CharM — A Model for Characterizing the Architecture of Service-based Systems

Thatiane de Oliveira Rosa

Thesis presented to the
Institute of Mathematics and Statistics
of the University of São Paulo
in partial fulfillment
of the requirements
for the degree of
Doctor of Science

Program: Computer Science
Advisor: Prof. Ph.D. Alfredo Goldman
Coadvisor: Prof. Ph.D. Eduardo Martins Guerra

São Paulo - SP
July 28, 2023

CharM — A Model for Characterizing the Architecture of Service-based Systems

Thatiane de Oliveira Rosa

This version of the thesis includes the corrections and modifications suggested by the Examining Committee during the defense of the original version of the work, which took place on July 28, 2023.

A copy of the original version is available at the Institute of Mathematics and Statistics of the University of São Paulo.

Examining Committee:

- Prof. Ph.D. Eduado Martins Guerra (Co-Advisor) - UNIBZ
- Prof. Ph.D. Elisa Yumi Nakagawa - ICMC-USP
- Prof. Ph.D. Kelly Rosa Braghetto - IME-USP
- Prof. Ph.D. Thais Vasconcelos Batista - UFRN
- Prof. Ph.D. Thelma Elita Colanzi - UEM

Acknowledgments

First, I thank God for always being present in my life, guiding me along good paths, giving me great opportunities like this one, and giving me the strength to conquer my dreams.

To my parents, Estevam and Maria Neuza, my brothers Thaíse and Stefan, my nephew Tarso, and my niece Laura, for all their love, care, friendship, support, encouragement, trust, and understanding. Thank you for always being there, encouraging me, welcoming me, understanding my absence at different times, allowing me to share my anxieties, and celebrating every achievement I reached.

I would like to thank my husband and great friend, Bruno Vilar, for all his love, affection, friendship, encouragement, patience, concern, understanding, and willingness to help emotionally and professionally. Thank you for listening, understanding, and calming me down.

I am grateful to my friends. To my great and dear childhood friend Francine for always rooting for me, listening to me, advising, encouraging, and cheering me up. I also thank the dear friends I made during this doctoral journey: Graziela Tonin, João Daniel, Luis Araujo, Mairieli Wessel, Renato Cordeiro, and Samuel Plaça. I am very grateful to you for all the conversations and advice, and for all the moments I had the opportunity and the privilege of working, learning, and exchanging experiences with you. Mainly, I am very grateful to have shared intense, challenging, sad, and happy moments with Luis and Mairi. I feel that I have gained another brother and sister.

I also thank my English teacher, Raphaela Mathias, for all the teachings and all the patience she had in revising my texts countless times. I also thank my psychologist, Priscila Simião, for helping me through the arduous process of self-knowledge, which was essential to overcoming many of the enormous challenges I faced during my Ph.D. Thank you both for encouraging me and cheering me on.

I would like to thank Professor Alfredo Goldman for believing in my potential and

accepting to be my advisor, allowing me to do my Ph.D. at one of the best universities in the world. I am very grateful for challenging myself (in the most different ways), for encouraging me, for opening so many doors to so many incredible opportunities, for worrying about me, for trying to understand me, for listening to me, and for allowing me to exchange learning (scientific and life) like you.

I would also like to thank my co-advisor, Professor Eduardo Guerra, for his excellent guidance, as well as for his professional advice and all his attention and care. I would also like to thank Professor Filipe Correia for his guidance during the survey. Thank you very much for all your patience, respect, and attention. I consider myself very privileged to have had the opportunity to work and learn so much from you both.

I would also like to thank Professors Carolyn Seaman, Fabio Kon, João Eduardo Ferreira, Mauricio Aniche, Sinai Robins, and Xiaofeng Wang, with whom I had the opportunity to learn so much in subjects I studied during my Ph.D.

Thank you very much to the teachers who agreed to contribute to this research as members of the evaluation committee for this Thesis. Thank you for sharing your knowledge and experience at this important time in my life. Thank you very much, teachers Elisa Nakagawa, Kelly Braghetto, Thais Batista, and Thelma Colanzi. I really admire the work of each of you.

I would also like to thank the Federal Institute of Tocantins – IFTO for granting me the license so that, for four years, I could dedicate myself exclusively to my Ph.D.

Finally, I would like to thank all my family, friends, Ph.D. and work colleagues, and students who have always cheered for my success.

Resumo

Thatiane de Oliveira Rosa. **CharM - Um Modelo para Caracterizar a Arquitetura de Sistemas Baseados em Serviços**. Tese (Doutorado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

A arquitetura baseada em serviços surgiu para auxiliar profissionais a superar desafios tais como dificuldade para escalar o software, baixa produtividade e forte dependência entre elementos de um sistema. Microsserviços é um estilo arquitetural baseado em serviço que oferece vantagens como escalabilidade, agilidade, resiliência e reutilização. Esse estilo arquitetural tem sido bem aceito e utilizado na indústria, assim como tem sido alvo de diversos estudos acadêmicos. No entanto, ao analisar o estado da arte e da prática, percebe-se que existe um limite nebuloso ao tentar classificar e caracterizar a arquitetura de sistemas baseados em serviços. Além disso, é possível perceber que é difícil analisar as perdas e os ganhos para tomar decisões quanto ao projeto e evolução desse tipo de sistema. Alguns exemplos concretos dessas decisões estão relacionados ao tamanho dos serviços, como eles se comunicam e como os dados devem ser divididos/compartilhados. Com base nesse contexto, desenvolvemos o CharM, um modelo de caracterização da arquitetura de sistemas baseados em serviços, que adota diretrizes de microsserviços. Para atingir esse objetivo, seguimos as diretrizes da *Design Science Research* em cinco iterações, compostas por revisões de literatura *ad-hoc*, discussões com especialistas, dois estudos de caso e um questionário. A principal contribuição desta tese é o CharM, que é um modelo de caracterização arquitetural de fácil compreensão, que auxilia profissionais com diferentes perfis a compreenderem, documentarem e manterem a arquitetura de sistemas baseados em serviços.

Palavras-chave: Arquitetura de Software. Sistema Baseado em Serviço. Microsserviço. Modelo de Caracterização

Abstract

Thatiane de Oliveira Rosa. **CharM — A Model for Characterizing the Architecture of Service-based Systems**. Thesis (Doctorate). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

Service-based architecture emerged to overcome software development challenges, such as difficulty to scale, low productivity, and strong dependence between elements. Microservice is a service-based architectural style that offers advantages, such as scalability, agility, resilience, and reuse. This architectural style has been well accepted and used in industry and has been the target of several academic studies. However, analyzing the state of the art and practice, we can notice a fuzzy limit when trying to classify and characterize the architecture of service-based systems. Furthermore, it is possible to realize that it is difficult to analyze the trade-offs to make decisions regarding the design and evolution of this kind of system. Some concrete examples of these decisions are related to how big the services should be, how they communicate, and how the data should be divided/shared. Based on this context, we developed the CharM, a model for characterizing the architecture of service-based systems that adopts microservices guidelines. To achieve this goal, we followed the guidelines of the Design Science Research in five iterations, composed of *ad-hoc* literature reviews, discussions with experts, two case studies, and a survey. The main contribution of this thesis is the CharM, which is an easily understandable architectural characterization model that helps professionals with different profiles to understand, document, and maintain the architecture of service-based systems.

Keywords: Software Architecture. Service-based System. Microservice. Characterization Model

List of Figures

1.1 Overview of the research design 4

3.1 Design process of the CharM 20
3.2 CharM's dimensions, metrics, and microservices guidelines. 24
3.3 Ruler of size dimension. 25
3.4 Ruler of data source coupling dimension. 25
3.5 Ruler of synchronous coupling dimension. 26
3.6 Ruler of asynchronous coupling dimension. 27
3.7 Overview of the Pingr architecture (notation inspired by Merson and Jospeh Yoder (2019)). 28
3.8 User Service characterization. 29
3.9 Ping Service characterization. 30
3.10 Chat Service characterization. 31
3.11 Feed Service characterization. 32
3.12 Pingr characterization. 33

4.1 Key steps of the multiple case study 36
4.2 InterSCity case study steps. 37
4.3 Resource Discovery characterization. 39
4.4 Actuator Controller characterization. 40
4.5 Data Collector characterization. 41
4.6 Resource Adaptor characterization. 42
4.7 Resource Catalog characterization. 43
4.8 InterSCity characterization. 44
4.9 Overview of the InterSCity architecture (notation inspired by Merson and Jospeh Yoder (2019)). 45
4.10 Evaluation of the CharM's usefulness, ease of understanding, and coherence – InterSCity case study. 47
4.11 Uses of the CharM – InterSCity case study. 48

4.12 Hierarchical organization of the uses of the CharM – InterSCity case study. 49
4.13 Improvement suggestions to the CharM – InterSCity case study. 51
4.14 Industry case study steps. 54
4.15 Overview of Company A's systems architecture. 58
4.16 Service A characterization. 59
4.17 Service B characterization. 60
4.18 Service C characterization. 61
4.19 Service D characterization. 62
4.20 Service E characterization. 63
4.21 Service F characterization. 64
4.22 Service G characterization. 65
4.23 Service H characterization. 67
4.24 Search System characterization. 68
4.25 Overview of the Search System architecture (notation inspired by Merson and Jospeh Yoder (2019)). 69
4.26 Evaluation of the CharM's usefulness, ease of understanding the dimensions, ease of understanding the results, and coherence – Industry case study. 71
4.27 Uses of the CharM – Industry case study. 72
4.28 Hierarchical organization of the uses of the CharM – Industry case study. 73
4.29 CharM's main categories of uses. 74
4.30 Improvement suggestions to the CharM – Industry case study. 75

5.1 Overview of the research design of the survey. 80
5.2 Relationship between participants' resident country and CharM's evaluation. 81
5.3 Research variables. 83
5.4 Summary of participants general profile, according to gender, location, and education level. From top to bottom, we present the (a) gender identify, (b) participant location, and (c) highest completed education level. All percentages were rounded based on simple rules of rounding numbers. . 85
5.5 Summary of participants professional actuation profile. (a) work field in the past 2 years, (b) professional role in the past 2 years, (c) work sector in the past 2 years, and (d) technological layer in the past 2 years. The sum of percentages exceeds 100% as participants could select multiple answers. All percentages were rounded based on simple rules of rounding numbers. 86

5.6 Summary of participants self-declared experience with service-based architecture. (a) range of time of experience with service-based architecture, (b) experience with different service-based architectural styles - it was possible to select multiple layers, and (c) experience level with Microservice-Based Architectural Style. All percentages were rounded based on simple rules of rounding numbers. 86
5.7 Participants' perspective to which extent the CharM supports understanding a service-based architecture. All percentages were rounded based on simple rules of rounding numbers. 87
5.8 Participants' perspective to which extent the CharM supports the maintenance of a service-based architecture. All percentages were rounded based on simple rules of rounding numbers. 88
5.9 Participants' perspective to which extent the CharM supports the communication of a service-based architecture to stakeholders. All percentages were rounded based on simple rules of rounding numbers. 89
5.10 Participants' perspective on how easy it is to understand the CharM. All percentages were rounded based on simple rules of rounding numbers. . 90
5.11 The CharM's evaluation according to participants' self-declared years of experience with service-based architecture. All percentages were rounded based on simple rules of rounding numbers. 91
5.12 The CharM's evaluation according to participants' self-declared level of experience with microservices. All percentages were rounded based on simple rules of rounding numbers. 92

6.1 Comparison of related research. 100

B.1 Pattern diagram of the selected patterns (inspired by Richardson (2020)) 115
B.2 Trade-offs diagram 117

D.1 This diagram is a simplified and anonymized replica of the diagram available in the company repository. 131

E.1 Number of operations of each InterSCity service. 140
E.2 Relationship between services and data sources of the InterSCity. 141
E.3 Degree of the synchronous importance and synchronous dependence of each service of the InterSCity. 142
E.4 Relationship between services requesting operations and services receiving requests in the InterSCity Platform. 142
E.5 Synchronous importance degree of each InterSCity service. 143

E.6 Synchronous dependence degree of each InterSCity service. 143
E.7 Degree of the asynchronous importance and asynchronous dependence of each service of the InterSCity. 144
E.8 Relationship between services that publish on a message queue and services that are subscribed to receive messages from a queue in the InterSCity Platform. 144
E.9 Asynchronous importance degree of each InterSCity service. 145
E.10 Asynchronous dependence degree of each InterSCity service. 145
E.11 Main structural elements of the InterSCity architecture. 146
E.12 Interactions between the InterSCity services (inspired by Richardson, 2018). 147

F.1 Number of operations of each Search System service. 150
F.2 Relationship between services and data sources of the Search System. . . 151
F.3 Degree of the synchronous importance and synchronous dependence of each service of the Search System. 152
F.4 Relationship between services requesting operations and services receiving requests in the Search System. 153
F.5 Synchronous importance degree of each Search System service. 154
F.6 Synchronous dependence degree of each Search System service. 154
F.7 Degree of the asynchronous importance and asynchronous dependence of each service of the Search System. 155
F.8 Relationship between services that publish on a message queue and services that are subscribed to receive messages from a queue in the Search System. 156
F.9 Asynchronous importance degree of each Search System service. 157
F.10 Asynchronous dependence degree of each Search System service. 157
F.11 Interactions between the services and the message queue. 158
F.12 Interactions between the services of the Company A's Search System (inspired by Richardson, 2018). 159

List of Tables

2.1 Software architecture definitions 8

4.1 Participants profile – InterSCity case study. 46
4.2 Participants profile – Industry case study. 70

A.1 Candidate metrics - Second version of the CharM 110

Contents

1 Introduction **1**

1.1 Research Objective . . . 2
1.2 Research Design . . . 3
1.3 Originality . . . 5
1.4 Relevance . . . 5
1.5 Main Contributions . . . 6
1.6 Thesis Structure . . . 6

2 Background **7**

2.1 Software Architecture . . . 7
2.2 Monolithic Architectural Style . . . 9
2.3 Service-Based Architectural Styles . . . 11
2.3.1 Microservices Architectural Style . . . 12
2.4 Chapter Summary . . . 17

3 The CharM – Characterization Model **19**

3.1 Design Process . . . 19
3.2 Definition . . . 21
3.3 Dimensions . . . 23
3.4 Fictional Scenario to Demonstrate the CharM Application . . . 27
3.5 Chapter Summary . . . 33

4 Multiple Case Studies **35**

4.1 InterSCity Case Study . . . 36
4.1.1 Research Design . . . 36
4.1.2 Characterization of the Architecture of the InterSCity Platform . . . 38
4.1.3 Evaluation Results . . . 45
4.1.4 Threats to Validity . . . 52
4.2 Industry Case Study . . . 53

4.2.1 Research Design . . . 53
4.2.2 Overview of the Company A's Systems . . . 57
4.2.3 Characterization of the Architecture of the Search System . . . 57
4.2.4 Evaluation Results . . . 70
4.2.5 Threats to Validity . . . 76
4.3 Multiple Case Study Discussion . . . 77
4.4 Chapter Summary . . . 78

5 Survey **79**
5.1 Methodology . . . 80
5.2 Sampling . . . 81
5.3 Research Variables . . . 82
5.4 Data Analysis . . . 83
5.5 Execution and Replication . . . 83
5.6 Limitations and Threats to Validity . . . 84
5.7 Evaluation Results . . . 85
5.7.1 Participants Profile . . . 85
5.7.2 The Uses of the CharM . . . 87
5.7.3 The Ease of Understanding of the CharM . . . 90
5.7.4 The Use and Ease of Understanding of the CharM According to Professional Experience . . . 90
5.8 Discussion . . . 92
5.9 Chapter Summary . . . 95

6 Related Research **97**
6.1 Architectural Recovery . . . 97
6.2 Architectural Evaluation . . . 98
6.3 Architectural Migration . . . 100
6.4 Comparison of Related Research . . . 100
6.5 Chapter Summary . . . 101

7 Conclusion **103**
7.1 Contributions . . . 104
7.2 Scientific Publications . . . 105
7.2.1 Published or Submitted Papers . . . 105
7.2.2 Papers in Progress . . . 107
7.3 Future Work . . . 107

Appendices

A *Ad-hoc* Review of Metrics **109**

B Method for Architectural Trade-off Analysis **113**

B.1 Demonstration – Architectural Trade-off Analysis with Microservices Structural Attributes 114

B.2 Discussions 121

C Roadmap for the Manual Collection of a Service's Metrics **123**

D Supplementary Material for the Case Studies **125**

D.1 InterSCity Case Study - Interview Script - CharM Evaluation 125

D.2 Interview Code Book of the InterSCity Case Study 127

D.3 Industry Case Study - Interview Script - System Architecture Overview . 129

D.4 Industry Case Study - Interview Script - Sub-system Architecture Overview 130

D.5 Industry Case Study - Interview Script - Metric Collection 132

D.6 Industry Case Study - Interview Script - CharM's Evaluation 133

D.7 Interview Code Book of the Company A Case Study 136

E Complementary Visualizations of the Characterization of the InterSCity Architecture **139**

F Complementary Visualizations of the Characterization of the Search System **149**

References **161**

Chapter 1

Introduction

Service-based architecture (SBA) emerged to overcome common challenges to monolithic software, such as difficulty in maintaining and scaling the software, low productivity, and strong dependence between elements (Richards, 2015; Bogner, Wagner, et al., 2017a). Among service-based architecture advantages are greater development agility, scalability, resilience, reuse, and support for technological heterogeneity (Mahmood, 2007; Newman, 2021; innoQ, 2015; Richardson, 2018). SOA (Service-Oriented Architecture) (Natis and Schulte, 2003), Self-Contained Systems (innoQ, 2015), and, more recently, microservice (Lewis and M. Fowler, 2014) are examples of service-based architectural styles. These architectural styles are based on the principles of decomposing complex systems into services, loosely coupled, and communicating by messages. In this work, we use "service-based architecture" to denominate architectural styles that follow these principles.

Despite the benefits presented, architecting and developing service-based systems (SBS) is complex. Among the main challenges faced today are the complexity of the development process, difficulty in defining the size and level of coupling of the services, maintaining data consistency, and the necessity to have a robust and automated infrastructure (Mahmood, 2007; Richards, 2015; innoQ, 2015; Newman, 2021; Soldani et al., 2018; Ford, 2018). Furthermore, experience reports from companies, such as Amazon (Kolny, 2023), Istio (Mendonça et al., 2021), Segment (InfoQ, 2020), and Uber (Highscalability, 2020) reveal the confusion and highlight the difficulty in designing, understanding, and characterizing the architecture of a given service-based system.

The Prime Video team migrated the architecture of the audio/video quality inspection tool from microservices to a monolith. In the published report, the company explained that the scaling bottlenecks and high cost to scale were some reasons for this architectural migration. However, the company also clarified that they never intended nor designed such a tool to run at a large scale. The strategy adopted by the Prime Video team to deal with the identified problems was to unite the three main components of the tool in a single process. With that, the orchestration logic was simplified, and data transfer now occurs inside the process memory. This architectural change resulted in a 90% reduction in infrastructure cost and a scalability increase (Kolny, 2023). The Istio development team thought that microservices would be the right architectural solution for their system. However, since all the components were installed and operated by a single team or individual, they realized

that it was not worth paying the price for the greater operational complexity inherent to microservices. Faced with this scenario, the team analyzed the related trade-off, rethought the system, and migrated to a monolithic architecture (Mendonça et al., 2021).

In the case of Segment, to solve fault isolation and operational overhead problems, the team decided to migrate its monolith to microservices. After three years of migration, the team concluded that microservices were not the most suitable solution for the Segment. Upon realizing this, the team again adopted a monolithic style more consciously, understanding the real problem to be solved and the advantages and disadvantages of this new migration (InfoQ, 2020). On Uber, one of its teams, which currently adopts the microservices architectural style, reported moving many of its microservices to what they called "macroservices", which would be "well-sized services". One of the justifications for this change is the high complexity of managing thousands of microservices (Highscalability, 2020). It is important to note that all these reports are somehow associated with microservices since this is considered the latest trend in software development (Zhou et al., 2023).

In addition to these issues, when analyzing the discussions presented by Nadareishvili et al. (2016) and Newman (2021), as well as the variety of terms that emerged recently related to service-based architectures (such as macroservice), it is possible to notice that there is a fuzzy limit when trying to classify and characterize the architecture of service-based systems. Therefore, this scenario demonstrates the value of a solution that characterizes a service-based system's architecture, since it can help to make grounded design decisions, find acceptable architectural solutions, and know the related trade-off to better meet the desired quality attributes for a system.

1.1 Research Objective

Faced with the presented context, the main goal of this thesis is ***to develop a model for characterizing the architecture of service-based systems, adopting microservices guidelines***.

It is important to clarify that in this research, a characterization model is defined as a way of describing and explaining something, listing some main qualities of the analyzed thing. Therefore, it is useful for identifying problems and solutions in the context where it is applied. The model we developed is named CharM and is organized into four dimensions, allowing professionals to characterize a service-based system's architecture based on static metrics related to the structural attributes of size and coupling.

Thus, a model such as the CharM could be used to understand the architecture of a service-based system and aid in the process of architectural communication, documentation, and maintenance. Furthermore, consequently, our model could support professionals in identifying architectural trade-offs and in grounded making architectural decisions.

1.2 Research Design

To achieve the research goal, we applied and evaluated the CharM through multiple case studies and a survey. We aim to validate possible uses of the CharM and evaluate the ease of understanding its structure and the architectural characterization generated from its metrics. From the two case studies, we intend to answer the following research questions:

- RQ1: "What uses of the CharM are perceived by participants?"
- RQ2: "What is the participants' perception of the architectural characterization generated from the CharM?"
- RQ3: "What aspects of the CharM can be improved?"

After analyzing the preliminary results of the case studies, we investigated the following research questions during a survey application:

- RQ4: "To which extent does the CharM support understanding a service-based architecture?"
- RQ5: "To which extent does the CharM support a service-based architecture maintenance?"
- RQ6: "To which extent does the CharM support communicating a service-based architecture to stakeholders?"
- RQ7: "How easy is it to understand the CharM?"
- RQ8: "Does the participants' experience influences the perceived usefulness and ease of understanding of the CharM?"

We conducted an empirical study combining different research methods to attain the proposed objective and address the presented research questions. Thus, the process of developing and evaluating our characterization model was based on the Design Science Research paradigm's guidelines (Hevner et al., 2004). The methods used during this process include exploratory and descriptive research combined with bibliographical and documentary research procedures. Besides that, we conducted two case studies and a survey and adopted quantitative and qualitative methods to analyze the results. Figure 1.1 illustrates an overview of the steps that compose the research.

We started the studies with exploratory and descriptive research. At first, we carried out an *ad-hoc* bibliographical review where we consulted primary and secondary scientific studies, as well as analyzed industrial experience reports. Our goal was to map key concepts and understand the importance of software architecture. Additionally, we investigated the workings, advantages, and challenges of monolithic, SOA, and Self-Contained Systems. At this stage, we also aimed to identify relevant microservices characteristics and guidelines. Therefore, from this investigation and discussions with experts in software architecture, we identified interesting structural attributes and thus refined the research problem. From this, we elaborated the first version of the CharM.

In the following step, we performed a new *ad-hoc* literature search, where we also

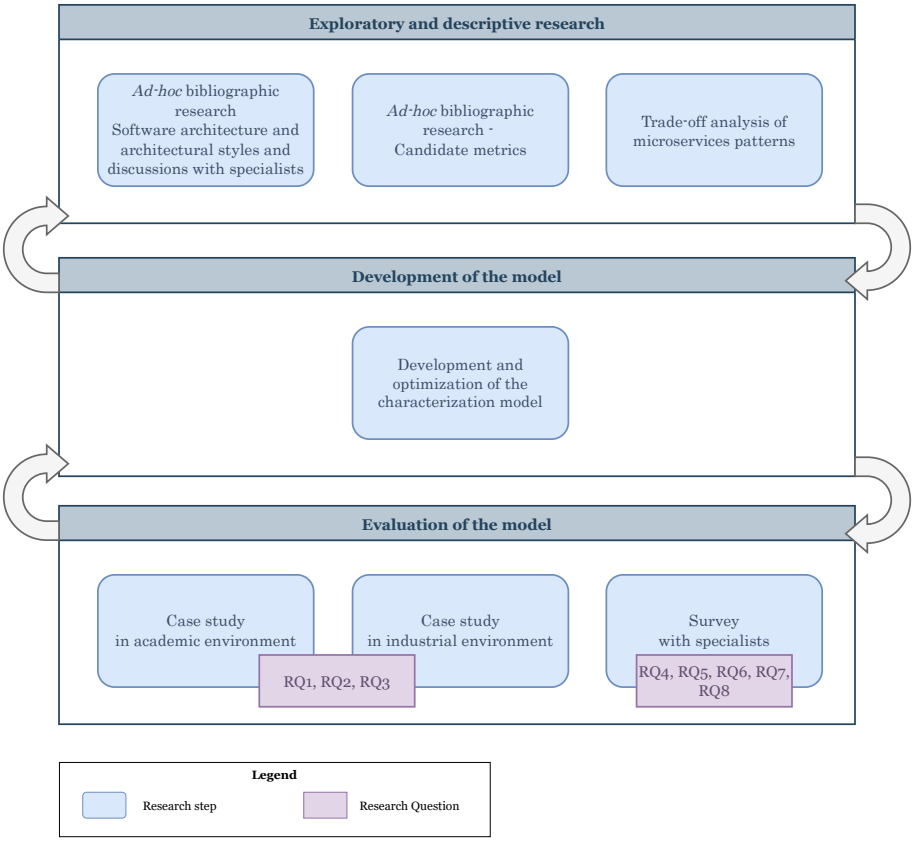


Figure 1.1: *Overview of the research design*

consulted primary and secondary scientific studies, as well as the documentation of architectural patterns for microservices. At this step, we aimed to identify and select metrics relevant to the context of this research related to the structural attributes of size and coupling. As a result, we obtained a list of viable candidate metrics to compose the CharM. Based on these results, we refined our characterization model.

Next, we performed a systematic analysis to identify the influence exerted by microservice patterns on structural attributes related to size, coupling, and data source sharing. From this analysis, we mapped trade-offs of the analyzed architectural patterns. In addition, we refined the research problem and identified points that could be improved in the CharM, generating its second version.

Afterward, we begin the process of evaluating our characterization model. For this, we carried out two case studies (Runeson et al., 2012) and a survey (Callegaro et al., 2015; Baltes and Ralph, 2020). In the first case study, we used the CharM to characterize the architecture of the InterSCity Platform for smart cities (Esposte, 2018), developed in an academic environment. A group of 11 professionals evaluated the results generated from the CharM. At the end of this case study, we discovered some possible uses of the CharM. We also evaluated its ease of understanding and identified points for improvement. From this, we generate the third version of our model.

In the second case study, we used the CharM to characterize the architecture of the search system of an online handcrafts marketplace. During this study, we interviewed 18

employees of the company, of which six evaluated the generated architectural characterization. In the end, we confirmed and discovered uses for our model and evaluated its ease of understanding. Furthermore, we got new suggestions for improvements. Based on these results, we elaborated the CharM's fourth version. In both case studies, we conducted a set of semi-structured interviews (a total of 37) and qualitatively analyzed the transcripts using open and axial coding procedures (Corbin and A. Strauss, 2015; Stol et al., 2016). After the two case studies, we obtained answers to research questions 1, 2, and 3.

Based on the results obtained from the two case studies, we designed a survey to obtain an evaluation of our model by a wider audience. Our survey was targeted at professionals who work with the architecture of service-based systems. In all, we received 58 valid responses from professionals in ten countries. From the answers obtained, we validated some previously identified uses for the CharM, as well as its ease of understanding. We also analyzed whether the results (uses and ease of understanding) varied according to the interviewee's professional experience. Furthermore, we discovered new uses and collect new suggestions for improvement. With that, we generated the fifth version of the CharM. We analyzed quantitatively and qualitatively the data collected during this evaluation stage. From the survey, we obtained answers to research questions 4, 5, 6, 7, and 8.

1.3 Originality

By analyzing the state of the art, we identified solutions that aid in mitigating the problem of characterizing the architecture of service-based systems. These solutions have varied objectives and may support architectural analysis, recovery, understanding, evaluation, or migration (Granchelli et al., 2017a; Zdun et al., 2017; Engel et al., 2018; Mayer and Weinreich, 2018; Cardarelli et al., 2019; Alshuqayran, 2020; Auer et al., 2021; Peng et al., 2022; Cerny, Amr S. Abdelfattah, et al., 2022).

However, none of the analyzed solutions focuses on architectural characterization, as the CharM. Furthermore, another characteristic that distinguishes our model from most of the other analyzed solutions is code and infrastructure independence. This feature enables the adoption of the CharM to characterize both the architecture of a service-based system being designed and systems already in the production environment. It is also worth mentioning that the CharM explores different and didactic ways for architectural visualization of services and systems.

1.4 Relevance

Our work contributes to the state of the art and practice by proposing an empirically validated theoretical and conceptual model to characterize the architecture of service-based systems. Our model aids software architects in better understanding the structural characteristics of a given system and, consequently, identifying structural aspects that must be improved or maintained. Besides that, the CharM facilitates the mapping and measurement of different impacts generated in the software architecture. It can also support architectural decisions that consider different quality attributes to balance the independence and collaboration of services for a given system.

1.5 Main Contributions

The main scientific and technical contributions presented during this research can be outlined as follows:

- **[Primary scientific contribution] Model to characterize service-based systems architecture:** We developed the CharM, a theoretical, conceptual, flexible, and extensible model. Currently, this model comprises dimensions and metrics related to the structural attributes of size and coupling. Based on the three evaluation stages in which it was submitted (Chapters 4 and 5), the CharM proved to be easy to understand, as well as useful for understanding and maintaining a service-based architecture, and communication of a service-based architecture to stakeholders;
- **[Primary technique contribution] Architecture documentation of service-based systems:** By considering all the generated artifacts, as well as the received feedback during the evaluation stages, the CharM is an approach that documents the architecture of service-based systems in a consistent and valuable way.
- **[Secondary scientific contribution] Systematic method for trade-off analysis of the adoption of architectural patterns:** we developed a step-by-step method that guides software architects in identifying architectural patterns that best meet the needs of a given project. We described this method in Appendix B;
- **[Secondary technique contribution] Trade-off analysis of the microservices patterns that influences the structural attributes of size and coupling:** We believe that the results obtained during one of the exploratory research stages of this thesis and described in one of our published papers (Oliveira Rosa, Daniel, et al., 2020) help in decision-making in the context of service-based systems. Since it demonstrates the impact of certain microservice patterns on the analyzed structural attributes;

1.6 Thesis Structure

We organized the remainder of this thesis as follows. In Chapter 2, we explore fundamental concepts in the context of this research. Thus, we begin this chapter by defining software architecture. Soon after, we explore monolithic and service-based architectural styles. We highlighted microservices, presenting their characteristics, advantages, and challenges. In Chapter 3, we present our model's design process, define what the CharM is, describe each of its dimensions, and demonstrate an example of applying our model in a fictional scenario. In Chapter 4, we detail the two case studies we carried out to evaluate the CharM and present and discuss the results achieved. In Chapter 5, we describe the design and execution of the survey and also present and discuss the results achieved. In Chapter 6, we present related research and compare the CharM to solutions that can also support professionals in assessing a service-based system's architecture. Finally, in Chapter 7, we summarize this thesis, present some considerations of the research questions, and propose future work.

Chapter 2

Background

In this chapter, we provide a literature review of the concepts that underlie this research. First, we present some main software architecture definitions, emphasizing the importance of this field for the development process. Next, we detail monolithic and service-based architectural styles. We present definitions, characteristics, advantages, and challenges for each of these architectural styles. Among the service-based styles, we explored the microservices architectural style more deeply.

2.1 Software Architecture

According to Perry and Wolf (1992), the software architecture area emerged to facilitate the understanding of system requirements, as well as to serve as a technical basis for software design and support process management and cost estimation. This area was also created to allow software reuse (or some parts of it) and to be a source for dependency analysis and project consistency. M. Fowler (2002) claims that two aspects are common to define software architecture: parts that compose a system and decisions that are difficult to change. Table 2.1 presents some definitions of software architecture, published by researchers who are the reference in this study area.

From Table 2.1, we define *software architecture as the hierarchical organization of an information system's elements (e.g., modules, components, and services), which illustrates the structure and form of interaction of these elements. The architecture also specifies fundamentals, proprieties, rules, and constraints that guide the design and software evolution. Moreover, the architecture is essential to relate the software's characteristics to its implementation (coding).*

By describing and exposing important aspects of the software, a well-defined software architecture helps with complexity management, avoiding problems during the development process, and facilitating software understanding, reuse, development, analysis, evolution, and maintenance (C. Hofmeister et al., 1999; Clements, Kazman, et al., 2002; Garland and Anthony, 2003; Garlan, 2014; Cervantes and Kazman, 2016). An appropriate architecture may help to ensure that a system satisfies quality attributes, such as performance, reliability, portability, scalability, and interoperability. On the other hand, an

Definition	Reference
"…software architecture involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns".	Shaw and Garlan, 1996
"…architecture is the hierarchical structure of program components (modules), the manner in which these components interact and the structure of data that are used by the components".	Pressman, 2001
"…our definition of architecture ...the structure or structures of the system, each of which comprise elements, the externally-visible behavior of those elements, and the relationships among them".	Clements, Bachmann, et al., 2002,
"The Software Architecture is the set of software components, subsystems, relationships, interactions, the properties of each of these elements, and the set of guiding principles that together constitute the fundamental properties and constraints of a software system or set of systems".	Garland and Anthony, 2003
"…structure illuminates the top-level design decisions, including things, such as how the system is composed of interacting parts, what are the principal pathways of interaction, and what are the key properties of the parts and the system as a whole …Software architecture typically plays a key role as a bridge between requirements and implementation".	Garlan, 2014
"The software architecture of a system is the set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both".	Cervantes and Kazman, 2016
"software architecture consists of the structure of the system …, combined with architecture characteristics ("-ilities") the system must support, architecture decisions, and finally design principles".	Richards and Ford, 2020
"fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution".	ISO/IEC/IEEE42010, 2022

Table 2.1: *Software architecture definitions*

improper architecture may be catastrophic for the project, generating complex software that is difficult to maintain and evolve (Clements, Bachmann, et al., 2002; Garlan, 2014; Cervantes and Kazman, 2016).

Martin (2018) confirms this idea by indicating that a proper and well-organized architecture facilitates software comprehension, development, and deployment, which

contributes to minimizing cost over software "lifetime" and maximize the developer productivity. Therefore, the main aim of an architecture is to offer support for the whole software development cycle. Clements, Bachmann, et al. (2002) complement claiming that an architecture enables greater control of the software elements and complex aspects, thus allowing to focus on essential elements and their interactions, not on irrelevant details.

Therefore, have adequate architecture is a critical success factor in the software development process (Shaw and Garlan, 1996). For this reason, the relevance of studies about architecture applied to areas, such as cloud computing, internet of things, autonomous systems, and developing ecosystems with end-user are growing, and the theoretical and practical challenges area diverse (Garlan, 2014). However, it is common to start software development without having a clear and well-defined architecture (Foote and Joseph Yoder, 1997; Garlan, 2014; Richards, 2015). Such a practice results in disorganized source-code and system modules with poorly defined responsibilities and relationships, a scenario known as the anti-pattern "big ball of mud" (Foote and Joseph Yoder, 1997; Richards, 2015).

In this context, in order to define the most appropriate architecture to the business objectives and expected features for the software, architectural styles and patterns are adopted (Richards, 2015). According to Taylor et al. (2009), in short, an architectural pattern is a "collection of architectural design decisions that are applicable to a recurring design problem". Sommerville (2011) explains that architectural patterns emerge from successful software development experiences. Besides that, a pattern comprises a stylized and generic description (organization, strengths, and weaknesses) and presents a set of experienced and tested good practices in different software projects and environments. In contrast, Taylor et al. (2009) define architectural style as a named collection of design decisions applicable in a given development context. The authors further explain that, at the same time, an architectural style constrains design decisions and generates beneficial qualities for the system within a context. Nitto and Rosenblum (1999) complement by stating that architectural "styles are a mechanism for categorizing architectures and for defining their common characteristics".

There are several architectural styles. However, in this thesis, we are going to address only the monolith (Section 2.2) and service-based (Section 2.3) styles which are aligned with the characterization model we propose. Thus, the following section presents an overview of monolithic architecture.

2.2 Monolithic Architectural Style

According to C. U. Press (2019), monolithic is an adjective that characterizes something "very large, united, and difficult to change". Monolithic systems in software architecture are applications with a single executable or implementable component (Richardson, 2018). Newman (2019) considers software as a monolith when all its functionalities have to be deployed together. Richards (2015) defines monolithic applications as a set of "tightly coupled components that are part of a single deployable unit", which is "difficult to change, test, and deploy".

Villamizar et al. (2015) and Kalske (2017) explain that a monolithic application has a single code base, which serves several different services and interfaces, such as REST (Representational State Transfer), APIs (Application Programming Interface), and HTML (HyperText Markup Language) pages. For Lewis and M. Fowler (2014), monolithic applications are usually composed of three main parts (layers), which form a single unit. The three parts are: (*i*) the user interface (front-end technologies), (*ii*) a database (usually a relational database), and (*iii*) the back-end code (single logical executable), which will run on the server side.

There are different types of monoliths. According to Newman (2019), probably the most common type is single-process – the classic. One of the variations of this type is known as modular monolith. In modular monoliths, the single process is made up of separate modules, which can be worked independently. However, each module has to be combined, forming a single deployment unit (Newman, 2019).

According to Richardson and Smith (2016), adopting the monolithic architectural style makes sense when the software is lightweight and simple. Some strengths of classic monolithic applications are that they are simple to develop, quick to test, deploy, and scale, as well as easy to perform radical modifications (for example, changing the database) (Kalske, 2017; Richardson, 2018). Among the advantages of classic monoliths, listed by Newman (2019), are the simplicity of monitoring, troubleshooting, and code reuse (within the monolith itself). M. Fowler (2015) explains that adopting a monolithic architecture is a good strategy to start the development of a simplistic version of a system, better to understand its context, divisions, and functions. Newman (2019) also states that adopting modular monoliths can favor the independent work of teams that deal with different contexts and aspects of a system.

On the other hand, when the software turns into large and complex, the monolithic architecture becomes inadequate (Acevedo et al., 2017; Richardson, 2018). The main challenges are related to the possibility of high coupling between the system elements, making it difficult to understand, develop, update, test, and deploy the software. Furthermore, it favors the occurrence of cascading failures and, at the same time, makes it difficult to isolate such failures (Villamizar et al., 2015; Bonér, 2016; Richardson and Smith, 2016). Other points that are also considered challenging in this architectural style are: complexity of understanding the functioning and integration of all parts of the software (Bonér, 2016; Richardson, 2018); difficulty and high risk of failures in performing maintenance and continuous deployment (Lewis and M. Fowler, 2014; Newman, 2021; Bonér, 2016; Kalske, 2017; Richardson, 2018); high cost and inefficiency in scaling software (Lewis and M. Fowler, 2014; Newman, 2021; Bonér, 2016; Ren et al., 2018; Richardson, 2018); limited possibilities of changing the adopted technologies (such as programming language, database, and framework) (Lewis and M. Fowler, 2014; Newman, 2021; Richardson, 2018); it takes a long time to load all the application code in the IDE (Integrated Development Environment), reducing the developers' productivity (Richardson, 2018); execution of automated tests in the whole system may be a time-consuming task (Lewis and M. Fowler, 2014; Kalske, 2017); and high complexity in aligning tasks between different teams working on the same project (Kalske, 2017; Richardson, 2018).

After presenting definitions, as well as some advantages and challenges related to

the monolithic architectural style, in the next section, we are going to explore some service-based architectural styles.

2.3 Service-Based Architectural Styles

Inspired by modular software development (Parnas et al., 1985), the service-based architecture emerged to overcome challenges common to the monolithic style, such as difficulty in maintaining and scaling the software, low productivity, and strong dependence between elements (Richards, 2015; Bogner, Wagner, et al., 2017a).

The service-based architecture composes the service-oriented computing (SOC) paradigm, which emerged as a way to develop maintainable and interoperable software (Bogner, Wagner, et al., 2017a). In SOC, services are the "fundamental elements for developing of applications/solutions" (Papazoglou, 2003). According to Papazoglou (2003), in this paradigm, services perform from simple functions to complex business processes. Bogner, Wagner, et al. (2017a) explain that services should be "self-contained, composable, technology-neutral, loosely coupled, and allow for location transparency". Richards (2016) indicates that some common characteristics in service-based architectures are modularity and distribution.

Service-Oriented Architecture (SOA), Self-Contained Systems (SCS), and more recently, microservice (MS) are examples of service-based architectural styles. According to Natis and Schulte (2003) and Bianco et al. (2007), SOA is a client/server design approach where an application is composed of users (clients) and providers (servers) of software services that emphasize the adoption of flexible coupling between components and independent interfaces. Krafzig et al. (2005) explain that SOA designs systems as a network of related services, where each service provides a specified functionality in a well-defined interface. Newman (2015) complements by stating that services collaborate to provide a final set of resources (solution).

Self-Contained Systems aims to separate the functionalities of a complex system into several autonomous systems, which have their "own user interface, specific business logic, and separate data source". Such autonomous systems communicate/collaborate through RESTful HTTP (Hypertext Transfer Protocol) or lightweight mechanisms, mostly asynchronously (innoQ, 2015).

According to Lewis and M. Fowler (2014), microservice is a way to design and develop software as suites of independently deployable small services. The authors also explain that each service is built around business capabilities, runs in its own process, and uses lightweight communication mechanisms. Lewis and M. Fowler (2014) base their microservice definition on nine characteristics that they observe as common in this type of architecture.

These three architectural styles follow principle, such as decomposing complex systems into services (basic system unit), loosely coupled, and communicating by messages. They also share the same principle: dividing complex systems into components which communicate in some way.

There is a series of studies that, in some way, discuss and compare the characteristics

of these different service-based styles, such as the work by Salah et al. (2016), Wolff (2016a), Wolff (2016b), Cerny, Donahoo, and Pechanec (2017), Rademacher et al. (2017), Shadija et al. (2017), Cerny, Donahoo, and Trnka (2018), Baresi and Garriga (2020), and Raj and Sadam (2021). Since the microservice is considered one of the most popular service-based architectural styles, we explore it further, presenting its definition, characteristics, advantages, and challenges in Section 2.3.1.

2.3.1 Microservices Architectural Style

According to Richardson and Smith (2016), the microservices style favors developing software with more agility, scalability, and maintainability. There is still no exact, complete, and consensual definition for the microservices architectural style. Newman (2021) claims that microservices are modeled around a business domain and "are an approach to distributed systems that promote the use of finely grained services that can be changed, deployed, and released independently". The author also highlights that two of the main qualities of microservices are loose coupling and high cohesion (Newman, 2015). Bonér (2016) advocates that in a microservices-based architecture each service must have its own data, and be independent, scalable, and resilient to failure. In one of the explanations presented by Dragoni et al. (2017), microservice is defined as "a distributed application where all its modules are microservices". Richardson (2018) explains that a microservices-based application is structured "as a collection of loosely coupled, independently deployable services".

Some of the main characteristics of the Microservices Architectural Style are:

- Software modularization into independent services (development, testing, deployment, scaling) (Lewis and M. Fowler, 2014; Jaramillo et al., 2016; Nadareishvili et al., 2016; Dragoni et al., 2017)
- Small and isolated services (Lewis and M. Fowler, 2014; Jaramillo et al., 2016; Nadareishvili et al., 2016)
- Services organized around business domain (Lewis and M. Fowler, 2014; Jaramillo et al., 2016; Nadareishvili et al., 2016; Dragoni et al., 2017; Newman, 2021)
- Services with high cohesion (Thönes, 2015; Jaramillo et al., 2016; Nadareishvili et al., 2016; Dragoni et al., 2017; Newman, 2021)
- Loose coupling services (Jaramillo et al., 2016; Dragoni et al., 2017; Newman, 2021)
- Smart services and dumb pipes (Lewis and M. Fowler, 2014)
- Open, standardized, and lightweight communication mechanisms (Lewis and M. Fowler, 2014; Nadareishvili et al., 2016)
- Decentralized governance and data management (Lewis and M. Fowler, 2014; Nadareishvili et al., 2016)
- Technological heterogeneity (Jaramillo et al., 2016; Nadareishvili et al., 2016)
- "Focused on product, not project" (Lewis and M. Fowler, 2014)

- Multi-functional teams (Lewis and M. Fowler, 2014)
- Automated infrastructure (Lewis and M. Fowler, 2014; Nadareishvili et al., 2016)
- Software designed to evolve (Lewis and M. Fowler, 2014; Dragoni et al., 2017)
- Resilient software (Lewis and M. Fowler, 2014; Dragoni et al., 2017)

In line with these characteristics, some of the advantages of developing microservices-based systems are:

- Agility: allows organizations to meet market demands by quickly delivering new software features, as well as identifying and making technical and business adjustments with more agility (Richards, 2015; Villamizar et al., 2015; Alshuqayran, Ali, et al., 2016; Balalaie, Heydarnoori, and Jamshidi, 2016; S. J. Fowler, 2016; Guo et al., 2016; Killalea, 2016; Nadareishvili et al., 2016; Pahl and Jamshidi, 2016; Richardson and Smith, 2016; Singleton, 2016; Vural et al., 2017; Soldani et al., 2018);
- Deployment autonomy: new features are created or replaced in a production environment faster and with a lower risk of failure (Le et al., 2015; Newman, 2015; Richards, 2015; Villamizar et al., 2015; Hassan and Bahsoon, 2016; Jaramillo et al., 2016; Nadareishvili et al., 2016; Richardson and Smith, 2016; Richardson, 2018; Soldani et al., 2018; Taibi et al., 2018);
- Ease of continuous deployment: from deployment autonomy, it is possible to rebuild and redeploy an entire service without depending on and affecting other parts of the software (Krylovskiy et al., 2015; Newman, 2015; Richards, 2015; Alshuqayran, Ali, et al., 2016; S. J. Fowler, 2016; Killalea, 2016; Pahl and Jamshidi, 2016; Richardson and Smith, 2016; Singleton, 2016; Dragoni et al., 2017; Vural et al., 2017; Richardson, 2018; Taibi et al., 2018);
- Management autonomy: favors software efficiency and reduces unavailability time (Nadareishvili et al., 2016; Soldani et al., 2018);
- Small and focused: each service has its scope well delimited and aligned with the business. Therefore, development teams become more organized and prepared to meet demands quickly and are encouraged to develop more complex solutions iteratively (Le et al., 2015; Newman, 2015; Nadareishvili et al., 2016; Richardson and Smith, 2016; Singleton, 2016; Dragoni et al., 2017; Richardson, 2018; Soldani et al., 2018);
- Replaceability: the ease of replacement and reuse of software services promotes continuous integration, makes the development process more agile, as well as reduces and improves the management of technical debt ratios, making the software more reliable (Krylovskiy et al., 2015; Le et al., 2015; Newman, 2015; S. J. Fowler, 2016; Hassan and Bahsoon, 2016; Nadareishvili et al., 2016; Singleton, 2016; Dragoni et al., 2017; Soldani et al., 2018);
- Simplicity: developing services with loose coupling and high cohesion simplify the implementation of new functionalities and favor increasing the product's local

quality (Le et al., 2015; Richards, 2015; Nadareishvili et al., 2016; Richardson and Smith, 2016; Soldani et al., 2018; Taibi et al., 2018);

- Technological heterogeneity: it makes it possible to choose the most appropriate technologies (tools, programming languages, database management system) for each context. Furthermore, it makes it easier to change and adopt different technologies when necessary (Lewis and M. Fowler, 2014; Newman, 2015; Krylovskiy et al., 2015; Villamizar et al., 2015; Le et al., 2015; Nadareishvili et al., 2016; Hassan and Bahsoon, 2016; Balalaie, Heydarnoori, and Jamshidi, 2016; Richardson and Smith, 2016; S. J. Fowler, 2016; Singleton, 2016; Killalea, 2016; Dragoni et al., 2017; Taibi et al., 2018; Richardson, 2018; Soldani et al., 2018);
- Greater efficiency: the infrastructure costs and the chances of software being unavailable due to service failures are low (Villamizar et al., 2015; Nadareishvili et al., 2016);
- Failure resilience: if a service fails, it is possible to isolate it and keep the rest of the software working. This provides high availability rates and contributes to a good user experience (Krylovskiy et al., 2015; Newman, 2015; Villamizar et al., 2015; Alshuqayran, Ali, et al., 2016; Jaramillo et al., 2016; Killalea, 2016; Nadareishvili et al., 2016; Pahl and Jamshidi, 2016; Vural et al., 2017; Richardson, 2018; Soldani et al., 2018; Taibi et al., 2018);
- Scalability: the division into services makes it possible to allocate infrastructure resources in a customized way (expansion or reduction) and consequently adequately meet the access load in each service (Newman, 2015; Richards, 2015; Villamizar et al., 2015; Alshuqayran, Ali, et al., 2016; S. J. Fowler, 2016; Nadareishvili et al., 2016; Pahl and Jamshidi, 2016; Richardson and Smith, 2016; Singleton, 2016; Dragoni et al., 2017; Vural et al., 2017; Richardson, 2018; Soldani et al., 2018);
- Testability: the local service testing process is simplified, as each service can be tested separately and independently, which reduces the risk of implementation failures (Richards, 2015; S. J. Fowler, 2016; Killalea, 2016; Nadareishvili et al., 2016; Singleton, 2016; Dragoni et al., 2017);
- Maintainability: elements, such as software division into services and deployment autonomy facilitate the software maintenance process (Le et al., 2015; Richards, 2015; Villamizar et al., 2015; Alshuqayran, Ali, et al., 2016; Richardson and Smith, 2016; Dragoni et al., 2017; Richardson, 2018; Soldani et al., 2018; Taibi et al., 2018);
- Flexibility: based on interoperability, the team responsible for the development of each service has the freedom to choose the technology that best suits the needs of the service and the team's technical knowledge (Killalea, 2016; Pahl and Jamshidi, 2016; Richardson, 2018; Taibi et al., 2018);
- Composability: having services with loose coupling and high cohesion makes parts of the software reusable in different ways and for different purposes, which reduces rework and increases the agility of software development and maintenance (Newman, 2015; Singleton, 2016; Taibi et al., 2018);

- Organizational alignment: it allows the team to better align the architecture with the real needs of the business, which optimizes the infrastructure use, the entire development process, management, and team effort (Krylovskiy et al., 2015; Newman, 2015);
- Reliability: based on testability and scalability, more stable and reliable software is obtained (Alshuqayran, Ali, et al., 2016; Singleton, 2016; Soldani et al., 2018).

