilizes model-driven development to address issues of DDD and microservices. Performs analysis on how to derive microservices and infrastructure from domain model and proposes MDD tools for such use cases.	
Source	IEEE	

Table A.26. Filled Data Extraction 26

Data Item	Value	Additional notes
Reference	[Petr17]	
Study	Model-based engineering for microservice architectures using Enterprise Integration Patterns for inter-service communication	
Year	2017	
Author	Roland Petrasch	
Institution	Department of Computer Science, Thammasat University	
Country	Thailand	
Modeling and Composition approach	MDD	
Orchestration approach	–	
Main Contribution	Introduces a static model to enable model-to-model transformation and code generation for microservices by utilizing UML. Formalizes microservices modeling and their communications.	
Source	IEEE	

Table A.27. Filled Data Extraction 27

Data Item	Value	Additional notes
Reference	[RaGa15]	
Study	A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development	
Year	2015	
Author	Mazedur Rahman, Jerry Gao	
Institution	San Jose State University	
Country	USA	
Modeling and Composition approach	Behavior-Driven Development	
Orchestration approach	–	
Main Contribution	Utilizes Behavior Driven Development to apply automated acceptance testing to microservices.	
Source	IEEE	

Table A.28. Filled Data Extraction 28

Appendix B

Data Item	Value	Additional notes
Tool Name	Cadence	
Year of First Release	2017	
Contributor(s)	Maxim Fateev	
Institution	Uber	
Country	USA	
Repository	https://github.com/uber/cadence	

Table B.1. Filled Data Extraction for Tools 1

Data Item	Value	Additional notes
Tool Name	Conductor	
Year of First Release	2016	
Contributor(s)	-	
Institution	Netflix	
Country	USA	
Repository	https://github.com/Netflix/conductor/	

Table B.2. Filled Data Extraction for Tools 2

Data Item	Value	Additional notes
Tool Name	RockScript	
Year of First Release	2017	
Contributor(s)	-	
Institution	Rockscript	
Country	-	
Repository	https://github.com/rockscript/rockscript/	

Table B.3. Filled Data Extraction for Tools 3

Data Item	Value	Additional notes
Tool Name	Ballerina	
Year of First Release	2017	
Contributor(s)	Sanjiva Weerawarana, James Clark, Sameera Jayasoma, Hasitha Aravinda, Srinath Perera, Frank Leymann	
Institution	Iaas Stuttgart University, WSO2	
Country	Germany, Sri Lanka	
Repository	https://github.com/ballerinaplatform/ballerina-lang/	

Table B.4. Filled Data Extraction for Tools 4

Data Item	Value	Additional notes
Tool Name	Baker	
Year of First Release	2017	
Contributor(s)	-	
Institution	ING Bank	
Country	Netherlands	
Repository	https://github.com/ing-bank/baker	

Table B.5. Filled Data Extraction for Tools 5

Data Item	Value	Additional notes
Tool Name	Toki	
Year of First Release	2017	
Contributor(s)	-	
Institution	Xogroup	
Country	USA	
Repository	https://github.com/xogroup/toki	

Table B.6. Filled Data Extraction for Tools 6

Data Item	Value	Additional notes
Tool Name	Zeebe	
Year of First Release	2017	
Contributor(s)	-	
Institution	Camunda	
Country	Germany	
Repository	https://github.com/zeebe-io/zeebe	

Table B.7. Filled Data Extraction for Tools 7

Appendix C

Study	Author	Organization	Country
[RaGa15]	Mazedur Rahman, Jerry Gao	San Jose State University	USA
[GFZV16]	Panagiotis Gouvas, Eleni Fotopoulou, Anastasios Zafeiropoulos, ConstantinosVassilakis	Ubitech Ltd., R&D Department	Greece
[MMPP16]	Andrea Melis, Silvia Mirri, Catia Prandi, Marco Prandini, Paola Salomoni	Department of Electrical and Information Engineering, Department of Computer Science and Engineering-University of Bologna	Italy
[SMMR16	Larisa Safina, Manuel Mazzara, Fabrizio Montesi, Victor Rivera	Innopolis University, University of Southern Denmark	Denmark, Russia
[Fran17]	Paolo Di Francesco	Gran Sasso Science Institute	Italy
[HaAB17	Sara Hassan, Nour Ali, Rami Bahsoon	School of Computer Science University of Birmingham School of Computing, Engineering and Mathematics University of Brighton	UK
[KWGJ17]	Sander Klock, Jan Martijn E. M. van der Werf , Jan Pieter Guelen, Slinger Jansen	Utrecht University, Princetonplein AFAS Software	The Netherlands
[KoSi17]	Richard O. Sinnott, Zhanibek Kozhirbayev	Faculty of information technologies, L.N. Gumilyov Eurasian National University, Department of Compuing and Information Systems, The University of Melbourne	Kazakhstan, Australia
[RaSZ17]	Florian Rademacher, Sabine Sachweh Albert Zuendorf	Department of Computer Science University of Applied Sciences and Arts Dortmund, Department of Computer Science and Electrical Engineering University of Kassel	Germany