Despite the cited benefits, developing microservices-based software also has a set of challenges. Thus, it is relevant to know them to be able to compare, treat, and make conscious decisions about the trade-offs. Some challenges related to microservices are:

- Define service size and coupling: identifying the scope and delimiting the "size" and the coupling level of a microservice are some of the biggest challenges of the microservices style. This is because, according to the level of coupling and cohesion of the services, aspects, such as scalability, performance, reuse, maintainability, and testability are directly affected (Richards, 2015; S. J. Fowler, 2016; Hassan and Bahsoon, 2016; Jaramillo et al., 2016; Singleton, 2016; Richardson, 2018; Soldani et al., 2018)
- Data consistency: the fact that software databases are distributed across services makes managing and maintaining data consistency a major challenge in microservices' architectures (Newman, 2015; Alshuqayran, Ali, et al., 2016; Bonér, 2016; Richardson and Smith, 2016; Richardson, 2018; Soldani et al., 2018);
- Service discovery: it is necessary to define consistent communication patterns for consumer services to discover the location of other services and endpoints. Besides that, it is relevant to adopt correct strategies to report the discovery and availability of services (Richards, 2015; Alshuqayran, Ali, et al., 2016; Balalaie, Heydarnoori, and Jamshidi, 2016; Bonér, 2016; S. J. Fowler, 2016; Richardson and Smith, 2016; Singleton, 2016; Sun et al., 2016);
- Tracking and recording of services: it is essential to adopt techniques and mechanisms for tracking, auditing, and recording services (such as logs), to understand how microservices-based software works (Newman, 2015; Alshuqayran, Ali, et al., 2016);
- Performance: the constant and intense data-sharing between services may cause overloads and delays in the communication process. Consequently, the performance of the software may be impaired (Lewis and M. Fowler, 2014; Richards, 2015; Alshuqayran, Ali, et al., 2016; Singleton, 2016; Soldani et al., 2018);
- Performance monitoring: since performance is a weak point in microservices-based software, therefore, it is relevant to use tools to measure and understand the performance of each service and the software as a whole (Krylovskiy et al., 2015; Savchenko et al., 2015; Alshuqayran, Ali, et al., 2016; Jaramillo et al., 2016; Richardson and Smith, 2016; Singleton, 2016; Sun et al., 2016; Soldani et al., 2018);
- Complex development process: the development of a single service is simple. However, the development process of distributed software, composed of several services,

is a challenging and complex activity (Newman, 2015; Villamizar et al., 2015; Jaramillo et al., 2016; Nadareishvili et al., 2016; Richardson and Smith, 2016; Singleton, 2016; Soldani et al., 2018; Taibi et al., 2018);

- Complex deployment: it is recommended to choose adequate platforms and tools as well as follow guidelines that facilitate infrastructure management and service scaling to thus own the continuous deployment activity as an advantage of microservices-based software (Newman, 2015; Balalaie, Heydarnoori, and Jamshidi, 2016; Bonér, 2016; S. J. Fowler, 2016; Richardson and Smith, 2016; Richardson, 2018);

- Network dependency: since the services communicate via a network, the correct functioning, software availability, and communication interval between services directly depend on the network's stability and latency (Newman, 2015; Alshuqayran, Ali, et al., 2016; Taibi et al., 2018);

- Need for automation: since many services and relationships must be managed and monitored, thus, automating activities and processes is critical in microservices-based software (Krylovskiy et al., 2015; Savchenko et al., 2015; Jaramillo et al., 2016; Richardson and Smith, 2016; Taibi et al., 2018);

- Failure tolerance: the microservices style is cloud-native, where cascading failures are common. Thus, it is essential to develop adequate mechanisms for tracking, interrupting, and recovering from infrastructure failures (Lewis and M. Fowler, 2014; Newman, 2015; Alshuqayran, Ali, et al., 2016; Bonér, 2016; S. J. Fowler, 2016; Jaramillo et al., 2016; Killalea, 2016; Richardson and Smith, 2016; Singleton, 2016; Sun et al., 2016; Soldani et al., 2018);

- Security: security is critical in microservices-based software due to the different ways and levels of data sharing between services. This fact requires the implementation of additional and more sophisticated security mechanisms (Richards, 2015; Alshuqayran, Ali, et al., 2016; Bonér, 2016; Sun et al., 2016; Dragoni et al., 2017; Soldani et al., 2018);

- Complex integration tests: having many services and relationships between them increases the complexity of integration tests in microservices-based systems (Jaramillo et al., 2016; Richardson and Smith, 2016; Dragoni et al., 2017; Richardson, 2018; Soldani et al., 2018; Taibi et al., 2018);

- Experienced team: it is necessary to have an experienced team to deal with the high degree of complexity of the infrastructure, development, and management in microservices-based systems (Villamizar et al., 2015; Balalaie, Heydarnoori, and Jamshidi, 2016; Jaramillo et al., 2016; Taibi et al., 2018);

- Organizational structure: for the successful adoption of the microservices style, it is necessary that the structure and communication strategy of the company and its development teams would be reorganized (S. J. Fowler, 2016).

2.4 Chapter Summary

In this chapter, we conceptualized software architecture, underlining the importance of this area in software development. We also explored definitions and characteristics of relevant architectural styles in this research's context. We further explored the microservices-based architectural style, which is currently considered the latest trend in software development. Thus, we sought to understand the main characteristics, advantages, and challenges of microservices to base the structure of our characterization model of the architecture of service-based systems.

After exploring the fundamental concepts related to this research context, in the next chapter, we describe the proposed model to characterize the architecture of service-based systems.

Chapter 3

The CharM – Characterization Model

This chapter presents the design process of the proposed model, explains what the CharM is, describes its four dimensions, and demonstrates an example of its application in a fictional scenario.

3.1 Design Process

In order to develop a model for characterizing the architecture of service-based systems, our study follows the seven guidelines of the Design Science Research (G1: Design as an Artifact; G2: Problem Relevance; G3: Design Evaluation; G4: Research Contributions; G5: Research Rigor; G6: Design as a Search Process; G7: Communication of Research), proposed by Hevner et al. (2004). According to Engström et al. (2020), the Design Science Research (DSR) is a paradigm focused to solve real-world problems and commonly used in research in the engineering and information systems fields. Hevner et al. (2004) explain that the DSR is a problem-solving paradigm, in which kernel theories are "applied, tested, modified, and extended through the experience, creativity, intuition, and problem solving capabilities of the researcher". Furthermore, the authors emphasize that, from the adoption of the DSR, it is possible to iteratively build and evaluate artifacts (constructs, models, methods, or instantiations). As already cited, the artifact developed in this study is a model, which was built and evaluated in five iterations.

The first version of the proposed model emerged from *ad-hoc* bibliographic research (Section 2.3.1), which was focused on the analysis of microservices style definitions and characteristics and from discussions with researchers specialized in software architecture. This version of the model is available in a previous paper (Oliveira Rosa, Goldman, et al., 2020).

Following this, we extended the *ad-hoc* bibliographic research to identify candidate metrics for each model dimension (Appendix A). It is worth clarifying that, during the CharM design, the main criteria for selecting its metrics were: to be related to the dimensions of the model and the components analyzed, as well as to be extracted independently

of technology, source code, or infrastructure. During the second iteration, we still analyzed (trade-off) the influence of a set of microservices patterns on the structural attributes of size and coupling (Appendix B). The results of this trade-off analysis were also published in a previous paper (Oliveira Rosa, Daniel, et al., 2020). At the end of this iteration, we obtained a second version of the model with a more concise scope, more refined dimensions, and a list of viable candidate metrics. Version 2 of the model is available in another previous paper (Rosa et al., 2020).

In the third iteration, the model was submitted for a first evaluation. We carried out a case study, where we adopted the second version of the model to characterize the architecture of a platform for smart cities, called InterSCity (Esposte, 2018), developed as part of an academic research project. After applying the model, the architectural characterization was evaluated by 11 project members. At the end of the iteration, we obtained feedback regarding the model uses and ease of understanding, as well as collected suggestions for improvements.

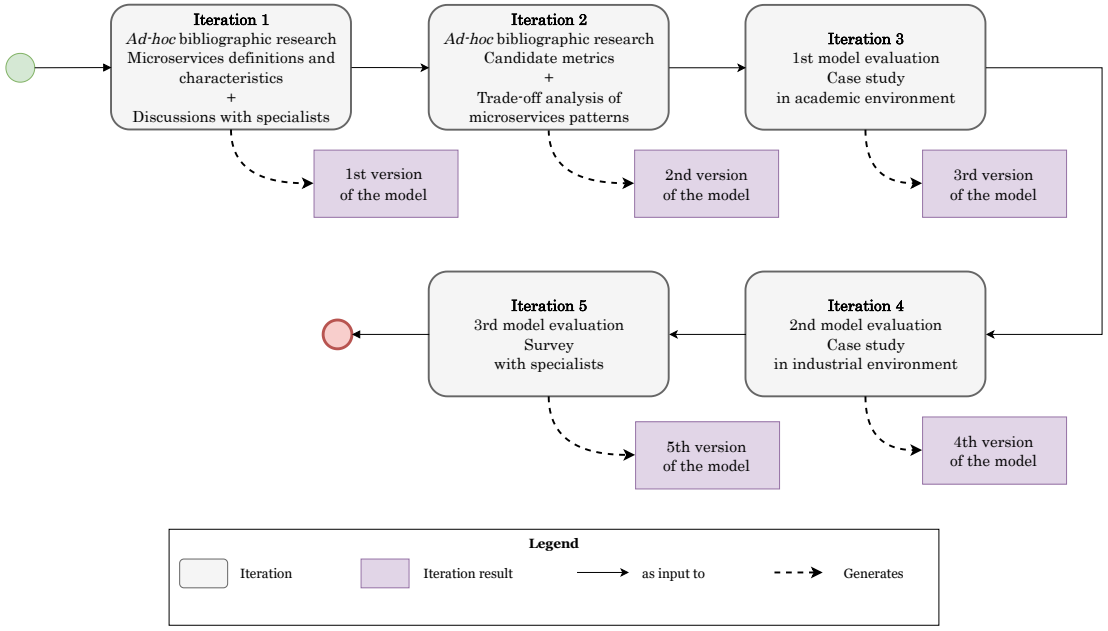


Figure 3.1: *Design process of the CharM*

We refined the model based on the feedback received during the third iteration. Herewith, we added new metrics, rethought and adjusted the visual presentation of some results' characterization, and optimized the model's explanation. Thus, we generated a third version, called CharM. We submitted this version to an evaluation through a case study in an industrial environment. We carried out this new case study in an online handcrafts marketplace with more than 7 million announced products produced by more than 100,000 active sellers. Since 2018, the architecture of this system is being migrated from the monolithic style to a service-based approach.

In order to make the case study feasible, the CharM was applied to only one part of the system, composed of eight services. After applying the CharM, the generated architectural characterization was evaluated by six development team members. At the end of this iteration, we also obtained feedback regarding the model uses and ease of understanding,

as well as collected suggestions for improvements. The CharM evaluation result through the two case studies is described in Chapter 4.

Based on the CharM's evaluation feedback received during the fourth iteration, we identified new uses and improved the results visualization and model explanation. Therefore, we started the fifth and final iteration, in which we evaluated the CharM through a survey, in which we obtained 58 answers (detailed in Chapter 5). The process design of the CharM is illustrated in Figure 3.1.

Given the main goal of this research, it is possible to verify that the built and evaluated artifact is a model, that satisfies the G1 guideline of the DSR. The arguments that we presented in the Section 1 demonstrate that it is still difficult to classify and characterize the architecture of service-based systems, so we meet the G2 guideline. In this section, we describe the rigor of the artifact (model) design (G5), its development (G6), and evaluation (G3) process. In Sections 1.5, 4.3, and 5.8, we discuss the research contributions (G4). This thesis and the previous papers Oliveira Rosa, Goldman, et al. (2020), Oliveira Rosa, Daniel, et al. (2020) and Rosa et al. (2020) communicate our results, which complies with the G7 guideline. Thus, we satisfied the seven DSR guidelines proposed by Hevner et al. (2004).

3.2 Definition

Before defining the CharM, it is important to understand what a *characterization model* is. According to the Cambridge Dictionary (C. U. Press, 2022), *characterization* can be defined as "the way in which something is described by stating its main qualities". According to the Oxford Learner's Dictionaries (O. U. Press, 2022), a *model* can be defined as "a simple description of a system, used for explaining how something works [...]". From Hevner et al. (2004) perspective, model-type artifacts "aid problem and solution understanding and frequently represent the connection between problem and solution components enabling exploration of the effects of design decisions and changes in the real world". Thus, we can define a *characterization model* as a way to describe and explain something, stating some of its main qualities. A characterization model also helps to understand the problem of something that is analyzed and consequently identify possible solutions and make conscious decisions.

Therefore, faced with the challenges presented in Chapter 1, the CharM is a characterization model created to describe and explain the architecture of service-based systems, standing some of its main qualities. From this, the CharM helps professionals identify problems and viable solutions for the architecture of a system and aid the decision-making process. The current version of the CharM focuses on the structural attributes of size and coupling and collects static metrics from the analysis of architectural design artifacts or APIs (Application Programming Interface) and configurations files.

Since microservices are the current trend for service-based systems development (O. Zimmermann, 2017; Francesco et al., 2017; Bushong, Amr S Abdelfattah, et al., 2021; Vera-Rivera et al., 2021), we used some of their guidelines to design the CharM. We extracted five microservices guidelines from the definitions and some characteristics presented in Section 2.3.1. During the guideline extraction process, we also analyzed defi-

nitions and characteristics of modular and other service-based approaches to understand their similarities and differences compared to microservices. Based on this investigation, we identified that microservices, SOA (Service-Oriented Architecture), and other service-based approaches share the same principle: dividing complex systems into smaller components, called services, which communicate in some way.

To define our model dimensions, we considered challenging design decisions in service-based systems, especially in the microservices context. Thus, from discussions grounded by the studies of Lewis and M. Fowler (2014), Hassan and Bahsoon (2016), Bonér (2016), Soldani et al. (2018), and Newman (2021), we identified that some of the main microservices challenges are related to the structural attributes of size and coupling (but not limited just to these), such as the following: defining the services' size, managing distributed data, and defining the services' coupling level. It is worth citing that such challenges are not exclusive to the microservices' context. The studies of Sneed, 2006; Mahmood, 2007; Perepletchikov et al., 2007 address the importance of size and coupling attributes in SOA and indicate them as challenging. We also identified that patterns, such as Business Microservice, Database per Service, Event Sourcing, and Saga aid professionals deal with these challenges (Oliveira Rosa, Daniel, et al., 2020) in microservice-based systems. A study by Bogner, Wagner, et al. (2017a), focused on metrics, reinforces the importance of understanding size and coupling characteristics. This study demonstrated that these two structural attributes are among the characteristics most commonly found in the literature related to the maintainability of service-based systems. Then, from the analysis of the microservices definitions, characteristics, challenges, and patterns, we extracted the following guidelines that found the CharM: *(i) small and independent services; (ii) loose coupling; (iii) lightweight communication mechanisms; (iv) deployment independence;* and *(v) decentralized data management.*

The CharM can be used to support different stages of the software life cycle, such as its design-time, architecture mapping [1], and architectural evolution. Besides that, it was created to enable the analysis of different components and perspectives according to the interest scope. Therefore, one of the goals of the CharM is to facilitate the architectural analysis of a system and identify if its characteristics are closer or farther from the listed microservices guidelines. Another goal is to facilitate the identification of the architectural trade-offs and make viable their balancing. To achieve these goals, the CharM is composed of dimensions and metrics that help to verify whether the architecture is adequate to meet the desired non-functional requirements, which, in some way, are influenced by the structural attributes of size and coupling. It is important to clarify that in the current version of the CharM, the metrics of each dimension are collected manually (the roadmap for the manual metrics collection is available in Appendix C). For this, the professionals can access and explore both architectural design artifacts and (if available) APIs, configuration files, or source code of the system.

It is also worth citing that we designed the CharM to be simple and independent of technology, source code, or infrastructure. Therefore, in its current version, we did not explore some critical features in service-based systems such as cohesion. We chose

[1] **Architecture mapping** means identifying the elements that compose the architecture of a system and how these elements relate to each other.

not to incorporate cohesion in the CharM because we identified that the related metrics (that we have studied so far) tend to make our model more complex and dependent on technology/source code. However, this does not mean the cohesion aspect cannot be incorporated later.

It must be made clear that the CharM is not a tool-based approach but a theoretical and conceptual model designed to be applied at different software life cycle stages. Besides that, this model could be used as a reference for grounding and developing other solutions with lower levels of abstraction. In the current version of the CharM, we are not interested in guaranteeing the precision and objectivity of the collected metrics nor analyzing data related to technological heterogeneity, performance, and message size.

Furthermore, it is not the CharM's goal to characterize all service-based architectures from the same perspective, ignoring their differences. Nor is it the aim of the CharM to label a given architecture as good or bad or to recommend specific architectural changes. The CharM's goal is to provide information that helps professionals better understand the architecture of a system and support them in making more grounded decisions. It is also important to explain that although the CharM is rooted in microservices guidelines, its application is not limited to systems that follow this architectural style since it analyzes structural attributes considered interesting and challenging in different service-based approaches.

Therefore, we designed the CharM balancing the yearnings for it to be a lean and comprehensive (applicable to different service-based architectural styles) solution. Furthermore, this model was created to be easy to understand, flexible, and expandable to incorporate new dimensions and metrics.

3.3 Dimensions

The four dimensions composing the current version of the CharM are *size, data source coupling, synchronous coupling, and asynchronous coupling*. As illustrated in Figure 3.2, each dimension of the CharM was inspired by at least two of the five selected microservices guidelines (Section 3.2). The *Size* dimension is directly related to the guidelines *small and independent services* and *deployment independence*. The guidelines *decentralized data management* and *loose coupling* are present in the *Data Source Coupling* dimension. The *Synchronous Coupling* and *Asynchronous Coupling* dimensions are directly related to the guidelines *loose coupling* and *lightweight communication mechanisms*.

Each dimension is composed of metrics and it is possible choose which components[2] and perspectives[3] will be evaluated. The components that can be analyzed using the CharM are services and modules. In the context of this research, inspired by Richardson (2018), a service is an application with cohesive and well-defined responsibility that implements functionalities related to business tasks. Inspired by Martin (2018), a module is a cohesive set of services deployed together, that is, a deployment unit. Regarding the perspectives, in

[2] **A component** is a part or element of a system. The components analyzed with the CharM are modules and services.

[3] **A perspective** is the way in which a particular system component is analyzed. The CharM permits some external and internal analyses.

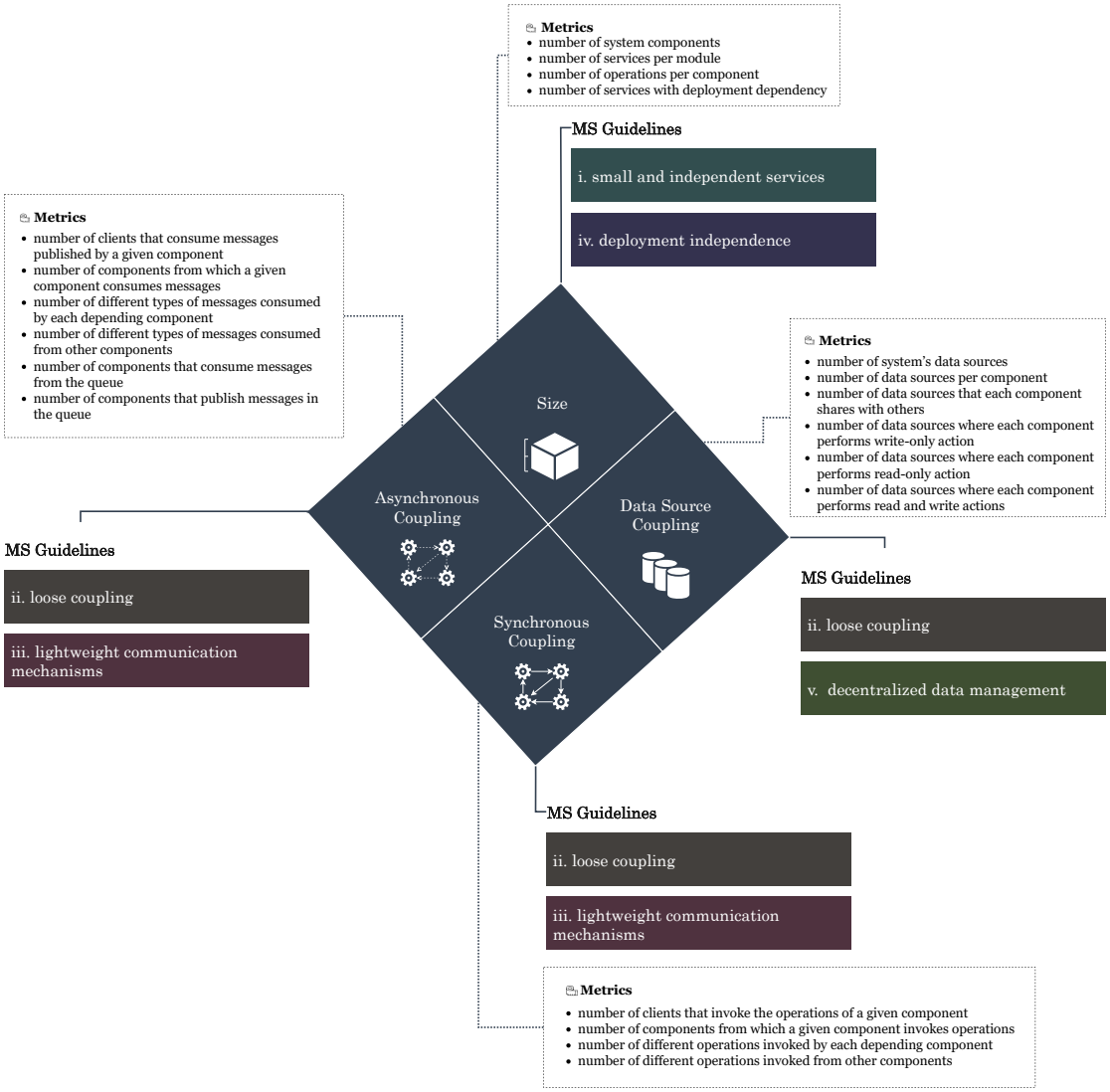


Figure 3.2: *CharM's dimensions, metrics, and microservices guidelines.*

the CharM context, we can analyze a component in a particular way, *i.e.*, from external [4], internal [5] or both views.

The *Size dimension* aims to characterize the size and composition of different system components and compare them. The components that can be analyzed are services and modules. The metrics that compose this dimension are: *the number of system components*, *the number of services per module*, *the number of operations*[6] *per component*, and *the number of services with deployment dependency*. According to the size of the component, we can obtain different advantages and disadvantages. Therefore, having access to these metrics enables more conscious architectural decisions related to the size of the components. For example, having a single deployment unit, which implements all business functionalities,

[4] **External perspective** pertains to the components outside the context of a system.

[5] **Internal perspective** pertains to the components inside the context of a system.

[6] **An operation** is a unit of functionality provided by a component (inspired by Shim et al. (2008)). In this version of the CharM, an operation is considered a synonym of endpoint.

may facilitate the integration of the components but it may hinder scalability. On the other hand, having each business functionality implemented in a different service, which is an independent deployment unit, may hinder the integration of the components but it may favor the scalability. This scenario is illustrated in Figure 3.3.

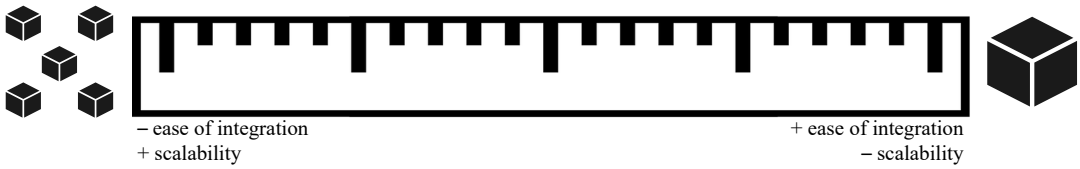


Figure 3.3: *Ruler of size dimension.*

The goal of the *Data Source Coupling dimension* is to characterize the strategy for sharing data sources[7] between the components of a system. The components that can be analyzed are services and modules. Besides that, the desired components can be analyzed from external, internal, or both perspectives. The metrics of this dimension are: *the number of system's data sources*, *the number of data sources per component*, *the number of data sources that each component shares with others*, *the number of data sources where each component performs write-only action*, *the number of data sources where each component performs read-only action*, and *the number of data sources where each component performs read and write actions*. Hence, from this set of metrics, we can obtain and evaluate different views of data source coupling between the components of a system and choose a more appropriate sharing strategy. For example, if all system components share a single source, the data coupling degree between the system components will be high, but this strategy may decrease the complexity of managing data consistency. On the other hand, if each component has its own exclusive data source, this may decrease the coupling between system components. However, it may increase the complexity of managing data consistency. Figure 3.4 exemplifies this case.

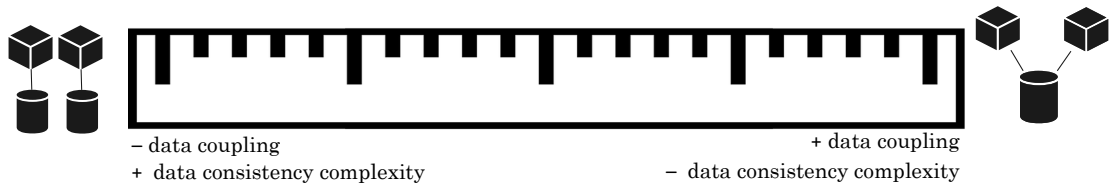


Figure 3.4: *Ruler of data source coupling dimension.*

The *Synchronous Coupling dimension* aims to characterize the synchronous interactions between the components of a system. In this research, a synchronous interaction occurs when a component x makes a request to a component y and waits for a response (inspired by Richardson (2018)). With the CharM, we can analyze the synchronous coupling of services and modules. Moreover, the selected components can be analyzed from external, internal, or both perspectives. The metrics adopted in this dimension are: *the number of clients that invoke the operations of a given component*, *the number of components from which a given component invokes operations*, *the number of different operations invoked by each depending component*, and *the number of different operations invoked from other components*.

[7] **Data source** is where it stores the data used by the components of a system

Therefore, this set of metrics helps to understand the synchronous dependency relationship between the components of a system, as well as the number of operations involved in each interaction. Consequently, it provides relevant information to guide decisions related to synchronous coupling trade-offs. For example, if all services in a system communicate with all the other services synchronously, then message exchange happens almost instantly, but system performance may be impaired. On the other hand, if there is no synchronous interaction between services, then the individual performance of each service may be better, but this strategy may cause a lag in the communication between the services. This situation is illustrated in Figure 3.5.

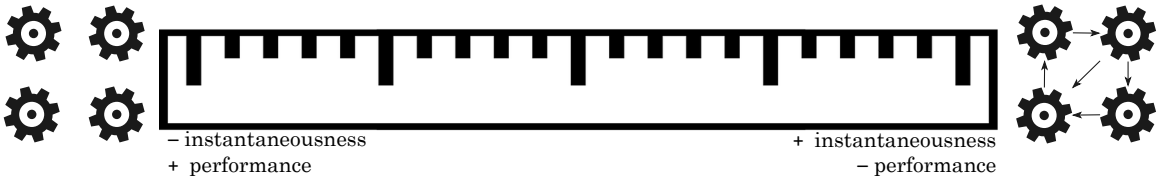


Figure 3.5: *Ruler of synchronous coupling dimension.*

The goal of the *Asynchronous Coupling dimension* is to characterize the asynchronous interactions between the components of a system. Unlike synchronous, an asynchronous interaction occurs when a component x sends a message to a component y and does not wait for a response (inspired by Richardson (2018)). In this research, the main interest is in interactions via message queues [8]. The components that can be analyzed in this dimension are services and modules. Besides that, the selected components can be analyzed from external, internal, or both perspectives. The metrics that compose this dimension are: *the number of clients that consume messages published by a given component*, *the number of components from which a given component consumes messages*, *the number of different types of messages consumed by each depending component*, *the number of different types of messages consumed from other components*, *the number of components that consume messages from the queue*, and *the number of components that publish messages in the queue*. Thus, these metrics help to understand the asynchronous interactions between system components, the number of different messages involved in each interaction, and the relationship between components and the message queue. Ergo, they provide relevant information to support decisions related to asynchronous coupling and deal with the trade-offs. For example, if all services in a system communicate with all other services asynchronously, then the operational complexity may increase but, at the same time, it may loosen the coupling. On the other hand, if there is no asynchronous interaction between services, the coupling may be higher, however, operational complexity may decrease. Figure 3.6 exemplifies this scenario.

The CharM is an adaptable model, which means that, according to need and interest, the professional can choose which dimensions, metrics, components, and perspectives will be analyzed. Therefore, more valuable metrics may vary depending on the scenario. Thus, one of the main goals of the CharM is to provide professionals with information that allows a better understanding of the system architecture. Besides that, from the generated

[8] We have not considered other ways of asynchronous interaction in the current version of the CharM, but we may incorporate them in future versions.

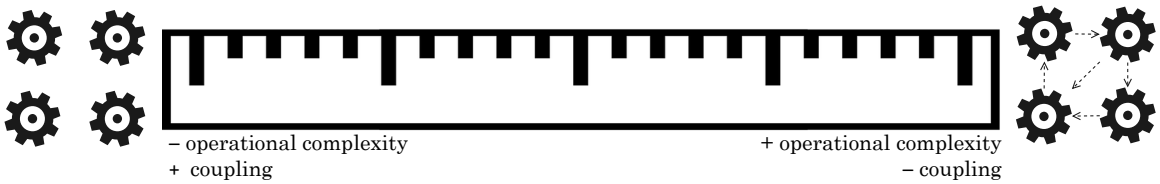


Figure 3.6: *Ruler of asynchronous coupling dimension.*

information, the CharM can help search for an appropriate balance for the software project, not tending to any of the ruler edges. Although there are other relevant dimensions and metrics, the scope of the current version of the model will be limited to those already described. Therefore, the model can be extended to include other dimensions, metrics, and elements in the future. It is also worth citing that it is outside this thesis's scope to identify and compare metrics with better results. In the next section, we present a fictitious CharM application scenario, where we illustrate the characterization result, as well as exemplify how our model can be useful to improve and evolve the architecture of a service-based system.

3.4 Fictional Scenario to Demonstrate the CharM Application

To demonstrate the application of the CharM dimensions and metrics, we created a fictional system named Pingr. A more detailed version of this demo scenario is available on YouTube[9]. The Pingr is a social network in the microblog format, where users can make posts limited to 160 characters. Each post is called *ping* and can receive *likes* and *pongs*. A *pong* means that other users shared a *ping*. The historical set of *pings* is displayed in a post feed, called *main table*. To post *pings*, people need to create an account. After creating an account, a user can post *pings* and follow other users. Users can also interact privately by chat. The quality requirements of Pingr are availability, scalability, performance, failure resilience, security, and privacy. Figure 3.7 presents an overview of the Pingr architecture.

As illustrated in Figure 3.7, there are four internal services, represented by the green circles. These services are *User*, *Chat*, *Ping*, and *Feed*. There is also one external service responsible for some authentication tasks. This external service is the gray circle. The white rectangles around the circles represent the modules, that is, deployment units. The yellow cylinders represent the data sources. The continuous blue lines represent the synchronous interactions between the services. The red dotted lines represent the asynchronous interactions between the services. Cardinalities with a blue background represent the number of operations involved in synchronous interactions. Cardinalities with a red background represent the number of messages involved in asynchronous interactions.

At first, we present the characterization of the services, followed by the characterization of the Pingr architecture, based on the 5^{th} version of the CharM. The scales used in the

[9]Demonstration of application of the CharM: https://youtu.be/bK9Yg9jmQXY

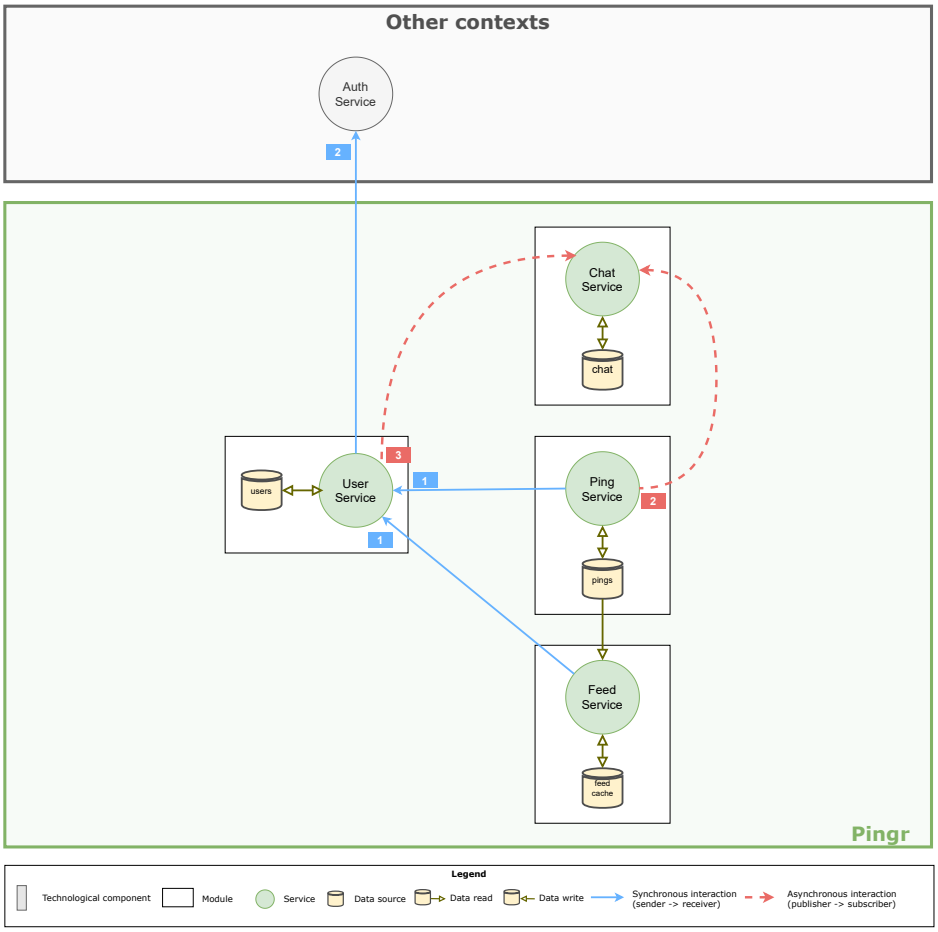


Figure 3.7: *Overview of the Pingr architecture (notation inspired by Merson and Jospeh Yoder (2019)).*

rulers of each dimension take data from the Pingr itself as a reference. Thus, the ruler of the size dimension ranges from 0 to 9 (maximum number of services operations). The scale used in the data source, synchronous, and asynchronous coupling dimensions ranges from 0 to 4 (number of services in the Pingr).

The **User Service** is the biggest service in the Pingr system, with nine operations. It accesses one exclusive data source performing read and write actions. Related to synchronous coupling, it is the service with the biggest internal importance and is the only one with external dependence. It is one of the services that perform asynchronous interactions, publishing three topics on the message queue that the Chat Service consumes. Figure 3.8 illustrates the User Service characterization, based on CharM dimensions and metrics.

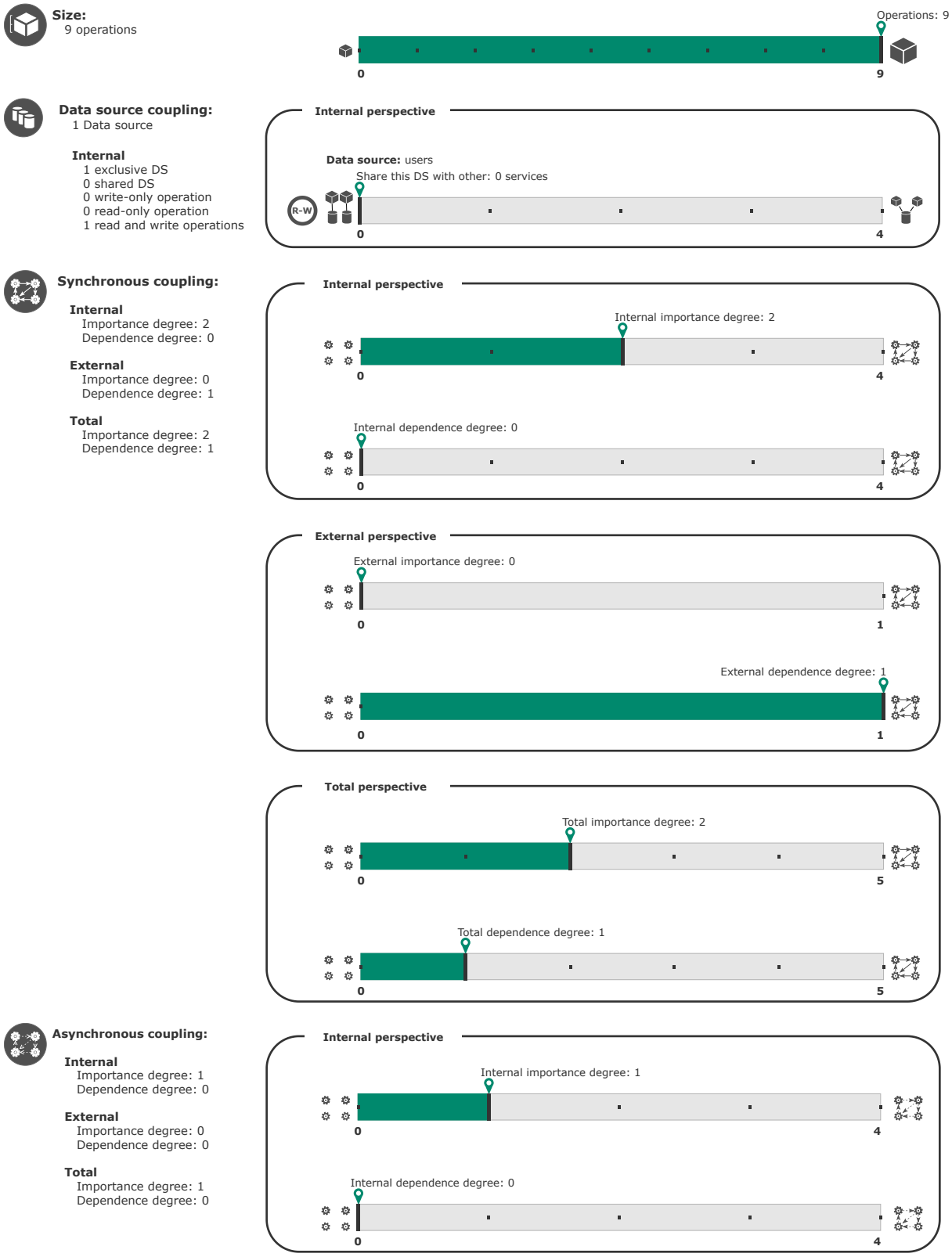


Figure 3.8: *User Service characterization.*

The **Ping Service** is the second-biggest service in the Pingr system, with five operations. It accesses one data source, which it shares with the Feed Service, performing read and write actions. It has a synchronous dependency on User Service. It is one of the services that perform asynchronous interactions, publishing two topics on the message queue that the Chat Service consumes. Figure 3.9 illustrates the Ping Service characterization.

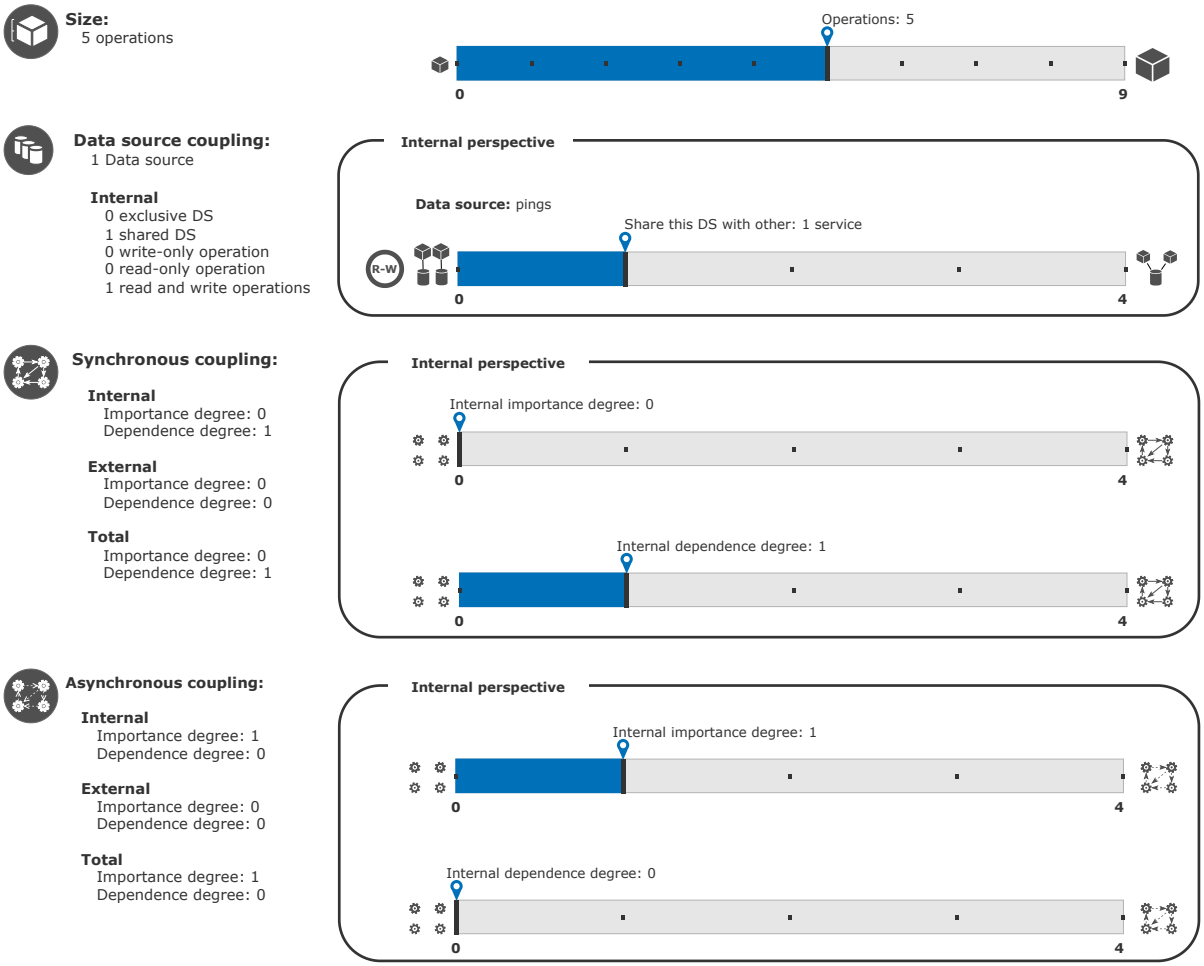


Figure 3.9: *Ping Service characterization.*

The **Chat Service** has three operations. Access one exclusive data source performing read and write actions. It does not have synchronous coupling. It is the service with the biggest asynchronous dependency, consuming topics published by User Service and Ping Service. Figure 3.10 illustrates the Chat Service characterization

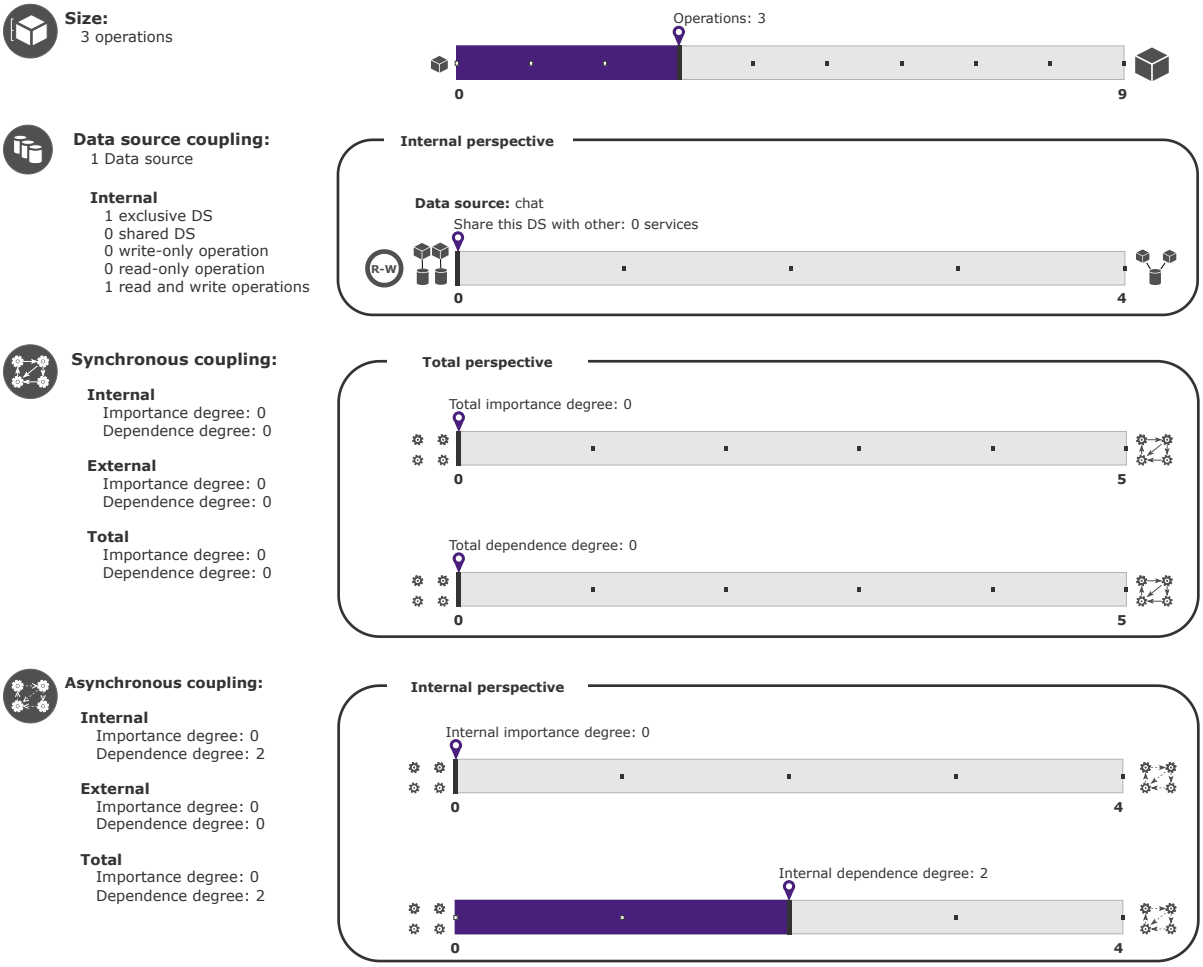


Figure 3.10: *Chat Service characterization.*

The **Feed Service** is the smallest service in the Pingr system, with two operations. It accesses two data sources, one exclusive and one shared with the Ping Service. It has a synchronous dependency on User Service. It does not have asynchronous coupling. Figure 3.11 illustrates the Feed Service characterization

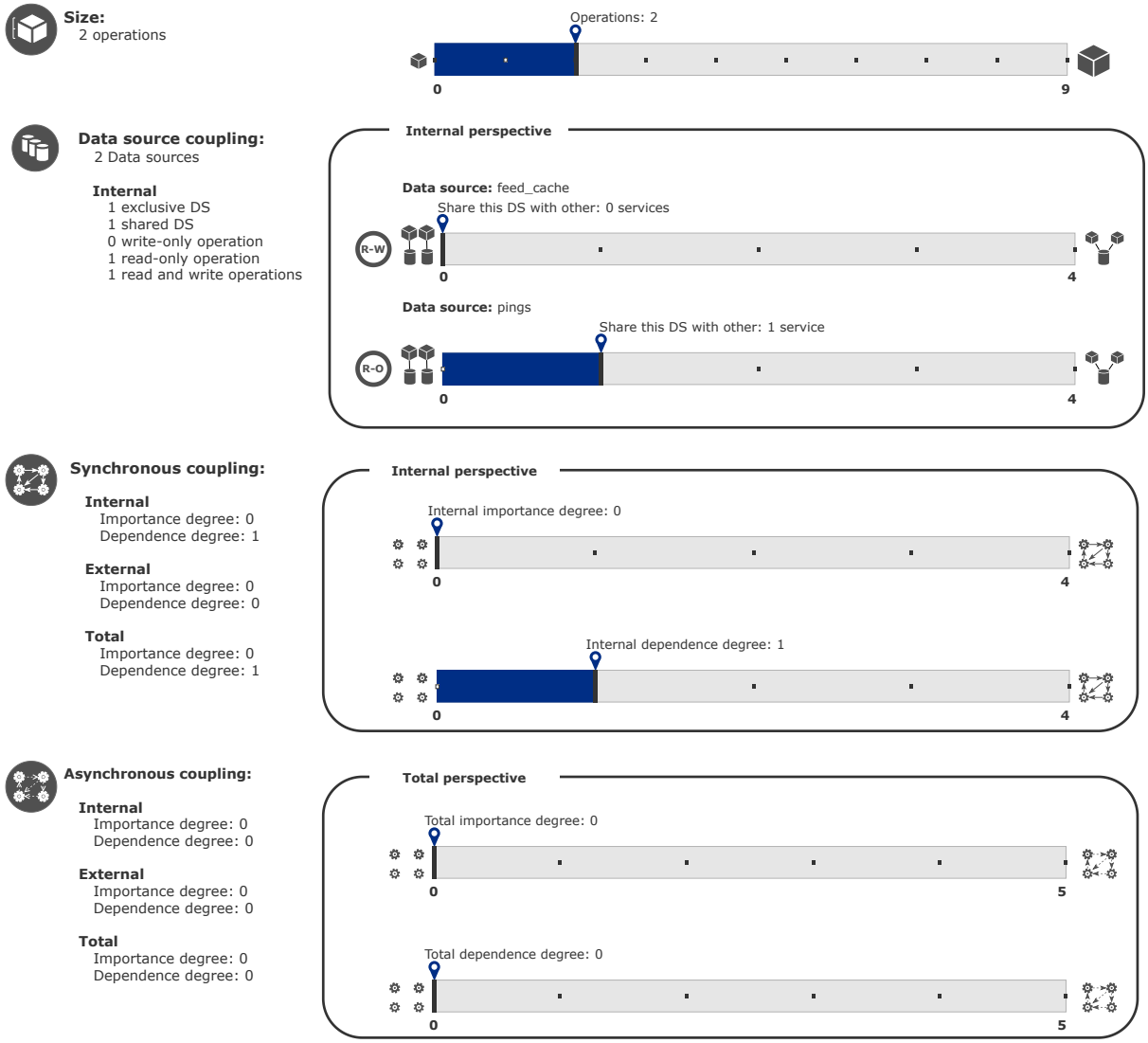


Figure 3.11: *Feed Service characterization.*

The Pingr is a service-oriented system composed of four services. Each module is composed of a single service. User Service is the biggest, with nine operations, while the smallest service (Feed Service) has only two operations. The system has four data sources, one of which is shared between two internal services. 60% of interactions between services are synchronous. On the other hand, 40% of interactions between services are asynchronous. Figure 3.12 illustrates the characterization of the Pingr system, based on the CharM dimensions and metrics.

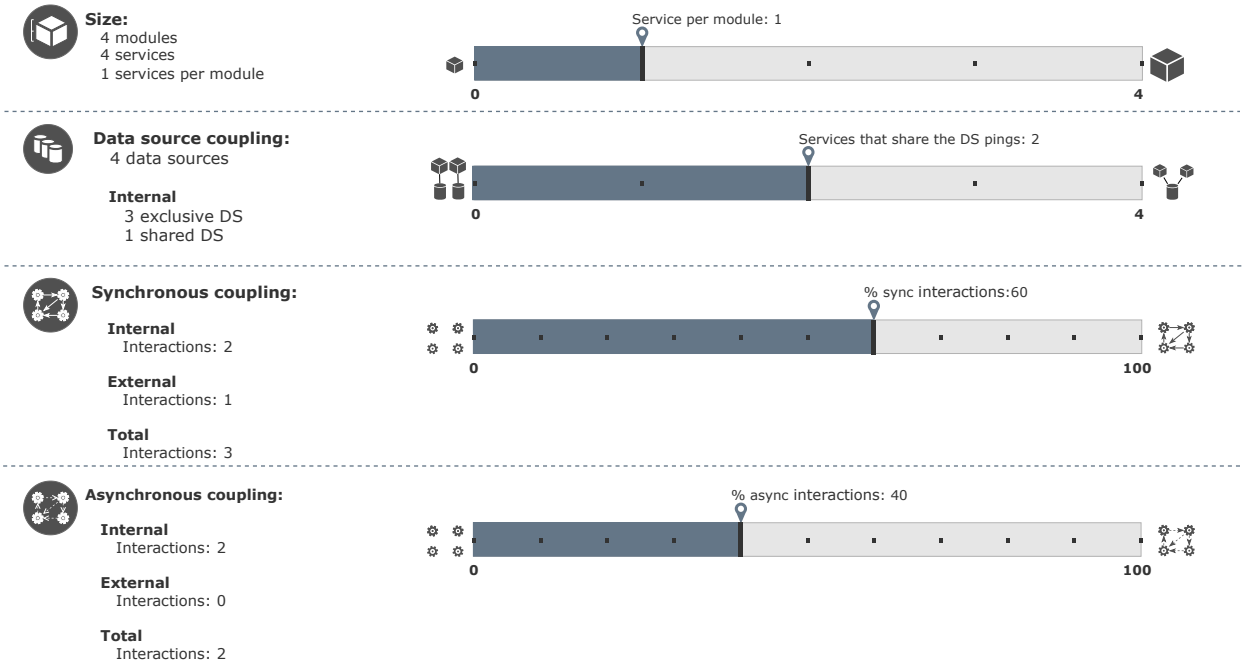


Figure 3.12: *Pingr characterization.*

Given that Pingr is a fictitious system, we can consider it to be at design time. Thus, the characterization generated from the CharM could be very valuable in assessing whether the designed architecture is adequate to meet the pre-established quality requirements. Furthermore, the metrics, rulers, and graphs generated could guide and support reflections and discussions about this architecture.

The characterization also demonstrates that most of the Pingr services interact synchronously. This characteristic can directly influence the system's performance. Since performance is a quality attribute listed as critical in Pingr, it is worth reflecting on whether it is advantageous to transform some synchronous interactions into asynchronous ones. For example, the coupling between the services Ping and User and Feed and User.

3.5 Chapter Summary

This chapter described the CharM design and evolution steps carried out so far. We also delimited what the CharM is and what it is not. In addition, we listed the microservices guidelines that ground the current version of our model. Next, we described the dimensions currently composing the CharM, presenting their goals, metrics, and usefulness. Finally, we demonstrated the application of the CharM in a fictitious scenario.

After presenting our characterization model, in the next chapter, we detail the two case studies carried out to evaluate the CharM and thus identify possible uses and ease of understanding degree.

Chapter 4

Multiple Case Studies

Considering the research goal, it is crucial to evaluate the CharM, in order to identify its possible uses and if this model is easy to understand. As described in Section 3.1, we submitted the CharM to three evaluation stages. In this chapter, we present the results of the first two evaluations, carried out through two case studies. The first one was executed in an academic environment. The second case study was performed in an industrial environment. Based on this, we defined the following research questions (RQs):

- **RQ1:** ***What uses of the CharM are perceived by participants?*** — Considerations: We want to identify tasks where the CharM can be useful. The main points that we explored are architectural understanding and evolution.
- **RQ2:** ***What is the participants' perception of the architectural characterization generated from the CharM?*** — Considerations: We are interested in finding out if the architectural characterization generated by the CharM is easy to understand and if it is coherent with the reality that each participant knows.
- **RQ3:** ***What aspects of the CharM can be improved?*** — Considerations: We are keen to map aspects of the CharM that can be changed or included to make it more useful and easier to understand.

In each case study, we selected a system to apply and evaluate the CharM. We adopted the multiple case study research method, objecting to conducting an in-depth evaluation of our model and obtaining a higher level of generality about the found outcomes. To then use these results as a basis for generating a survey, as well as to perform triangulation of the CharM evaluation. Such triangulation is relevant, as it will contribute to increasing the degree of reliability of the findings (Easterbrook et al., 2008).

We define three key steps for the multiple case study (Figure 4.1). The first consists of obtaining a general understanding of the architecture of the system chosen for the study. In the next step, the CharM is applied to characterize the architecture of such a system. Finally, we invited project members to evaluate the architectural characterization generated from our model. In each case study, we refine these steps according to the complexity of the context. The main techniques we adopted to collect data were document analysis and semi-structured interviews. To analyze the CharM evaluation data, we adopted open and

axial analysis procedures and the constant comparison method.

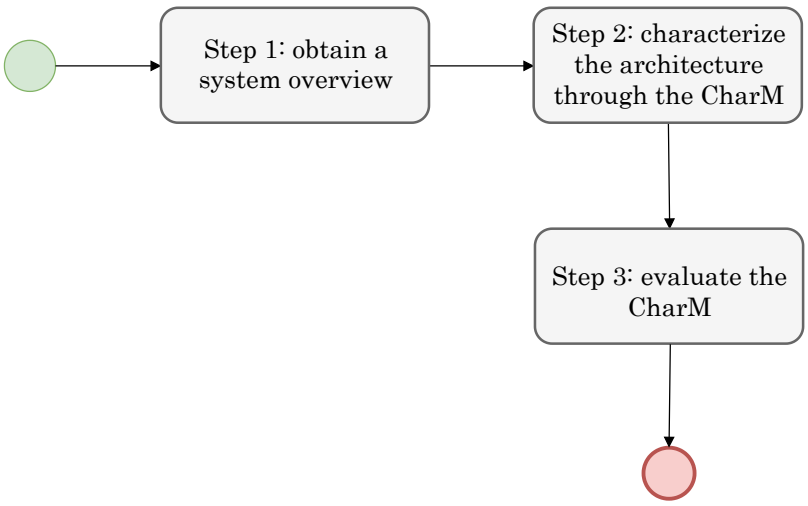


Figure 4.1: *Key steps of the multiple case study .*

In the following sections, we describe the two case studies carried out, detailing the applied research design as well as the obtained results.

4.1 InterSCity Case Study

We conducted the first case study in an academic project named InterSCity. The InterSCity is a Brazilian collaborative research project that aims to create a platform that enables the development of robust, integrated, sophisticated applications for the smart cities of the future (Batista et al., 2016). The platform designed in this project was also named InterSCity. Thus, the InterSCity Platform was created to support collaborative work development and enable experiments in the smart cities field. Furthermore, it is worth mentioning that some important quality requirements considered and analyzed during the design of this platform were scalability and evolvability (Esposte, 2018).

The architecture of the InterSCity Platform is microservices-based and has adopted the following principles: modularity via services; distributed models and data; decentralized evolution; reuse of open source projects; adoption of open standards; asynchronous versus synchronous; and stateless services. Besides that, its architecture provides a set of high-performance cloud-based mechanisms to manage heterogeneous IoT (Internet of Things) resources, data storage and management, and context-aware resource discovery (Esposte, 2018). We chose the InterSCity Platform to carry out this case study because it has a service-based architecture, is open-source, is constantly evolving, has extensive documentation, and is composed of a few services.

4.1.1 Research Design

To achieve the proposed goal and answer the RQs 1, 2, and 3, we followed four main steps in this study, as illustrated in Figure 4.2. This case study ran from May to August 2020.

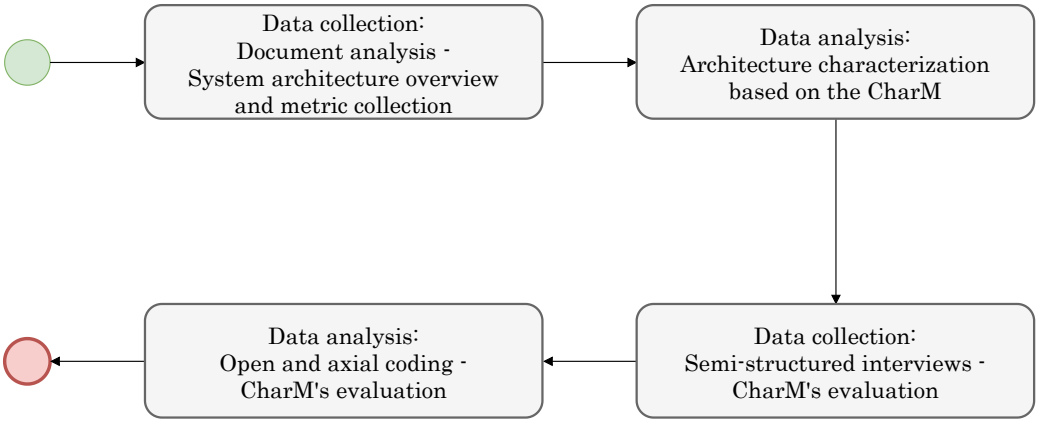


Figure 4.2: *InterSCity case study steps.*

At first, we carried out a document analysis to understand the goals of the InterSCity Platform, obtain an overview of its architecture and collect the metrics of the CharM. During this step, we studied a set of scientific work related to the development of this platform [1]. From this, we identified the main elements of its architecture, especially its services and their respective responsibilities. We also analyzed the documentation and source code available in the GitLab repository [2]. One of the objectives was to manually collect the CharM's metrics. This task was performed by two researchers (one of them is the author of this thesis) in a dynamic similar to the agile practice of Pair Programming. We examined and discussed each service of the platform configuration files and source code [3].

In the next step, based on the data collected during document analysis and following the dimensions and metrics of the CharM, we characterized the architecture of the InterSCity Platform. The result of this characterization is detailed in Section 4.1.2.

After completing the characterization of the InterSCity Platform's architecture, we collected data to evaluate the CharM. For this, we conducted semi-structured interviews. The respondents were invited via email. To contact them, we asked the InterSCity project research group for suggestions from members who could collaborate with our study. From these indications, we invited 11 (all male) platform contributors to evaluate the result generated by the CharM. After accepting the invitation, each participant received the interview protocol (Appendix D.1) and a video [4] explaining the CharM and the characterization of the InterSCity Platform architecture. It is worth clarifying that the protocol was subjected to a pilot test with one interviewee and then was minor adjusted.

We started the interviews by briefly explaining the research objectives and guidelines.

[1] Software Platforms for Smart Cities: Concepts, Requirements, Challenges, and a Unified Reference Architecture (E. F. Z. Santana et al., 2017) and A Scalable Microservice-based Open Source Platform for Smart Cities (Esposte, 2018).

[2] InterSCity repository: gitlab.com/interscity/interscity-platform/interscity-platform

[3] A summary of the collected data, based the CharM, is available at: https://drive.google.com/file/d/1qVo2j7raTaOLUZYPu53PgiJCGoAHfTyK/view?usp=sharing

[4] The video explaining our model and the characterization of the architecture of the InterSCity Platform (in Portuguese) is available at: https://drive.google.com/file/d/1XzQORjPuJuPw7Yor5X2tpef3cghON2P/view?usp=sharing

The interview script was composed of four sets of questions: (i) the respondent's experience, (ii) the respondent's perception of the InterSCity Platform, (iii) evaluation of the CharM and the architectural characterization generated from it, and (iv) possibilities of architectural evolution based on the result generated by the CharM.

The 11 interviews took place between July and August 2020 and were conducted and recorded remotely via Google Meet after the respondents' consent. On average, the interviews lasted 34 minutes. In all, the 11 interviews lasted 6 hours and 22 minutes. It is worth mentioning that the sample of participants in this study step is non-probabilistic and combines convenience and referral-chain types (Baltes and Ralph, 2020). Furthermore, our sample size is aligned with evidence from the anthropology field, which indicates that 10-20 knowledgeable people are sufficient to uncover and understand the core categories in any study of lived experience (Bernard, 2011).

In the last step of the InterSCity case study, we analyzed the data collected from the interviews using open and axial coding procedures (Corbin and A. Strauss, 2015; Stol et al., 2016). We started by applying the open coding procedure, from which we mapped some use categories of the CharM. For this, we performed an iterative process of inductively coding the transcription of one interview at a time. Following, we did the axial coding, where we further analyzed and reviewed the interviews to identify relationships between the categories that emerged from the open coding analysis. The author of this thesis conducted the preliminary analyses and multiple meetings with two other experienced researchers (advisor and co-advisor of this thesis) to discuss and increase the results' reliability and mitigate bias (Patton, 2014). Furthermore, throughout the coding process, we adopted the constant comparison method (Glaser and A. L. Strauss, 2017), whereby we continuously compare the results of an interview with those obtained in the previous ones.

Due to confidentiality reasons, we do not share the interviews' transcription. However, we made it publicly the code book available (Appendix D.2).

4.1.2 Characterization of the Architecture of the InterSCity Platform

To characterize the architecture of the InterSCity Platform, we analyzed (documentation and code) each of its five services. The metrics collected were arranged in rulers to illustrate the service's profile in each of the CharM dimensions. The scales used in the rulers of each dimension take data from the InterSCity Platform itself as a reference. Thus, the ruler of the size dimension ranges from 0 to 12 (maximum number identified of services operations). The scale used in the data source, synchronous, and asynchronous coupling dimensions ranges from 0 to 5 (number of the platform services). At first, this section presents the characterization of the services, followed by the characterization of the platform architecture, based on the 5^{th} version of the CharM.

Characterization of the Resource Discovery: It is the smallest service on the platform, with just one operation. It accesses one exclusive data source, performing read and write actions. It is the service with the biggest synchronous dependence degree since it invokes operations from two other services. Furthermore, it is the only service that does not have asynchronous coupling. Figure 4.3 illustrates the Resource Discovery characterization.

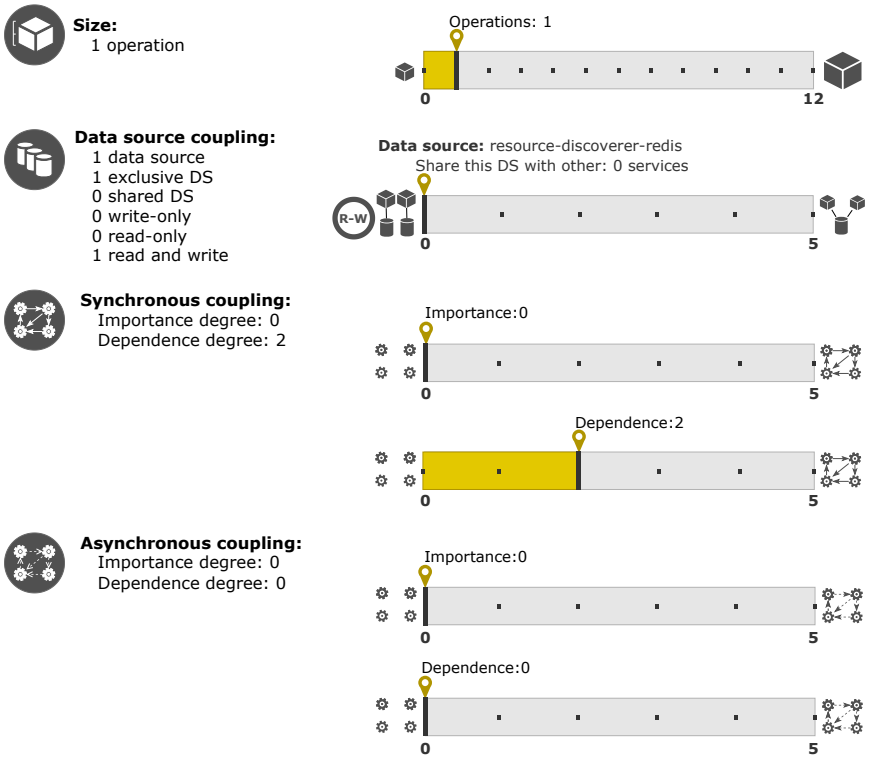


Figure 4.3: *Resource Discovery characterization.*

Characterization of the Actuator Controller: It is the second-smallest service of InterSCity, with two operations. It has one exclusive data source, which performs read and write actions. It is the only service on the system which does not have synchronous connections. Concerning asynchronous connections, only one service consumes messages published by it in one of its topics. In contrast, it consumes messages published by two other services on the platform, and it is one of the services with the highest degree of asynchronous dependence on the system. Figure 4.4 illustrates the Actuator Controller characterization.

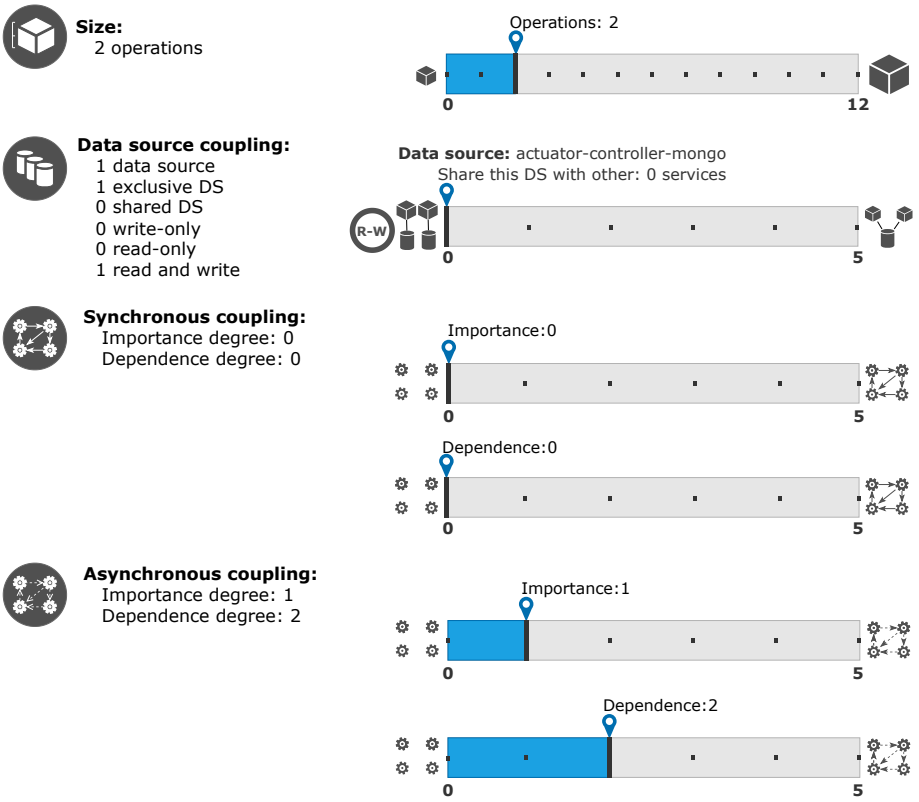


Figure 4.4: *Actuator Controller characterization.*

Characterization of the Data Collector: It is the third-smallest service on the platform. It accesses two exclusives data sources, performing read and write actions. Only one service requests synchronous messages from it. Among the services with an asynchronous connection, it is the only one that just consumes but does not publish messages. Furthermore, it is one of the services with the highest degree of asynchronous dependence on the platform. Figure 4.5 illustrates the Data Collector characterization.

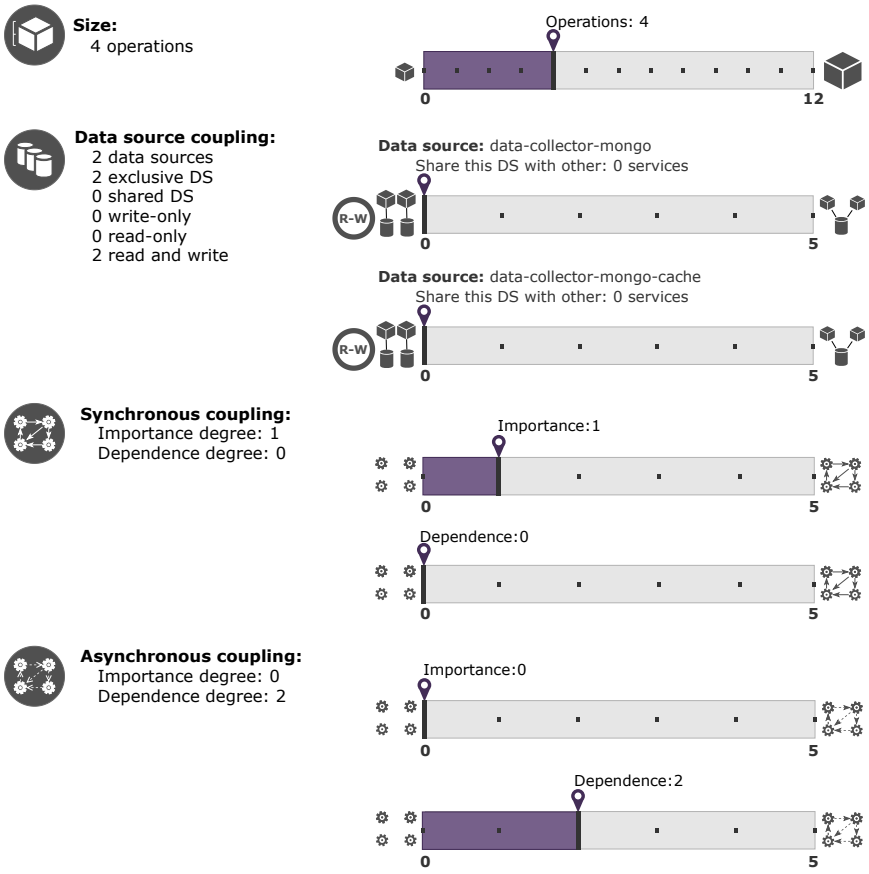


Figure 4.5: *Data Collector characterization.*

Characterization of the Resource Adaptor: It is the second-biggest InterSCity's service, with 8 operations. It has two exclusive data sources, performing read and write actions. Furthermore, it is one of the platform's services that makes synchronous requests. It is the service with the highest asynchronous importance degree, since three other services consume the messages it publishes. Furthermore, it is subscribed to a topic published by another service of the system. Figure 4.6 illustrates the Resource Adaptor characterization.

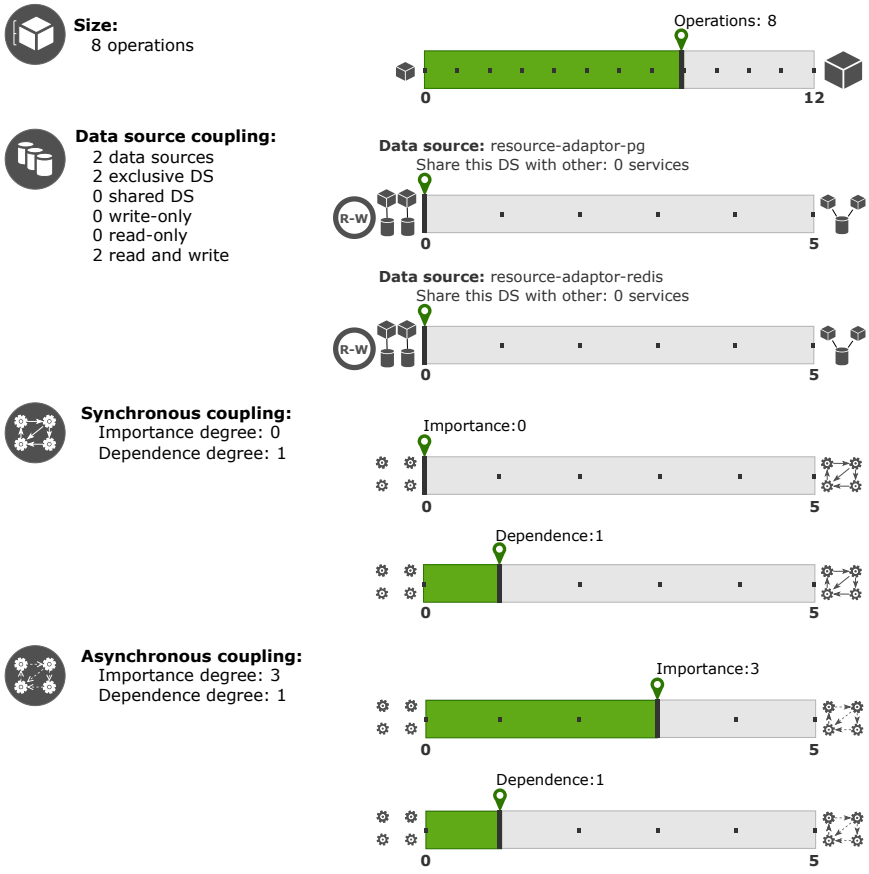


Figure 4.6: *Resource Adaptor characterization.*

Characterization of the Resource Catalog: It is the biggest service in the system, with 12 operations. It accesses two exclusives data sources, performing read and write actions. It is the service with the biggest synchronous importance degree since two other services invoke its operations. It is also the service with the biggest asynchronous importance degree since two other services consume its messages. Figure 4.7 illustrates the Resource Catalog characterization.

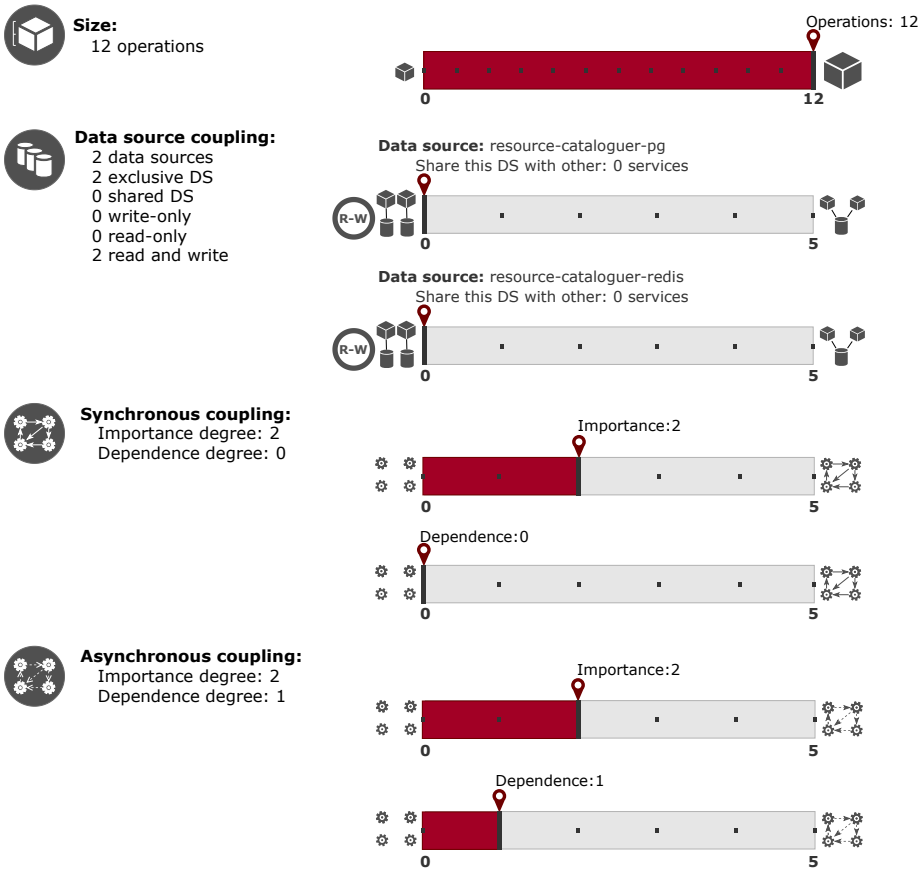


Figure 4.7: *Resource Catalog characterization.*

After mapping and understanding the profile of each service of the InterSCity, we could characterize this platform's architecture. As in the services' characterization, we arranged the collected metrics in rules. The rules of the size and data source dimensions have a scale ranging from 0 to 5 (number of platform services). The scale used in the rules of the synchronous and asynchronous coupling dimensions ranges from 0 to 100 to represent the percentage of each type of communication.

Characterization of the InterSCity Platform: The InterSCity platform can be classified as a small service-oriented system since each module comprises a single service. Resource Catalog is the biggest service, with 12 operations, while the smallest service (Resource Discovery) has only one operation. The system has eight data sources. The platform services do not share these data sources with each other. Approximately 33% of interactions between services are synchronous. On the other hand, 67% of interactions between services are asynchronous. Figure 4.8 illustrates the characterization of the architecture of the InterSCity Platform, based on the CharM dimensions. Figure 4.9 presents an overview of the platform architecture, containing the main structural elements analyzed with the CharM.

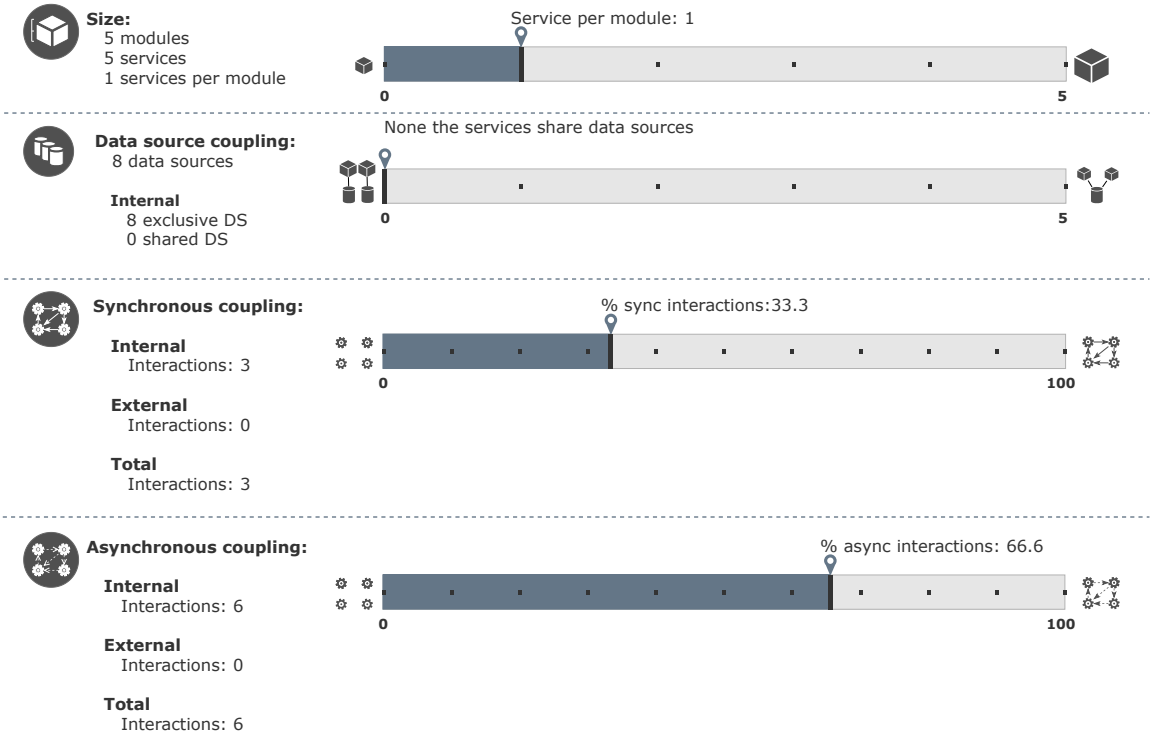


Figure 4.8: *InterSCity characterization.*

Appendix E complements the characterization of the platform and its services through visualizations that consider the different CharM dimensions, present different perspectives, and facilitate comparisons.

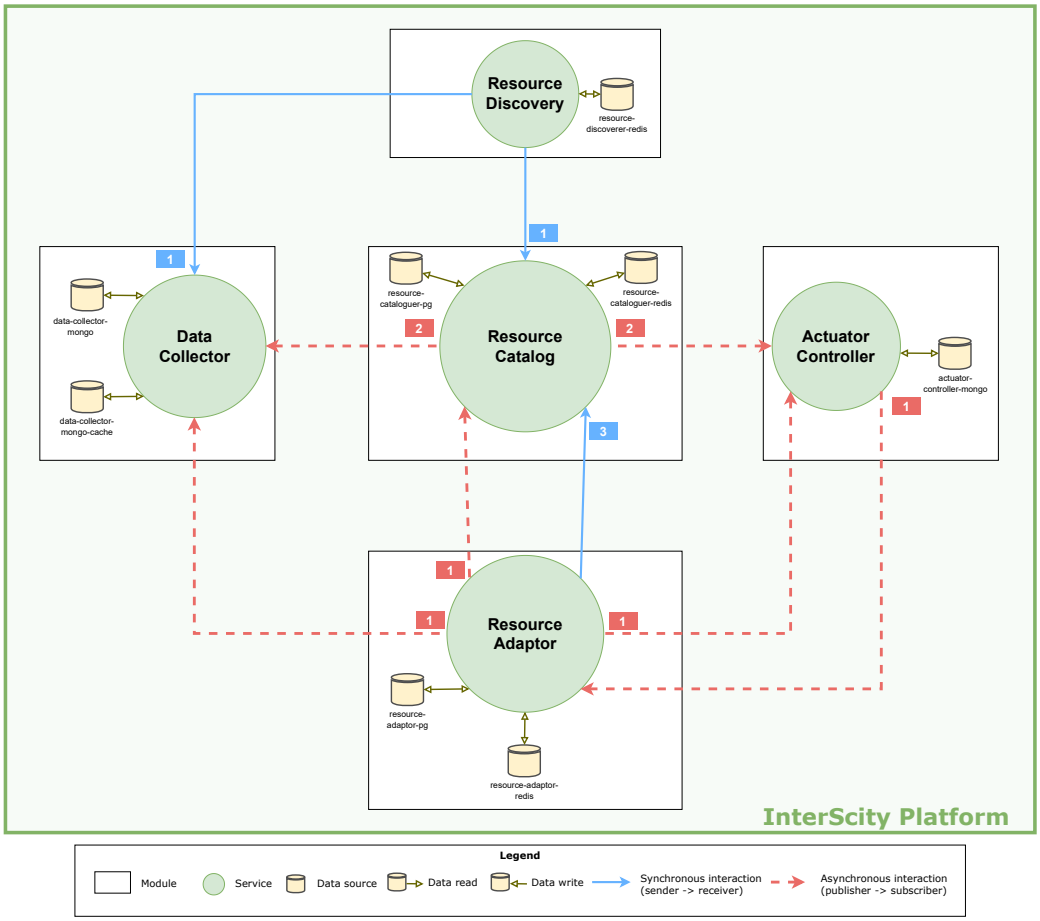


Figure 4.9: *Overview of the InterSCity architecture (notation inspired by Merson and Jospeh Yoder (2019)).*

4.1.3 Evaluation Results

Participants Profile

We interviewed 11 professionals with different experiences contexts (academic and industrial) and knowledge levels (from novices to specialists). These professionals acted in different roles in the project, such as manager, architect, consultant, developer, or external contributor. The interviewees' experience with service-based architecture varies from 5 months to 32 years (≃ 6 years and 1 month on average). The working time with the InterSCity Platform varies from 5 months to 2 years (≃ 1 year and 4 months on average). During this case study, 7 out of 11 respondents were working with the platform. The other 4 respondents have worked with the InterSCity some years before, mainly in the design stage. Table 4.1 details interviewees' profile.

Participant ID	Role	Experience with SBA (months)	Experience with the Platform (months)	Experience context	Current member?
P1	Manager	26	26	Academic	Yes
P2	External contributor	5	5	Academic	Yes
P3	Architect and developer	72	30	Academic and industrial	No
P4	Architect and developer	48	30	Academic and industrial	No
P5	Architect	48	24	Academic	No
P6	Consultant and developer	84	12	Academic and industrial	Yes
P7	External contributor	5	5	Academic	Yes
P8	External contributor	30	30	Academic and industrial	No
P9	Consultant	84	18	Academic and industrial	Yes
P10	External contributor	384	36	Academic and industrial	Yes
P11	External contributor	24	18	Academic and industrial	Yes

Table 4.1: *Participants profile – InterSCity case study.*

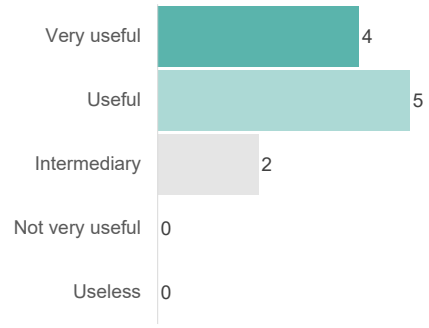
Evaluation of the CharM

We invited the interviewees to evaluate the CharM. Some aspects investigated at this stage were usefulness, ease of understanding, and coherence. In the following sections, we describe the results of this evaluation.

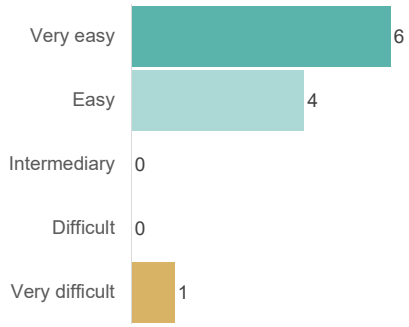
The Uses of the CharM

Regarding usefulness, we asked the participants how much our model helped them understand the platform's architecture. The average score was 4.2, on a scale from 1 (Useless) to 5 (Very useful). As detailed in Figure 4.10a, nine respondents indicated that the CharM is very useful (four – P1, P2, P9, and P11) or useful (five – P3, P5, P6, P7, and 10) for understanding the system architecture. Only the interviewees P4 and P8 indicated that the model's usefulness is intermediate.

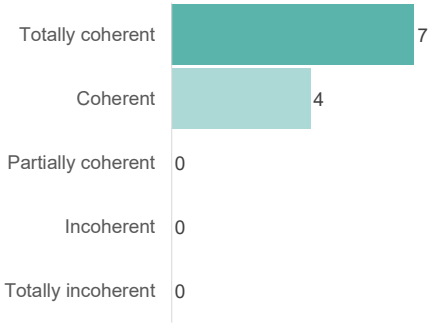
Respondents P4 and P8 argued that for someone already familiar with the InterSCity Platform, the CharM is not so useful for understanding its architecture. However, when considering new members, these interviewees indicated that our model could be advantageous and help them understand the system's architecture. Interviewee P4 considers that the CharM helps identify elements that compose the architecture that require attention and possibly need maintenance. P4 also explained that our model could be adopted to guide architectural evolution (at a high level of abstraction). Because they believe that with the CharM, it is possible to discover differences between the services of architecture, as well as identify services that could be divided. Nevertheless, they pondered that our model does not help discover elements missing in the architecture.



(a) *Usefulness for understanding the architecture.*



(b) *Ease of understanding the results.*



(c) *Coherence of the architectural characterization.*

Figure 4.10: *Evaluation of the CharM's usefulness, ease of understanding, and coherence – InterSCity case study.*

From the point of view of interviewee P8, the CharM helps understand the relationship between architecture elements, as well as identify couplings. Furthermore, P8 agrees with P4 that our model can guide the process of architectural evolution and maintenance, and facilitate the identification of services that can be divided. P8 further stated that the CharM was useful to confirm previous intuitive knowledge that they already had.

During the interviews, we asked participants to indicate aspects in which the CharM could be useful. In all, respondents indicated 18 uses for our model. We detailed this result in Figure 4.11. Nine of the 11 participants responded that the CharM could help *"understand the architecture"* (P1, P2, P3, P5, P6, P7, P9, P10, and P11) and to *"guide the architectural evolution in a high level of abstraction"* (P1, P3, P4, P6, P7, P8, P9, P10, and P11). It is worth mentioning that the interview protocol had questions directly related to our model's usefulness for the architecture's evolution and understanding. Since we were interested in investigating these two key uses in this case study.

We also asked participants if the CharM helped discover new information about the architecture. Six participants confirmed this use (P2, P4, P5, P7, P10, and P11). Nevertheless, we verified that the other five participants (P1, P3, P6, P8, and P9) indicated that the model did not help discover new information. Among these interviewees, P1 and P8 explained that despite this, the CharM was useful to confirm their intuitive knowledge about the platform architecture. Participants P3, P6, and P9 reported that they already had a deep knowledge of the architecture, so they considered that the model did not add new information.

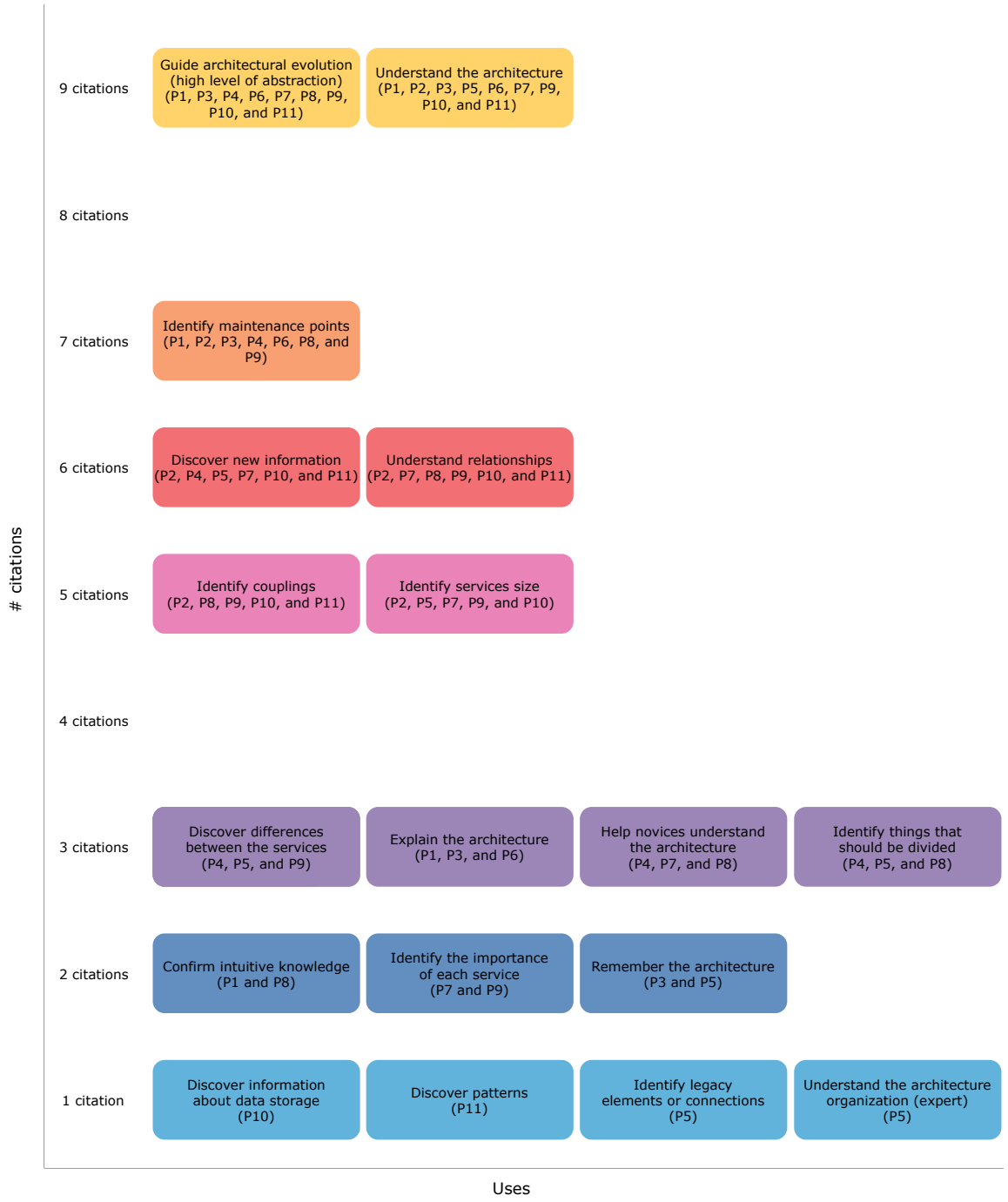


Figure 4.11: *Uses of the CharM – InterSCity case study.*

Other uses of the CharM mentioned by more than half of the interviewees were: the possibility of *identifying maintenance points* (seven - P1, P2, P3, P4, P6, P8, and P9); and *understanding relationships* (six - P2, P7, P8, P9, P10, and P11). Participant P3 stated that, although they consider that our model helps to understand the architecture and identify points of evolution and maintenance, they believe that since the CharM adopts static metrics, it would only be used sporadically.

We hierarchically organized the uses of the CharM mentioned during the interviews. It

is worth noting that during this case study, we investigated three key uses for our model: understanding the architecture, discovering new information, and guiding architectural evolution. From this, we identified the secondary uses related to these key uses. The result is illustrated in Figure 4.12.