Study	Author	Organization	Country
[SaSR17]	Neha Singhal, Usha Sakthivel, Pethuru Raj	Dept. of ISE Engineering and Dept. of CSE Rajarajeswari, College of Engineering SRE Division Reliance Jio Infocomm.ltd(RJIL)	India
[StOb17]	Roy Oberhauser, Sebastian Stigler	Computer Science Department, Aalen University, Aalen, Germany	Germany
[VCTG17]	Massimo Villari, Member, Antonio Celesti, Giuseppe Tricomi, Antonino Galletta, Maria Fazio	Department of Engineering, University of Messina	Italy
[GuLJ18]	Carlos Guerrero, Isaac Lera, Carlos Juiz	Computer Science Department, Balearic Islands University	Spain
[LTOG18]	Duc-Hung LUONG, Huu-Trung THIEU, Abdelkader OUTTAGARTS Yacine GHAMRI-DOUDANE	Software Defined Mobile Network - SDMN Team Nokia Bell Labs, Laboratory University of La Rochelle	France
[MFCL18]	Shang-Pin Ma, Chen-Yuan Fan, Yen Chuang, Wen-Tin Lee, Shin-Jie Lee, Nien-Lin Hsueh	Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung Department of Software Engineering and Management, National Kaohsiung Normal University, Kaohsiung Department of Computer Science and Information Engineering, National Cheng Kung University, Department of Computer Science and Information Engineering, Feng Chia University	Taiwan
[SeGM18	Stefano Sebastio, Rahul Ghosh, Tridib Mukherjee	London Institute of Mathematical Sciences, Big Data Labs, American Express, Conduent Labs India	UK, India
[TaLP18]	Davide Taibi, Claus Pahl,Valentina Lenarduzzi	Tampere University of Technology,	Italy, Finland

Study	Author	Organization	Country
		Libera Università di Bozen-Bolzano	
[Arel18]	Damian Arellanes, Kung-Kiu Lau	School of Computer Science The University of Manchester	UK
[KoDK18]	Hadi Razzaghi Kouchaksaraei, Tobias Dierich, Holger Karl	Paderborn University	Germany
[Sorg17]	Jonas Sorgalla	University of Applied Sciences and Arts Dortmund, Institute for Digital Transformation of Application and Living Domains	Germany
[MGMR18]	Davi Monteiro, Rômulo Gadelha, Paulo Henrique M. Maia, Lincoln S. Rocha, and Nabor C. Mendonça	State University of Ceará, Federal University of Ceará, University of Fortaleza	Brazil
[InSi18]	Kasun Indrasiri, Prabath Siriwardena	WSO2	Sri Lanka
[ThVS18]	Kleanthis Thramboulidis, Danai C. Vachtsevanou, Alexandros Solanos	Electrical and Computer Engineering, University of Patras	Greece
[MaCL17]	Genc Mazlami, Jürgen Cito, Philipp Leitner	Software Evolution and Architecture Lab Department of Informatics University of Zurich	Switzerland
[RaSS18]	Florian Rademacher, Sabine Sachweh, Jonas Sorgalla	Dortmund University of Applied Sciences and Arts	Germany
[Petr17]	Roland Petrasch	Department of Computer Science, Thammasat University	Thailand
[GLMM17]	Claudio Guidi, Ivan Lanese, Manuel Mazzara, and Fabrizio Montesi	italianaSoftware, Focus Team, University of Bologna/INRIA, Innopolis University, University of Southern Denmark	Italy, Russia, Denmark
[BRBC16]	Elyas Ben Hadj Yahia , Laurent Reveillere, Yerom-David Bromberg, Raphael Chevalier, Alain Cadot	LaBRI, Universite de Bordeaux, IRISA, Universit´e de Rennes	France

Table C.1. Filled Data Synthesis Form for Table 3.6

Bibliography

[Arel18] ARELLANES, DAMIAN: Analysis and Classification of Service Interactions for the Scalability of the Internet of Things (2018), S. 8

[Awss00] AWS Step Functions. *Amazon Web Services, Inc.* [Online]. URL: https://aws.amazon.com/step-functions/. [Accessed: 08-Jan-2019].