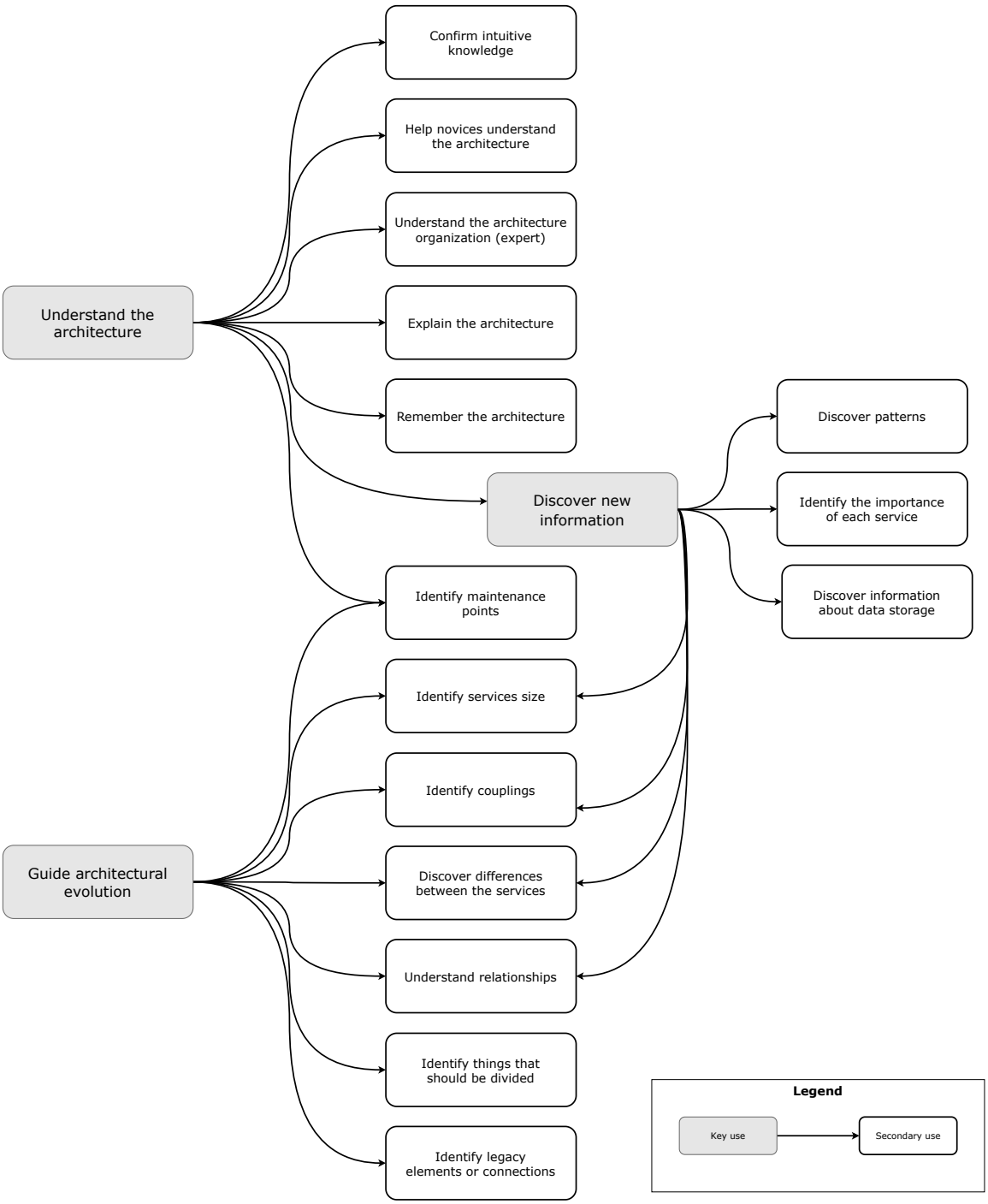


Figure 4.12: *Hierarchical organization of the uses of the CharM – InterSCity case study.*

Ease of Understanding the Results of the CharM

We also asked respondents about the ease of understanding the result generated by the CharM. The average score given by participants was 4.3, on a scale from 1 (Very difficult) to 5 (Very easy). As detailed in Figure 4.10b, six respondents indicated that the result generated by the CharM is very easy to understand (P1, P5, P6, P7, P8, and P9) and four consider it easy (P2, P3, P4, and P10). Only interviewee P11 claimed that the result generated by our model is difficult to understand. However, P11 explained that we first need to understand the microservice architectural style, which is complex. In order to, then, we can be able to understand the data presented by the CharM. This is evidenced in the following excerpt from the interview: "*You must have a background in architecture focused on microservices to understand the model, and even then, doubts can still arise*".

When evaluating the ease of the model, interviewees P8 and P9 highlighted that the graphic elements of the CharM contributed a lot to the metrics' understanding. This opinion is evidenced in the following excerpts: P8: "*[...] the graphs presented are very simple to understand*". P9: "*[...] the graphs you created were very easy to understand and compare*".

Coherence of the Architectural Characterization

Participants were also invited to assess the coherence of the architectural characterization obtained from the CharM. On a scale from 1 (Totally Incoherent) to 5 (Totally coherent), the average score was 4.6. As detailed in Figure 4.10c, seven interviewees considered the architectural characterization "totally coherent" with the reality they knew (P1, P2, P5, P7, P9, P10, and P11). The other interviewees (P3, P4, P6, and P8) evaluated the characterization as "coherent".

Although the model result was evaluated as coherent, when considering the architectural elements, participant P6 indicated that they believe it is essential to add the representation of the API Gateway and the Broker in the characterization. Participant P8 stated that they were surprised by the characterization coherence. P8 explained that they did not believe the adopted metrics could generate such a precise result. The remaining participants did not add comments on the characterization coherence.

Improvement Suggestions

During the interview, we asked participants to suggest improvements to our model. Altogether, respondents presented 18 types of suggestions for improvement, which we organized into four groups, illustrated in Figure 4.13.

Seven different participants suggested presenting additional information in the model. The improvement suggestion most cited by the interviewees (four - P1, P3, P4, and P5) was to present more details about the services since it would complement the context and contribute to the architectural understanding. Some of the additional information cited were: the responsibility of the service, the role played (e.g., aggregator), and the functional and non-functional requirements. Respondents also suggested providing more details about the type of DBMS (Database Management Systems) (four - P4, P6, and P8) used in the system, as well as the connections between services (two - P3 and P9).

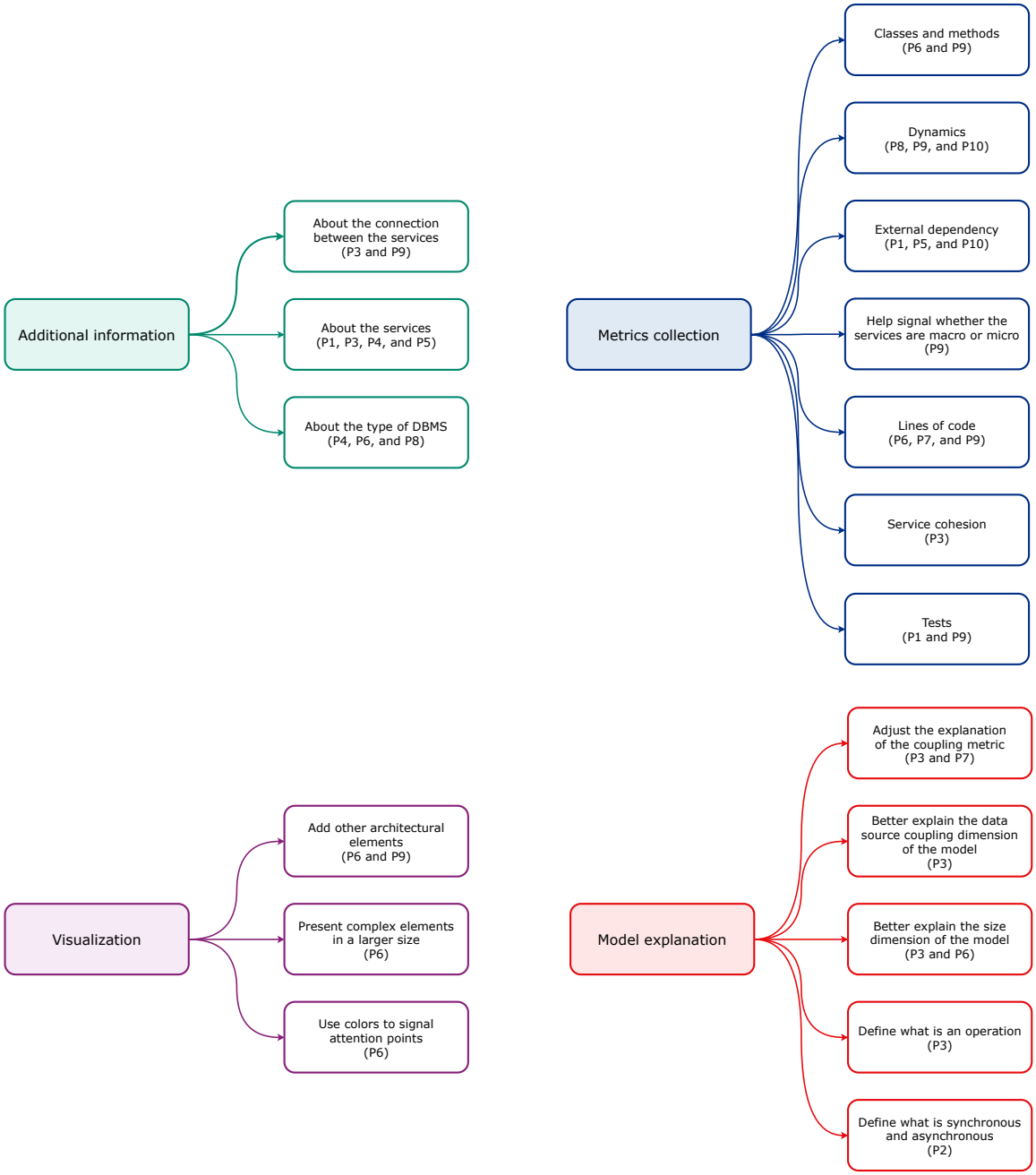


Figure 4.13: *Improvement suggestions to the CharM – InterSCity case study.*

Respondents also suggested collecting some metrics that they considered interesting for the model. Respondents P8, P9, and P10 believe that the collection of dynamic metrics can offer a more comprehensive and accurate view of the system architecture, favoring the identification of attention points. Participants P1, P5, and P10 recommended collecting metrics from external dependencies to obtain a more complete view of the coupling of services. Respondents P6, P7, and P9 believe collecting line-of-code metrics could be interesting. Other proposed metrics are related to classes, methods, and tests, as well as cohesion and metrics for classifying a service into macro or micro.

Participants also suggested improvements to the architecture visualization. Respon-

dents P6 and P9 indicated that it would be interesting to add the representation of elements, such as Brokers and Gateways to the diagrams. Interviewee P6 also recommended differentiating more complex elements by size and using colors to signal attention points in the architecture.

Furthermore, during the interviews, we mapped points that were confused during the model explanation. We noticed that there was some difficulty in understanding the coupling metrics. It was also necessary to resolve some doubts related to the size dimension, as well as the dimension of data sources coupling. An explanation of the concept adopted for operation and synchronous and asynchronous communication in the context of the model was also requested.

Based on the suggestions received, we created a new version of the CharM that incorporated some of the improvements indicated by the interviewees. We added new information related to services and their couplings. We also made improvements to the metrics collected, and we welcome the suggestion to add the external coupling metric. Nonetheless, we did not add other suggested metrics, such as dynamics, tests, and lines-of-code, since we consider these outside the scope of this research. Furthermore, we accepted all suggestions for improvements regarding the explanation of the model and some related to the results visualization.

4.1.4 Threats to Validity

This section addresses the main threats to the validity of this case study.

Internal: There may have been failures in interpreting the documents or communicating with respondents. Besides that, the documentation analyzed to characterize the architecture of the InterSCity Platform may be out of date. To mitigate the possible problem of documentation inconsistency, we analyzed the platform source code, the documentation available in the repository, and related scientific work. During the interviews, the developers were asked about the coherence of the characterization. To mitigate communication and interpretation failures, part of the data collection was carried out and discussed with the help of another researcher in the software architecture area. Furthermore, we sought to confirm with the interviewees the interpretation of what was said. Another adopted strategy was that after preliminary analyses, the results were discussed with the advisor and co-advisor of this thesis.

External: The profile of the interviewees can interfere with the evaluation results since professionals with more experience have a more profound knowledge of software architecture. However, the heterogeneity of profiles is desired because it is possible to verify the model's usefulness for different professionals. That is, from professionals with detailed architecture knowledge to beginners in the project. Thus, we could replicate this study with development teams from other service-based systems. Another relevant point is that the InterSCity Platform is microservices-based. The trade-offs related to this architectural style are still being investigated and are therefore not fully or widely known. Thus, we believe that this fact can influence the interviewees' perception of the architectural decisions discussed.

Construct: We discussed the best way to formulate the interview questions to reduce

possible biases. Furthermore, we carried out a pilot test with a first interview. Based on this test, we reviewed the questions and made some adjustments. An important issue to highlight is that, during the interviews, the interviewer could explain and resolve any doubts if the interviewee did not correctly understand the main meaning of any question. Another limitation is that the scope of this case study is academic. Until now, the InterSCity Platform has not been deployed in a production environment. Thus, the InterSCity was just used in simulation and restricted academic/scientific scenarios. Despite these limitations, we consider the studied system appropriate for conducting a first case study and collecting relevant insights.

4.2 Industry Case Study

We carried out the second case study in Company A (fictitious name for confidentiality reasons). Company A is an online handicraft marketplace founded in 2008. In 2009, it reached the mark of 100,000 products sold and 10,000 active sellers. As of 2022, it has over 7 million products announced, produced by over 100,000 active sellers.

According to the software engineers team, some of the most relevant quality requirements for Organization A's systems are scalability, maintainability, usability (UX), security, availability, resilience to failures, testability, and continuous delivery. In 2018, the Information Technology (IT) team began a process of migrating the software architecture from a monolithic to a service-based style. One of the main motivations for starting such migration was the difficulty of working (maintaining and evolving) with the monolith. Furthermore, it was necessary to increase system performance and team productivity. Thus, we chose Company A's marketplace for this case study because it is in the process of architectural style migration (monolith to service-based). Besides that, the IT team was willing to collaborate with this study.

4.2.1 Research Design

To achieve the objective of this thesis and answer the RQs 1, 2, and 3, during the industry case study, we followed the eight steps presented in Figure 4.14. This case study ran from December 2020 to July 2021.

Before starting this case study, we met with the CTO (Chief Technology Officer) and a software engineer from Company A. During this meeting, we presented the research objectives, the planned investigation roadmap, and the expected results. In the end, we obtained authorization to start our case study and defined the list of employees who would be interviewed during the first step.

The first step goal was to obtain an overview of Company A's systems and identify the main contexts. Thus, between December 2020 to February 2021, we interviewed 12 team leaders (11 men and 1 woman) previously indicated by the CTO. The interviews were conducted via Google Meet. On average, each interview lasted 1 hour. In all, the 12 interviews lasted 12 hours and 10 minutes.

We started the interviews by briefly explaining the research objectives and the interview script. The interview protocol (Appendix D.3) was composed of three sets of

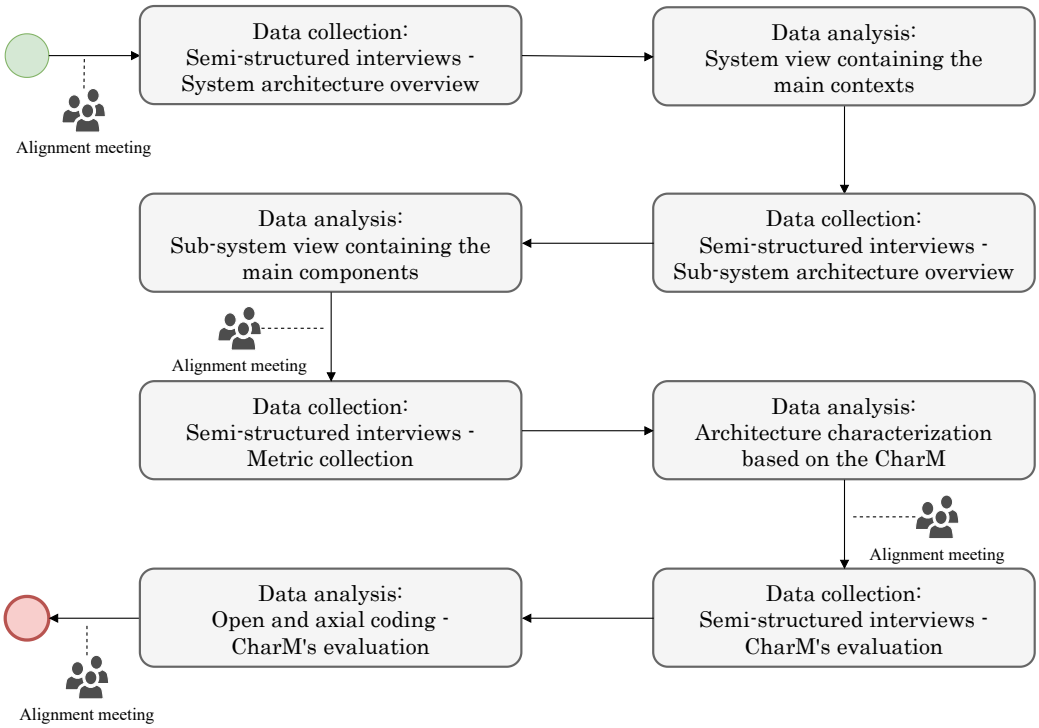


Figure 4.14: *Industry case study steps.*

questions related to (i) respondent's experience, (ii) respondent's perception of the company's systems architecture, and (iii) respondent's perception of the architectural migration process. With Company A and the interviewees' consent, all interviews were recorded. Besides the interviews, we also analyzed a series of architecture documents shared by the company.

Following, we analyzed the data collected during the first step and discussed the preliminary results with the software engineer, the same one who participated in the first meeting. With this, we improved the generated artifacts and obtained an overview of Company A's systems architecture. Furthermore, we defined the context in which the CharM would be applied to do its evaluation. We present the obtained results of this step in Section 4.2.2.

Besides obtaining an overview of the system architecture and mapping its main contexts, the previous step allowed us to identify that the architectural migration process started on the *Search System*. Therefore, we delimited this case study to such a system since, among the analyzed contexts, we considered it the most suitable to apply the CharM due to the variety of services, providing a wider adoption of our model.

Thus, in the third step of this case study, we interviewed three experienced members (all male) of the team of *Search System*. The objective was to obtain an overview of such a system and identify its main components. The interviews took place in February 2021. As in the first step, the interviews were conducted via Google Meet. On average, these interviews lasted 1 hour and 18 minutes. In all, the three interviews lasted 3 hours and 54 minutes. In this step, the interview protocol (Appendix D.4) was composed of two sets of questions related to (i) the respondent's experience and (ii) the respondent's perception of

the *Search System* architecture. All interviews were also recorded after Company A's and interviewees' consent.

In the fourth step, we analyzed the data collected from the previous interviews. This data revealed the components and the structure of the *Search System*. Fundamental information for planning and executing the next step, where the CharM metrics were collected.

In April 2021, we presented the obtained results to the CTO, the product manager, the tech lead of information retrieval, and a software engineer of Company A. During this meeting, we discussed the results and their relevance to the company, as well as aligned the next steps of the case study.

One of the goals of the fifth step was to collect the CharM metrics manually. For this, during May 2021, we interviewed six members (all male) of the *Search System* team. Different from previous steps, in this one, the professionals were interviewed in pairs. Only one of the interviews was individual. This strategy was adopted to reduce the risk of failures in the metrics collection process. Thus, in all, the six professionals were interviewed, in a total of four sections (three sections with two professionals and a single one with only one professional). As in the previous steps, the interviews took place via Google Meet. On average, each interview section lasted 1 hour and 6 minutes. In all, the four interviews sections lasted 3 hours and 39 minutes. The interviews of this step were also recorded after obtaining the consent of Company A and the interviewees.

The questions asked during the interviews are directly related to the CharM metrics. To answer each question of the interview protocol (Appendix D.5), team members accessed the necessary architectural documents, source code, and system configuration files. The author of this thesis followed and discussed the entire process of looking for and verifying the requested data. After the six interviews, we held a moment to discuss and resolve doubts with a member appointed by the team.

In the next step, based on the data collected and following the dimensions and metrics of the CharM, we characterized the architecture of the *Search System*. The result of this characterization is detailed in Section 4.2.3.

In June 2021, we held a new meeting to share and discuss the obtained results with the company. The CTO, the chief architect, the product manager, the tech lead of information retrieval, and a software engineer from Company A participated in this meeting. During this meeting, we also aligned the last steps of the case study, which focused on evaluating the CharM.

After completing the characterization of the *Search System* architecture, we collected data to evaluate the CharM. For this, we conducted semi-structured interviews. We invited six members (5 men and 1 woman), with different experience levels, of the *Search System* team to evaluate the result generated by the CharM. These members were indicated by the participants of the last results-sharing meeting. Each interviewee received the interview protocol (Appendix D.6) and a video explaining the CharM and the characterization of the *Search System* architecture.

We started the interviews by briefly explaining the research objectives and guidelines.

The interview script was composed of three sets of questions: (i) the respondent's experience, (ii) evaluation of the CharM and the architectural characterization generated from it, and (iii) possibilities of architectural evolution based on the result generated by the CharM.

The six interviews took place in July 2021 and were conducted and recorded remotely via Google Meet after Company A and respondents consented. On average, the interviews lasted 38 minutes. In all, the six interviews lasted 3 hours and 51 minutes.

As in the InterSCity case study, here we analyzed the data collected from the six interviews using open and axial coding procedures (Corbin and A. Strauss, 2015; Stol et al., 2016). We started by applying the open coding procedure, from which we mapped some use categories of the CharM. For this, we performed an iterative process of inductively coding the transcription of one interview at a time. Following, we did the axial coding, where we further analyzed and reviewed the interviews to identify relationships between the categories that emerged from the open coding analysis. The author of this thesis conducted the preliminary analyses and multiple meetings with two other experienced researchers (advisor and co-advisor of this thesis) to discuss and increase the results' reliability and mitigate bias (Patton, 2014). Furthermore, throughout the coding process, we adopted the constant comparison method (Glaser and A. L. Strauss, 2017), whereby we continuously compare the results of an interview with those obtained in the previous ones.

Due to confidentiality reasons, we do not share the interviews' transcription. However, we made it publicly the code book available (Appendix D.7).

In December 2021, we presented the final results of the case study to Company A's software engineering team. At the time, more than 60 members participated. The goal of this meeting was to share the main results found. In the end, we received interesting feedback, such as that the characterization generated from the CharM fostered discussions about software architecture and motivated the investigation of attention points and architectural evolutions in the *Search System*.

During the seven months of this case study, we conducted 26 interviews, in which we spoke with 18 different people [5] (16 men and 2 women). In all, were 23 hours and 35 minutes of interviews. In addition, we shared the research progress through four alignment meetings with managers, leaders, and other company employees.

The sample of participants in this case study is non-probabilistic and combines convenience, purposive, and referral-chain types (Baltes and Ralph, 2020). Furthermore, our sample size is aligned with evidence from the anthropology field, which indicates that 10-20 knowledgeable people are sufficient to uncover and understand the core categories in any study of lived experience (Bernard, 2011).

[5] Detailed list of interviewees is available at:
https://docs.google.com/spreadsheets/d/1hKq71rmWQknrrLJsqkcM9F3xfJdrrAH1q59uFeftLy0/edit?usp=sharing

4.2.2 Overview of the Company A's Systems

As already mentioned, Company A is an online handicraft marketplace. Based on the interviews with the 12 team leaders, we mapped that for such a business to work well, there is a set of eight fundamental systems. According to the interviewees, one of these eight systems is considered a monolith and the others follow a service-based approach. This scenario exists because the systems are going through an architectural migration (from the monolithic to the service-based style). One of the primary motivations for this migration was the difficulty of working with the *Monolith*.

Although there has been a reduction in the development flow in *Monolith*, the main business rules are still within it. This system is the main integration point among the other seven systems. In addition, it has a robust relational database, which is also manipulated by the other seven systems and is used as a query source for the data enrichment system. The *Monolith* is also the only one that interacts directly with the old front-end rendering system.

Each of the other seven systems corresponds to specific contexts: *Search*, *Recommendation*, *Categories*, *Payment*, *Shipping*, *Collection*, and *Marketing*. During the interviews, we noticed that the *Search*, *Recommendation*, and *Categories* systems are considered more mature in service-based development. These three systems are also different from the others since they interact with a set of *machine learning models*. *Search System* stands out the most among the seven service-based systems since it was the first context to be separated from the *Monolith* and is currently formed by a larger set of heterogeneous services. For this reason, the *Search System* is the only one among the service-based systems that interacts directly with the *new front-end services* and the *API gateway*.

Communication between the eight systems and other components occurs asynchronously using the *Message Broker*, synchronously, and through a data source. Message Broker is an essential technological component that enables the communication between the systems and from the systems with *external services*, *data lake*, *data warehouse*, and *data enrichment system*. Another crucial technological component is the *API gateway*, created as soon as the architectural migration process started, aiming to help in the *Monolith* strangulation process, being the front door to interact with Company A's systems. Figure 4.15 shows an architecture overview of Company A's systems.

After understanding Company A's main systems, we delimited our case study to the *Search System*. This system is one of the service-based systems with greater maturity and a greater number of heterogeneous services. Therefore, it is suitable for the application and evaluation of the CharM. Thus, in the next section, we present the characterization of the architecture of Company A's *Search System* based on the CharM.

4.2.3 Characterization of the Architecture of the Search System

To characterize the architecture of the *Search System*, we analyzed some documents provided by Company A and carried out a guided exploration of the source code of the eight services through interviews with the team members. As in the InterSCity case study,

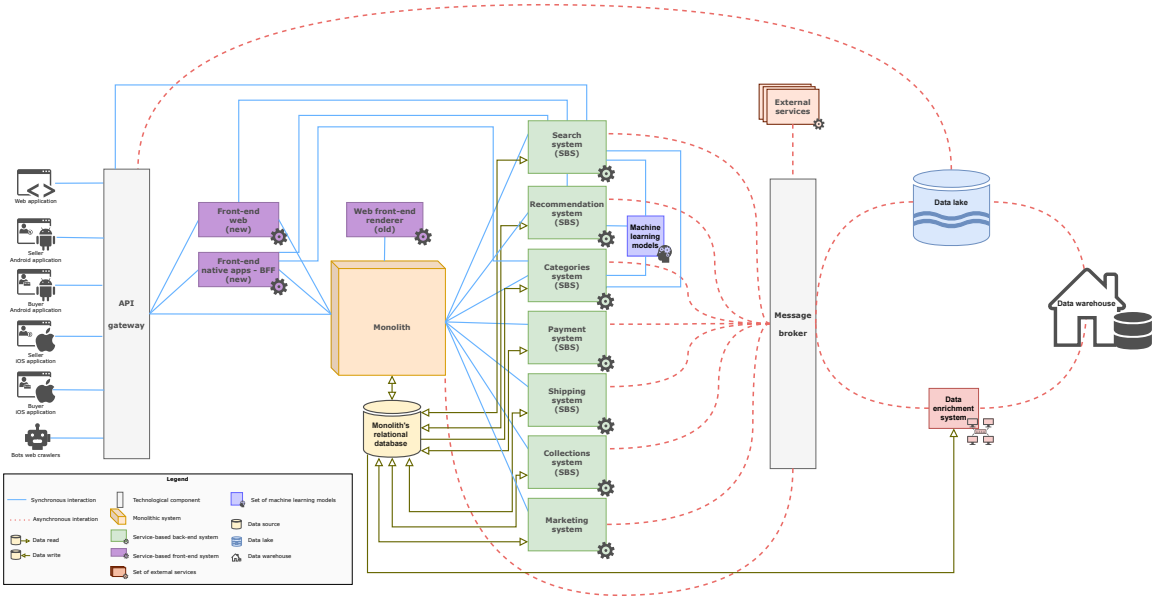


Figure 4.15: *Overview of Company A's systems architecture.*

the metrics collected were arranged in rulers to illustrate the service's profile in each of the CharM dimensions. The scales used in the rulers of each dimension take data from the *Search System* itself as a reference.

The ruler of the size dimension ranges from 0 to 26 (maximum number identified of services operations). The scale used in the data source, synchronous, and asynchronous coupling dimensions varies according to the analyzed perspective. Considering the internal perspective, the scale ranges from 0 to 8 (number of internal services on the *Search System*). The ruler of the external perspective ranges from 0 to 4 (number of external identified components). Withal, the values of the total perspective ruler vary from 0 to 12 (sum of the number of internal services and external components). It is important to note that the ruler for each perspective is exhibited only if there are values and components for analysis. Based on the 5^{th} version of the CharM, in this section, we first present the services' characterization, then the *Search System's* characterization.

Characterization of the Service A: It has a single responsibility. It is one of the services that does not expose operations. This service does not access any data sources and has no synchronous coupling. It is the only service with asynchronous importance and dependency, publishing 1 topic to the message queue, which is consumed by the Service B. And consuming 1 topic from the message queue, published by the technology component Debezium (Change Data Capture Platform). Figure 4.16 illustrates the Service A characterization.

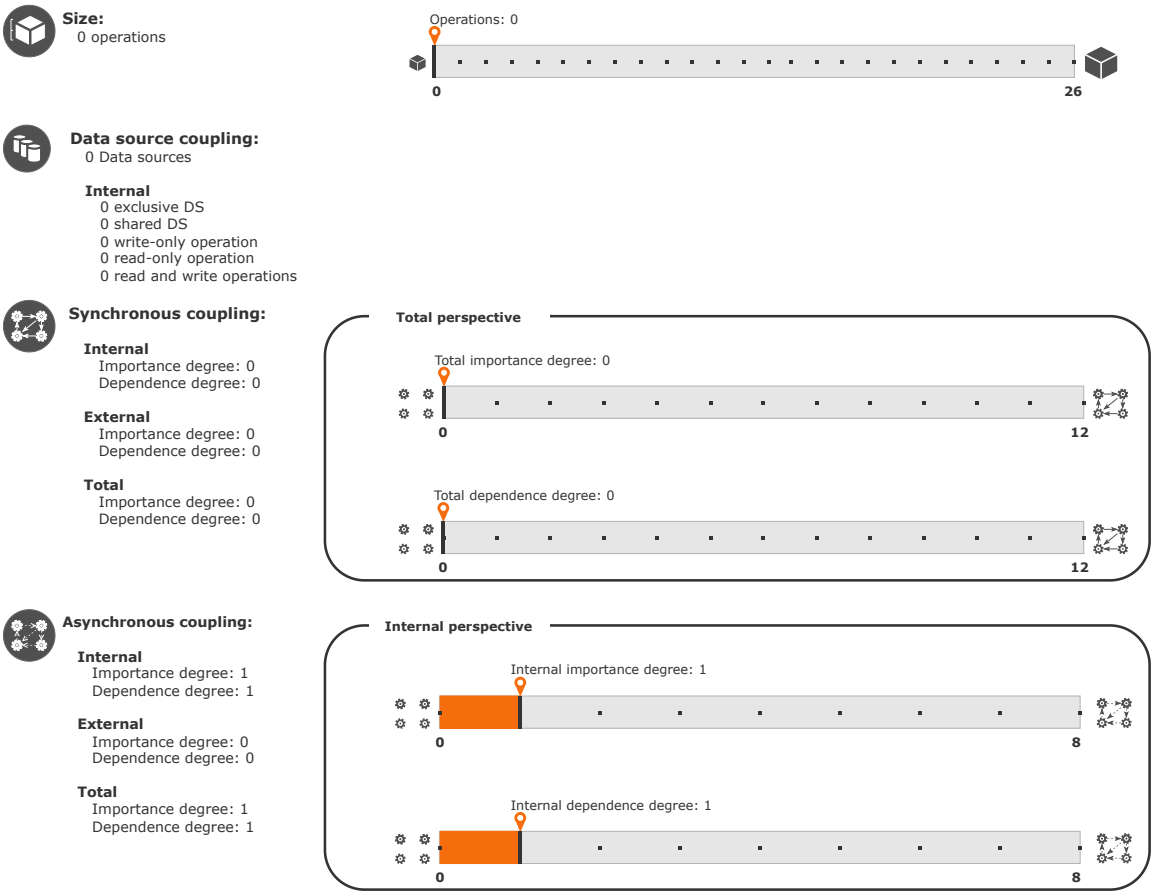


Figure 4.16: *Service A characterization.*

Characterization of the Service B: It has a single responsibility. It is one of the services that does not expose operations. Access 1 data source, performing write-only action. However, it shares access to this data source with 4 other services (Service C, Service F, Service G, and Service H). It has no synchronous coupling. It is one of the few services that performs asynchronous interactions, consuming 1 topic from the message queue, published by the Service A. Figure 4.17 illustrates the Service B characterization.

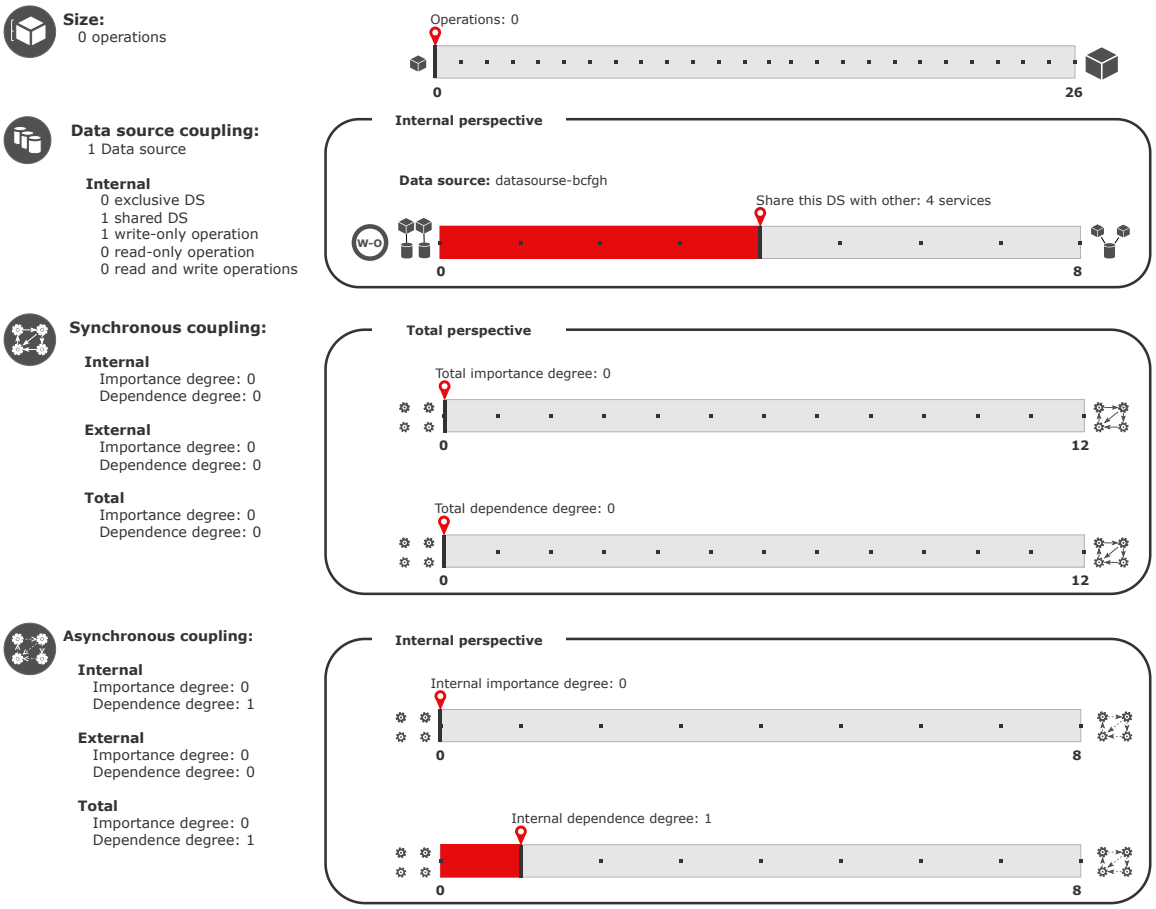


Figure 4.17: *Service B characterization.*

Characterization of the Service C: It has a single responsibility. It is one of the services that does not expose operations. Access 1 data source, performing write-only action. However, it shares access to this data source with 4 other services (Service B, Service F, Service G, and Service H). It has no synchronous coupling. It is one of the few services that performs asynchronous interactions, consuming 1 topic from the message queue, which is published by the Debezium technological component. Figure 4.18 illustrates the Service C characterization.

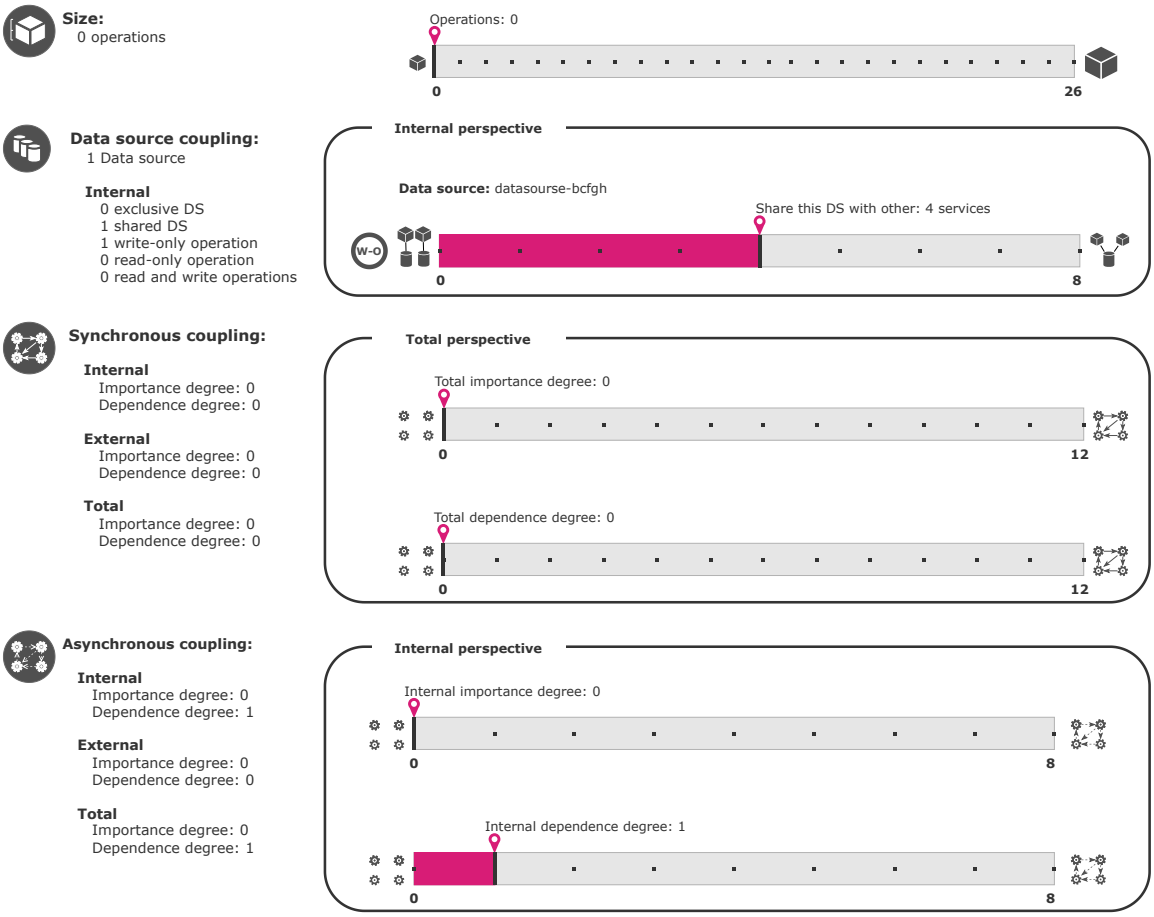


Figure 4.18: *Service C characterization.*

Characterization of the Service D: It has a single responsibility. It is one of the services with only 1 operation. Access 8 data sources. 7 of which are exclusive, and 1 is shared with the Service H. It performs read and write actions in 1 data source and performs read-only action on the other 7 data sources. It has no synchronous coupling. It is one of the few services with asynchronous coupling, consuming 1 topic from the message queue, published by the Service H. Figure 4.19 illustrates the Service D characterization.

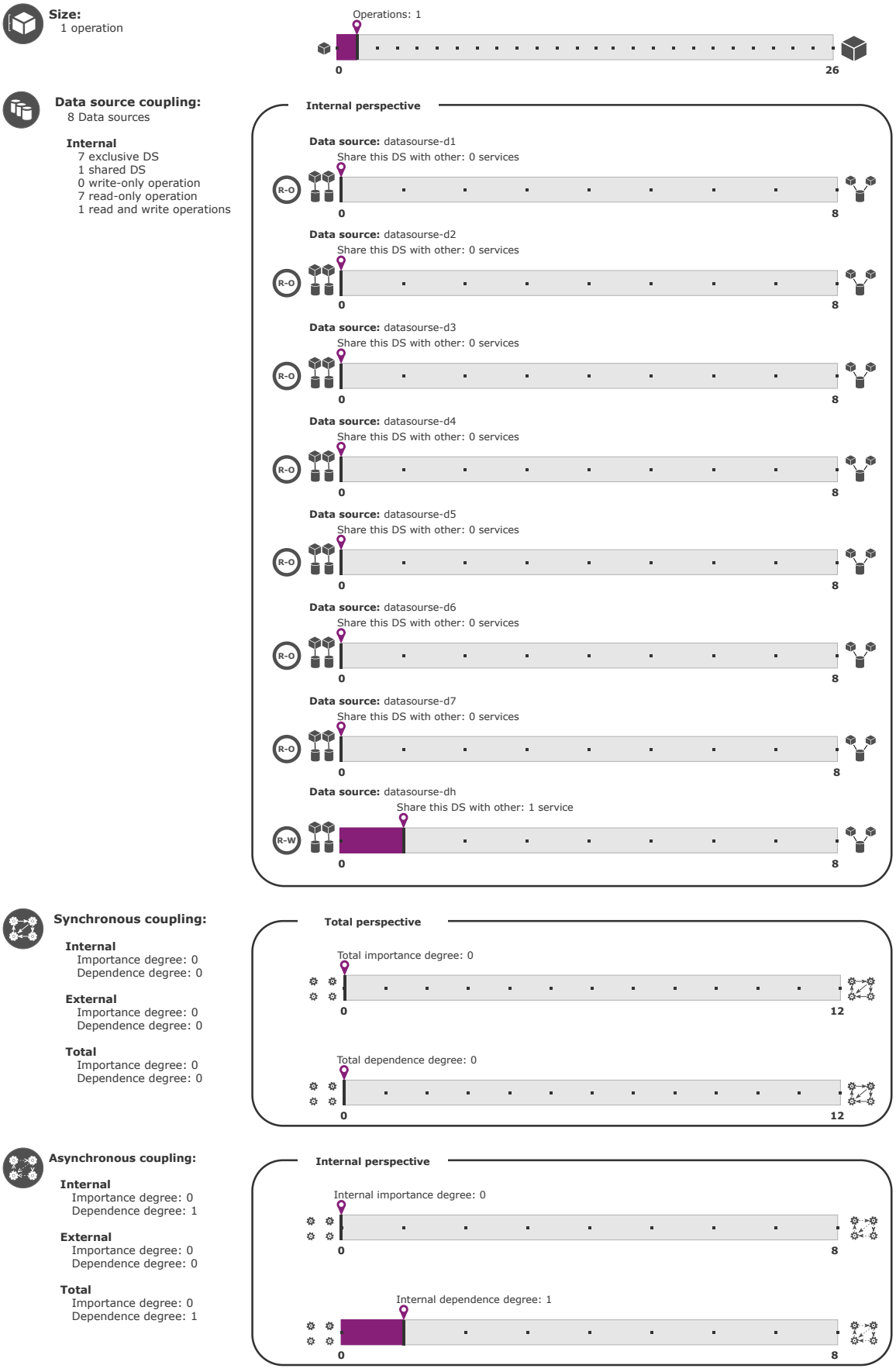


Figure 4.19: *Service D characterization.*

Characterization of the Service E: It has a single responsibility. It is one of the services with only 1 operation. Access 2 exclusive data sources, performing read-only action. It has synchronous coupling only with the Monolith. It has no asynchronous coupling. Figure 4.20 illustrates the Service E characterization.

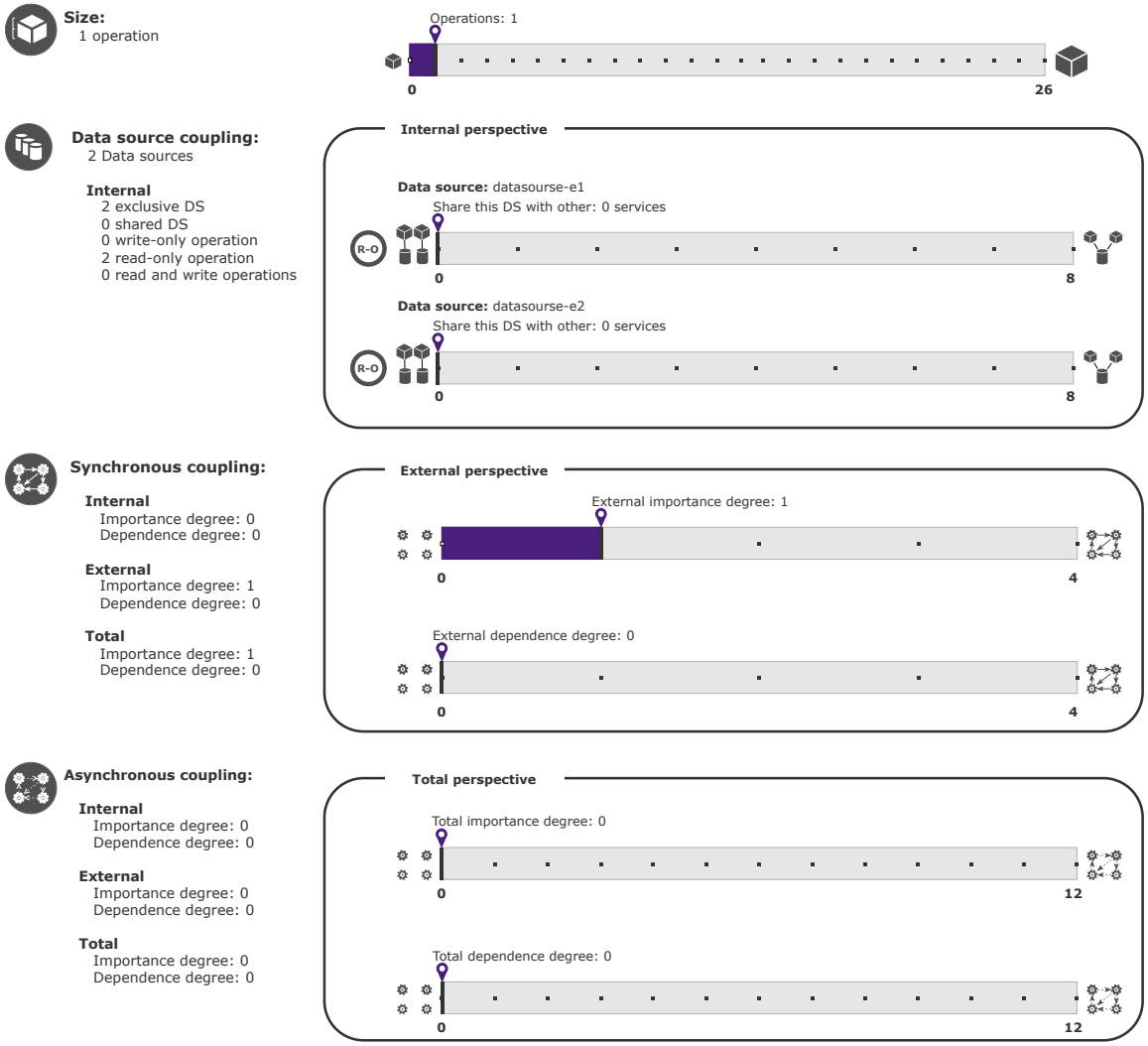


Figure 4.20: *Service E characterization.*

Characterization of the Service F: It has a single responsibility. It is one of the services with only 1 operation. Access 3 data sources, 2 exclusive and 1 shared with 4 other services (Service B, Service C, Service G, and Service H). It has no synchronous dependence. However, its operation is invoked by the Service G and the Service H. It has no asynchronous coupling. Figure 4.21 illustrates the Service F characterization.

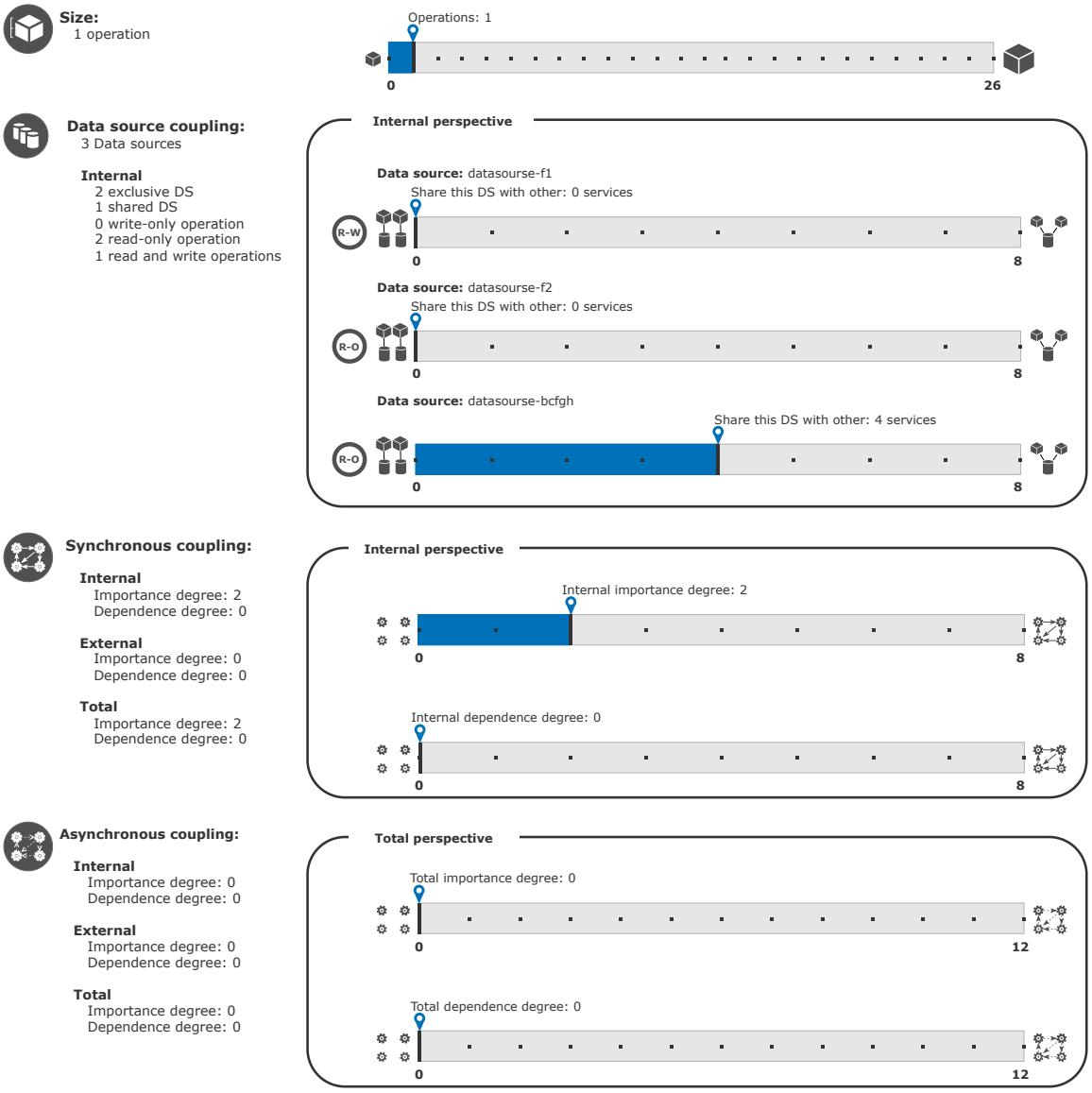


Figure 4.21: *Service F characterization.*

Characterization of the Service G: It has a single responsibility. It is the second-biggest service but has only 2 operations. Access 1 data source, performing read-only action. However, it shares access to this data source with 4 other services (Service B, Service C, Service F, and Service H). Considering the synchronous coupling, internally, it depends on the Service F. And externally, the Monolith invokes it. It has no asynchronous coupling. Figure 4.22 illustrates the Service G characterization.

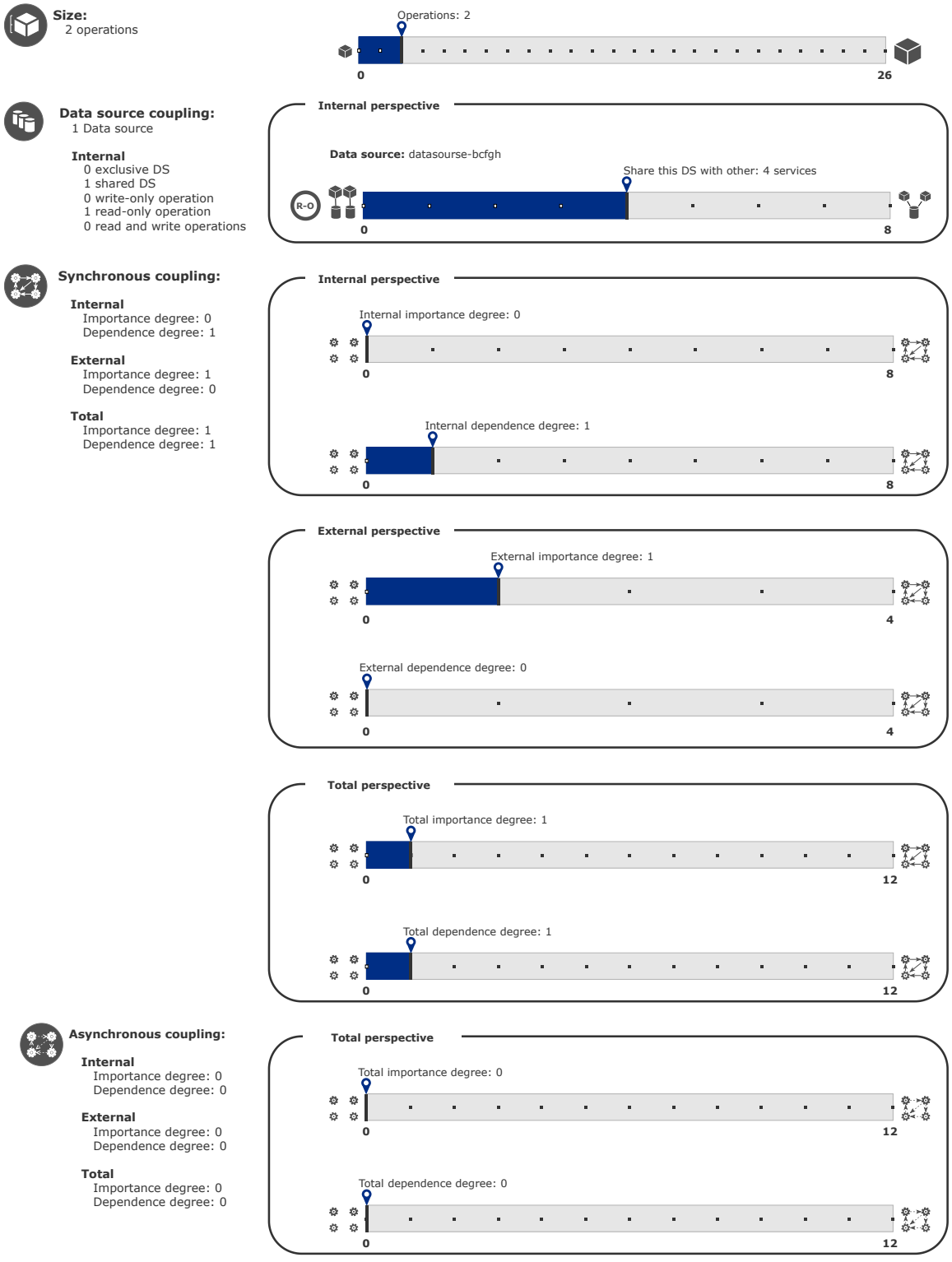


Figure 4.22: *Service G characterization.*

Characterization of the Service H: It has 16 responsibilities and is the biggest service in the system, with 26 operations. Access 7 data sources by performing read-only action. 5 data sources are exclusive, but it shares the other 2. One of these data sources is shared with 4 different services (Service B, Service C, Service F, and Service G) and another with 1 service (Service D). Regarding synchronous coupling, it is the service with the highest total (internal and external) dependence and importance. Furthermore, it has a high coupling (external) with the Monolith since 23 of its operations are invoked by this system. It is one of the few services that performs asynchronous interactions, publishing 1 topic to the message queue, which is consumed by the Service D. Figure 4.23 illustrates the Service H characterization.

Figure 4.23: *Service H characterization.*

After mapping and understanding the profile of each eight services, we could characterize the architecture of the *Search System*. As in the services' characterization, we arranged the collected metrics in rules. The rules of the size and data source dimensions have a scale ranging from 0 to 8 (number of system services). The scale used in the rules of the synchronous and asynchronous coupling dimensions ranges from 0 to 100 to represent the percentage of each type of communication.

Characterization of the *Search System*: When considering the number of services per module, it can be considered a small service-oriented system since each module is composed of a single service. When analyzing the number of operations, the biggest service is the Service H, with 26 exposed operations, while the second-biggest service (Service G) has only 2 operations. The system has 18 data sources, 2 of which are shared between different internal services. One of these data sources is shared by 5 services. Approximately 66.6% of interactions between the services are synchronous. On the other hand, approximately 33.3% of interactions between the services are asynchronous. Figure 4.24 illustrates the characterization of the architecture of the *Search System*, based on the CharM dimensions. Figure 4.25 presents an overview of the system architecture, containing the main structural elements analyzed with the CharM.

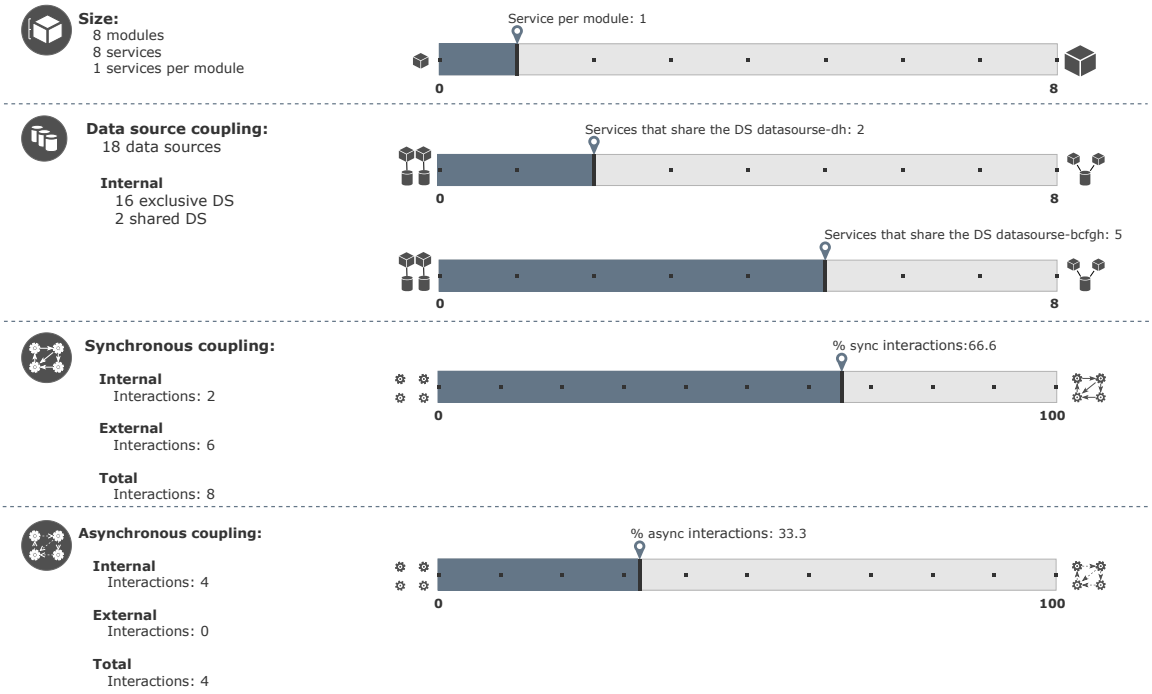


Figure 4.24: *Search System characterization.*

Appendix F complements the characterization of the Company A' *Search System* and its services through visualizations that consider the different CharM dimensions, present different perspectives, and facilitate comparisons.

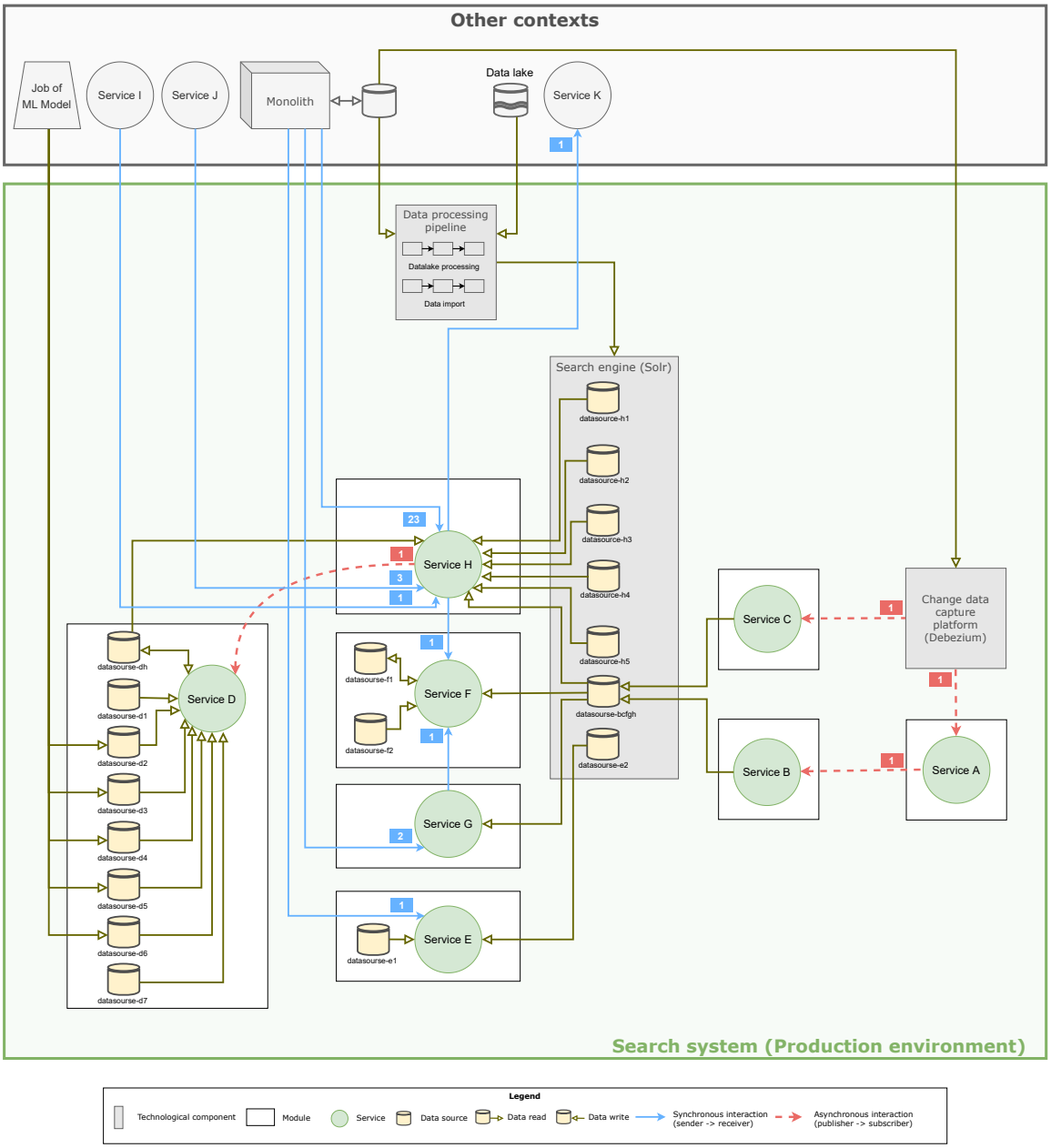


Figure 4.25: *Overview of the Search System architecture (notation inspired by Merson and Jospeh Yoder (2019)).*

4.2.4 Evaluation Results

Participants Profile

Six Company A employees evaluated the CharM. The time of experience (self-declared) in the IT area of the interviewees varies from 3 to 19 years (10 years on average). When asked about their degree of experience with software architecture [6], one considered them self an advanced beginner, two declared themselves competent, two others as proficient, and one as an expert. Their self-declared time of experience with service-based architecture ranges from 2 to 6 years (4 years on average). Related to their performed role, one declared to be a chief architect, four software engineers, and one group leader. Respondents were also asked how much they knew about the architecture of the *Search System* [7]. On a five-point scale ranging from very low to very high, three considered that they have median knowledge, two reported that they have a high degree of knowledge of the architecture of this system, and two others stated that they have a very high degree of knowledge. Table 4.2 details the profile of professionals who evaluated the CharM.

Participant ID	Role	Experience in IT area (years)	Experience with Software Archtecture	Experience with SBA (years)	Knowledge about the Search System architecture
P3	Software engineer	18	Competent	6	Very high
P12	Group leader	8	Proficient	3	Median
P13	Software engineer	13	Proficient	6	Very high
P15	Software engineer	3	Competent	2	High
P17	Software engineer	4	Advanced beginner	2	Median
P18	Chief architect	19	Expert	6	Median

Table 4.2: *Participants profile – Industry case study.*

Evaluation of the CharM

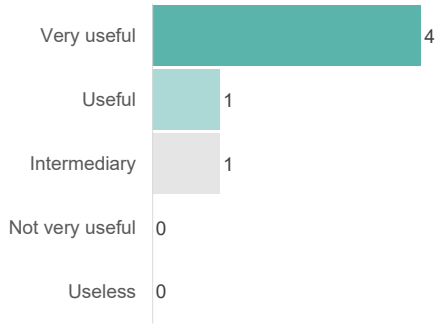
As in the first case study, we asked respondents to evaluate the CharM's usefulness, its ease of understanding, and the coherence of its result. However, unlike the InterSCity case study, we asked participants to evaluate separately the ease of understanding the dimensions and the result generated by the model. In the following sections, we describe the results of this assessment.

[6] We adopted the five stages of skill acquisition proposed by Dreyfus et al. (1986). Thus, in the interviews we defined this fives stages as follows: *novice* - I understand the concept of software architecture, but I have never made architectural decisions in real environments; *advanced beginner* - I have experience in real scenarios and have already contributed to some architectural decision-making, but without much confidence; *competent* - I have already contributed to some architectural decision-making in different contexts and real scenarios, but I still have some difficulties; *proficient* - daily I make decisions related to software architecture consciously and with a good degree of confidence; and *expert* - I deal with different computer systems with different architectures, and I can make decisions confidently and without significant difficulties.

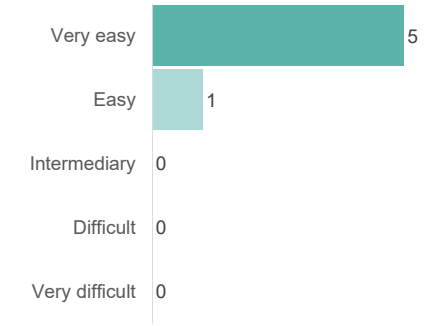
[7] We adopted the following five-point Likert scale for the interviewees to indicate their level of knowledge about architecture: *very low* - I understand the purpose of this system, but I do not know its architecture; *low* - I know a little about the architecture of this system (I know about the existence of some services), but I have not had the opportunity to work in practice yet; *average* - I know the architecture of this system (not in detail) and I have already had the opportunity to work on some services; *high* - I know the architecture of this system and I have had the opportunity to work on most of its services; and *very high* - I work with this system daily and am very familiar with the architecture and services that comprise it.

The Uses of the CharM

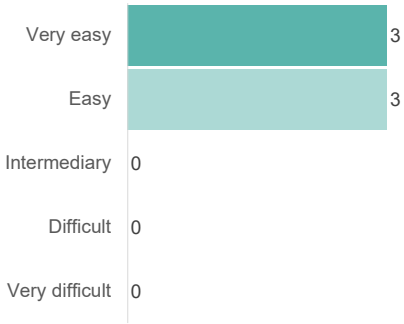
Using a scale from 1 (useless) to 5 (very useful), we asked participants how much our model helped them understand the architecture of the *Search System*. The average score obtained was 4.5. As detailed in Figure 4.26a, five respondents indicated that the CharM was very useful (four – P3, P12, P15, and 17) or useful (one – P18) for understanding the system architecture. Only interviewee P13 considered the usefulness of the model intermediary for this task.



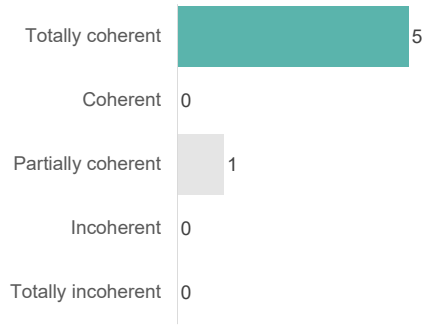
(a) *Usefulness for understanding the architecture.*



(b) *Ease of understanding the dimensions.*



(c) *Ease of understanding the results.*



(d) *Coherence of the architectural characterization.*

Figure 4.26: *Evaluation of the CharM's usefulness, ease of understanding the dimensions, ease of understanding the results, and coherence – Industry case study.*

P13 argued that they were thoughtful when they noticed that, in the characterization generated by the CharM, three services (A, B, and C) of the *Search System* did not have any operations. They commented that they understand that the metric *"number of operations"* of the *Size dimension* of the model only considers exposed endpoints. However, they believe that it would be valuable to expand the concept of "operation" used in the context of the CharM, embracing other types. Despite this reflection, P13 considered that our model was very useful for understanding the architecture in a macro way and could support the onboarding process of new team members.

We then asked the interviewees, in a broader way, what could the usefulness of the CharM be. As illustrated in Figure 4.27, in all, 24 uses were indicated by them. Among these, 14 were also mentioned in the first case study. The most cited uses of the CharM in this second case study were *"understanding architecture"* and *"guiding architectural evolution at a high level of abstraction"*, mentioned by all six interviewees. Considering

these uses, it is worth highlighting some excerpts from the interviews. P3 stated the following *"In systems like this, where there are several components, several services, and interactions between them[…] The model presents a very nice view of things that I, really, didn't pay attention to before"*. P18 said that the CharM *"serves as a kick-start to identify the priorities to analyze in an architectural evolution. It makes it very explicit where the biggest problems are. And from there, you can make the necessary plan"*.

Figure 4.27: *Uses of the CharM – Industry case study.*

The other two most cited uses were *"identify points that need maintenance"* (five - P3, P12, P13, P17, and P18) and *"help novices understand the system architecture"* (five - P3, P13, P15, P17, and P18). Regarding identifying maintenance points, P3 stated the following *"I think the work you've done makes it very clear to see the yellow (or redder) flags for us to act on"*. About the CharM helping novices to understand the architecture, P17 explained the following *"When I joined the team, it was all very confusing for me, because the system is very big and complex. I think with this model, I was able to understand much better who depends on whom, and who is important to whom. So, I think it was really helpful for me[…] thus, I think this model may be very useful in onboarding new members"*.

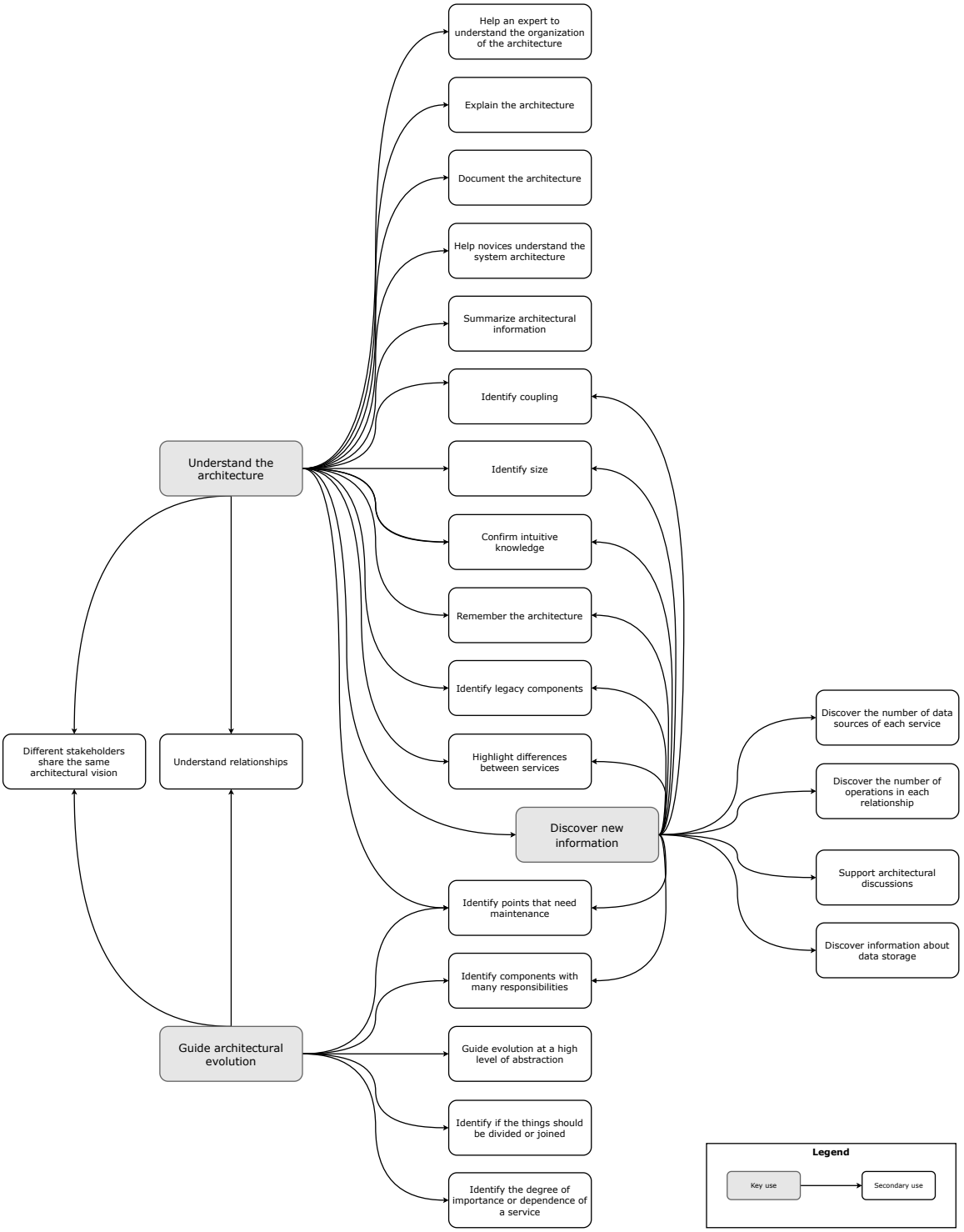


Figure 4.28: *Hierarchical organization of the uses of the CharM – Industry case study.*

It is worth mentioning that, as in the previous case study, the interview protocol had questions directly related to the usefulness of our model for architectural understanding and evolution. Besides that, we asked if the CharM would be useful for discovering something new about architecture. In this case study, four participants (P3, P15, P17, and P18) indicated that the CharM was useful for discovering new things, such as the services'

size and coupling. Figure 4.28 illustrates the hierarchical organization of the CharM's uses mentioned in this case study.

After identifying the uses cited in this case study and the previous one, we performed a new analysis. From this, we reorganized, relabel, and grouped such uses into three main categories: *understanding a service-based architecture*, *service-based architecture maintenance*, and *communicating a service-based architecture to stakeholders*. We carried out this process in three rounds of discussions with the support of a specialist researcher in software architecture. The result of this categorization is shown in Figure 4.29.

Communicating a SBA to stakeholders	- Different stakeholders share the same architectural vision
	- Documenting the architecture of a system
	- Explaining the architecture of a system for novices
	- Explaining the architecture of a system for technical stakeholders
	- Obtaining different views (local and global) of the architecture of a system
	- Summarizing architectural information
	- Supporting architectural discussions
	- Supporting the architecture communication
SBA maintenance	- Identifying structural characteristics that can be adjusted
	- Supporting system quality assessment
	- Supporting the architectural maintenance process
	- Supporting the process of architectural evolution
Understanding a SBA	- Confirming intuitive knowledge about the architecture of a system
	- Discovering new information about the architecture of a system
	- Evaluating the architecture of a system that is being designed
	- Identifying adopted architectural patterns
	- Identifying different characteristics between services of a system
	- Identifying the size of services of a system
	- Mapping the architecture of a system that alread exists
	- Remembering the architecture of a system
	- Understanding the architecture of a system
	- Understanding the asynchronous coupling between services of a system
	- Understanding the data sharing strategy adopted
	- Understanding the synchronous coupling between services of a system

Figure 4.29: *CharM's main categories of uses.*

Ease of Understanding the CharM

In this case study, we investigated the ease of understanding CharM's dimensions as well as its result generated. This investigation indicates that the respondents tend to consider that it is very easy to understand the dimensions of the CharM since the average score assigned was 4.8 (Figure 4.26b) on a scale from 1 (Very difficult) to 5 (Very easy). Of six participants, five (P3, P12, P13, P15, and P18) considered it very easy to understand the dimensions of our model and one (P17) considered it easy. P17 explained that when watching the video that explains the model, they took some time to assimilate all the information. However, when they saw the application of the CharM to characterize the *Search System's* architecture, they understood all the dimensions well.

Concerning the ease of understanding of the result generated by the CharM, the average score was 4.5 (we used the same scale from 1 to 5). As illustrated in Figure 4.26c, three participants (P3, P12, and P13) considered it very easy to understand the result generated by our model and three others (P15, P17, and P18) rated this same aspect as easy. Participants who rated it as easy argued that some elements of the presentation of results could be improved. We are going to explore such elements in the last subsection of Section 4.2.4.

Coherence of the Architectural Characterization

When evaluating the coherence of the *Search System's* architectural characterization, the average score attributed by the participants was 4.6 on a scale from 1 (Totally incoherent) to 5 (Totally coherent). Figure 4.26d illustrates that five respondents (P3, P12, P15, P17, and P18) considered the characterization "totally coherent" with the reality they knew. Only P13 evaluated the characterization as "partially coherent".

We asked P13 what led them to this perception. They explained that they considered the size characteristic of services A, B, and C inconsistent with the reality they knew. They presented the same justification we explored earlier when we presented the uses of the CharM related to the *"number of operations"* metric of the *Size dimension*.

Improvement Suggestions

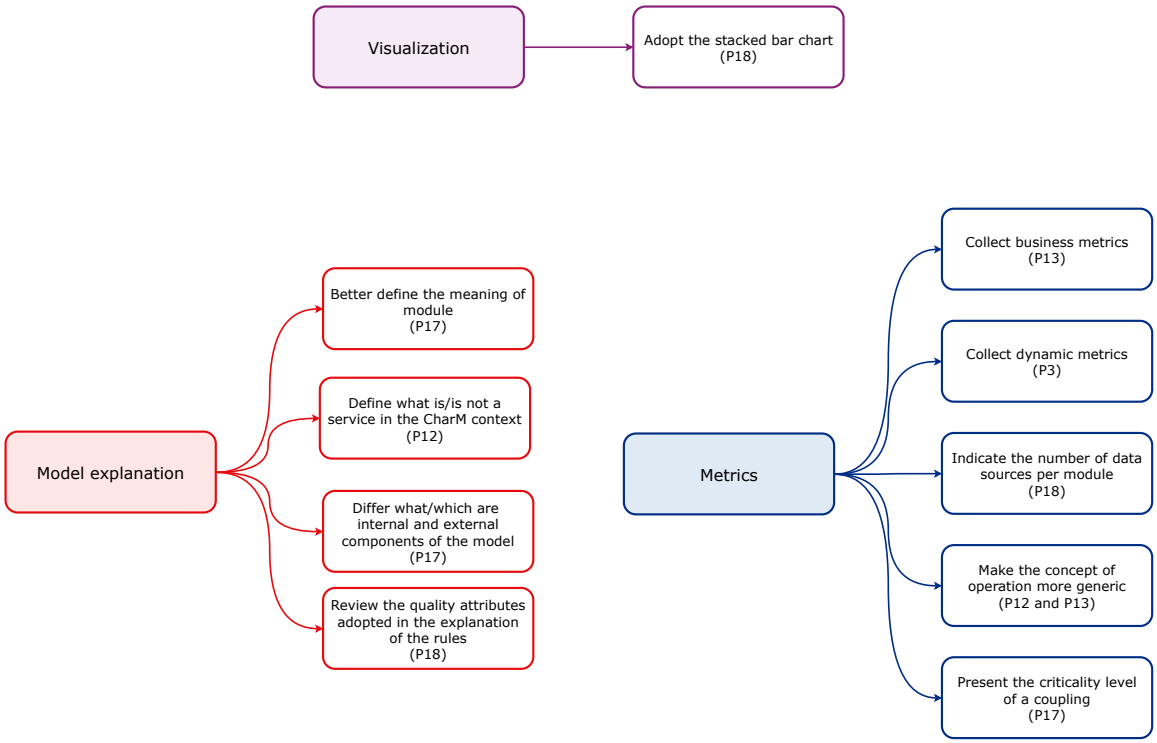


Figure 4.30: *Improvement suggestions to the CharM – Industry case study.*

In order to upgrade the CharM, we asked interviewees for suggestions for improvement. As illustrated in Figure 4.30, we received ten suggestions, which we organized into three

groups. Participant P18 suggested adopting the stacked bar graph to illustrate the size ratio of the components in the system as a whole, thus improving the visualization of the generated result. We have already performed some tests considering this suggestion but have not yet incorporated it into the CharM.

Participants P12, P17, and P18 made four suggestions related to explaining the model. Three of these suggestions were related to the module and service definition, and the other one was to the quality attributes adopted to explain the *Synchronous coupling dimension*. We have already incorporated all these suggestions for improvement.

Most of the suggestions we received were related to metrics. P12 and P13 suggested making the operation concept more generic in the CharM, not limited to just exposed endpoints. We are investigating this suggestion further to verify how we can incorporate it into the CharM. The suggestions for collecting dynamic and business metrics presented, respectively, by P3 and P13, are outside the context of the CharM, but it is a research front included as future work. The other suggestions have not yet been incorporated into the CharM. When comparing with the first case study, the only suggestion for improvement that was repeated was to incorporate dynamic metrics.

4.2.5 Threats to Validity

In this section, we present the main threats to the validity of this case study.

Internal: The author of this thesis carried out all stages of data collection and analysis. Therefore, it is possible that, at some point, there may have been failures in the documents' interpretation or in the communication with the interviewees. This scenario may have affected the architectural characterization and the CharM evaluation result. To mitigate the risk of generating an inconsistent characterization, we reviewed the collected metrics data as well as questioned respondents about the coherence of the result generated from the CharM. As for possible failures in communication, we seek to confirm with the interviewees the interpretation of what was said. Furthermore, after preliminary analyses, the results were discussed with the advisor and co-advisor of this thesis.

External: The participants' backgrounds and experiences may influence our model's evaluation result. However, we consider that having heterogeneity of profiles is favorable for the CharM assessment process. This situation allows us to verify its usefulness for novices to experts, as well as for professionals with different levels (from low to high) of knowledge of the architecture of the studied system.

Construct: The scripts of the interviews were developed based on the evolution of the script adopted in the first case study. Furthermore, the scripts were discussed and reviewed by the advisor and co-advisor of this thesis and by professionals from *Company A*. This review process, combined with pilot tests, was useful in finding a better way to formulate the questions and reducing possible biases. Besides that, during the interviews, the interviewer could explain and resolve any doubts in case the interviewee had difficulties interpreting any questions.

4.3 Multiple Case Study Discussion

By adopting multiple case studies as a research strategy, we could deeply evaluate different aspects of the CharM. From this, we obtained valuable answers to the three research questions posed at the beginning of this chapter.

RQ1: *What uses of the CharM are perceived by participants?* Considering the two case studies in all, respondents perceived 28 uses for our model. Between these uses, we mapped six that were most frequently cited, *(i) guide evolution at a high level of abstraction* and *(ii) understand the architecture*, both with 15 citations, *(iii) identify points that need maintenance* with 12 citations, *(iv) discover new information* with 10 citations, and *(v) identify size* and *(vi) understand relationships*, both with 9 citations. From a refined analysis, we merged the CharM's uses into 24 and grouped them into three categories that emerged: *understanding a service-based architecture*, *service-based architecture maintenance*, and *communicating a service-based architecture to stakeholders*. The categories of uses that we mapped indicate that the CharM, similar to approaches, such as MicroART (Granchelli et al., 2017a), MAAT (Engel et al., 2018), MiSAR (Alshuqayran, Ali, et al., 2018), and MM4S (Bogner, Wagner, et al., 2017b), somehow contributes to professionals (with different levels of experience) understanding and maintaining the architecture of a service-based system. Such approaches are going to be explored in Chapter 6.

Furthermore, we obtained evidence that our model can support discussions and reflections and, consequently, help for more grounded and mature architectural decision-making, especially those related to size and coupling attributes. Aligned with this reasoning, according to Perepletchikov et al. (2007) and Bogner, Wagner, et al. (2017a), knowing structural characteristics may help maintain and improve the quality of a system. Additionally, one of the studies by Bogner, Wagner, et al. (2017a) points out that the structural attributes of size and coupling are among the most commonly found characteristics in the literature on service-based systems' maintainability.

RQ2: *What is the participants' perception of the architectural characterization generated from the CharM?* Interviewers in the two case studies tended to consider that it was easy to understand the result generated by the CharM. Despite this, it is essential to consider that, to interpret the generated characterization correctly, the professional must first know and understand our model's dimensions and metrics and have basic knowledge of service-based architecture. Thus, evaluating the ease of understanding the model's structure and the result generated by the CharM is relevant because this aspect can influence its acceptance (Davis, 1989). Furthermore, this evaluation may help us identify points that should be simplified or better explained in the CharM. It can also reveal professionals' profiles for whom the CharM may be more appropriate.

Another perception of the participants is that the characterization generated by the CharM was coherent with the reality they knew. This is evidence that the metrics adopted in each dimension of the CharM are appropriate to characterize the architecture, considering the attributes of size and coupling. However, this does not mean that the set of dimensions and metrics adopted is the only suitable one.

Developing an approach that is easy to understand and generates coherent results is critical to making it more useful and fostering its adoption.

RQ3: ***What aspects of the CharM can be improved?*** In both case studies, we received suggestions for metrics to be added to the CharM, as well as to improve the explanation of some aspects of the model and the presentation of results. Besides that, in the first case study, the interviewees suggested that we should add some information they considered missing and would be adding value to the CharM.

The suggestions for improvements helped us enhance our model from the first to the second case study and made us reflect on different aspects, such as scope delimitation, target audience, data collection, and presentation strategy.

From a practical point of view, given the difficulty in understanding and characterizing the architecture of service-based systems, perceived in reports, such as those made by Amazon (Kolny, 2023), Mendonça et al. (2021), Segment (InfoQ, 2020), and Uber (Highscalability, 2020), the results obtained from the two case studies indicate that the CharM may contribute to face the challenges related to this nebulous context. However, it is still relevant to evolve this model, apply it in other real environments and carry out broader studies to evaluate it such as a survey.

From a research perspective, adopting the multiple case studies method allowed us to identify and discuss the usefulness of the CharM and to comprehend and reflect on other important aspects, such as ease of understanding and coherence of the generated results. During the research, we also noticed that the CharM helped us to identify and understand some challenges related to the service-based systems' metrics collection process.

4.4 Chapter Summary

This chapter described how we conducted case studies in two service-based systems, one developed in an academic environment and the other in an industrial environment. During such case studies, we applied the CharM to characterize the architecture of the two studied systems and discussed the generated results. From this, we investigated deeply and found 24 uses for our model, and also identified that professionals with different profiles tend to consider that CharM's structure and generated results are easy to understand. We also identified limitations and mapped and implemented suggestions for improvement in the CharM. At the end of this chapter, we answered research questions RQ1, RQ2, and RQ3.

Chapter 5

Survey

In this chapter, we present the results of the third and final evaluation carried out through a survey. Based on this, we defined the following research questions (RQs):

- **RQ4:** ***To which extent does the CharM support understanding a service-based architecture?*** — Considerations: we are interested in verifying the usefulness of the CharM for different aspects of understanding the architecture. From identifying and knowing its elements, interactions, and characteristics to the adopted patterns and the related trade-offs.

- **RQ5:** ***To which extent does the CharM support a service-based architecture maintenance?*** — Considerations: we want to analyze the usefulness of the CharM for corrective, evolutionary, and preventive maintenance. Thus, some of the considered points are the identification of adjustments and the quality assessment of the architecture.

- **RQ6:** ***To which extent does the CharM support communicating a service-based architecture to stakeholders?*** — Considerations: we are keen to validate the usefulness of the CharM in disseminating architectural information, helping tasks, such as study, explanation, documentation, summarization, and discussion about the architecture.

- **RQ7:** ***How easy is it to understand the CharM?*** — Considerations: we are interested in finding out the level of ease of understanding of the CharM, considering its dimensions and metrics and the results it generates.

- **RQ8:** ***Does the participants' experience influences the perceived usefulness and ease of understanding of the CharM?*** — Considerations: We want to verify if the time of experience with service-based architecture and the level of experience with microservices affect the participants' perception of the different uses of the CharM and the ease of understanding of this model.

In this chapter, we describe the methodological aspects of the survey and present and discuss the generated results.

5.1 Methodology

We used an online survey targeting professionals working with the architecture of service-based systems. Furthermore, the collected data were analyzed quantitatively and qualitatively. Figure 5.1 shows an overview of the research design employed in the survey.

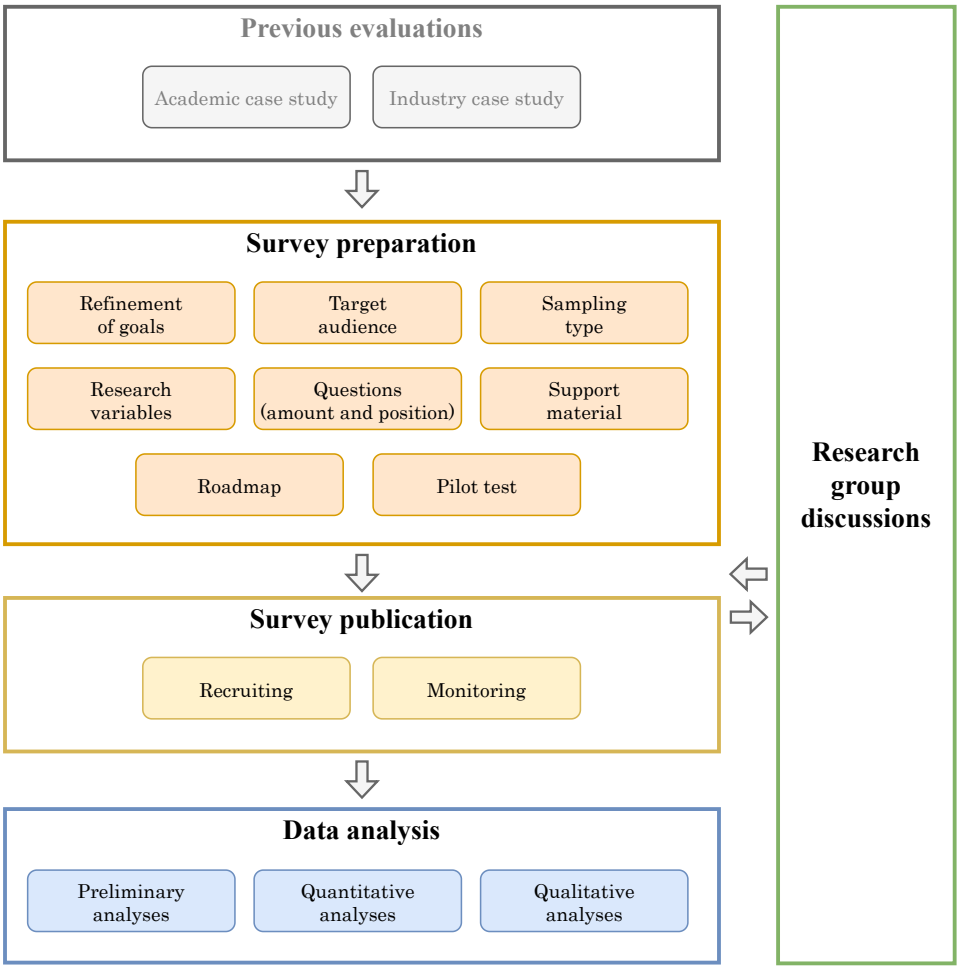


Figure 5.1: *Overview of the research design of the survey.*

As already described (Section 3.1), before the survey, we carried out previous evaluations of the CharM based on a case study in an academic context and another in the industry (Chapter 4). The obtained results in these two case studies were used as input to structure the survey.

During the preparation phase, we used materials (Callegaro et al., 2015; Baltes and Ralph, 2020) that enable us to plan and develop a survey with appropriate strategies that was, at the same time, complete, simple and objective. The main tasks performed at this phase were related to the discussions and refinement of goals, delimitation of the target audience that would respond to the questionnaire, and choosing the most appropriate strategy for selecting the population sample (Section 5.2). We also discussed and delimited the independent and dependent variables of the study (Section 5.3), developed the support

material (videos and descriptive documents), and defined the questions and roadmap for the survey presentation (Section 5.5). In the end, we carried out a pilot test round with three professionals and adjusted the survey based on their feedback.

The next phase was the survey publication, which was available for four months. The main activities carried out were recruitment, which involved disseminating the survey to professionals working with the architecture of service-based systems, and the constant monitoring of the responses received.

As we received answers, we performed some preliminary analyses to identify relevant initial insights and evidence of a state of saturation. After four months, the number of responses stabilized in 58, even with divulgation efforts. Therefore, we move on to the data analysis phase using both a quantitative and a qualitative approach. More details about data analysis are presented in Sections 5.4 and 5.7.

5.2 Sampling

The sample of participants in this study is non-probabilistic and combines convenience and referral-chain types (Baltes and Ralph, 2020). We adopted the open invitation strategy (Wagner et al., 2020). Thus, the survey was sent out via email to professional contacts in the software development area and working in industry or academia. Furthermore, we shared it on social networks, forums, and discussion lists related to software development and architecture. Recipients were invited to answer the survey and forward it to their peers. In an attempt to obtain the participation of professionals with the desired profile, we present a brief description of the profile of the target population in the research invitation. We adopted these strategies to have a quick send-out and obtain a diverse set of participants. The survey was viewed 446 times (unique clicks) in 29 countries [1].

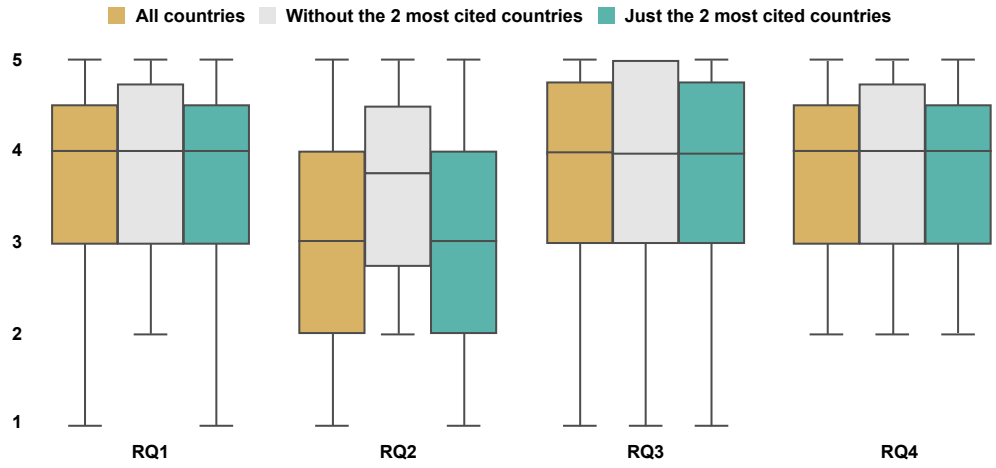


Figure 5.2: *Relationship between participants' resident country and CharM's evaluation.*

[1] These data was extracted from Rebrandly. List of countries where the survey was viewed: Australia, Brazil, Canada, Chile, Colombia, France, Germany, Ghana, Hong Kong, Hungary, India, Italy, Kenya, Mexico, Nepal, Netherlands, Pakistan, Portugal, Romania, Russian Federation, Slovakia, South Korea, Spain, Sweden, Switzerland, United Arab Emirates, United Kingdom, United States of America, and Viet Nam.

In the end, we received 58 valid answers from participants across ten countries. We found that 77% (45) of respondents are concentrated in two countries. Concerned with verifying whether the language background and the questionnaire distribution would affect the survey results, we grouped the evaluation results, considering only these two countries with the highest number of participants (both with the same native language but from different continents). As Figure 5.2 illustrates, by ignoring the most frequent countries (grey box), we notice that the data distribution has not changed drastically, maintaining the trend of positive evaluation of the different aspects of the CharM.