[Bake17] *ING Baker.* ING Bank. [Online]. URL: https://github.com/ing-bank/baker/. [Accessed: 08-Jan-2019].

[Ball17] *Ballerina*. WSO2. [Online]. URL: https://ballerina.io/. [Accessed: 08-Jan-2019].

[BRBC16] BEN HADJ YAHIA, ELYAS ; RÉVEILLÈRE, LAURENT ; BROMBERG, YÉROM-DAVID ; CHEVALIER, RAPHAËL ; CADOT, ALAIN: Medley: An Event-Driven Lightweight Platform for Service Composition. In: BOZZON, A. ; CUDRE-MAROUX, P. ; PAUTASSO, C. (Hrsg.): *Web Engineering*. Cham : Springer International Publishing, 2016 — ISBN 978-3-319-38791-8, S. 3–20

[Cond16] *Conductor*. Netflix. [Online]. URL: https://netflix.github.io/conductor/. [Accessed: 08-Jan-2019].

[DGLM16] DRAGONI, NICOLA ; GIALLORENZO, SAVERIO ; LLUCH-LAFUENTE, ALBERTO ; MAZZARA, MANUEL ; MONTESI, FABRIZIO ; MUSTAFIN, RUSLAN ; SAFINA, LARISA: Microservices: yesterday, today, and tomorrow (2016)

[DüHo17] DÜLLMANN, THOMAS F. ; VAN HOORN, ANDRÉ: Model-driven Generation of Microservice Architectures for Benchmarking Performance and Resilience Engineering Approaches. In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion - ICPE '17 Companion*. L'Aquila, Italy : ACM Press, 2017 — ISBN 978-1-4503-4899-7, S. 171–172

[Fran17] FRANCESCO, P. D.: Architecting Microservices. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, S. 224–229

[GFZV16] GOUVAS, PANAGIOTIS ; FOTOPOULOU, ELENI ; ZAFEIROPOULOS, ANASTASIOS ; VASSILAKIS, CONSTANTINOS: A Context Model and Policies Management Framework for Reconfigurable-by-design Distributed Applications. In: *Procedia Computer Science* Bd. 97 (2016), S. 122–125

[GLMM17] GUIDI, CLAUDIO ; LANESE, IVAN ; MAZZARA, MANUEL ; MONTESI, FABRIZIO: Microservices: A Language-Based Approach. In: MAZZARA, M. ; MEYER, B. (Hrsg.): *Present and Ulterior Software Engineering*. Cham : Springer International Publishing, 2017 — ISBN 978-3-319-67425-4, S. 217–225

[GuLJ18] GUERRERO, CARLOS ; LERA, ISAAC ; JUIZ, CARLOS: Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications. In: *The Journal of Supercomputing* Bd. 74 (2018), Nr. 7, S. 2956–2983

[HaAB17] HASSAN, S. ; ALI, N. ; BAHSOON, R.: Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity. In: *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017, S. 1–10

[HoWo03] HOHPE, GREGOR ; WOOLF, BOBBY: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2003 — ISBN 0-321-20068-3 (cit. on pp. 113-115, 147-149, 439-442)

[InSi18] INDRASIRI, KASUN ; SIRIWARDENA, PRABATH: Integrating Microservices. In: *Microservices for the Enterprise: Designing, Developing, and Deploying*. Berkeley, CA : Apress, 2018 — ISBN 978-1-4842-3858-5, S. 167–217 (cit. on pp. 167, 208-209, 214-215)

[Jela11] JELASITY, MÁRK: Gossip. In: DI MARZO SERUGENDO, G. ; GLEIZES, M.-P. ; KARAGEORGOS, A. (Hrsg.): *Self-organising Software: From Natural to Artificial Adaptation*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011 — ISBN 978-3-642-17348-6, S. 139–162

[Joli07] Jolie Programming Language. [Online]. URL: https://www.jolie-lang.org/. [Accessed: 08-Jan-2019].

[Khan17] KHAN, A.: Key Characteristics of a Container Orchestration Platform to Enable a Modern Application. In: *IEEE Cloud Computing* Bd. 4 (2017), Nr. 5, S. 42–48

[Kitc00a] KITCHENHAM, BARBARA: Procedures for Performing Systematic Reviews, S. 33 (cit. on pp. 3-5, 7-10, 17-18, 22)

[Kitc00b] KITCHENHAM, BARBARA: Guidelines for performing Systematic Literature Reviews in Software Engineering, S. 44 (cit. on pp. 11-14, 36, 37)

[KKKR17] KISS, TAMAS ; KACSUK, PETER ; KOVACS, JOZSEF ; RAKOCZI, BOTOND ; HAJNAL, AKOS ; FARKAS, ATTILA ; GESMIER, GREGOIRE ; TERSTYANSZKY, GABOR: MiCADO—Microservice-based Cloud Application-level Dynamic Orchestrator. In: *Future Generation Computer Systems* (2017)

[KoDK18] KOUCHAKSARAEI, HADI RAZZAGHI ; DIERICH, TOBIAS ; KARL, HOLGER: Pishahang: Joint Orchestration of Network Function Chains and Distributed Cloud Applications (2018), S. 3

[KoSi17] KOZHIRBAYEV, ZHANIBEK ; SINNOTT, RICHARD O.: A performance comparison of container-based technologies for the Cloud. In: *Future Generation Computer Systems* Bd. 68 (2017), S. 175–182

[KWGJ17] Klock, S. ; Werf, J. M. E. M. V. D. ; Guelen, J. P. ; Jansen, S.: Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures. In: *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017, S. 11–20

[LeRo00] Leymann, Frank ; Roller, Dieter: *Production Workflow: Concepts and Techniques*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2000 — ISBN 0-13-021753-0 (cit. on pp. 1-10, 62, 346)

[LTOG18] Luong, D. ; Thieu, H. ; Outtagarts, A. ; Ghamri-Doudane, Y.: Cloudification and Autoscaling Orchestration for Container-Based Mobile Networks toward 5G: Experimentation, Challenges and Perspectives. In: *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, 2018, S. 1–7

[MaCL17] Mazlami, G. ; Cito, J. ; Leitner, P.: Extraction of Microservices from Monolithic Software Architectures. In: *2017 IEEE International Conference on Web Services (ICWS)*, 2017, S. 524–531

[MFCL18] Ma, S. ; Fan, C. ; Chuang, Y. ; Lee, W. ; Lee, S. ; Hsueh, N.: Using Service Dependency Graph to Analyze and Test Microservices. In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. Bd. 02, 2018, S. 81–86

[MGMR18] Monteiro, Davi ; Gadelha, Rômulo ; Maia, Paulo Henrique M. ; Rocha, Lincoln S. ; Mendonça, Nabor C.: Beethoven: An Event-Driven Lightweight Platform for Microservice Orchestration. In: Cuesta, C. E. ; Garlan, D. ; Pérez, J. (Hrsg.): *Software Architecture*. Cham : Springer International Publishing, 2018 — ISBN 978-3-030-00761-4, S. 191–199

[Micr00] *Microservices Pattern: Microservice Architecture pattern*. URL http://microservices.io/patterns/microservices.html. [Accessed: 08-Jan-2019].

[MMPP16] Melis, A. ; Mirri, S. ; Prandi, C. ; Prandini, M. ; Salomoni, P.: A Microservice-Based Architecture for the Development of Accessible, Crowdsensing-Based Mobility Platforms. In: *2016 International Conference on Collaboration Technologies and Systems (CTS)*, 2016, S. 498–505

[MoGZ14] Montesi, Fabrizio ; Guidi, Claudio ; Zavattaro, Gianluigi: Service-Oriented Programming with Jolie. In: Bouguettaya, A. ; Sheng, Q. Z. ; Daniel, F. (Hrsg.): *Web Services Foundations*. New York, NY : Springer New York, 2014 — ISBN 978-1-4614-7518-7, S. 81–107

[Netw18] *Tchannel*. Uber Open Source. [Online]. URL: https://github.com/uber/tchannel/. [Accessed: 08-Jan-2019].

[Newm15] Newman, Sam: *Building Microservices*. 1st. Aufl. : O’Reilly Media, Inc., 2015 — ISBN 1-4919-5035-8 (cit. on pp. 17–19)

[Obje18] *Joi hapi.js*. hapi.js. [Online]. URL: https://github.com/hapijs/joi/. [Accessed: 08-Jan-2019].

[Orch18] *ING Baker Documentation*. ING Bank. [Online]. URL: https://ing-bank.github.io/baker/. [Accessed: 08-Jan-2019].