5.3 Research Variables

Since we want to evaluate the proposed characterization model, we present a list of uses and ask the participants to which extent they consider the CharM to be useful for achieving each one. Furthermore, we wondered how easy it is to understand the model's dimensions, metrics, and results. Therefore, the dependent variables of this study are the *uses* and *ease of understanding of the CharM*. The independent variables are related to the participants' profile, considering the *self-declared time of experience with service-based architecture* and the *self-declared level of experience with microservices*.

The possible uses of the CharM were identified in the two preliminary case studies. The first case study was in an academic environment and the second was in the industry environment. In each case study, we applied the CharM to characterize the architecture of a service-based system. Then, the professionals who worked with the selected systems were invited to analyze the generated results and evaluate the usefulness and ease of understanding of the CharM. In the end, we identified 24 uses for the CharM (Figure 4.29). From this set, we included 21 uses in the survey [2] and organized them into three groups: *(i) understanding of a service-based architecture*, *(ii) maintenance of a service-based architecture*, and *(iii) communication of a service-based architecture to stakeholders*. From this, we extracted the dependent variables of this study.

We suppose that the usefulness and ease of understanding of the CharM could vary according to the user's experience. Therefore, we mapped the respondents' self-declared experience in two aspects. The first independent variable concerns the time of self-declared experience with service-based architecture. In this aspect, we do not use any specific reference or scale. So, we decided to use the following range: (a) *Less than 1 year*, (b) *1 to 3 years*, (c) *4 to 6 years*, (d) *7 to 9 years*, (e) *10 to 12 years*, (f) *13 to 15 years*, and (g) *15 years or more*. The other aspect analyzed was the self-declared level of experience with Microservices. For this, we adopted the five stages of skill acquisition proposed by Dreyfus et al. (1986): *novice*, *advanced beginner*, *competent*, *proficient*, and *expert*. Based on this, each level was defined as follows: *novice* - knows the microservice style's theory and principles and applies it in simple contexts; *advanced beginner* - has practical experience of adopting microservice style in real scenarios; *competent* - has experience of adopting

[2] We did not include three previously mapped uses for the CharM in the survey. We believe that to obtain a higher quality assessment of these uses, the respondents should have known and worked with the analyzed system before evaluating the CharM. Uses not included were: confirming intuitive knowledge about the architecture of a system; discovering new information about the architecture of a system; e remembering the architecture of a system.

microservice style in different real contexts and scenarios; *proficient* - deals with different microservice-based systems and can make decisions in a conscious manner and with a high degree of assertiveness; and *expert* - deals with different microservice-based systems and can make decisions in an assertive way and without significant difficulties. Figure 5.3 illustrates the dependent and independent variables of this study.

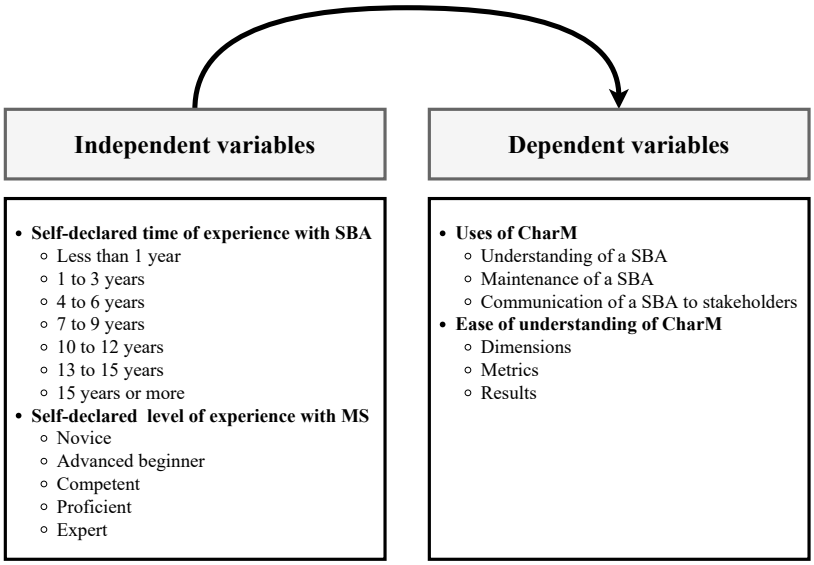


Figure 5.3: *Research variables.*

5.4 Data Analysis

At first, we analyzed the participants' profile (Section 5.7.1) to identify whether we obtained a varied sample. Next, we quantitatively and qualitatively analyzed participants' responses for each of the three CharM use groups (Section 5.7.2). At that moment, we also tried to identify if any profile was predominant, which, in some way, influenced the evaluation of the use groups. Finally, we also quantitatively and qualitatively analyzed the participants' responses regarding the ease of understanding of the CharM (Section 5.7.3). Once again, we verified if any profile of participants tended to consider the CharM either easier or more difficult to understand. Quantitative analyses were based on descriptive statistical techniques such as percentages. During the qualitative analyses, we examined the answers to the open questions. As we analyzed these answers, we organized the data into categories and checked for crossovers and associations. The objective was to identify insights that complement and help us explain the quantitative data. In the final analysis, answers marked as "I have no opinion about this point" were discarded since they did not reveal a positive or negative perception of using the CharM.

5.5 Execution and Replication

During this research, a semi-open questionnaire was developed, in which most questions were close-ended. We designed this questionnaire to be complete, simple, and objective. During its elaboration, there were constant discussions between the author of

this thesis, the advisor, the co-advisor, and an experienced researcher in service-based architecture, to refine and optimize different aspects of the questionnaire. Furthermore, we submitted it to a test phase to validate, when three professionals with the appropriate profile were invited to answer and evaluate the survey and then present their impressions and suggestions for improvement. The questionnaire was built using Google Forms and was available to the public from 2021-08-14 to 2021-11-30, during which time we received 58 valid responses.

The questionnaire was organized into seven sections. We provided a CharM presentation video [3] explaining its dimensions and metrics in the first section. In the second section, we presented a video [4] that demonstrated the result of the application of the CharM in a fictitious system. In the third section, we collected participants' demographic data. The focus of the fourth section was to collect information on the respondents' professional profile related to academic background, professional performance, and experience with service-based architecture and microservices. Next, we asked about the participants' architectural interests. In the sixth section, we presented a list of 21 uses and asked participants to signalize to which extent the CharM would be useful in each one. In this same section, we asked about the ease of understanding of the CharM's dimensions, metrics, and results. In the last section, the participants could present suggestions for improving the CharM and register general comments.

A replication package is openly available in our Zenodo repository [5], including the questionnaire, the dataset with the valid answers, and the quantitative analysis script.

5.6 Limitations and Threats to Validity

Internal: By adopting a survey as the CharM evaluation methodology, we limit ourselves to obtaining answers that reveal only the respondents' perceptions, which may not represent the opinion of the total population of professionals who work with service-based architecture. Therefore, to mitigate this threat, we developed a survey based on results that we collected in two previous case studies. This strategy contributed to directing the construction of the survey with really relevant information. Another possible threat is related to the quality of the received answers since participants could need additional explanations to interpret and answer the survey questions adequately. To minimize this risk, we submitted the survey to pilot tests to assess the ease of interpretation of the questions. Furthermore, we presented didactic videos explaining and applying the CharM to contribute to a better understanding of the model and related questions.

External: Considering that the survey was publicly available, using the open invitation strategy, no previous mechanisms were adopted to validate the participants' identities. Thus, we risk obtaining answers from professionals who do not have the profile and knowledge necessary to respond to the questions. To mitigate this threat, we briefly described the target population profile in the survey invitation and collected demographic information to identify if the participants' profile was adequate to answer the survey.

[3] https://youtu.be/VQRqG9hLBSQ
[4] https://youtu.be/bK9Yg9jmQXY
[5] https://doi.org/10.5281/zenodo.6590692

Construct: We adopted protocols for design, execution, and quantitative analysis to facilitate the survey replication. However, the open questions were analyzed qualitatively, which allows for subjective interpretations of some results and discussions.

5.7 Evaluation Results

In this section, we describe the results of the evaluation of the CharM, obtained through a survey. The main points that we explore are participants' profile and the uses of the CharM and its ease of understanding.

5.7.1 Participants Profile

Most participants identified themselves as male (86%), despite the low female participation rate, the numbers are aligned with a recent global census, which evidences that most software development professionals are males (Statista, 2022). We received responses from participants located in different world regions, but most live in Latin America (62%) or Europe (31%). Regarding education level, most participants claimed to have a master's degree (55%) or an undergraduate degree (29%). The participants general profile is presented in Figure 5.4.

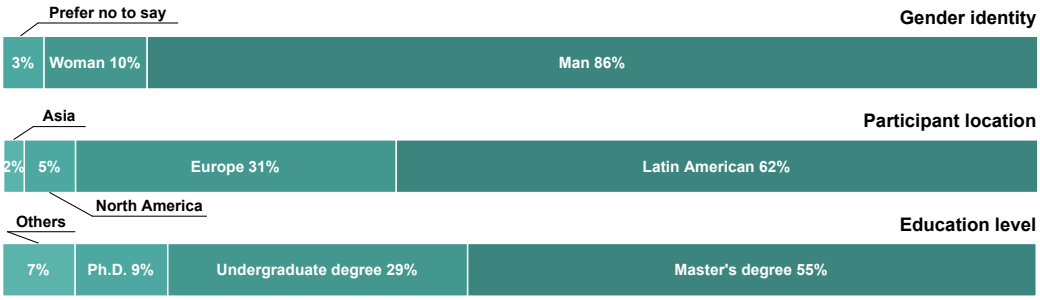


Figure 5.4: *Summary of participants general profile, according to gender, location, and education level. From top to bottom, we present the (a) gender identify, (b) participant location, and (c) highest completed education level. All percentages were rounded based on simple rules of rounding numbers.*

As illustrated in Figure 5.5, most respondents worked in the industry (79%) in the last two years, which is relevant for this research since we can obtain an assessment from a more practical point of view. Related to the role they perform, most declared to work as a developer (59%) or an architect (45%). The main sectors where survey respondents work are Financial (24%), Corporate (22%), Research (22%), Education (21%), and E-commerce (17%), which demonstrate a good range of sectors, which is also desirable for this research. Participants work in different technological layers, most focusing on the back-end (86%), infrastructure (53%), and web front-end (50%) layers. It is worth mentioning that in these questions, the participants could select multiple answers.

Regarding the self-declared experience with service-based architecture, we received a mixed set of responses from participants with different times of actuation, from less than 1 year to more than 15 years of experience. Respondents also have experience with

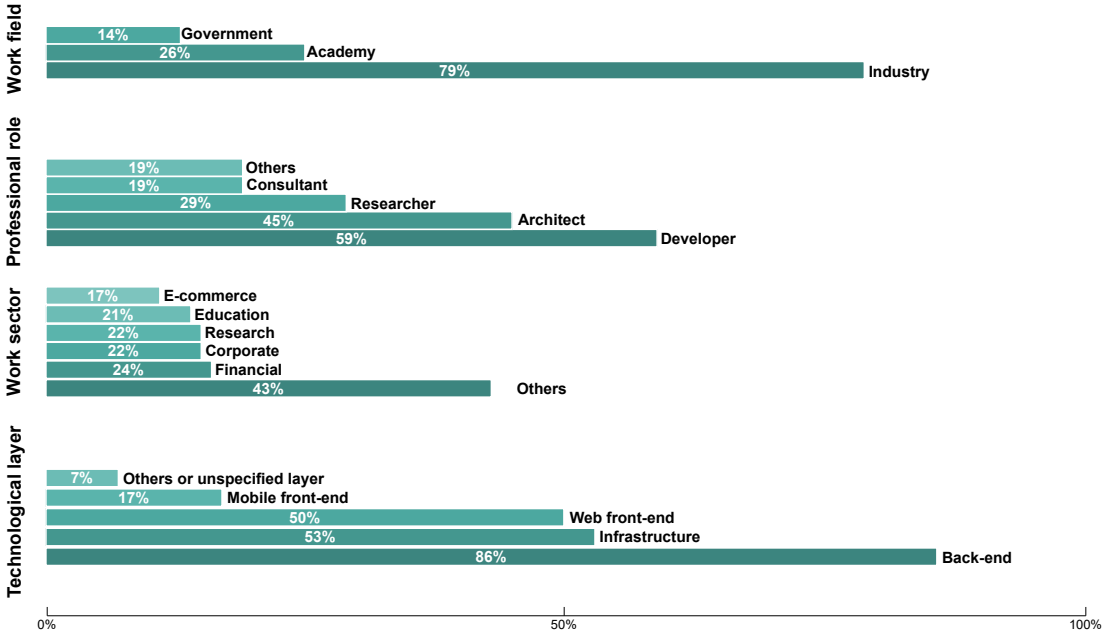


Figure 5.5: *Summary of participants professional actuation profile. (a) work field in the past 2 years, (b) professional role in the past 2 years, (c) work sector in the past 2 years, and (d) technological layer in the past 2 years. The sum of percentages exceeds 100% as participants could select multiple answers. All percentages were rounded based on simple rules of rounding numbers.*

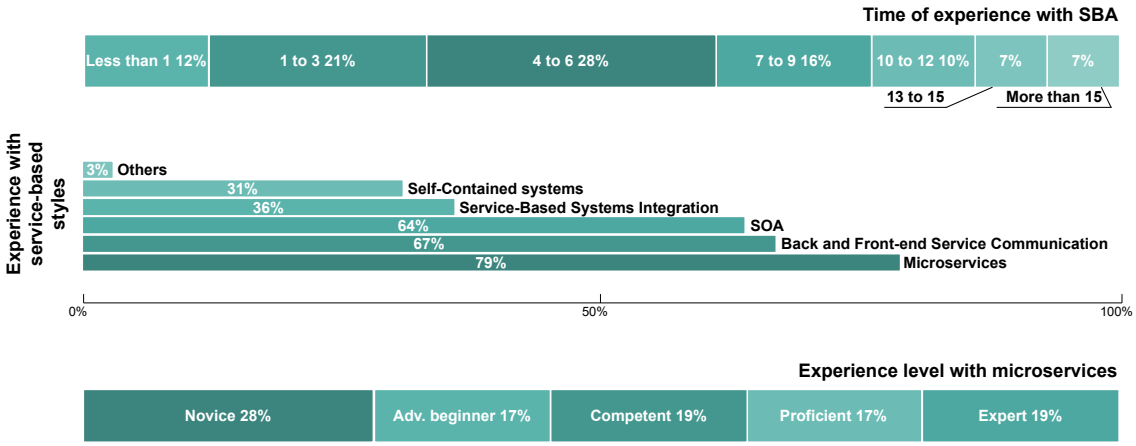


Figure 5.6: *Summary of participants self-declared experience with service-based architecture. (a) range of time of experience with service-based architecture, (b) experience with different service-based architectural styles - it was possible to select multiple layers, and (c) experience level with Microservice-Based Architectural Style. All percentages were rounded based on simple rules of rounding numbers.*

different service-based architectural styles, where the main are microservices (79%), back-end and front-end service communication (67%), and SOA (64%). In this question, the participants also could select multiple answers. We also obtained a relatively balanced number of respondents with different self-declared experience levels with the Microservices Architectural Style, since 28% declared themselves as a novice, 17% an advanced beginner, 19% a competent, 17% a proficient, and 19% an expert. Therefore, considering the variety of profiles and the number of survey participants, we have a good sample to evaluate the CharM. Figure 5.6 summarizes the self-declared experience of the participants with service-based architecture.

5.7.2 The Uses of the CharM

We presented a list of 21 possible uses for our model and asked participants which ones the CharM would be useful for. Based on previous investigation (described in the Chapter 4), these uses were organized into three groups: ***(i) understanding of a service-based architecture; (ii) maintenance of a service-based architecture; and (iii) communication of a service-based architecture to stakeholders***.

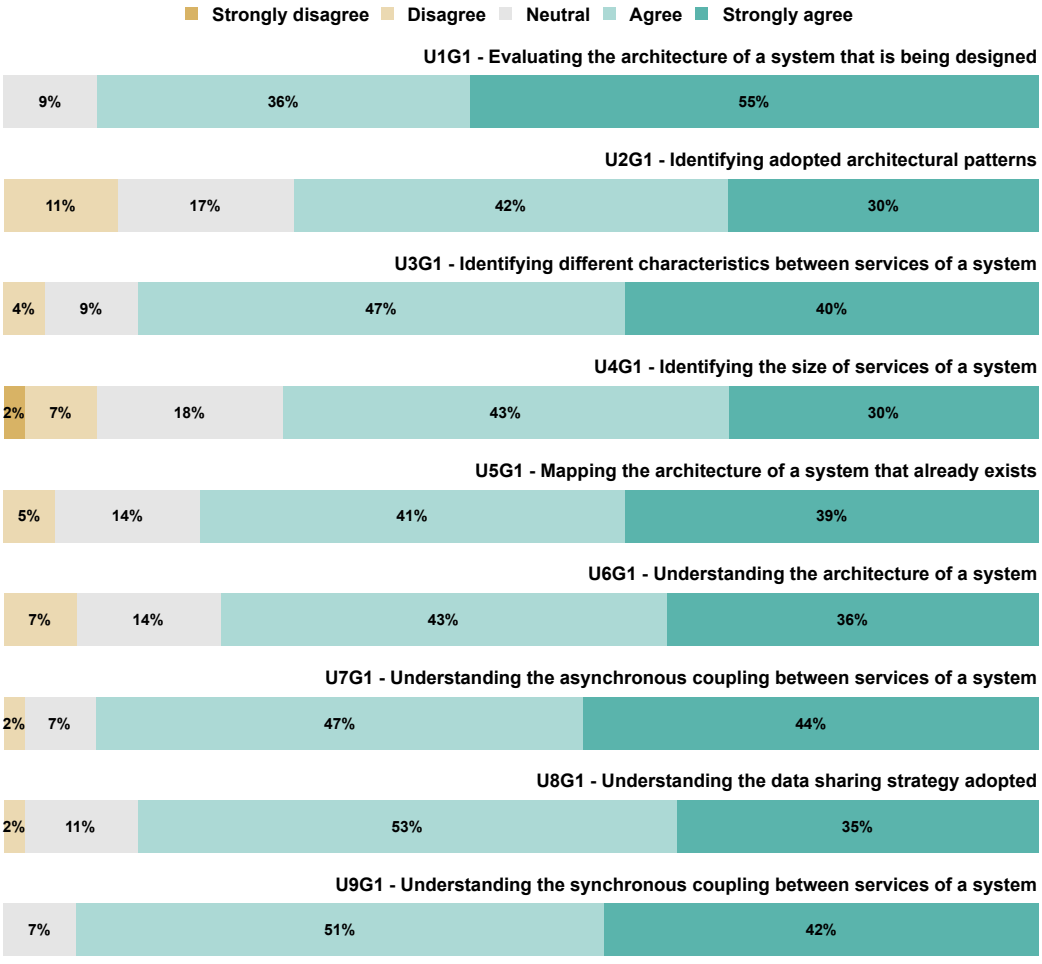


Figure 5.7: *Participants' perspective to which extent the CharM supports understanding a service-based architecture. All percentages were rounded based on simple rules of rounding numbers.*

As illustrated in Figure 5.7, most of the participants who gave their opinion[6] "agree" or "strongly agree" that CharM is useful for ***understanding the architecture of a service-based system***. Using it for *understanding the asynchronous/synchronous coupling* (U7G1 and U9G1) and for *evaluating the architecture of a system that is being designed* (U1G1) stand out, since over 90% of the participants consider that the CharM helps achieve these goals. In the case of uses U1G1 and U9G1, no participant disagreed that the CharM helps achieve them. On the other hand, using the CharM for *identifying adopted architectural patterns* (U2G1) and *identifying the size of services of a system* (U4G1) had the lowest levels of agreement since just over 70% of the participants claimed that the CharM helps achieve these goals. These last two uses also stood out, since ≈10% of the participants believe that the CharM is not useful to achieve these goals.

Based on the respondents' profile and answers to open questions, we tried to understand what caused this scenario. However, after analyzing the data, no predominant profile was identified. Furthermore, when analyzing the answers to open questions, we did not identify points that could explain why the participants considered the CharM less useful for achieving U2G1 and U4G1 than they did for the other goals.

Analyzing Figure 5.8, we can verify that at least 60% of the participants, who expressed their opinion, "agree" or "strongly agree" that CharM can help in some aspects related to ***architectural maintenance***. Its use for *identifying structural characteristics that can be adjusted* (U1G2) stood out in this group, since 84% of participants claimed that the CharM helps achieve this goal. In contrast, at least 11% of respondents disagreed that the CharM is useful for supporting system quality assessment (U2G2) and the architectural maintenance process (U3G2). As in the previous uses group, after analyzing the participants' profile and the open responses, we did not identify evidence that helped understand why a few participants considered the CharM not useful for achieving the goals U2G2 and U3G2.

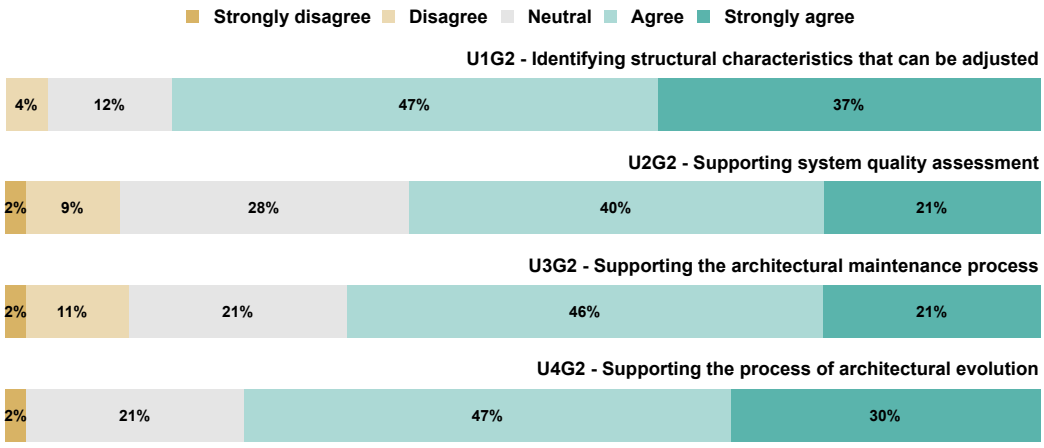


Figure 5.8: *Participants' perspective to which extent the CharM supports the maintenance of a service-based architecture. All percentages were rounded based on simple rules of rounding numbers.*

As presented in Figure 5.9, most of participants, who gave their opinion, "agree" or "strongly agree" that the CharM is useful for ***communicating a service-based architec-***

[6]Participants could choose the option "I have no opinion about this point".

ture to stakeholders. It is important to highlight that more than 90% of the participants think that the CharM can *supporting architectural discussions* (U7G3). Furthermore, at least 84% of respondents point out that the CharM is useful for *documenting the architecture of a system* (U2G3) and *supporting architectural communication* (U8G3). It is worth mentioning that in the case of uses U4G3, U7G3, and U8G3, no participant disagreed that the CharM helps achieve them. On the other hand, almost 20% of participants "disagree" or "strongly disagree" that the CharM can support *different stakeholders to share the same architectural vision* (U1G3). Furthermore, 15% of respondents believe that the CharM is not useful for *obtaining different views (local and global) of the architecture of a system* (U5G3).

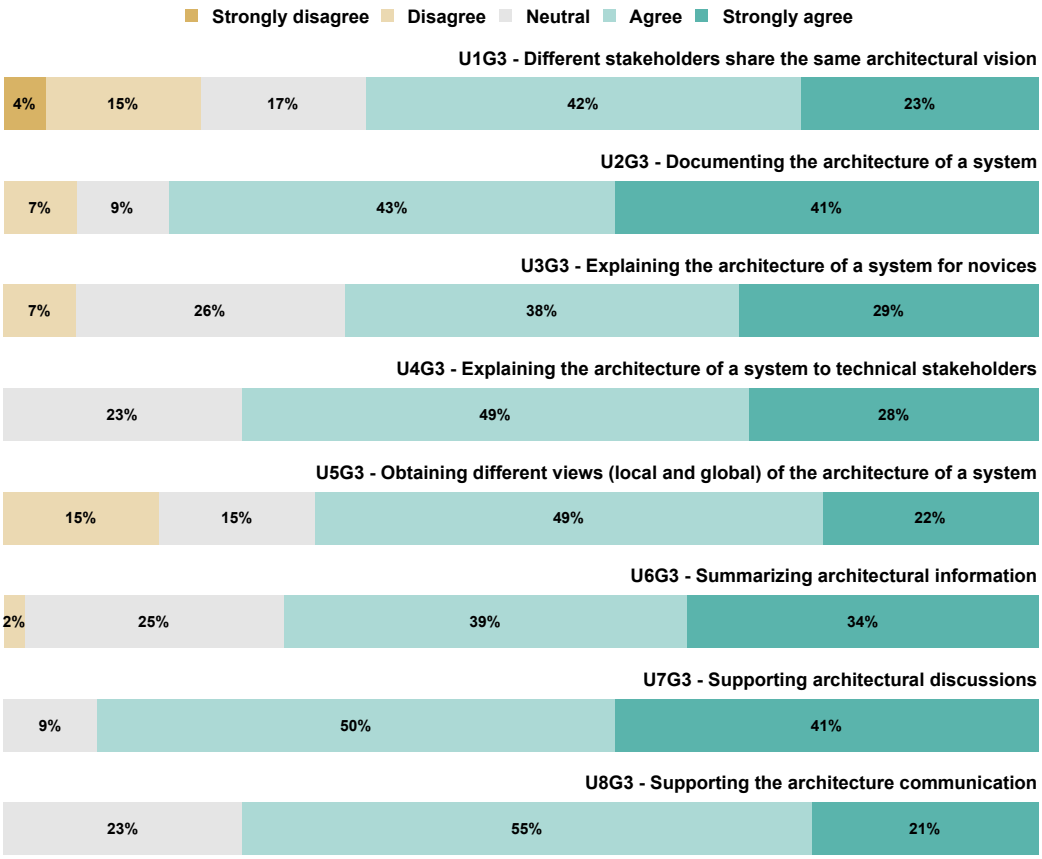


Figure 5.9: *Participants' perspective to which extent the CharM supports the communication of a service-based architecture to stakeholders. All percentages were rounded based on simple rules of rounding numbers.*

Considering the three analyzed groups, the uses U1G3 and U5G3 stood out with the highest rates of participants who disagreed that the CharM can help achieve these goals. Once again, we tried to understand what caused this rate of discordant participants. After analyzing these data, no predominant profile was identified. Furthermore, from the open responses, we did not identify evidence that could indicate why the participants considered the CharM not useful for achieving the objectives U1G3 and U5G3.

During the CharM uses assessment stage, we asked the participants if they identified any new uses, not previously listed in the survey. Four participants contributed, indicating new uses. P9 stated that the CharM is useful for *"supporting the architectural process definition at the project's beginning"*. P11 indicated that the CharM helps *"modernize and*

migrate strategies for pre-existing workloads". P21 reported that the proposed model might be useful in *"decision make before migrating"*. P49 indicated that the CharM might help *"spotting risks related to coupling, performance, and availability"*. Faced with the new uses identified by the participants, we believe that, probably, the CharM can be useful in other scenarios that adopt the same information already explored in the model.

5.7.3 The Ease of Understanding of the CharM

After identifying the uses, we asked participants how easy it is to understand the CharM's dimensions and metrics and its results. As illustrated in Figure 5.10, 91% of the participants considered it easy to understand the CharM's dimensions, with no participants stating that it is difficult to understand them. When asked about the CharM's metrics, most respondents (86%) also reported that they are easy to understand. Only one participant[7] believed that the CharM's metrics are difficult to understand. Regarding the understanding of the results generated by the CharM, 77% of the participants "agree" or "strongly agree" that they are easy to understand, with only 12% disagreeing.

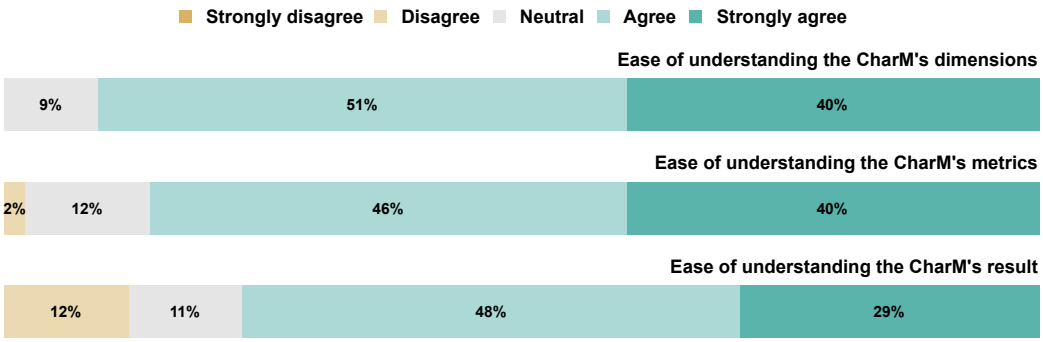


Figure 5.10: *Participants' perspective on how easy it is to understand the CharM. All percentages were rounded based on simple rules of rounding numbers.*

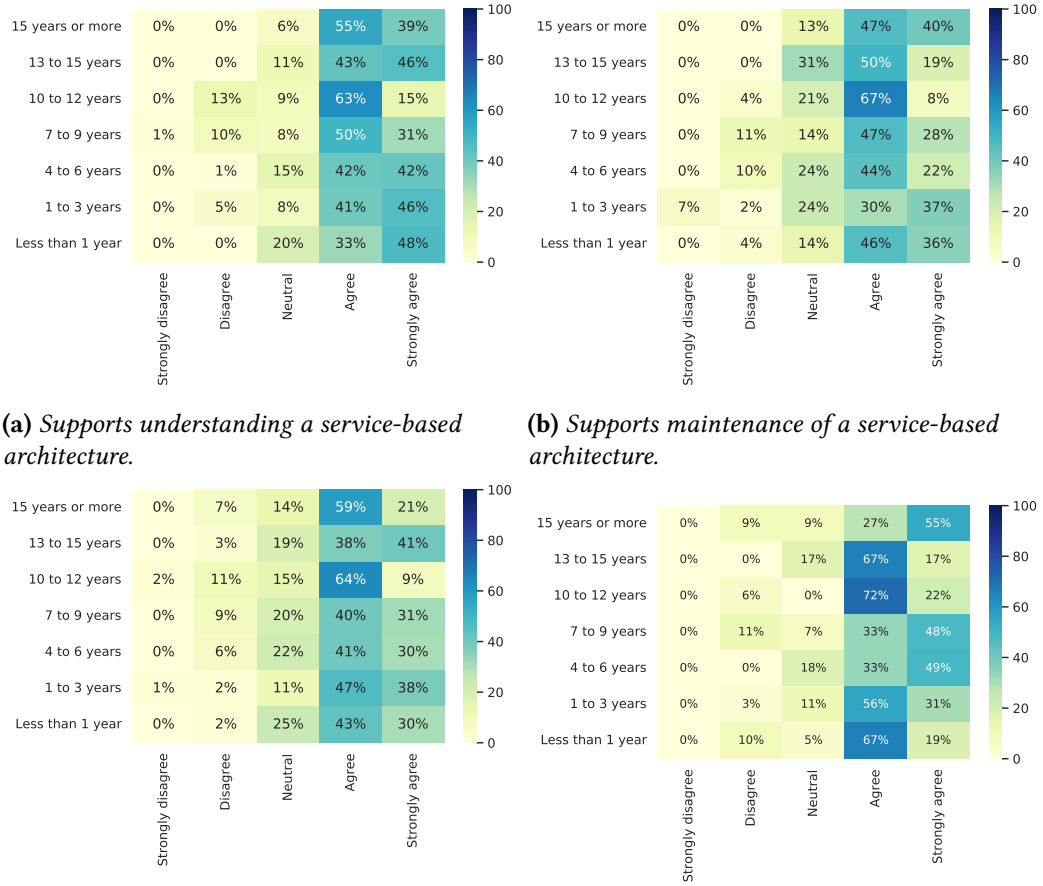
Some participants presented suggestions for improvements that may contribute to understanding the result generated by the model. Respondent P19 suggested presenting a diagnosis of the result, *"[...] saying whether the architecture is good or not [...]"*. Participant P24 stated that it would be interesting and informative to execute *"[...] the software during the presentation"* of the model. P48 claimed that *"[...] it would be great to have a "cheat sheet" or any other material with the summary of the CharM"*. P42 believes that *"[...] it might be helpful to have a kind of guideline that supports the interpretation of results"* generated by the CharM. P30 indicated that it is interesting to clarify *"in which scenarios, it is worth using the CharM"*.

5.7.4 The Use and Ease of Understanding of the CharM According to Professional Experience

In order to obtain other perspectives for the CharM evaluation, we explored the results considering the participants' self-declared experience with service-based architecture and

[7] P22 claimed that he has less than 1 year of experience with service-based architecture and that he is a novice in the microservice architectural style.

with microservices. Participants with 10 to 12 years of experience with service-based architecture tend to agree that the CharM is both useful for understanding the architecture of a service-based system (Figure 5.11a) and that it can support the maintenance process (Figure 5.11b) and architectural communication with stakeholders (Figure 5.11c). Furthermore, this same tendency can be verified regarding the proposed model's ease of understanding (Figure 5.11d).



(a) *Supports understanding a service-based architecture.*

(b) *Supports maintenance of a service-based architecture.*

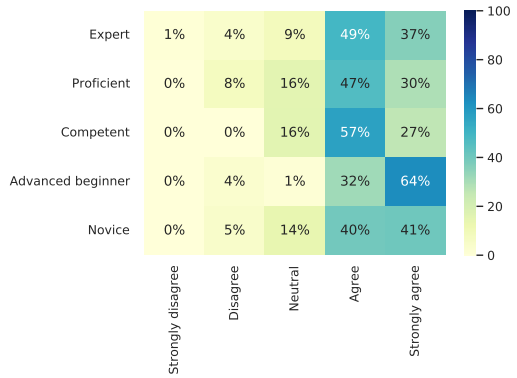
(c) *Supports communication of a service-based architecture to stakeholders.*

(d) *The ease of understanding.*

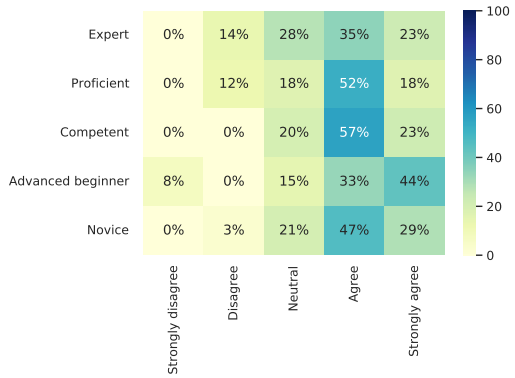
Figure 5.11: *The CharM's evaluation according to participants' self-declared years of experience with service-based architecture. All percentages were rounded based on simple rules of rounding numbers.*

Analyzing the level of experience with microservices, we can verify that the CharM can be very useful and easily understood by advanced beginners. Participants with this degree of experience with microservices tend to strongly agree that the CharM is easy to understand (Figure 5.12d) and useful for understanding (Figure 5.12a), maintaining (Figure 5.12b), and communicating (Figure 5.12c) the architecture of service-based systems.

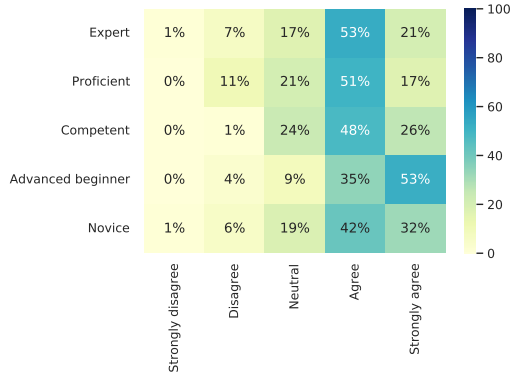
Given the results presented in Figures 5.11 and 5.12, we believe it is relevant to investigate further the perception of the CharM by professionals with different profiles. Furthermore, we are curious to find out whether or not there is some correlation between the independent variables *self-declared time of experience with service-based architecture* and *self-declared level of experience with microservices*.



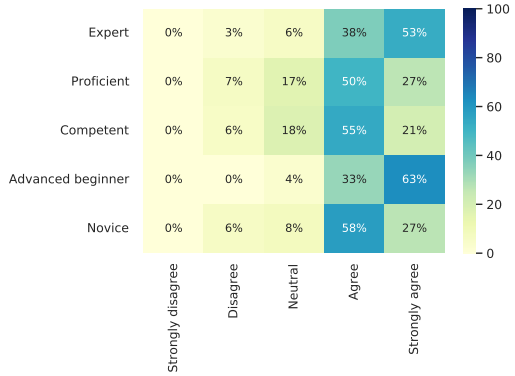
(a) *Supports understanding a service-based architecture.*



(b) *Supports maintenance of a service-based architecture.*



(c) *Supports communication of a service-based architecture to stakeholders.*



(d) *The ease of understanding.*

Figure 5.12: *The CharM's evaluation according to participants' self-declared level of experience with microservices. All percentages were rounded based on simple rules of rounding numbers.*

Therefore, considering the survey data, we found that regardless of the experience of the participants, the evaluation of the CharM's usefulness and ease of understanding tends to be positive. That is, the CharM tends to be useful and easy to understand by professionals with different times and levels of experience.

5.8 Discussion

There are approaches focused on the analysis, recovery, and understanding of the architecture of service-based systems (Granchelli et al., 2017a; Engel et al., 2018; Mayer and Weinreich, 2018; Alshuqayran, 2020), others are more focused on the evaluation of quality attributes (Cardarelli et al., 2019; Bogner, Wagner, et al., 2017b) and to the analysis of conformance to architectural patterns (Zdun et al., 2017), and there are others that support the architectural migration process (Auer et al., 2021). Despite this, experience reports from companies, such as Amazon (Kolny, 2023), Istio (Mendonça et al., 2021), Segment (InfoQ, 2020), and Uber (Highscalability, 2020) evidence the difficulty in characterizing and defining which architectural structure is the most suitable for a given system. Furthermore, based on the work by Nadareishvili et al. (2016) and Newman (2021)

and the variety of terms, such as nanoservice, microservice, and macroservice, we can notice that there is a fuzzy limit when trying to classify and characterize the architecture of service-based systems. Such a scenario is an indication that it is still necessary to invest in the development of approaches for evaluating, characterizing, and guiding the architecture of service-based systems.

That said, the CharM is being developed to help identify structural characteristics (size and coupling), which allows us to describe and understand the architecture of service-based systems, as well as their components. It is important to clarify that, different from approaches, such as MicroART (Granchelli et al., 2017a), MAAT (Engel et al., 2018), MicroQuality (Cardarelli et al., 2019), MiSAR (Alshuqayran, 2020), RAMA (Bogner, Wagner, et al., 2020), Prophet and Microvision (Cerny, Amr S. Abdelfattah, et al., 2022) the CharM is not a tool-based approach but a theoretical and conceptual model designed to be applied at different software life cycle stages. Hence, the CharM combines a part of the advantages of some approaches (explored in the Chapter 6) for architectural evaluation, recovery, and evolution. In addition, one of the differentials of this study (compared to the studies presented in the Chapter 6) is that we evaluated to which extent our approach (CharM) is useful for different purposes and is easy to understand.

Besides that, it is worth mentioning that we selected the microservices guidelines that underlie CharM through *ad-hoc* bibliographic research. Therefore, we did not adopt a strict protocol for mapping the microservices definitions and characteristics. Despite this, the entire process of designing the CharM was grounded on the seven Design Science Research guidelines. We also clarify that adopting the current version of the CharM in large-scale projects is unfeasible since the metrics are collected manually. Cardarelli et al. (2019) and Ntentos, Zdun, Plakidas, and Geiger (2021a) highlighted this same challenge of evaluating their approaches in large-scale systems.

Considering this scenario and from the analysis of survey results (Section 5.7), we could answer the five research questions specified at the beginning of this chapter.

RQ4: ***To which extent does the CharM support understanding a service-based architecture?*** Granchelli et al. (2017b) claim it is challenging to have a clear architectural overview of a microservice-based system. However, Mayer and Weinreich (2018) argue that it is essential to understand how a system is divided into components and how they interact. In addition, Bass et al. (2012) explain that by understanding the architecture of a system, it is also possible to understand the ability of its architecture to meet the quality attributes. Faced with this importance and challenge, the CharM was assessed as an approach that contributes to understanding the architecture of a service-based system, as well as the approaches proposed by Mayer and Weinreich (2018), Cardarelli et al. (2019), and Alshuqayran (2020). Especially, the survey participants believe that the CharM supports understanding the synchronous and asynchronous couplings between services, as well as is useful for evaluating the architecture of a system that is still being designed. This participants' perception is in line with some of the CharM objectives. Nevertheless, although there is a dimension of the CharM focused on the structural attribute of size, the survey respondents did not indicate the identification of the size of the components as one of the main uses of the CharM. This result may be related to the discussion of the subjectivity of defining the service's size, as explored by Newman (2021).

RQ5: ***To which extent does the CharM support a service-based architecture maintenance?*** Bogner, Wagner, et al. (2017b) explain that the maintainability, that is, the degree of effectiveness and efficiency in modifying, correcting, improving, extending, or adapting software is fundamental for organizations. Based on the survey participants' answers, the CharM can contribute to the architectural evolution of service-based systems, especially to identify structural characteristics that can be adjusted. But, different from the approach MM4S (Bogner, Wagner, et al., 2017b), system quality assessment is not among the main uses of the CharM (even with a positive acceptance). A possible explanation for this perception from some participants is that the results generated from the CharM do not show an explicit relationship with the non-functional requirements nor improvement suggestions, as in the approach proposed by Ntentos, Zdun, Plakidas, and Geiger (2021b). This interpretation is reinforced by the following comment from participant P54, who suggested *"modeling the data collected by the CharM to recommend architectural improvements according to the context and objective of each service-based system"*.

RQ6: ***To which extent does the CharM support communicating a service-based architecture to stakeholders?*** For Bass et al. (2012), the architecture of a system must be communicated clearly and unambiguously to all stakeholders. Considering this, the survey participants believe that the CharM can support the process of communicating service-based architecture to stakeholders, especially in architectural discussions and documentation. This participants' perception demonstrates that the CharM fulfills the objective of facilitating the analysis of the architecture of a service-based system, similar to the MicroART approach (Granchelli et al., 2017a). Although the CharM allows choosing the system's components and perspectives that will be analyzed, a small index (15%) of survey participants did not perceive the model's utility for obtaining different views of the architecture of a system. Therefore, we believe this may be a CharM use that can be better explored and explained in future versions.

RQ7: ***How easy is it to understand the CharM?*** The ease of understanding is an important aspect to be evaluated in new artifacts, as it helps determine their ease of use (Lederer et al., 2000) and can influence their perceived usefulness (Davis, 1989). In general, the survey participants indicated that the CharM is easy to understand. However, considering the analyzed aspects (dimensions, metrics, and results), the respondents indicated that they faced some difficulty understanding the generated results. We believe this difficulty is justified by the volume of data presented and the need to understand the analyzed system and relate all the metrics and perspectives presented.

RQ8: ***Does the participants' experience influences the perceived usefulness and ease of understanding of the CharM?*** We hypothesized that participants with different profiles would have different use perceptions of the CharM and that participants with less time and a low level of experience would tend to have more difficulty understanding the model. However, we found that the participants' profile did not significantly influence (negatively) their perception of usefulness and ease of understanding of the CharM. Despite this, we believe it is relevant to investigate further the perception of the CharM according to professional experience since this type of analysis can help better understand the results of the CharM's uses and identify improvements to be implemented. Some approaches, such as those proposed by Mayer and Weinreich (2018) and Auer et al. (2021), were also evaluated through a survey. However, the authors did not investigate whether the perception of

their approaches varied according to the participants' profiles. Auer et al. (2021)'s study only accepted experienced respondents without an academic background.

Given the positive evaluation of the CharM obtained from the survey, we strengthen the evidence that this model is useful to characterize the architecture of service-based systems, regardless of time and level of professional experience. Despite this, there are aspects of the CharM to be explored and improved. These aspects are mainly related to dimensions and metrics (type and collection type), presentation and explanation strategy, and the possibility of the model integration with existing technologies. To implement some of these improvements, we are studying strategies for collecting and analyzing metrics semi-automatically, as in the Mayer and Weinreich (2018), Cardarelli et al. (2019), Bogner, A. Zimmermann, et al. (2020), and Alshuqayran (2020) studies. In addition, we are also working on optimizing the generation of the views (rulers, graphs, and diagrams) of the CharM metrics (E. Santana et al., 2021).

5.9 Chapter Summary

This chapter presented the evaluation results of the CharM by a wider audience. Thus, we described the process of elaboration and application of a survey answered by 58 professionals who work with the architecture of service-based systems. In the end, we identified that survey participants consider the CharM useful for understanding a service-based architecture, maintaining a service-based architecture, and communicating a service-based architecture to stakeholders. Furthermore, in general, respondents indicated that the CharM is easy to understand. We also identified evidence that our model could be useful for professionals with different profiles (from novices to experts). At the end of this chapter, we answered the research questions RQ4, RQ5, RQ6, RQ7, and RQ8.

Chapter 6

Related Research

This chapter presents an overview of scientific studies that propose different approaches which, in some way, support assessing and characterizing the architecture of service-based systems. In the end, we compare these approaches and highlight the main differences between them and our study. We organize the related work into three groups: architectural recovery, architectural evaluation, and architectural migration. We adopted the main purpose identified for the research as a grouping criterion. We describe the study group focused on architectural recovery in the next section.

6.1 Architectural Recovery

The first two studies in the series published by Granchelli et al. (2017a) and Granchelli et al. (2017b) present an approach (MicroART) to assist in recovering and understanding the complexity of microservice-based systems architecture. The last one (Cardarelli et al., 2019) proposes an approach (MicroQuality) for specification, aggregation, and evaluation of quality attributes for the architecture of microservice-based systems. Such approaches are based on Model-Driven Engineering principles. To validate the proposed approaches, the authors carried out experiments with one open-source system and collected static and dynamic metrics in a semi-automatic way. Some findings of these studies are that the proposed approaches are useful for architectural understanding, documenting, and analyzing, unveiling dependencies between microservices from the development perspective, automatically recovering and measuring architectural models, and continuously assessing the quality of microservices.

Alshuqayran published two studies related to this research. The focus of the first study (Alshuqayran, Ali, et al., 2018) is to present an approach (MiSAR - Micro Service Architecture Recovery) based on Model-Driven Engineering (MDE) to recover the architecture of microservice-based systems. This approach was evaluated through empirical studies that analyzed eight microservice open-source projects. In the second publication (Alshuqayran, 2020), the author explains that the objective of the proposed approach is to comprehend the complexities of microservices by developing a bottom-up reverse engineering process. The evaluation of the approach proposed in the first study was complemented with an empirical study on nine open-source microservice-based projects

and a case study with a large open-source microservice-based system. Some of the listed uses for the approach are: to create architectural documentation, obtain performance diagnostics at the container level, identify several concepts and support the definition of the underlying features and behavior of microservice-based systems, and help identify and build the relations between services.

The studies by Bushong, Das, et al. (2022) and Cerny, Amr S. Abdelfattah, et al. (2022) present a technique for software architecture reconstruction based on static code analysis, which generates an architectural visualization in augmented reality. Cerny, Amr S. Abdelfattah, et al. (2022) developed two prototype tools. Prophet performs the static code analysis and Microvision generates the architectural visualization in augmented reality. The authors used a proof of concept and conducted case studies to evaluate the proposed approach. Some uses cited for this approach are extracting a view of the architecture before the system is deployed and detecting deviations in the planned architecture.

In the next section, we present a group of studies in which the main focus we identified is architectural evaluation.

6.2 Architectural Evaluation

The study conducted by Engel et al. (2018) presents an approach (MAAT - Microservice Architecture Analysis Tool) to evaluate microservice architectures based on principles, such as small size and loose coupling of the services and domain-driven design. To develop the proposed approach, the authors adopted the design of microservice architectures principles extracted from literature review, structured interviews with experts, and the GQM (Goal Question Metric) (Basili and Rombach, 1988) technique to derive the proposed approach's metrics. During the research, a case study was carried out with a project with 50 microservices, in which metrics were collected automatically through a dynamic analysis. Furthermore, discussions were held with project team members. In the end, some identified uses for the approach were to support the identification of hot spots in the architecture of a system, support the design and development of systems, and confirm perceptions of the system's team.

Another series of four scientific studies, led by Bogner, is also related to our work. The first study (Bogner, Wagner, et al., 2017b) proposes a practical maintainability quality model for services- and microservices-based system, named MM4S (Maintainability Model for Services). The study objective is to obtain a simple and practical tool for basic maintainability estimation, control, and specification in the context of services- and microservice-based systems. The authors performed an intra-company focus group combined with a literature review yielding the first requirements for the desired quality model. The second published study (Bogner, Schlinger, et al., 2019) in the series aims to calculate service-based maintainability metrics from the runtime data of microservice-based systems. To achieve this goal, they carried out an exploratory study with an example system which was used by six people. The focus of the third study (Bogner, Wagner, et al., 2020) is to calculate maintainability metrics from machine-readable interface descriptions of RESTful services using the approach RAMA (RESTful API Metric Analyzer). An evaluation was performed based on threshold benchmarking with 1,737 public RESTful APIs. In

the last study (Bogner, A. Zimmermann, et al., 2020), the authors present a continuous assurance method to support the identification and remediation of evolvability-related issues in service-based systems. For the final evaluation of this method, they plan to combine industry case studies with action research. Considering the aforementioned studies, some of the utilities mapped are: maintenance support of service-based systems, evaluation of maintainability of RESTful services, hot spot identification, and early quality evaluation.

The first study (Zdun et al., 2017) of another series, proposes a minimal set of constraints and metrics necessary to evaluate the conformance of an architecture to microservice patterns. To achieve this goal, the authors cataloged an initial set of constraints and metrics. Then, to identify the most significant ones, this initial set was used to manually evaluate 13 architecture models (extracted from the literature). The focus of the second study in the series (Ntentos, Zdun, Plakidas, and Geiger, 2021a) is a method for the semi-automatic detection and resolution of microservice patterns conformance violations. The authors tested this method on a set of 24 models of various degrees of pattern violations and architecture complexity. The last study of the series (Ntentos, Zdun, Plakidas, and Geiger, 2021b) aims to provide foundations for an automated approach for architectural reconstruction, assessing conformance to patterns and practices specific to microservice architectures, and detecting possible violations. To achieve this goal, the authors carried out experiments with a set of 27 models of microservice-based systems from third-party practitioners. Some of the uses of the approach, pointed out by the authors of this series of studies, are: to compare architecture conformance of the current design and possible refactorings, find the root cause of a violation of microservices patterns, measure the quality of microservice decomposition in software architecture models, and suggest possible improvements related to microservice coupling.

The research conducted by Mayer and Weinreich (2018) presented an approach that aims to extract and analyze the architecture of a microservice-based system based on a combination of static service information with infrastructure-related and aggregated runtime information. Such an approach is based on three levels in a microservice-based system: service, infrastructure, and interaction. The proposed approach was evaluated through a survey and interview study with 15 architects, developers, and operations experts. In the end, some of the uses identified for the approach were to provide a high-level overview of the system, analyze architecture evolution, provide an overview of a service's functionality, obtain automatic documentation of a microservice, and determine which microservices and their functions will be affected by changes.

Peng et al. (2022) proposed a trace analysis-based microservice architecture measurement approach. This approach was evaluated through three case studies. In the paper, the authors mentioned that the proposed approach is useful in characterizing the independence and complexity of the invocation chain of microservice architectures. Another cited utility was that the approach could help to identify microservice architecture issues caused by improper service decomposition and architectural degradation.

In the next section, we describe an identified research focused on architectural migration.

6.3 Architectural Migration

The work by Auer et al. (2021) presents a framework for supporting companies in discussing and analyzing the potential benefits and drawbacks of the migration and re-architecting process. During the study, a survey was carried out in the form of interviews with 52 professionals. The main use of the framework, pointed out by the authors, is to help companies avoid architectural to microservices migration if it is not necessary, especially when they might get better results by refactoring their monolithic system or re-structuring their internal organization.

After describing the identified related work, we compare such studies to ours in the next section and highlight the main differences.

6.4 Comparison of Related Research

In all, we identified and analyzed 19 related scientific work, organized into nine series of studies. Figure 6.1 compares our study with the related papers described in previous sections. From the investigation carried out to date, we have not identified approaches with the same objective as the CharM. That is, to characterize the architecture of service-based systems based on microservices guidelines and static metrics of the structural attributes of size and coupling.

Group	Study	Support understanding a SBA	Support the maintenance of a SBA	Support the communication of a SBA to stakeholders	Independent of code or infrastructure?
Architectural recovery	Granchelli et al. (2017a,b); Cardarelli et al. (2019)	Yes	Yes	Yes	No
	Alshuqayran et al. (2018); Alshuqayran (2020)	Yes	Yes	Yes	No
	Bushong et al. (2022); Cerny et al. (2022)	Yes	Yes	Yes	No
Architectural evaluation	Engel et al. (2018)	Yes	Yes	Yes	No
	Bogner et al. (2017, 2019, 2020a,b)	Yes	Yes	No	No
	Zdun et al. (2017); Ntentos et al. (2021a,b)	Yes	Yes	No	Yes
	Mayer and Weinreich (2018)	Yes	Yes	Yes	No
	Peng et al. (2022)	Yes	Yes	Yes	No
Architectural migration	Auer et al. (2021)	No	Yes	Yes	Yes
Architectural characterization	**This study**	**Yes**	**Yes**	**Yes**	**Yes**

Figure 6.1: *Comparison of related research.*

Although the analyzed approaches do not have the same objective as the CharM, they have some uses in common. Therefore, they can support the maintenance, understanding, or communication of the architecture of service-based systems. Thus, when analyzing the related studies [1], we can observe that all the other presented approaches, in some way, support the maintenance of the architecture of service-based systems. Nevertheless, we did not clearly identify that the approach proposed by Auer et al. (2021) supports understanding a service-based architecture and that the approaches presented in the

[1]The summary of the related studies analysis is available at: https://drive.google.com/file/d/1YutSuItDv6uhTtEGSwodbJclxaU3fao9/view?usp=sharing.

series of studies conducted by Bogner, Wagner, et al. (2017a) and Zdun et al. (2017) and Ntentos, Zdun, Plakidas, and Geiger (2021a) support the communication of a service-based architecture to stakeholders.

Another feature we observed when comparing our research with the others was whether the proposed approach's analysis depends on the software source code or infrastructure. We found that only our model and the approaches proposed by Zdun et al. (2017) and Ntentos, Zdun, Plakidas, and Geiger (2021a) and Auer et al. (2021) have this independence. This feature means that the CharM could be used at different software life cycle stages.

Another interesting point to note is that the CharM also explores different and didactic ways for architectural visualization of services and systems. When examining the analyzed studies, this aspect was considered in only seven (Engel et al., 2018; Mayer and Weinreich, 2018; Alshuqayran, 2020; Bogner, A. Zimmermann, et al., 2020; Ntentos, Zdun, Plakidas, and Geiger, 2021b; Bushong, Das, et al., 2022; Cerny, Amr S. Abdelfattah, et al., 2022) of the 19.

Furthermore, although the proposed approaches were evaluated empirically, we found that the use and ease of understanding aspects were not the focus of most related work. The only exception is Cerny, Amr S. Abdelfattah, et al. (2022)'s study, which sought to understand the usefulness of Microvision and whether its use was intuitive. That is, none of the other studies we analyzed have deeply investigated how useful the proposed approaches are for certain tasks or how easy these approaches are to understand. In contrast, one of the focuses of our work is to empirically evaluate the uses of the CharM and its ease of understanding.

6.5 Chapter Summary

In this chapter, we reviewed and compared solutions with similar goals to the CharM. However, after analyzing 19 scientific studies, we did not identify solutions whose main goal is to characterize the architecture of service-based systems. We also noticed that the mapped solutions aim at architectural recovery, evaluation, or migration. Most of these solutions depend on source code or infrastructure to achieve these goals. Therefore, considering this scenario, the CharM proved to be a valuable solution that may help professionals to characterize the architecture of service-based systems, being useful to understand, document, and maintain the architecture without depending on source code or infrastructure.

Chapter 7

Conclusion

This chapter concludes this thesis by reviewing the research questions and CharM's evaluation results. Furthermore, we point out the main contributions of this thesis and suggestions for future work.

Faced with the challenge of characterizing the architecture of service-based systems, we sought a way to mitigate this problem and, at the same time, support the architectural decision-making process. Therefore, the main objective of this thesis was to develop a model to characterize the architecture of service-based systems, adopting microservices guidelines. We called this model CharM.

To achieve this goal, we followed the seven Design Science Research guidelines, which allowed us to build and evaluate the CharM iteratively and incrementally, based on *ad-hoc* literature reviews, discussions with software architecture experts, multiple case studies, and a survey. Thus, this thesis defined the CharM and described its dimensions and metrics. In addition, we presented the result of this model's empirical evaluation through two case studies (the first in an academical and the second in an industrial environment) and a survey.

Based on the analysis and refinement of the CharM's application and evaluation results in the two case studies (Chapter 4), we mapped 24 possible uses for our characterization model. From this, we grouped them into the following three categories of uses (i) understanding a service-based architecture, (ii) maintaining a service-based architecture, and (iii) communicating a service-based architecture to stakeholders (RQ1). We also investigated the professionals' perceptions regarding the characterization generated by the CharM. We found that, in general, respondents considered easy to understand the result generated by our model. Besides that, respondents considered that the architectural characterization generated from the CharM is coherent with the reality they knew of the studied systems (RQ2). During the case studies, we also mapped aspects where the CharM could be improved. Thus, we received suggestions to add new metrics and other architectural information, as well as recommendations to improve the results presentation and the explanation of some aspects of the model (RQ3). The CharM was evaluated by 17 different professionals through the two case studies. The analyses and discussions that took place during these studies allowed us to evaluate different aspects of the CharM deeply.

Then, to evaluate the CharM from the perspective of a wider audience, we designed a survey (Chapter 5). This survey was answered by 58 professionals who work with service-based systems architecture. The results of this survey evidenced that our model is useful for understanding the synchronous and asynchronous couplings between services and evaluating a service-based system's architecture at design time (RQ4). Furthermore, another use of the CharM that stood out was the possibility of identifying structural characteristics, which can be adjusted (RQ5). The characterization generated by the CharM was also highlighted as useful to support discussions and compose architectural documentation (RQ6). It is also worth mentioning that the CharM had a mostly positive evaluation since it was considered useful for the 21 uses presented in the survey.

The survey results also reinforced that the CharM is easy to understand (RQ7). Furthermore, during the evaluation stages, we identified more evidence that the characterization generated by the CharM is useful for professionals with different profiles (RQ8), helping them to understand the architecture of a service-based system and in the process of architectural maintenance and communication.

Thus, we believe that the results presented in this thesis demonstrate that the CharM is a useful model to characterize the architecture of service-based systems and that it supports architectural decision making.

7.1 Contributions

The main contribution of this work is the CharM, which is a valuable resource that helps professionals with different profiles to understand, document, and maintain the service-based systems' architecture. Despite the positive result of the model evaluation, some improvements can be implemented, such as the following: not limiting the CharM dimensions to the structural attributes of size and coupling, exploring other types of metrics (static and dynamic), adopting strategies for semi-automatic collection of metrics as well as the automatic generation of views, and presenting results that relate explicitly to the influence of the architecture characteristics on the quality attributes.

A secondary contribution of this research was the developed systematic method for trade-off analysis of architectural patterns. Such a method may help in decision-making by signaling the adoption effects of certain architectural patterns on relevant quality attributes in a specific context. Despite the initial results being interesting, we believe it is relevant to conduct further investigations and submit our method to new validation steps.

Other relevant contributions directly or indirectly related to this thesis are:

- **[Scientific]** The literature review on software architecture, where we explored monolithic and service-based architectural styles and highlighted the characteristics, advantages, and challenges of microservices. In addition to the ad-hoc review about metrics of size, data source coupling, and synchronous and asynchronous coupling;
- **[Scientific]** The results of this thesis contributed to the development of an undergraduate research project and two master's degree dissertations. The undergraduate research project was funded by FAPESP (São Paulo Research Foundation) and aimed to develop the Sorting Hat tool to automate some tasks to assist in architectural

characterization with the CharM. One of the dissertations was developed in the graduate program in Computer Science at IME/USP, whose objective was automatically detecting patterns in microservices-based architectures. The other dissertation is being developed in the graduate program in Applied Computing at the Institute for Technological Research at the University of São Paulo (IPT/USP). In such work, the CharM is being adopted to characterize the architecture of a financial market system;

- **[Scientific and Technical]** Development of didactic videos to explain the CharM's structure and demonstrate its application's result. Such videos are useful for understanding and using our model;
- **[Technical]** The results generated from the CharM instigated and supported discussions by the Organization A team to adjust the Search System's architecture (one of the study objects in this thesis);
- **[Educational]** The studies carried out in this thesis were the basis for the planning and execution of two extension courses [1], where the author of this thesis acted as one of the teachers. One of the courses was named Complex Systems Development and lasted 60 hours (2019/1 and 2019/2 offers). The other course was named Agile Software Architecture, was offered in 2021/1, and lasted 30 hours. Both courses were offered in the summer course program at the University of São Paulo (USP);
- **[Educational]** The studies carried out in this thesis contributed to the creation of the course Laboratory of Complex Computational Systems [2] for the undergraduate in Computer Science at the Institute of Mathematics and Statistics of the University of São Paulo (IME/USP). The author of this thesis acted as a student monitor in the 2020/1 and 2021/1 course editions.

7.2 Scientific Publications

We generated a series of scientific papers at different steps of this research. Some of these papers are directly related to this thesis and correspond to results, which, in some way, are described throughout this document. Others are the unfolding steps of this research, developed in collaboration with other researchers. It is worth noting that some of these papers have already been published or submitted, and others are still in development.

7.2.1 Published or Submitted Papers

Paper 1. ROSA, Thatiane de Oliveira; GOLDMAN, Alfredo; GUERRA, Eduardo Martins. *How 'micro' are your services?* In: **IEEE International Conference on Software Architecture Companion (ICSA-C 2020)**. 2020.

[1] Extension courses program: https://drive.google.com/file/d/1onrw4-YOxyPX4FEMuIu8jrreNZwb9TIV/view?usp=sharing

[2] Undergraduate course program: https://bcc.ime.usp.br/principal/catalogo2017/disciplinas/MAC0475.html

We published this paper on the New and Emerging Ideas track of the ICSA. Our goal was to present the first version of our characterization model, developed from exploratory and descriptive research.

Paper 2. ROSA, Thatiane de Oliveira Rosa; DANIEL, João Francisco Lino; GUERRA, Eduardo Martins; GOLDMAN, Alfredo. *A Method for Architectural Trade-off Analysis Based on Patterns: Evaluating Microservices Structural Attributes*. In: **Proceedings of the European Conference on Pattern Languages of Programs**. 2020.

This paper presented a method for systematic trade-off analysis of architectural patterns. In such a study, we identified how different microservice patterns influence the structural attributes of size, coupling, and sharing data sources.

Paper 3. ROSA, Thatiane de Oliveira; GOLDMAN, Alfredo; GUERRA, Eduardo Martins. *Characterization and Evolution Model of Service-based Architecture Systems* (in Portuguese). In: **Workshop on Theses and Dissertations (WTDSOFT) - Brazilian Conference on Software: Theory and Practice (CBSOFT)**. 2020.

We presented in this paper the second version of our characterization model. We incorporated improvements from investigating metrics and architectural patterns in such a version.

Paper 4. SANTANA, Erick Rodrigues de; ROSA, Thatiane de Oliveira; DANIEL, João Francisco Lino; GOLDMAN, Alfredo. *Desenvolvendo o Sorting Hat: uma Ferramenta para Caracterização de Arquitetura Baseada em Serviços* (in Portuguese). In: **Undergraduate Research on Software Engineering Competition (CTIC-ES) - Brazilian Conference on Software: Theory and Practice (CBSOFT)**. 2021.

This paper is an offshoot of this thesis. The main objective of this paper is to present the Sorting Hat, a tool that helps in the process of characterizing the architecture of service-based systems. We present the advances in the process of automatically generating visualizations of the metrics of our characterization model. Furthermore, we describe a preview version of the automated data collector prototype.

Paper 5. VALE, Guilherme; CORREIA, Filipe Figueiredo; GUERRA, Eduardo Martins; ROSA, Thatiane de Oliveira; FRITZSCH, Jonas; BOGNER, Justus. *Designing Microservice Systems Using Patterns: An Empirical Study on Quality Trade-Offs*. In: **IEEE International Conference on Software Architecture (ICSA-2022)**. 2022.

This paper is an offshoot of the microservice patterns trade-off analysis step carried out in this thesis. It was developed with researchers from the University of Porto in Portugal and the University of Stuttgart in Germany. This paper presents an empirical study in the industry, aiming to identify and understand the relationships between microservices design patterns and the quality attributes they can support or hinder.

Paper 6. ROSA, Thatiane de Oliveira; GUERRA, Eduardo Martins; CORREIA, Filipe Figueiredo; GOLDMAN, Alfredo. *CharM — evaluating a model for characterizing service-based architectures*. In: **Journal of Systems and Software**, Volume 206, December 2023.

This paper presents the fifth version of our characterization model, named CharM, as

well as the results of its evaluation through a survey.

Paper 7. DANIEL, João Francisco Lino; ROSA, Thatiane de Oliveira Rosa; GOLDMAN, Alfredo; GUERRA, Eduardo Martins. *Towards the Detection of Microservice Patterns Based on Metrics.* **Accepted to be presented at the Euromicro Conference Series on Software Engineering and Advanced Applications - SEAA 2023.**

This paper is also an unfolding of the studies initiated in this thesis. Following a metric-based approach, we present an automatic detection tool for microservices patterns. The objective is to increase awareness of such patterns and help developers better understand a microservices-based architecture. At this moment, our tool works with the detection of five patterns. We evaluated this tool through two case studies.

7.2.2 Papers in Progress

We are currently working on developing three other papers:

- **Paper 8:** the main objective is to present the result of the evaluation of the CharM through the two case studies described in this thesis;
- **Paper 9:** aims to present the improvements implemented in the Sorting Hat tool (presented in Paper 4), considering both the automatic generation of visualizations and the automation of data and CharM's metrics collection;
- **Paper 10:** this paper aims to describe the results of applying the CharM to characterize the architecture of a financial market system. We intend to explore and evaluate suggestions for architectural evolutions from such characterization.

7.3 Future Work

Throughout the development of this research, we identified some gaps and opportunities for further research. Below we describe each of them:

- **Develop automation tools** related to collecting metrics from the model, generating views (rulers, graphs, and diagrams) of the characterization of the architecture of the analyzed system, and identifying architectural patterns from the metrics of the model. These activities are in progress and we have already obtained preliminary results. The first results were published in the paper of the E. Santana et al. (2021). We are currently working on the development of two other papers, in which we describe new advances;
- **Evolve the model** incorporating new dimensions related to different attributes, test the resource of analysis profiles, and submit it to new evaluation iterations (in new real environments and/or through a survey with more participants). During this process, we also plan to monitor the behavior and evolution of our model, as well as evaluate its continued use. Currently, we are adopting the fifth version of the CharM to characterize a part of a financial market system. This task consists of a master's thesis being developed within the Institute for Technological Research at the University of São Paulo (IPT/USP);

- **Incorporate dynamic metrics into the CharM**, which, combined with static metrics, may generate a richer and more consistent architectural characterization, making it possible to explore and understand a more extensive set of architectural characteristics of the system to be analyzed;
- **Develop a catalog of evolutions**, which, combined with the CharM, serve as a guide to adjust the architecture according to the desired architectural approach;
- **Evaluate the adoption of the CharM as one of the steps of an architectural migration process**, to help professionals analyze and understand how much the new architectural design meets the desirable quality requirements of a given system;
- **Evaluate and compare the architectural visualization generated from the CharM** with those generated from other studies to identify positive points and opportunities for improvements.

Appendix A

Ad-hoc Review of Metrics

Perepletchikov et al. (2007) explain that metrics can serve as early indicators of software quality characteristics, enabling the identification of potential problems from the early stages of the software life cycle. Thus, we performed an *ad-hoc* bibliographic research about metrics during the second iteration of the CharM design process. The objective of this research was to identify candidate metrics to be used in each dimension of our characterization model. Therefore, considering that we adopted microservices guidelines in the CharM, we directed our search toward metrics applicable to this architectural style.

During this investigation, we found over 50 metrics directly related to microservices. This set of metrics was varied, containing different types (static and dynamic), covering different scopes (element, service, operation, and system), applicable to different structural attributes (size, complexity, cohesion, and coupling), as well as dependent and independent of technology, source code, and infrastructure. During this investigation, we consulted the following academic materials and patterns documentation: Rud et al. (2006), Perepletchikov et al. (2007), H. Hofmeister and Wirtz (2008), Shim et al. (2008), Hirzalla et al. (2009), Zhang and Xinke (2009), Bogner, Wagner, et al. (2017a), Bogner, Wagner, et al. (2017b), Hutapea et al. (2018), Engel et al. (2018), Ntentos, Zdun, Plakidas, Meixner, et al. (2020), and Richardson (2020).

We defined the following criteria to select metrics: being related to the structural attributes of size and coupling, consequently with the dimensions of our model, as well as being independent of technology, source code, and infrastructure. Other aspects that we considered were simplicity and that the metrics should be useful in facilitating the interpretation of the service-based system's architecture. In Table A.1 we present the insight sources to the set of metrics incorporated into the second version of the CharM, which we described in paper Rosa et al. (2020).

CharM dimension	Candidate metrics' insight sources
Size	• Number of Services (NS) (H. Hofmeister and Wirtz, 2008) • Number of Operations (SIM NO) and System Size in Number of Services (SM SSNS) (Shim et al., 2008; Bogner, Wagner, et al., 2017a) • Number of Services (NOS) and Weighted Service Interface Count (WSIC) (Hirzalla et al., 2009; Bogner, Wagner, et al., 2017a; Bogner, Wagner, et al., 2017b) • Number of Services (NS) • Number of synchronous and asynchronous interfaces (Engel et al., 2018)
Data Source Coupling	• Service Interconnections with SharedDB (Ntentos, Zdun, Plakidas, Meixner, et al., 2020) • Pattern Shared database and Pattern Database per service (Richardson, 2020)
Synchronous Coupling and Asynchronous Coupling	• Absolute Importance of the Service (AIS) and Absolute Dependence of the Service (ADS) (Rud et al., 2006; Bogner, Wagner, et al., 2017a; Bogner, Wagner, et al., 2017b) • Weighted Extra-Service Incoming Coupling of an Element (WESICE) and Weighted Extra-Service Outgoing Coupling of an Element (WESOCE) (Perepletchikov et al., 2007; Bogner, Wagner, et al., 2017a) • Service Consumers (SC), Service Providers (SP), Coupling of Service (cos), and System's Service Coupling (SSC) (H. Hofmeister and Wirtz, 2008; Bogner, Wagner, et al., 2017a) • Coupling of Service (CS) and Importance of Service (IS) (Zhang and Xinke, 2009) • Number of synchronous and asynchronous dependencies (Engel et al., 2018)

Table A.1: *Candidate metrics - Second version of the CharM*

Inspired by Table A.1, we selected the following initial set of metrics to compose the second version of our architectural characterization model:

- **Size:** number of services per module and number of operations per service;

- **Data Source Coupling:** number of databases per module and number of modules that share the same database;
- **Synchronous Coupling:** number of client services that invoke the operations of a given service and number of services that a given service is dependent on;
- **Asynchronous Coupling:** number of different types of messages published by a given service and number of different types of messages consumed by a given service.

We detailed the results of evaluating the second version of our model with this set of metrics in Section 4.1.

Appendix B

Method for Architectural Trade-off Analysis

Faced with the difficulty of selecting the most appropriate architectural patterns, which guide the software architecture in the desired direction, we present in this appendix, a reproducible and systematic technique that aims to mitigate this problem. The focus of the proposed technique is to identify architectural patterns that affect a set of predefined quality attributes.

We present below a brief description of the step-by-step technique. Ideally, two researchers/professionals should perform the technique in order to minimize bias and provide more reliable findings. Thus, pattern selection (step 4) and trade-off analysis (step 5) must occur autonomously. Then, conflicting decisions must be discussed until a consensus is reached.

1. **Define the analysis objective:** Delimit the investigation scope, making clear the architectural style(s) that will be studied and what kind of trade-off we want to analyze. This will make it easier to perform steps 2 and 3;

2. **Select pattern collections sources related to the type of architecture investigated:** Define a search strategy. Choose keywords, types of materials (such as web pages and papers), consult documents and studies that investigate architectural patterns related to the desired scope;

3. **Choose quality attributes for trade-off analysis:** Based on researches and specifics of the software project, define the quality attributes that will be considered in the trade-off analysis. We imagine it is viable to perform this analysis considering up to 4 attributes, more than that would make the selection process very difficult;

4. **Select the patterns applicable to selected quality attributes:** Read the context, problem, and solution of each pattern. If more information is needed, read the positive and negative consequences of the pattern. Furthermore, check for duplicate patterns with different names. If existing, to avoid redundancy, only one of the patterns should be considered in the analysis. At first, each researcher must carry out the selection of patterns independently. Then, the obtained results must be

compared and conflict points resolved;

5. **For each selected pattern, check the trade-offs:** Based on the description of each pattern, analyze and record the impact generated by each structural attribute. In order to carry out this analysis, it is important to define parameters to classify the impact. Furthermore, as in the previous step, each researcher must carry out this analysis independently. Then, the obtained results must be compared, and conflict points resolved.

We hope that these steps help software architects obtain insights and evidence about the impact of architectural patterns on certain investigated quality attributes.

B.1 Demonstration – Architectural Trade-off Analysis with Microservices Structural Attributes

In this section, we demonstrate the process of applying our method for architectural trade-off analysis. To carry out this demonstration, we are going to analyze the trade-offs of adopting patterns on structural attributes of the microservices architectural style. The execution of each methods step is described below.

Analysis objective: Identify the influence that architectural patterns have on the structural attributes of microservices.

Patterns collections sources: We search for pattern collections on Google engine using the combination of the keywords microservice* and "architectural pattern*" (The symbol * allows capturing possible variations in search terms such as plural). Furthermore, we analyzed selected materials in systematic reviews that investigate microservices patterns. The criterion used to select the patterns collections was the description quality. To perform this trade-off analysis, we selected four microservices patterns collections:

- Microservice Architecture - A Pattern Language for Microservices, by Richardson (2020)
- Design Patterns for Microservices, by Wasson (2017);
- Implementation Patterns for Microservices Architectures, by Brown and Woolf (2016);
- Microservices Migration Patterns, by Balalaie, Heydarnoori, Jamshidi, et al. (2018).

After analyzing the selected collections, we identified 81 microservices patterns (considering repetitions).

Quality attributes: During this analysis, we are considering the following microservices structural attributes:

- **Module/service size**: We are going to analyze aspects related to services' and modules' size. For this, we are going to consider information, such as services' and modules' scope, number of service functions, and the number of services per module.

Therefore, the objective is to analyze if each pattern has an influence on the increase or decrease in the services' and modules' size;

- **Database sharing between modules/services**: We are going to consider information about the database architecture, especially about the number of databases shared between modules/services. Therefore, the objective is to analyze if each pattern influences the increase or decrease number of shared databases between modules/services;
- **Service coupling level**: We are going to analyze the degree of dependency/number of connections between services. Therefore, the objective is to analyze if each pattern influences the increase or decrease in the coupling level between services.

We selected these structural attributes based on researches related to microservices challenges. These researches indicate that these structural attributes have a relationship with the differentials and main characteristics of microservice architectural style.

Selected patterns: After reading the context, problem, and solution of each pattern and eliminating repetitions, we selected 24 patterns for trade-off analysis. The selection was carried out by two researchers to reduce bias.

Figure B.1 presents these patterns and illustrates the grouping (according to the kind of problem it solve) and relationship between them. To illustrate the relationship between the patterns, we follow the three forms of representation adopted by Richardson (2020): a successor pattern that solves problems identified in the application of a predecessor pattern; patterns that present alternative solutions to the same problem; and patterns that solve problems in the same area, which can be grouped.

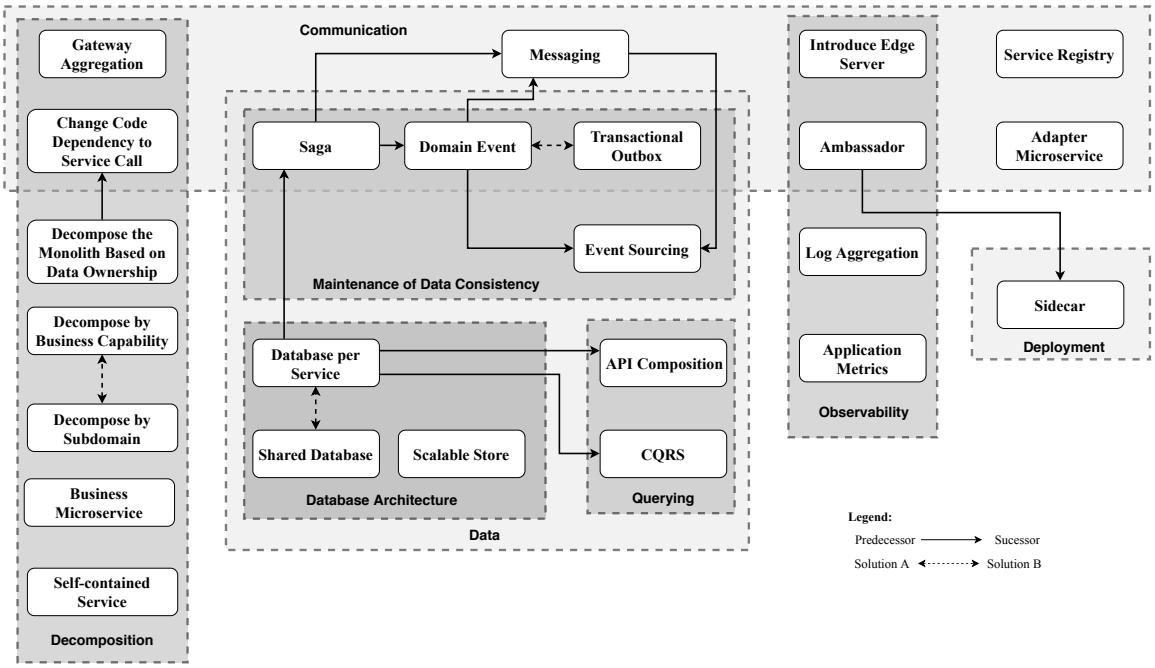


Figure B.1: *Pattern diagram of the selected patterns (inspired by Richardson (2020))*

Trade-off analysis: As previously defined, the objective of this analysis was to identify the influence those previously listed patterns have on **module/service size**, **database**

sharing between modules/services, and **coupling level**. To identify this influence, we read the description of each selected pattern. From this, we analyzed if the adoption of a particular pattern increased, decreased, or remained neutral on each structural attribute. It is noteworthy that, as in the previous step, to reduce bias, the trade-off analysis was performed independently by two researchers, then compared and discussed until consent was reached. Despite this, it is important to note that the results are limited to the researchers' knowledge and interpretation.

To illustrate the results of this analysis, we developed the trade-offs diagram (Figure B.2), which is an original contribution of this research. The trade-offs diagram must be created considering the relationship between the investigated quality attributes and their possible impacts in the architecture. Thus, the analyzed attributes represent the diagram dimensions, and the set of trade-offs must be arranged in these dimensions. Note that this diagram version can only represent three dimensions. This diagram aims to put together in one place all trade-offs of the analyzed attributes, facilitating the interpretation of results. The result of the trade-off analysis for each pattern is detailed below:

- Pattern that decreases the module/service size, number of database sharing between modules/services, and service coupling level:

- `Business Microservice`: *For each business function (some routine process or activity of an organization), an exclusive microservice is implemented, which encapsulates the business logic and makes it composable.* Since the aim is to define the service's scope, the tendency is to reduce its size. Furthermore, as the isolation of databases is done following the services' scope, this contributes to independent evolution, decreasing the number of shared databases and the level of coupling between services (Brown and Woolf, 2016).

- Patterns that decrease the module/service size and service coupling level, but maintain neutral the number of database sharing between modules/services:

- `Decompose by Business Capability`: *Each microservice represents a business capability.* Delimit the microservices scope according to business capability, makes them cohesive. Furthermore, it contributes to microservices isolation and coupling decrease. Nevertheless, identifying business capability is a subjective and challenging task (Richardson, 2020);

- `Decompose by Subdomain`: *Each microservice represents a Domain-Driven Design (DDD) subdomain.* Delimit the microservices scope according to the project domains, makes them cohesive. Furthermore, it contributes to microservices isolation and coupling decrease. Nevertheless, identifying project domains is a subjective and challenging task (Richardson, 2020) (a.k.a. Decompose the Monolith (Balalaie, Heydarnoori, Jamshidi, et al., 2018));

- `Decompose the Monolith Based on Data Ownership`: *Decompose a monolithic system according to the ownership of a cohesive set of data.* The monolith decomposition occurs around data ownership, which reduces the level of coupling because it reduces the intense exchange of data between microservices. However, this pattern is impractical in complex domain systems (Balalaie, Heydarnoori, Jamshidi, et al., 2018).

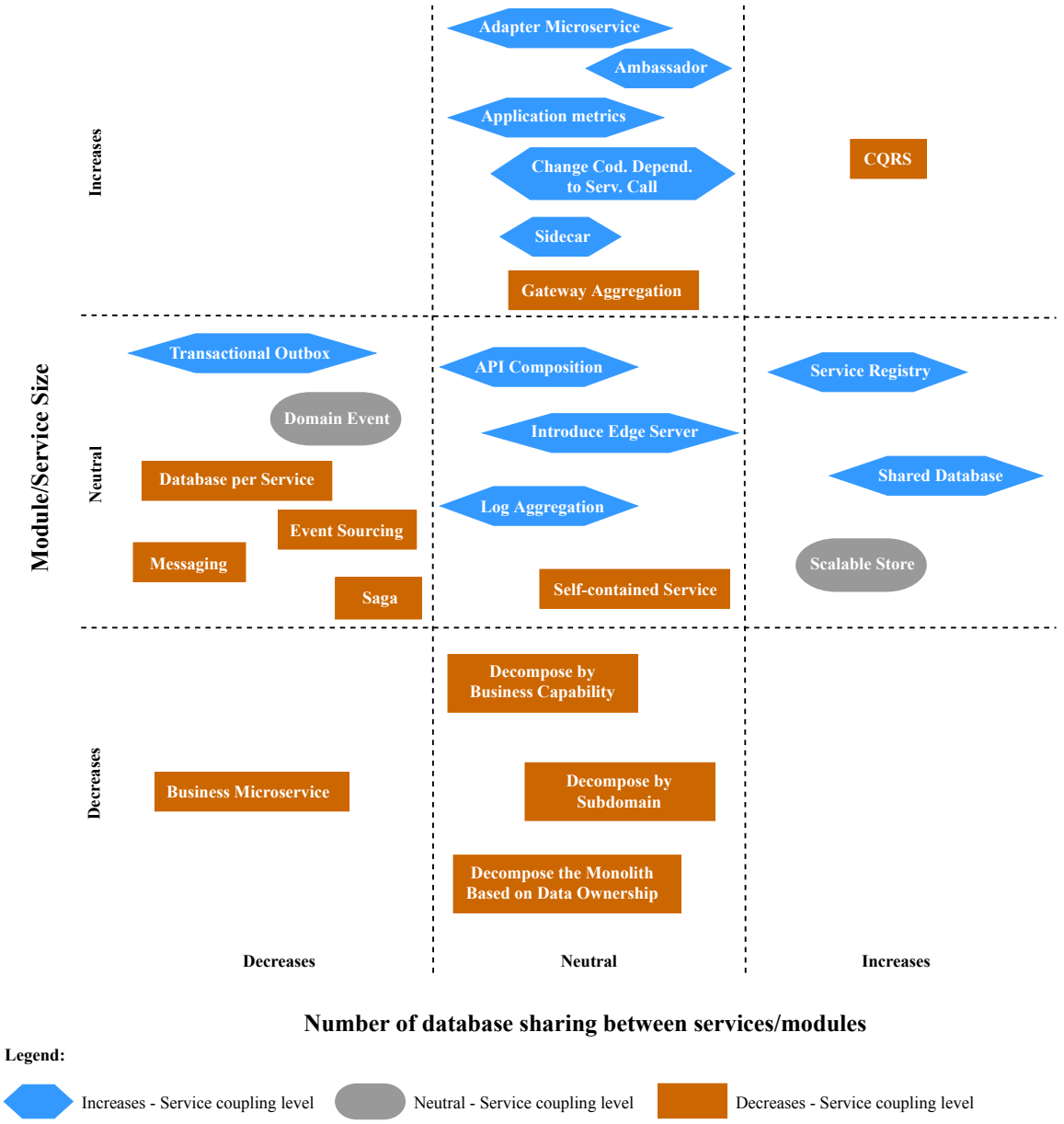


Figure B.2: *Trade-offs diagram*

- Patterns that maintain neutral the module/service size, but decrease the number of database sharing between modules/services and service coupling level:

- `Database per Service`: *Each microservice has its own private database, accessible only through its API.* The trend is to reduce the number of databases shared between microservices. Furthermore, it reduces the coupling between services and provides greater flexibility in the adoption of databases (each one can be different from the other). However, it is complex to guarantee transactions between different microservices (Richardson, 2020);
- `Event Sourcing`: *To reliably and atomically update the database, the state of a business entity is persisted as a sequence of events.* By persisting the status of an entity as a sequence of events, the need to share databases between microservices decreased.

Furthermore, it dramatically decreases the coupling between microservices. On the other hand, it is difficult to learn and implement, as well as creating eventual data consistency (Richardson, 2020);

- `Messaging`: *To implement inter-service communication, use asynchronous messaging.* By storing messages exchanged in a buffer, there is less need to share databases. Furthermore, it reduces the coupling between microservices, since it decouples the message senders from the recipients. Despite these advantages, it depends a lot on a message broker (which must be very resilient) (Richardson, 2020);

- `Saga`: *To maintain data consistency across services, business transactions that span multiple services are implemented as a sequence of local transactions.* Through the sequence of local transactions, it contributes to the reduction of the number of databases shared by different microservices. Furthermore, it provides a way to deal with data consistency between microservices, addressing the transaction problem, while increasing the complexity of communication (Richardson, 2020).

- Pattern that maintains neutral the module/service size and service coupling level, but decreases the number of database sharing between modules/services:

- `Domain Event`: *Organize the business logic of a microservice as a collection of DDD aggregates that publish an event whenever data changes.* The use of domain events and DDD terminology allows decoupling the connection between microservices based on the domain. Furthermore, when creating a collection of aggregated data, the tendency is to decrease the need to share databases between different microservices (Richardson, 2020).

- Patterns that maintain neutral the module/service size, decreases the number of database sharing between modules/ services, and increases the service coupling level:

- `Transactional Outbox`: *To maintain data consistency across services, messages are inserted into an outbox table as part of a local transaction, and a separate process publishes such messages to a message broker.* A message outbox is injected into the persistence command, based on the atomicity of the database command. The outbox is then read through a separate process to publish to a broker. This strategy provides greater flexibility and resilience to the event bus. However, the development process becomes more complicated (Richardson, 2020).

- Patterns that maintain neutral the module/service size and number of database sharing between modules/services, but decreases the service coupling level:

- `Self-contained Service`: *To handling with synchronous requests, design services that do not wait for the response from any other service.* By definition, it is a service that depends little on others, which reduces coupling but increases the microservices internal complexity (Richardson, 2020).

- Patterns that maintain neutral the module/service size and number of database sharing between modules/services, bat increase the service coupling

level:

- `API Composition`: *Implement queries invoking the microservices that own the data and executing an in-memory join.* When the data is separated into different microservices, it provides a way of unified consultation. On the other hand, it creates a service that is tightly coupled with others (Richardson, 2020);
- `Introduce Edge Server`: *Create a layer of indirection in the system, which can do dynamic routing based on a predefined configuration. This layer is useful for hiding internal complexity from end-users and monitoring overall services usage.* Adds an abstraction layer to the outside, which facilitates access for clients external, but concentrates much processing in a single point, which is tightly coupled to the others (Balalaie, Heydarnoori, Jamshidi, et al., 2018);
- `Log Aggregation`: *To understand the behavior of the application, use a centralized logging service that aggregates logs from each service instance.* Since it is a mechanism that centralizes logs, it is necessary to connect to all microservices, thus increasing the coupling level at a single point (Richardson, 2020) (a.k.a. Log Aggregator (Brown and Woolf, 2016)).

- ***Patterns that maintain neutral the module/service size and service coupling level, but increases the number of database sharing between modules/services:***

- `Scalable Store`: *The proposal is to have a distributed, scalable and resilient database. To achieve this, all states are placed in a Scalable Storage, being made available and shared by any number of application runtimes.* A distributed database is adopted, shared between instances of the same microservice, which, despite this, provides greater internal/implicit data consistency (Brown and Woolf, 2016).

- ***Patterns that maintain neutral the module/service size, but increase the number of database sharing between modules/services and service coupling level:***

- `Service Registry`: *For the client, find the available instances of a service, implement a database with the service instance locations.* The creation of a database that stores the microservices locations instances facilitates the discovery of such instances. On the other hand, a service (Registry) dependency is created, increasing the coupling. Furthermore, in the case of Registry failure, the problem of (eventual) data consistency arises (Richardson, 2020) (a.k.a. Introduce Service Registry (Brown and Woolf, 2016) or Introduce Service Registry Client (Balalaie, Heydarnoori, Jamshidi, et al., 2018));
- `Shared Database`: *Multiple microservices share a single database.* Provides ACID (Atomicity, Consistency, Isolation, and Durability) transactions and strengthens data consistency. However, considerably increases the coupling and database sharing between microservices (Richardson, 2020).

- ***Patterns that increase the module/service size and service coupling level, but maintain neutral the number of database sharing between modules/services:***

- `Adapter Microservice`: *Builds a simple adapter microservice that converts an API*

outside the microservice format to an API that the microservice client will expect. A new microservice – highly – coupled with a legacy service is added. On the other hand, a new communication interface with the rest of the system is gained, which is more flexible and consistent with the microservices architectural style (Brown and Woolf, 2016);

- `Ambassador`: *Acts as a proxy between the application and external services, aiming to make tasks, such as monitoring, logging, routing, and security, in an independent way.* Peripheral functionalities are removed from the business logic process, which increases the dependency between microservices. On the other hand, the development process is favored since a microservice is created to act as a Proxy. This pattern can be considered to be a superset of the sidecar, the difference of which is that it does not necessarily follow the life cycle of the main application (Wasson, 2017);

- `Application Metrics`: *Creates a service to gather statistics about individual operations, in either passive or active way.* Aggregates metrics into a single service, slightly increasing the number of microservices. Furthermore, to perform the metrics monitoring action, the level of coupling between microservices increases and is created a single point of failure (Richardson, 2020);

- `Change Code Dependency to Service Call`: *The strategy is to create services to reduce code dependency and facilitate independent scalability.* It transforms reused code into a requested service. This strategy can increase the number of microservices and the coupling between microservices and creates a single point of failure (Balalaie, Heydarnoori, Jamshidi, et al., 2018);

- `Sidecar`: *Processes or support services that are deployed with the main application, but provide isolation and encapsulation.* It is a subset of the ambassador pattern, the difference of which is that the sidecar necessarily follows the life cycle of the main application. Therefore, peripheral functionalities are removed from the business logic process, which increases the dependency between microservices. On the other hand, the development process is favored since a microservice is created to act as a Proxy (Wasson, 2017).

- Patterns that increases the module/service size, maintains neutral the number of database sharing between modules/services, and decreases the service coupling level:

- `Gateway Aggregation`: *Reduces communication between consumers and services. For this, requests from multiple microservices are aggregated into a single request.* It avoids a high rate of exchange of requests between client and server, adding several services in one, which deals with sub-requests. However, it creates a microservice that acts as a Gateway, which becomes a single point of failure (Wasson, 2017);

- Patterns that increases the module/service size and number of database sharing between modules/services, but decreases the service coupling level:

- `Command Query Responsibility Segregation (CQRS)`: *Creates a read-only copy of a database (constantly updated), where queries can be efficiently performed.* Increases scalability and decreases the coupling between microservices. However, it creates

> a new service responsible for the management of the historical database, which is shared with all microservices with "interest" in the domain, and which offers eventual consistency (since the Event Sourcing pattern is generally adopted in its implementation) (Richardson, 2020);

When analyzing the results more broadly, we realize that 11 out of 24 patterns increases the level of coupling between services. This point deserves attention, as this structural attribute is related to the loose coupling characteristic of the microservice architectural style.

We identified seven patterns that, when adopted, can increase the module/service size. These patterns typically perform monitoring activities, such as `Ambassador` and `Application Metrics`. When we analyzed the influence of patterns on databases, we found out that four patterns increase the number of databases shared between modules. Similar to the previous case, these patterns have a contrary effect on the independence proposed in the microservices definition.

We observe that patterns that reduce the shared use of databases (five out of seven) are mostly related to communication between services. Furthermore, four among these five are about exchanging asynchronous messages and being event-driven. From that, it is safe to assert the adoption of asynchronous event-based communication reduces the shared use of databases.

We confirmed that following a pattern to decompose the parts of the system into microservices leads to decoupled, smaller and more cohesive microservices. On the other hand, we could verify it does not directly affect database sharing.

The combination of patterns associated with asynchronous messaging – `Event Sourcing`, `Messaging`, `Saga`, and `Domain Event` – with patterns of decomposition and isolation – `Self-contained Service`, `Database per Service`, `Decompose by Business Capability`, `Decompose by Subdomain`, and `Decompose the Monolith Based on Data Ownership` – creates a powerful pattern set. The adoption of this set of patterns improves the analyzed attributes in this work. However, it is essential to note that a more careful study on their actual application is needed.

B.2 Discussions

After applying the proposed method and analyzing the results generated, we believe that we have obtained interesting insights, which can guide software architects to identify the most appropriate architectural patterns for a given context. However, since this is the first experiment using this method, we believe that the steps can be refined to improve the execution dynamics and the quality of the results.

Furthermore, we consider the proposed method drove the discussion to take pragmatic decisions about the analyzed patterns. That means the method conducted the discussion highlighting the trade-offs of each pattern and ending up in different outputs. Broadly, we realized that there are cases when the discussion brought by the method are enough to discard the hypothesis of adopting that pattern; there are cases when the method guides to the adoption of the analyzed pattern; and there are cases when neither is possible

and so further investigation is needed, such as an experiment or prototype. Either way, the application of the method conducts enlightening discussions around its purpose, the trade-offs of patterns.

It is important to note that during the method application, some difficulties were faced, the main one being related to the information subjectivity. Therefore, one of the important criteria for selecting patterns collections sources is the patterns description quality. Another important aspect is that the patterns selection and trade-off analysis are made by at least two people. This is important not only to reduce bias but to stimulate reflection and increase the analysis quality.

Appendix C

Roadmap for the Manual Collection of a Service's Metrics

Service name:

1. What are the responsibilities of this service?
2. What is the name of each operation (endpoint) exposed by this service?
3. What is the name of the services that invoke each operation of this service?
4. What is the protocol (REST, gRPC) used in each synchronous connection of invocation of this service?
5. What is the label of each topic published by this service in a message queue?
6. What is the label of each topic consumed by this service from a message queue?
7. What is the name of the source services that publish the topics consumed by this service?
8. What is the identifier of each data source accessed by this service?
9. What is the type (relational database, NoSQL, search engine, spreadsheet, text file, etc) of each data source accessed by this service?
10. What role (principal, cache) does each data source play in this service?
11. What type of action (read, write, or both) does this service perform on each data source it accesses?
12. What are the main technologies (programming language, IDE, etc) adopted to develop this service?
13. Must this service be deployed together with other services (i.e. is this service has deployment dependency)?
    - If so, what is the name of the other services that must be deployed together with it?

Appendix D

Supplementary Material for the Case Studies

This appendix presents complementary material to replicate the case studies.

D.1 InterSCity Case Study - Interview Script - CharM Evaluation

Authors: Thatiane de Oliveira Rosa, Eduardo Martins Guerra, and Alfredo Goldman

Starting the interview

- Remembering the terms of the consent form
- Explaining research objectives
- Explaining the research protocol
- Ask if you have any questions about the videos
- Asking permission to record

Part 1 - Interviewee experience

1. How long have you been working with systems with a service-based architecture?
2. Describe, in general terms, your trajectory with service-based systems, citing your main experiences.
3. How long have you been working/have worked with the InterSCity Platform?
4. What role(s) do you/did you play in the InterSCity Platform project?

Part 2 - General questions about the InterSCity Platform

1. Do you consider that the platform's current architecture is the most suitable for the purposes for which it is used? Justify your answer.

2. In your opinion, what are the strengths of the InterSCity Platform architecture?
3. In your opinion, what are the weaknesses of the InterSCity Platform architecture?
4. In your opinion, what quality requirements are desirable for the InterSCity Platform?
5. In your opinion, what quality requirements does the InterSCity Platform currently meet?
6. In your opinion, which architectural patterns are present in the InterSCity Platform?

Part 3 - Evaluation of the CharM and the architectural characterization

1. How do you assess the characterization of the platform architecture generated from our model?
    - From 1 (useless) to 5 (very useful), how useful is our model for understanding the architecture of the InterSCity Platform?
        - If it is useless, do you have any suggestions for improvement to make the characterization more useful?
    - From 1 (very difficult) to 5 (very easy), what is the degree of ease of understanding the result of the architectural characterization of the platform?
        - If it is very difficult or difficult, what point do you think got confused in the characterization?
    - From 1 (Totally incoherent) to 5 (Totally coherent), how coherent is the characterization result with the reality you know?
        - If it is totally incoherent or incoherent, which point is dissonant from reality?
2. Based on the characterization presented, do you believe that it was possible to:
    - Better understand the architecture of the platform or was the information indifferent?
    - Discover something new that you did not know about before or no new information was presented? If you discovered something new, could you tell us what it was?
    - Do you consider that the data presented can be used as a guide for the evolution of the platform architecture or are they insufficient to support decision-making?
3. Considering the structural attributes of size and coupling used as the basis of our research, do you have any suggestions for improving the characterization model presented? For example, analysis of other dimensions and/or metrics that you believe could be more useful for understanding the architecture and guiding its evolution.