[Petr17] PETRASCH, R.: Model-based engineering for microservice architectures using Enterprise Integration Patterns for inter-service communication. In: *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2017, S. 1–4

[RaGa15] RAHMAN, M. ; GAO, J.: A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development. In: *2015 IEEE Symposium on Service-Oriented System Engineering*, 2015, S. 321–325

[RaSS18] RADEMACHER, F. ; SORGALLA, J. ; SACHWEH, S.: Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective. In: *IEEE Software* Bd. 35 (2018), Nr. 3, S. 36–43

[RaSZ17] RADEMACHER, F. ; SACHWEH, S. ; ZÜNDORF, A.: Differences between Model-Driven Development of Service-Oriented and Microservice Architecture. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, S. 38–45

[Rock17] *RockScript.io*. RockScript.io. [Online]. URL: https://rockscript.github.io/. [Accessed: 08-Jan-2019].

[RSSZ18] RADEMACHER, FLORIAN ; SORGALLA, JONAS ; SACHWEH, SABINE ; ZÜNDORF, ALBERT: Towards a Viewpoint-specific Metamodel for Model-driven Development of Microservice Architecture. In: *CoRR* Bd. abs/1804.09948 (2018)

[SaSR17] SAKTHIVEL, U. ; SINGHAL, N. ; RAJ, P.: RESTful web services composition amp;amp; performance evaluation with different databases. In: *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, 2017, S. 1–4

[SeGM18] SEBASTIO, S. ; GHOSH, R. ; MUKHERJEE, T.: An Availability Analysis Approach for Deployment Configurations of Containers. In: *IEEE Transactions on Services Computing* (2018), S. 1–1

[SMMR16] SAFINA, L. ; MAZZARA, M. ; MONTESI, F. ; RIVERA, V.: Data-Driven Workflows for Microservices: Genericity in Jolie. In: *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, 2016, S. 430–437

[Sorg17] SORGALLA, JONAS: AjiL: A Graphical Modeling Language for the Development of Microservice Architectures (2017), S. 4

[StOb17] STIGLER, SEBASTIAN ; OBERHAUSER, ROY: Microflows: Enabling Agile Business Process Modeling to Orchestrate Semantically-Annotated Microservices: In: *Proceedings of the Seventh International Symposium on Business Modeling and Software Design*. Barcelona, Spain : SCITEPRESS - Science and Technology Publications, 2017 — ISBN 978-989-758-238-7, S. 19–28

[TaLP18] TAIBI, DAVIDE ; LENARDUZZI, VALENTINA ; PAHL, CLAUS: Architectural Patterns for Microservices: A Systematic Mapping Study: In: *Proceedings of the 8th International Conference on Cloud Computing and Services Science*. Funchal, Madeira, Portugal : SCITEPRESS - Science and Technology Publications, 2018 — ISBN 978-989-758-295-0, S. 221–232

[ThVS18] THRAMBOULIDIS, K. ; VACHTSEVANOU, D. C. ; SOLANOS, A.: Cyber-physical microservices: An IoT-based framework for manufacturing systems. In: *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, 2018, S. 232–239

[Toki17] *Toki*. XO Group. [Online]. Available: https://github.com/xogroup/toki/. [Accessed: 08-Jan-2019]. *Microservice orchestration engine utilizing a rules engine: xogroup/toki* : XO Group, 2017

[Uber17] *Uber Cadence*. Uber Open Source. [Online]. URL: https://github.com/uber/cadence/. [Accessed: 08-Jan-2019].

[VCTG17] VILLARI, M. ; CELESTI, A. ; TRICOMI, G. ; GALLETTA, A. ; FAZIO, M.: Deployment orchestration of microservices with geographical constraints for Edge computing. In: *2017 IEEE Symposium on Computers and Communications (ISCC)*, 2017, S. 633–638

[WEPL18] WEERAWARANA, SANJIVA ; EKANAYAKE, CHATHURA ; PERERA, SRINATH ; LEYMANN, FRANK: Bringing Middleware to Everyday Programmers with Ballerina. In: WESKE, M. ; MONTALI, M. ; WEBER, I. ; VOM BROCKE, J. (Hrsg.): *Business Process Management*. Cham : Springer International Publishing, 2018 — ISBN 978-3-319-98648-7, S. 12–27

[Zeeb17] Zeebe. *Camunda*. [Online]. URL: https://zeebe.io/. [Accessed: 08-Jan-2019].

All links were last followed on January 11, 2019

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Place, date, signature