Part 4 - Evolution of the InterSCity platform architecture

1. Considering the metrics presented and analyzing each service, is there any aspect (size, data source sharing, synchronous or asynchronous coupling) that you believe could be changed to improve the architecture of the platform?

- (if yes) What do you think can be changed?
- (if yes) Do you have any ideas or suggestions on how this change could be made?

2. Considering the metrics presented and analyzing the whole platform, is there any aspect that you believe can be changed to improve the architecture of the platform? For example, increase/decrease the number of services or change the data source sharing scheme.
    - (if yes) What do you think can be changed?
    - (if yes) Do you have any ideas or suggestions on how this change could be made?

Closing

1. Do you have any questions for me or final comments?
2. Can you suggest other people who can be interviewed and contribute to our research?

D.2 Interview Code Book of the InterSCity Case Study

This appendix presents the code book generated from the qualitative analysis of the interviews carried out during the InterSCity case study. This material serves to understand the codes and categories identified during the qualitative analysis. As well as obtain details of which participants cited each code, the number of citations of each code, and examples of excerpts from the interviews from which each code emerged.

Code category	Category description	Code	# of participants who cited the code	Participants id who cited the code	# code citations	Code example
Uses of the model	Tasks or situations in which the model can be useful for professionals who work with SBA	Helps an expert to understand the organization of the architecture	1	P5	1	I think it is quite useful to understand how the architecture is organized, but if it is seen by someone who already knows the platform well. [P5]
		To understand relationships	6	P2, P7, P8, P9, P10, and P11	7	I actually got to know more about how my microservices relate. I think this was a new thing for me, I had no idea about the dependencies... about the communications between the microservices. [P10]
		To confirm intuitive knowledge	2	P1 and P8	3	I believe that this vision seems to have reinforced the idea I already had of the platform. For me, the visualizations generated by the model confirmed that the platform is like that. Because it was a third party stating things I knew intuitively. It showed me through metrics that what I perceived was really true. Because until then, what I saw was code, not values. [P8]
		To discover patterns	1	P11	1	From the characterization, I could understand that one of the architecture patterns is the database per service. [P11]
		To discover differences between services	3	P4, P5, and P9	3	The idea of comparing, I had never thought of… when you are implementing you have an idea of which is the biggest or which is the smallest service. But I consider it is cool that you can make a ranking with this information. You can have an ideia of what is bigger and what is smaller, so I think that is cool. [P5]
		To discover information about data storage	1	P10	1	In fact, I ended up getting to know more about how my microservices are related... and also about the database, where they store the information. [P10]
		To discover new information	6	P2, P4, P5, P7, P10, and P11	6	I didn't know which was the biggest or the smallest service. I belive this was a new thing for me. [P2]
		To understand the architecture	9	P1, P2, P3, P5, P6, P7, P9, P10, and P11	20	I consider this type of architecture study very important, because when we begin to work, even though we've had some contact with it (kind of architecture)... To do a project within the platform, you have to understand everything at once, it's quite complex... pretty hard. So I think this kind of analysis can be useful. I don't know what kind of documentation you could generate with your research, but I think it could be very useful if you had some kind of report that could help a person, for example, who will have contact with InterSCity or another platform, could help us, right away, to understand what is happening there, more or less. [P7]
		To explain the architecture	3	P1, P3, and P6	3	If I need to present the InterScity architecture to someone, I would definitely like to have this kind of information. [P1]
		To guide evolution at a high level of abstraction	9	P1, P3, P4, P6, P7, P8, P9, P10, and P11	10	Of course, there is a limit to this, but I think it is possible to look at this model and say, let's try to work in this sense, to reduce the synchronous and increase the asynchronous, when necessary. [P1]
		To identify the importance of the services	2	P7 and P9	2	There are some services that I didn't know, that were just as important, such as the Resource Catalog. [P7]
		To identify coupling	5	P2, P8, P9, P10, and P11	6	I could see better how everything was related and how they worked. How big were they or how much communication did they have with each other. I considered it very interesting. [P2]
		To identify legacy things	1	P5	1	One thing that I think can also help is in identifying legacy stuff. For example, I have a service that I know no longer does something it did one day, and then I see it still has asynchronous communication. [P5]
		To identify things should be divided	3	P4, P5, and P8	4	Maybe it can be easy to identify services that are very big, that access many things, that are very complex. [P5]
		To identify points that need maintenance	7	P1, P2, P3, P4, P6, P8, and P9	11	With this model, we can identify some things that can be improved, and some others that are going well. [P1]
		To identify size	5	P2, P5, P7, P9, and P10	6	I had never counted how many operations each microservice makes available. I consider this interesting information. [P5]
		To help novices understand the system architecture	3	P4, P7, and P8	3	I believe that for those who do not know the platform, the presented characterization would give to them a better understanding of the platform. [P8]
		To remember the architecture	2	P3 and P5	3	As it had been a while since I worked with the platform, there were things I didn't remember. So, it was good that I could remember. [P5]
Aspects in which the model is not useful	Tasks or situations where the characterization generated from the model is not useful	The presented information rarely will be accessed	1	P3	2	The model gives us a very static view of a given moment, and then you wouldn't use that information as much anymore, just from time to time. [P3]
		Did not add new information	5	P1, P3, P6, P8, and P9	7	Considering that I already knew the architecture, it was a good visual presentation, but it didn't bring me any new information. [P6]
		Not helpful for understanding the architecture (for someone who already knows)	1	P4	2	So, related to the understanding, I think it doesn't help much. If you don't know anything, I think it helps, but for those who already know the architecture, I don't think it helps much. Because I consider that it is missing detailing of the architectural elements. [P4]
		Not helpful to find things that are missing	1	P4	1	I think the model doesn't help to find out what's missing in the architecture. [P4]
Model improvements suggestions	Aspects that can be improved in the model to make it more useful and easier to understand	Complementary information - to present complementary information about the service	4	P1, P3, P4, and P5	12	Maybe, with more complementary information from each service, it becomes very useful for a project. Because I imagine viewing this on an interactive website. For example, you have the synchronous coupling here, then I click there and see what these couplings are, what these calls are, or the line of code that makes this call, something like that. Or, asynchronous coupling, I click there and see the events it depends on, for example. I find this super helpful. [P3]
		Complementary information - to present relationship information	2	P3 and P9	4	It would be interesting if there were more details in the diagram, saying what the communications are. Because there are some that have both synchronous and asynchronous communication... Besides having an arrow, saying that this system communicates with this one, also saying which is, for example, the endpoint that this service is calling from the other. This would give to us a good overview. [P9]
		Complementary information - to present information about the type of DBMS	3	P4, P6, and P8	4	Another suggestion, in the database part, where you define whether each service has an individual database or not. I think it makes sense to say which type of database (relational database, NoSQL). It is a new level of detail, but if you look at the platform proposal, it is a level of detail that is interesting because it makes a difference whether you use one database and not another. [P4]
		Improve the explanation - to adjust the presentation of coupling metrics	2	P3 and P7	2	Maybe being a little more visual on the “who depends more on whom” part can make it a little easier. [P7]
		Improve the explanation - to define the meaning of operation	1	P3	1	The size of the service, for me, was confusing. What would an operation be? It got a little confusing for me. And many times I couldn't map very well...I didn't know how to map very well what an operation would be. [P3]
		Improve the explanation - to define the meaning of synchronous and asynchronous	1	P2	1	The only thing you can present in more detail is explaining what is synchronous and asynchronous. Because I was a little confused about what was each one. [P2]
		Improve the explanation - better explain the first dimension	2	P3 and P6	3	I had one doubt at the end of the video, when you explained that you compared the characterization of each service with themselves. In the beginning, you were saying that the Resource Adapter is smaller, compared to the Resource Catalog. You were following the line of reasoning in this sense... For example, the Actuator Controller has two operations, and you said that it is smaller because compared to the Resource Catalog it had far fewer operations, it was too far to the left on the ruler. But at the end of the video, when you give the overview of the architecture, you suggest that the platform is small, but I couldn't understand when compared to what? [P6]
		Improve the explanation - better explain the second dimension	1	P3	1	Actually, I have some suggestions. I think there are things that are very important, for example, sharing databases, it tends to be an anti-pattern... it's an anti-pattern... So the tendency would be for you not to do that. So, maybe, sharing databases gain a very big dimension here. Because my point is, isn't it as important as any other anti-pattern? [P3]
		Metrics - to adopt some responsibility (cohesion) metric	1	P3	1	I don't know if the operation is a good metric for the size of the service. I don't know if it's so relevant. For example, sometimes I may have forty operations, but the responsibility of the service is only one. I know it seems a bit contradictory, but perhaps evaluating responsibility, in my opinion, is more important than evaluating the number of operations... Sometimes I have a service that has two operations, but one operation has nothing to do with the other, so this means lower cohesion... I think there are other more interesting metrics for the size of the service. In my opinion, it should be the size of the responsibility of the service or the weight of the service or of the responsibility. [P3]
		Metrics - to present information related to tests	2	P1 and P9	3	I think if you added some information, such as if there are test classes or if there are integration tests between these microservices, I think it would be cool. [P1]
		Metrics - to present metrics related to classes and methods	2	P6 and P9	3	I thought that maybe it would be possible to calculate some simpler metrics. Such as lines of code, number of classes, number of methods per class, and other cohesion and coupling metrics of classes and methods. And try to create some formula that would be possible to infer the module drawing size.[P6]
		Metrics - to present line of code metrics	3	P6, P7, and P9	3	What comes to me instantly in my mind, is something like lines of code. I don't know if it can be useful or if it can be a fake thing. [P7]
		Metrics - to present some metric that signals whether the service is micro or not	1	P9	1	Another thing I was thinking... I don't know if it's a question that can be answered by your model. But, a question that I would find very interesting would be if there was a way for the model to answer me, if these microservices make sense, or if they are too micro. A way to calibrate thinking based on these metrics that you collected. It would provide a basis for making a decision about the size of each service. [P9]
		Metrics - to collect dynamic metrics	3	P8, P9 and P10	5	Maybe another dimension that might be interesting is knowing how scalable each of the services is. Because if I have a bottleneck in the Data Collector, I can just upload more Data Collector instances, or I have to look elsewhere. But, I don't know if this is too out of your research scope. [P9]
		Metrics - to map external dependencies	3	P1, P5, and P10	4	A point that I think is a little complex, but that would help a lot, would be to start an analysis of the dependencies of each of these microservices with external APIs. Not all, since I think they can be in the number of gigantic granularity. But, at least the main ones. So, it would be good for this model to indicate the amount or what these external dependencies are. The APIs that are used by the microservices. I think this would help a lot, especially for those who think about adopting the platform or not.[P1]
		Visualization - to present complex things in a larger size	1	P6	2	For example, if the Resource Catalog has many more lines of code than the Data Collector, then it would be presented in a larger size than the Data Collector. [P6]
		Visualization - to present more elements that compose the architecture	2	P6 and P9	3	I don't know if it's the goal, but there's a lack of information on how messages are exchanged. Not in the sense that they are synchronous or asynchronous. But, for example, there is a message broker, and the database is Mongo. I don't know if it makes sense to talk about tools in this characterization. But, I think that, to really understand how the system is working, some more details are important. I think the gateway was also missing, which would be important to understand. [P6]
		Visualization - use colors to warn attention points	1	P6	1	I also think you could use some colors. I don't know if you thought about it too. For example, something that appears to be very large and complex can get bigger and redder in the sense of alert. [P6]
Architectural evolution	Aspects in the InterSCity architecture that the interviewees perceived should evolve, after analyzing the characterization generated from the model	To adjust coupling	4	P1, P6, P8, and P9	4	It certainly can be used as a reference to try to reduce synchronous couplings and increase, a little, the asynchronous ones. [P1]
		To divide the Resource Catalog service	2	P3 and P4	2	For me, the clearest thing is to separate the Catalog into more things. This service does a lot of things. I don't remember 100%, but I remember that one thing it did more was: both to register resources, and to check which resource does such a thing. It kind of has a piece of the Resource Discovery inside it, and I think that's very wrong. [P4]
		To review the database sharing strategy	2	P1 and P11	4	I can say that the fact that it uses an exclusive database for each of the microservices, for me, is a problem. This should be changed. Since we are in the smart cities context, the system must be very dynamic and large. So having a database available and consistent is one of the main factors. [P11]
		To change the architectural style to the monolithic	1	P6	1	I consider, in fact, it could be just one service. Although the services make sense and are logically divided, they are not complex services. So, you end up uploading different applications, which are not complex, but which add a lot of complexity to the architecture and to the understanding of the platform. In a monolithic system, in this case, I think you would probably have no performance losses and it would have a lot of gains. Such as ease of securing the platform and also introducing new people to work on the platform. [P6]

D.3 Industry Case Study - Interview Script - System Architecture Overview

Authors: Thatiane de Oliveira Rosa, Eduardo Martins Guerra, and Alfredo Goldman

Starting the interview

- Remembering the terms of the consent form
- Explaining research objectives
- Explaining the research protocol
- Asking permission to record

Part 1 - Interviewee experience

1. How long have you been working with systems with a service-based architecture?
2. Describe, in general terms, your trajectory with service-based systems, citing your main experiences.
3. How long have you been working/have worked in Company A?
4. What role(s) do you/did you perform in Company A?

Part 2 - General questions about the Company A systems

1. In your opinion, what are the main objectives of the systems developed by Company A?
2. What are the main technical challenges related to these systems?
3. Could you list the main features of these systems?
4. What are the desirable quality requirements for Company A's systems?
5. Among the mentioned quality requirements, which are currently met?
6. Among the mentioned quality requirements, which are not currently met?
7. What are the strategies (organizational and technical – practices, methods, patterns) adopted to develop Company A's systems?
8. From your point of view, give me a high-level explanation of the current architectural solution (if you prefer, you can draw or show me diagrams that already exist). Please, in this explanation, point out the main structural elements (and their objectives) (services, APIs, modules, databases, and relationships).
    - Considering this solution, could you tell me which is the most present architectural style? (layered, hexagonal, service-based, microservices, mixed).
9. In your opinion, what are the strengths of the current architecture of Company A's systems?

10. In your opinion, what are the weaknesses of the current architecture of Company A's systems?
11. From your point of view, give me a high-level explanation of the idealized architectural solution (if you prefer, you can draw or show me diagrams that already exist). Could you tell me the main points that will be changed/affected? Please, in this explanation, point out the new elements (and their objectives) and elements that will be excluded (reason) (services, APIs, modules, databases, and relationships).
    - Considering this new solution, could you tell me which is the most present architectural style? (layered, hexagonal, service-based, microservices, mixed).
12. In your opinion, what are the strengths of the idealized architecture for Company A's systems?
13. In your opinion, what are the weaknesses of the idealized architecture for Company A's systems?

Part 3 - Architectural migration process

1. In your opinion, what motivated the architectural style migration?
2. In your opinion, was this migration process a good decision?
3. Besides the service-based approach, were other alternatives considered?
4. Could you describe, in general terms, the steps of this migration process?
    - Please, cite the steps completed, the steps that are being executed now, and those that are yet to be executed.
5. What benefits did you perceive, until this moment, from the architectural migration?
6. In your opinion, what are the main challenges you have faced from architectural migration?

Closing

1. Do you have any questions for me or final comments?
2. Can you suggest other people who can be interviewed and contribute to our research?

D.4 Industry Case Study - Interview Script - Sub-system Architecture Overview

Authors: Thatiane de Oliveira Rosa, Eduardo Martins Guerra, and Alfredo Goldman

Starting the interview

- Remembering the terms of the consent form
- Explaining research objectives

- Explaining the research protocol
- Asking permission to record

Part 1 - Interviewee experience (just for new participants)

1. How long have you been working with systems with a service-based architecture?
2. Describe, in general terms, your trajectory with service-based systems, citing your main experiences.
3. How long have you been working/have worked in Company A?
4. What role(s) do you/did you perform in Company A?

Part 2 - Questions about the architecture of the Search System

1. In your opinion, what are the main technical challenges faced by your team to develop the system/services that your team is responsible for?
2. Considering the diagram below, available in the team's repository, could you briefly explain its elements and the interaction between them?

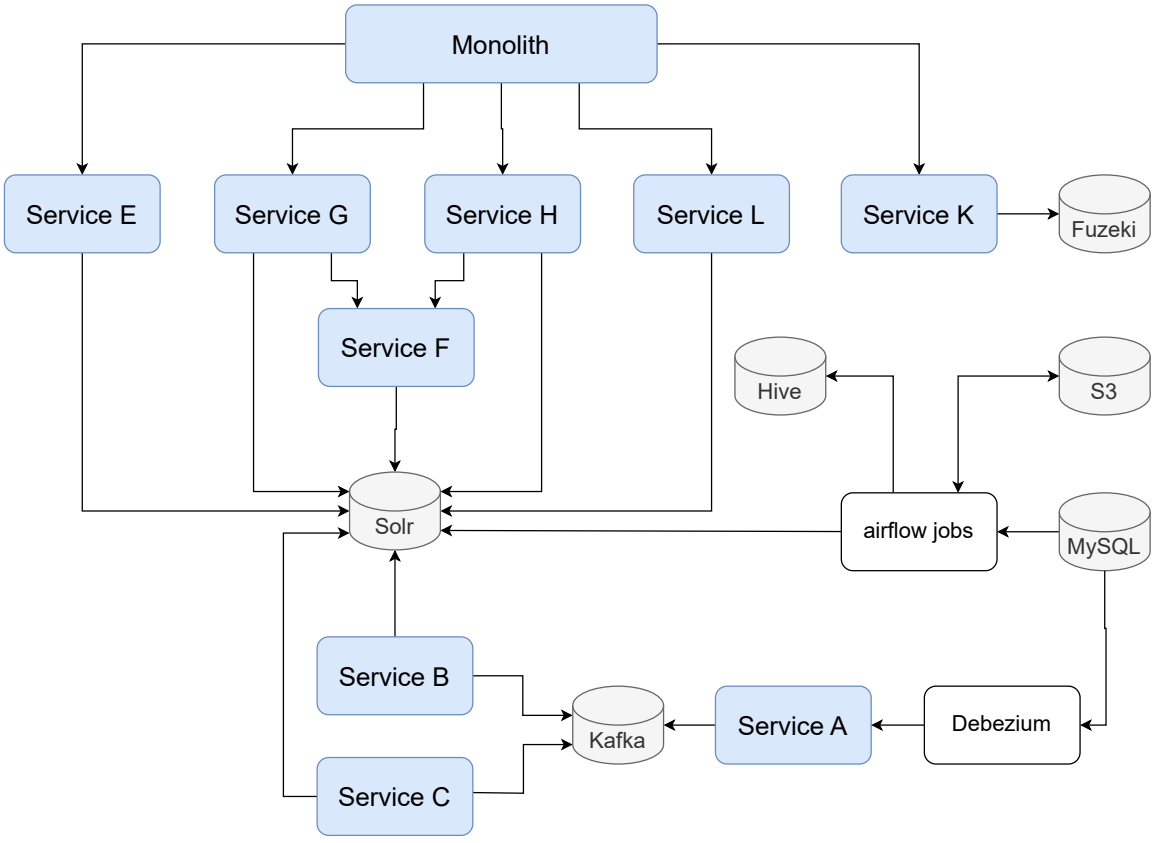


Figure D.1: *This diagram is a simplified and anonymized replica of the diagram available in the company repository.*

3. Still considering the diagram, which of the illustrated elements do you identify as a service that composes the search context?

4. Is there any other service that was not illustrated in this diagram but is your team's responsibility?
5. Is there a document where I can obtain a description of the responsibilities of each service that you mentioned?
    - If not, could you suggest someone who can give me this answer (the services' responsibilities)?
6. Which data sources are accessed by the services that you previously mentioned/identified?
7. Do any of the services you previously mentioned/identified access any data sources outside the search context?
    - If yes, what are these services?
8. Are there external elements (services from other contexts, third-party services, databases, ML models) that interact with the services of the search context?
    - If yes, could you indicate which services of the search context that have external interaction?

Closing

1. Do you have any questions for me or final comments?

D.5 Industry Case Study - Interview Script - Metric Collection

Authors: Thatiane de Oliveira Rosa, Eduardo Martins Guerra, and Alfredo Goldman

Starting the interview

- Remembering the terms of the consent form
- Explaining research objectives
- Explaining the research protocol
- Asking permission to record

For each previously identified service, ask the following questions

1. What are the responsibilities of this service?
2. What is the name of each operation (endpoint) exposed by this service?
3. What is the name of the services that invoke each operation of this service?
4. What is the protocol (REST, gRPC) used in each synchronous connection of invocation of this service?
5. What is the label of each topic published by this service in a message queue?

6. What is the label of each topic consumed by this service from a message queue?
7. What is the name of the source services that publish the topics consumed by this service?
8. What is the identifier of each data source accessed by this service?
9. What is the type (relational database, NoSQL, search engine, spreadsheet, text file, etc) of each data source accessed by this service?
10. What role (principal, cache) does each data source play in this service?
11. What type of action (read, write, or both) does this service perform on each data source it accesses?
12. What are the main technologies (programming language, IDE, etc) adopted to develop this service?
13. Must this service be deployed together with other services (i.e. is this service has deployment dependency)?
    - If so, what is the name of the other services that must be deployed together with it?
14. Do you have any other comments about an important service characteristic we did not address in the previous questions?

D.6 Industry Case Study - Interview Script - CharM's Evaluation

Authors: Thatiane de Oliveira Rosa, Eduardo Martins Guerra, and Alfredo Goldman

Starting the interview

- Remembering the terms of the consent form
- Explaining research objectives
- Explaining the research protocol
- Ask if you have any questions about the videos
- Asking permission to record

Part 1 - Interviewee experience

1. What role(s) do you/did you perform in Company A?
2. How long have you been working in IT area?
3. How long have you been working with systems with a service-based architecture?
4. How experienced do you consider yourself in the software architecture area?

- Novice – I understand the concept of software architecture, but I have never made architectural decisions in real environments
- Advanced beginner – I have experience in real scenarios and have already contributed to some architectural decision-making, but without much confidence
- Competent – I have already contributed to some architectural decision-making in different contexts and real scenarios, but I still have some difficulties
- Proficient – daily I make decisions related to software architecture consciously and with a good degree of confidence
- Expert – I deal with different computer systems with different architectures, and I can make decisions confidently and without significant difficulties

5. What is your knowledge level of the Search System architecture?
    - Very low – I understand the purpose of this system, but I don't know its architecture
    - Low – I know a little about the architecture of this system (I know some services), but I haven't had the opportunity to work in practice yet
    - Moderate – I know the architecture of this system (not in detail) and I've had the opportunity to work on some services
    - High – I know the architecture of this system and have had the opportunity to work on most of its services
    - Very high – I work daily with this system and I know very well the architecture and services that compose it

Part 2 - Evaluation of the CharM and the architectural characterization

1. From 1 (very difficult) to 5 (very easy), what is the degree of ease of understanding the dimensions of the model?
    - Did any part of the explanation get confused?
2. From 1 (very difficult) to 5 (very easy), what is the degree of ease of understanding the characterization of the Search System architecture (result generated by the model)?
    - Did any of the presented results get confused?
3. From 1 (useless) to 5 (very useful), what is the degree of usefulness of the model for understanding the architecture of the Search System?
    - Do you have any suggestions for improvement to make the characterization more useful?
4. From 1 (Totally incoherent) to 5 (Totally coherent), what is the degree of coherence of the characterization with the reality you know?
    - If there is any incoherence, which point is dissonant from reality?

5. In your opinion, based on the characterization presented, what is the use of the proposed model for Company A?

6. Based on the characterization presented, do you believe that it was possible to better understand the architecture of the Search System or was the information indifferent?

7. Based on the characterization presented, do you believe that it was possible to discover something new that you did not know before or that no new information was presented?

    - If you discovered something new, tell us what it was.

    - If nothing new was discovered, was the characterization helpful in confirming intuitive knowledge or remembering the architecture?

8. Do you consider that the data presented can be used as a guide for the evolution of the Search System architecture or are they insufficient to support decision-making?

9. Considering that we are performing a static analysis based on the structural attributes of size and coupling, do you have any suggestions for improving the characterization model presented? For example, analysis of other dimensions and/or metrics that you believe could be more useful for understanding the architecture and guiding its evolution.

Part 3 - Evolution of the Search System architecture

1. When analyzing the characterization of each service, is there any aspect (size, data source sharing, synchronous or asynchronous coupling) that you believe could be changed to improve the architecture of the Search System?

    - (if yes) What do you think can be changed?

    - (if yes) Do you have any ideas or suggestions on how this change could be made?

2. When analyzing the characterization of the search system as a whole, is there any aspect that you believe can be changed to improve the architecture of the Search System? For example, increase/decrease the number of services, change the sharing scheme of the data sources, and review the percentage of synchronous and asynchronous couplings among other things.

    - (if yes) What do you think can be changed?

    - (if yes) Do you have any ideas or suggestions on how this change could be made?

Closing

1. Would you like to add any remarks that were not possible to cite before or present a final comment?

D.7 Interview Code Book of the Company A Case Study

This appendix presents the code book generated from the qualitative analysis of the interviews carried out during the Company A case study. This material serves to understand the codes and categories identified during the qualitative analysis. As well as obtain details of which participants cited each code, the number of citations of each code, and examples of excerpts from the interviews from which each code emerged.

Code category	Category description	Code	# of participants who cited the code	Participants id who cited the code	# code citations	Code example
Uses of the model	Tasks or situations in which the model can be useful for professionals who work with SBA	To help novices understand the system architecture	5	P3, P13, P15, P17, and P18	5	I think this model may be very useful in onboarding new members. While watching the video (about the characterization of the architecture), I thought “Wow, P13 could use this a lot to show the architecture to the new members that will join our team”. [P17]
		To help an expert to understand the organization of the architecture	1	P18	1	I think that this model is very useful for anyone who has some experience in software development but doesn't know the systems, since it gives this high-level view. [P18]
		To understand relationships	3	P3, P12, and P17	6	I think that this model gives us a very interesting view of how the relationship between services is. And it is very cool. [P3]
		To confirm intuitive knowledge	4	P3, P12, P13, and P17	4	I didn't know all the components that well. So, the model presented to me some new information, even in points in which I had some intuition. For example, I already had an intuition that the system is strongly dependent on Solr. So that was, for me, pretty cool. Since there (in the characterization) we clearly see this dependence. [P3]
		To discover information about data storage	2	P17 and P18	2	With the model, I realized that some data source couplings had a consumer-producer characteristic since some services have data source coupling. I think Service D and Service H are examples of this. Both have a coupling to the same data source. Service D writes and Service H reads. I realized that they couple by data source into a consumer-producer model. This brought me some insights. To mainly observe this relationship between data sources: who is producing, who is consuming, who is reading and writing from the same data source... [P18]
		To discover new information	4	P3, P15, P17, and P18	5	Something new that the model brought to me... I didn't know the number of things that Service H does. Knowing the exact number gave a lot of strength for this to be well-concretized. Of course, there's a bad smell here. [P3]
		To discover the number of data sources of each service	2	P3 and P17	2	I realized that Service H communicates with seven data sources and that's a lot. [P17]
		To discover the number of operations in each relationship	1	P3	1	One thing I found interesting was knowing the exact number of operations in each service interaction... [P3]
		To highlight differences between services	2	P13 and P18	2	I was aware that Service H was the biggest service. I just didn't know it was so much bigger than the other services. I don't know if it was because I believed that the other search services were bigger or if I really had no idea of the size of Service H. Seeing the result generated by the model, I think that the size of Service H is totally disproportionate compared to others. [P18]
		To different stakeholders share the same architectural vision	1	P3	2	I think the model is a great starting point for everyone to share the same vision of the system architecture. [P3]
		To document the architecture	1	P3	1	In the company, we have lean teams with high turnover. So, there are a lot of things that we depend on senior members to do. So, when we have this kind of documentation (generated by the model) I think it helps the team to share the same view of the architecture and reduces the worries of senior people. [P3]
		To support architectural discussions	2	P12 and P13	3	The result of the model matches a lot with a discussion that we had with the team. And it kind of shows what we discussed, but in another way. So it corroborates what we have been discussing. It is a view from outside, related to the things that we have been discussing to say “Look, we have discussed and there is another metric here that also says that this makes sense” [P13]
		To understand the architecture	6	P3, P12, P13, P15, P17, and P18	7	I think that the model is useful to understand the architecture of a system. Because one thing that I think is very difficult, personally... In systems like this, where there are several components, several services, and interactions between them... is how these interactions are and etc. The model presents a very nice view of things that I, really, didn't pay attention to before. [P12]
		To explain the architecture	3	P13, P17, and P18	3	The model can be used to explain the architecture during onboarding, but only to show the design and complexities and other things related... [P13]
		To guide evolution at a high level of abstraction	6	P3, P12, P13, P15, P17, and P18	15	I will go back several times in the same spots that I think it is very interesting. Service H with the highest score shows that it is the most complex. And this evidences that it could be done differently and that it is necessary to improve a lot of the things at this point. And the opposite too, that there are some services that are so small... and maybe it could make them healthier. So, in that sense, I found it quite useful in order to guide an architectural evolution. [P13]
		To identify coupling	3	P12, P17, and P18	5	When I joined the team, it was all very confusing for me, because the system is very big and complex. I think with this model, I was able to understand much better who depends on whom, and who is important to whom. So, I think it was really helpful for me. The model helps me to be able to see the whole case and where all these services fit in, and who they talk to. [P17]
		To identify if the things should be divided or joined	4	P3, P12, P13, and P17	5	Considering the size dimension, you can see, for example, that Service H is a candidate for us to refactor. We can extract other services from it. [P12]
		To identify components with many responsibilities	2	P3 and P13	3	I didn't know the number of things that Service H does. Knowing the exact number gave a lot of strength for this to be well-concretized. Of course, there's a bad smell here. [P3]
		To identify legacy components	1	P18	2	Looking at the relationship between Services H and D, I saw that between them there are two types of coupling... If Service D writes to a source and Service H reads from the same source, there may be some part of Service H that is in the scope of Service D. But the code is inside Service H, because it wasn't migrated or because of some mess... So, the model helped me to see some of those bounded contexts inside the Search System. The smaller contexts have a certain mixture... It looks like some extraction was started, but not ended. [P18]
		To identify the degree of importance or dependence of a service	2	P3 and P12	2	Based on the model, I was able to observe the services' dependencies. If many services depend on another service, this shows some architecture fragility that also leads to an evolution of the team going to a more autonomous side. Instead of having authority. [P12]
		To identify points that need maintenance	5	P3, P12, P13, P17, and P18	14	I think the work you've done makes it very clear to see the yellow (or redder) flags for us to act on. The difference was very clear, looking at all the services. Service H is the microlitter.... and this is within a context of a lean system, which is the Search... Another point that became clear... this pain or discomfort is the issue of database coupling. ... I think it was very clear that we have a serious problem with database coupling. [P3]
		To identify size	4	P3, P13, P17, and P18	4	With the model, I discovered the real amount of Service H operation... I was a little shocked by the number of operations of Service H. [P17]
		To remember the architecture	4	P3, P13, P15, and P17	4	As we often do not work with all of these services all the time, we may forget some of them... So, it (the model) was very useful for me to remember and see where they (the services) were properly fitted. [P17]
		To summarize architectural information	2	P3 and P13	2	I think it was really nice to see everything in one place. It was very good to have this information centralized. Because it is difficult to have a total understanding of all these services. I think the gain of this analysis is like, “Presto, it's on the screen”... as a whole thing. So, at this point, I think it (the model) is a great tool. [P3]
Model improvements suggestions	Aspects that can be improved in the CharM to make it more useful and easier to understand	Improve the explanation - better define the mening of module	1	P17	1	There are two points where I ended up getting a little lost... one of them is the part where you explain about the service per module. We have eight services, one service per module... I ended up getting a little lost here. [P17]
		Improve the explanation - to differ what are and which are the internal and the external components	1	P17	1	I don't know if you talked about the difference between internal and external services. There is a part in the video where you talked a lot about internal and external services, and I didn't really know how to differentiate which they were. I could only understanded, when you showed the results in the table. So, I don't know if it was a little lacking in the intro or if I ended up not paying enough attention. [P17]
		Improve the explanation - what is and what is not a service in the context of the CharM	1	P12	2	About the service definition. When I started watching the videos, I was looking at everything as a service, every component of the architecture. For example, for me, Data import would also be a service. After your explanation, I understood that in your service definition the Data import does not fit in it. But perhaps it is important to make this clearer in your explanation. [P12]
		Improve the explanation - to review the quality attributes adopted in the explanation of the rules	1	P18	1	At the time when you explain synchronous coupling. In your ruler, there was a relationship between (if I'm not mistaken) reuse and I don't remember which was the second one... In my understanding, I don't think that the fact of it being a synchronous or asynchronous coupling is a thing that will impact the reuse. For example, for me, I believe that... the synchronous and asynchronous complement are measurements that make a lot of sense to me, but I think the word “reuse” in the ruler took my attention a little bit. Because, in my understanding, I think that, for example, the granularity of one of the services has a much greater impact on reuse than whether it is synchronous or asynchronous... So, I didn't feel that the fact of being synchronous or asynchronous would impact the reuse metric. [P18]
		Metrics - to present the criticality level of a coupling	1	P17	1	The only improvement I could think of was the issue of dependency on one service on another. That is if a given dependency causes instability or unavailability of the dependent service. Because sometimes something depends on Service H, but if Service H doesn't respond, that's fine. Thus, this thing is not so dependent... For example, if Service F doesn't respond, Service H shouldn't cause an unavailability problem. Service H shouldn't be left out just because Service F couldn't fix that. So, I think that presenting the criticality of this coupling between services would be interesting. [P17]
		Metrics - to collect business metrics	1	P13	1	One thing that the model does very well is to look a lot at the metrics of what would be the tech part (technology in fact). You look at the number of connections (synchronous, asynchronous, and others). Maybe, one point could be added to the model... The product team helps us a lot in how we will architect the business. So, I think it would be very interesting to have something more in the model in terms of metrics, which could also help people in the business area. With a slightly higher level of abstraction, not just so technical as it is today. [P13]
		Metrics - to collect dynamic metrics	1	P3	1	I thought like this: how much is each of these services accessed? Because we understand here how they relate. But, suddenly, knowing which services are most accessed, within the chain... maybe it gives a sign of importance... I don't know if this is another type of analysis, more related to performance. [P3]
		Metrics - to indicate the number of data sources per module	1	P18	2	During the analysis, you talked a lot about data sources and how many of those services share a given set of data sources. But, having prior knowledge, I know that those data sources are all on the same database server. It's like having multiple schemas within the same database server... I know there are four services attached to a given data source. And it tells me here I have a problem, I have to break this. But if I look at the module level, I'll notice that out of the 18 data sources, I believe about 10 are in the same module. That is, 10 of them are in the same Solr. This could help me to identify that there is a module that is very critical, there. Because if it goes down, I would lose 10 data sources at once. So even if I didn't have sharing between these data sources (which is not the case), if they each had a data source, but they're all on the same server, I have a single point of failure. I think that's the only point that wasn't clear in the analysis and it was in the diagram. When you show the diagram, this is explicit, but in the model description, I don't remember you mentioning this explicitly. I think for some scenarios, maybe the data sources would have to have the same “treatment” as the services, as in… there is a coupling, but this coupling exists between the data sources, but the module also has a certain level of importance because it helps me identify single points of failure in the architecture. [P18]
		Metrics - to make the concept of operation more generic	2	P12 and P13	5	I was a little confused because there were some services in which the number of operations was equal to zero. This situation left me confused and thoughtful about what operations are. Basically, you defined it as endpoints, right? This, for me, raised some doubts. That's because the REST endpoint, which is what we have, is one way, but not the only one. Because, for example, if it was a gRPC call... it's another way of communication, but, in a generic way, I would consider it an operation. But, in the case of these services, it was zero... I think it was because you didn't consider another way of communication, which is via event. In that context, for me, it's the same thing. That is, the endpoint, gRPC or event. [P13]
		Visualization - to adopt the stacked bar chart to illustrate the size ratio of the components in the whole system	1	P18	1	One thing that I think would help me... There was a moment in the presentation when you showed a bar graph that had Service G, all the services, and the big Service H bar. In my way of analyzing, instead of being a bar chart with each service with its bar... if you were shown the same information in a stacked chart, would be better. It would be Service H and just above Service G... and on top of Service G another one... Then I could look and say “Ok, I can see the height of the Search System bar”. So, “The Search System bar has 30 operations”, then I look and Service H has 26 out of 30. Wow! This is a problem. 26 out of 100 is different from 26 out of 30. Its representation and how bad this level is... If I compare it to the second... The first thing you did already gave me that impression right away. Which was: the first is Service H, the second is Service G. And then you look and say “Wow, it really is much bigger than the other”, right? But, in the total universe, how much does it represent? I missed that a little bit during this characterization presentation. [P18]
Architectural evolution	Aspects in the Search System (of Company A) architecture that the interviewees perceived should evolve, after analyzing the characterization generated from the CharM	To rethink the data source coupling of Service H	1	P17	1	I believe that Service H's database sharing can be changed. [P17]
		To rethink the data source coupling between multiple services and the Solr	2	P3 and P13	2	We actually have a lot of coupling with Solr. So Solr is our big point of failure. If Solr falls, everyone falls with it. So, cool that the end result of your analysis clearly showed this. It's the very turning point. So, one thing that I think contributes to what we want to do in the team is to have more autonomy. So, having a Solr for each of the services… This is something that P2 has always defended a lot. And I share this idea because it makes sense. Even more so in a system like Search, which is super important. [P3]
		To rethink the couplings of Service F	1	P17	1	I think a bit of Service F could be changed. Service F is responsible for correcting the terms of a search. But it will only return the correction of that term the next time a new search is attempted... I think that two services share this with Service F, which is Service H and Service G, so I think there may be a way for both of them to obtain the same results. I don't know what to do about it... I have no idea how to try to solve this, but I think it's an interesting thing to think about. [P17]
		To rethink Service H synchronous couplings	1	P17	1	I believe that another thing that can also be changed is the synchronous coupling of Service H. [P17]
		To rethink the contexts stored in the datasourse-bcfgh	3	P12, P13, and P18	4	I think these are the two points that we should attack in order to change right away. Which is to reduce the size of Service H, dividing it into more services. But doing this will theoretically increase the number of services coupling to datasourse-bcfgh... So solving Service H will move to the next level. I also need to resolve the datasourse-bcfgh. So… it would have to be one thing being done at the same time. Not only breaking Service H, but also breaking datasourse-bcfgh eventually. So, those two things, I think I could see it very clearly in the model. [P18]
		To rethink Service H contexts and responsibilities	5	P12, P13, P15, P17, and P18	7	I would improve Service H. Service H is a small microlith and the model showed this and confirmed what we had already been discussing. It has several architectural problems. For example, poorly defined responsibility and consequently have several contexts... It ends up increasing the load of the service itself to the detriment of that. So, we can improve a lot on that. Strangling the Service H. [P15]
		To rethink Service G contexts and responsibilities	1	P17	1	There is an overlap between what Service G does and what Service H does. The team doesn't know if certain actions should be in Service G or Service H. So, I think it is necessary to rethink the context of Service G to not have this overlap. [P17]
		To review couplings (increase the proportion of asynchronous interactions)	2	P12 and P18	2	I think that one thing that can be changed is related to coupling. I believe that the ideal would be to increase the amount of couplings that are asynchronous. Instead of synchronous couplings. I see there are a lot of things that are synchronous because... it's the pattern way of doing things that aren't necessarily supposed to be that way. Because I believe that this is a way for us to increase the resilience, for example, of the services, because they are coupled... Disregarding Service H, to make it simpler, I believe that reducing the couplings that are synchronous and increasing couplings that are asynchronous would be a feature that would be more interesting. Because I believe that everything related to services that are external to the search, theoretically, will have to be asynchronous. Everything external will have to be asynchronous. Within the Search System, there are a few things that I believe have to be synchronous. And, today, I think they are still... There are a lot of synchronous things that should be asynchronous. [P18]

Appendix E

Complementary Visualizations of the Characterization of the InterSCity Architecture

In addition to the rulers, the CharM is composed of a set of other artifacts that help visualize the architectural characteristics of the systems and services analyzed. This appendix presents the architectural visualization artifacts generated from the InterSCity case study. Such artifacts are organized based on the four CharM dimensions.

Size Dimension

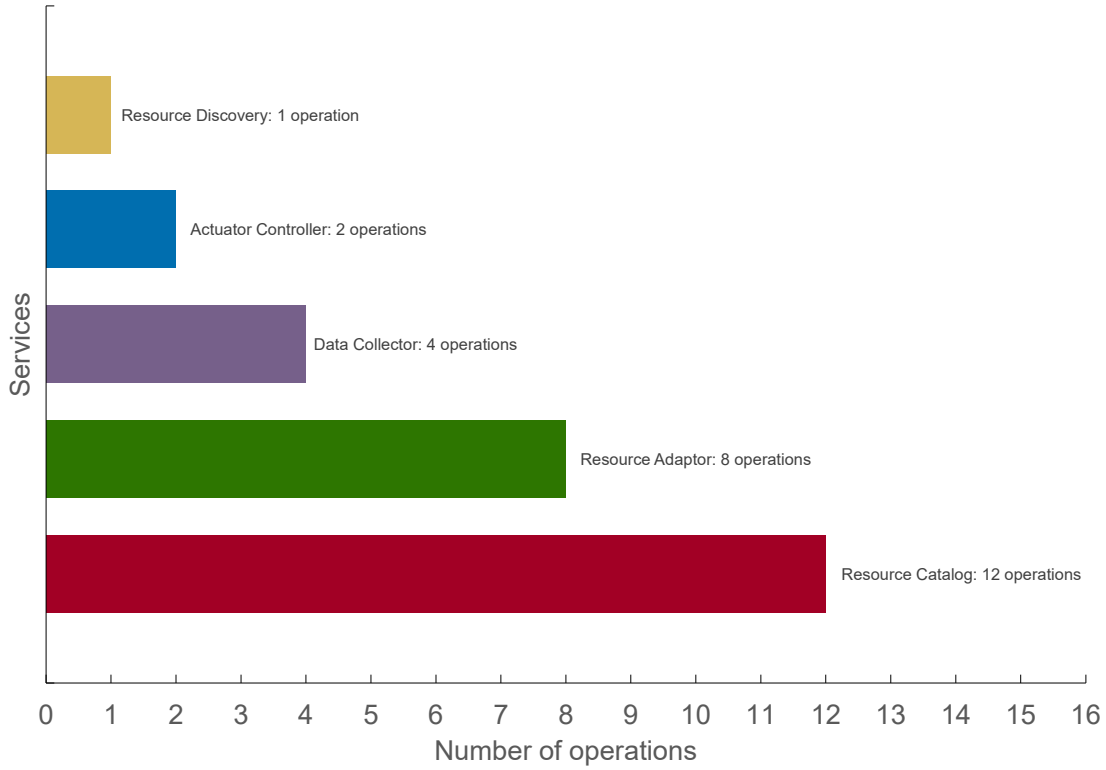


Figure E.1: *Number of operations of each InterSCity service.*

Data Source Coupling

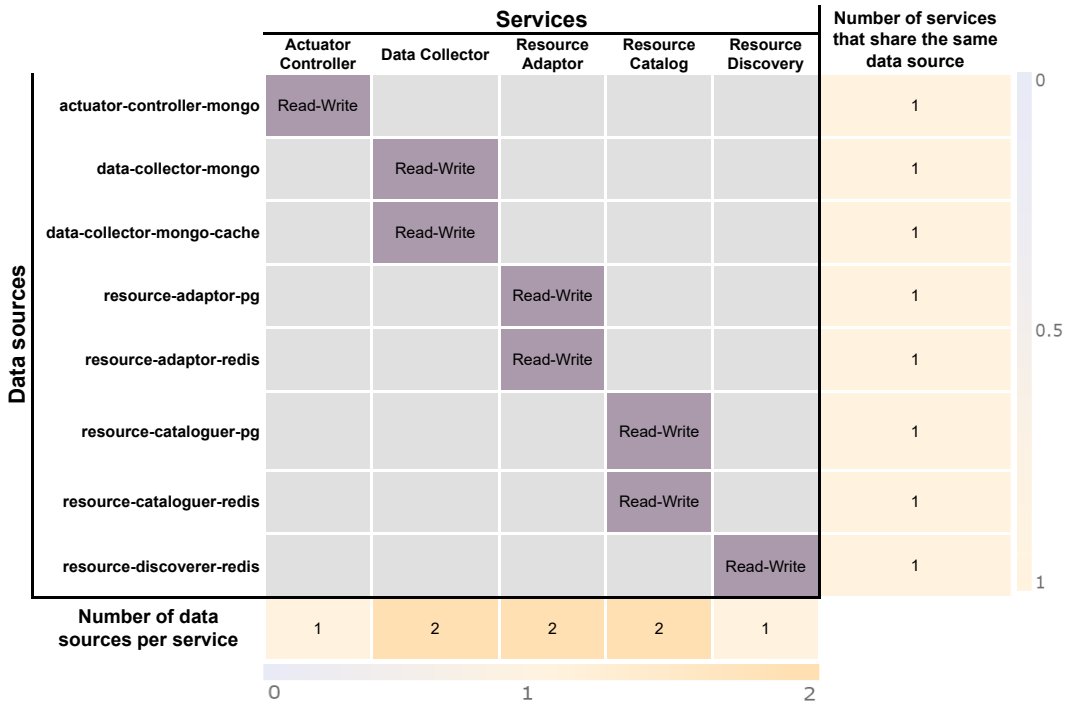


Figure E.2: *Relationship between services and data sources of the InterSCity.*

Synchronous Coupling

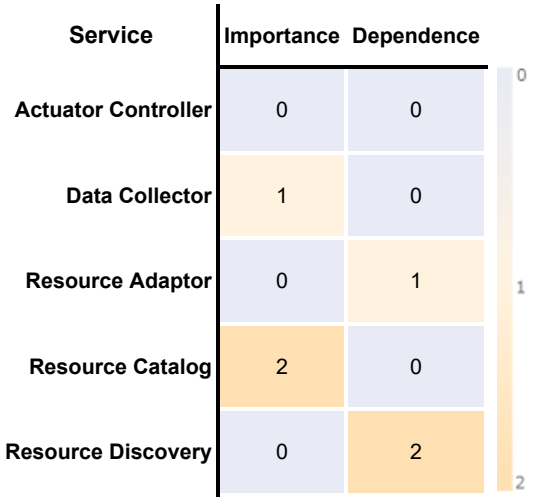


Figure E.3: *Degree of the synchronous importance and synchronous dependence of each service of the InterSCity.*

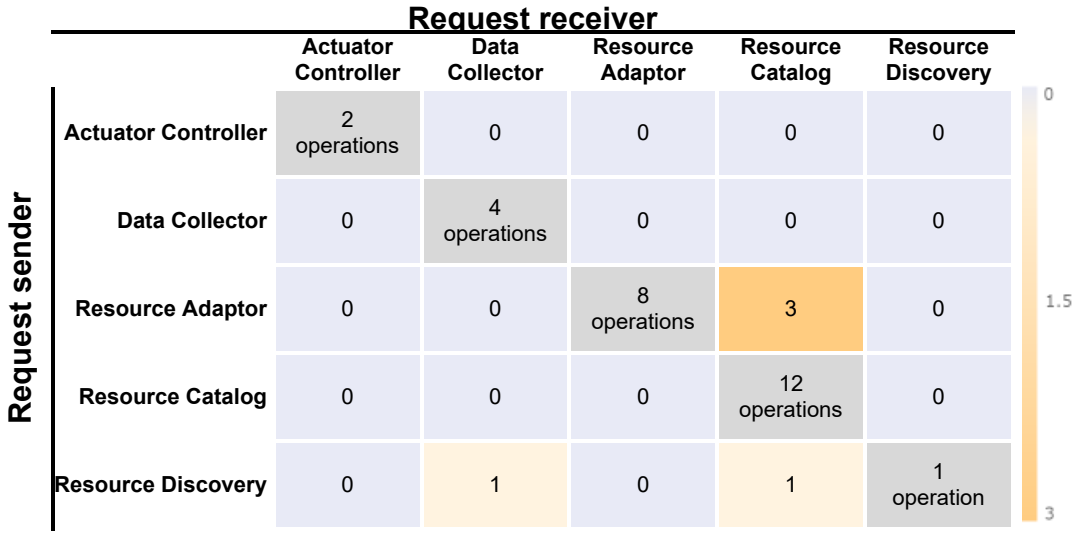


Figure E.4: *Relationship between services requesting operations and services receiving requests in the InterSCity Platform.*

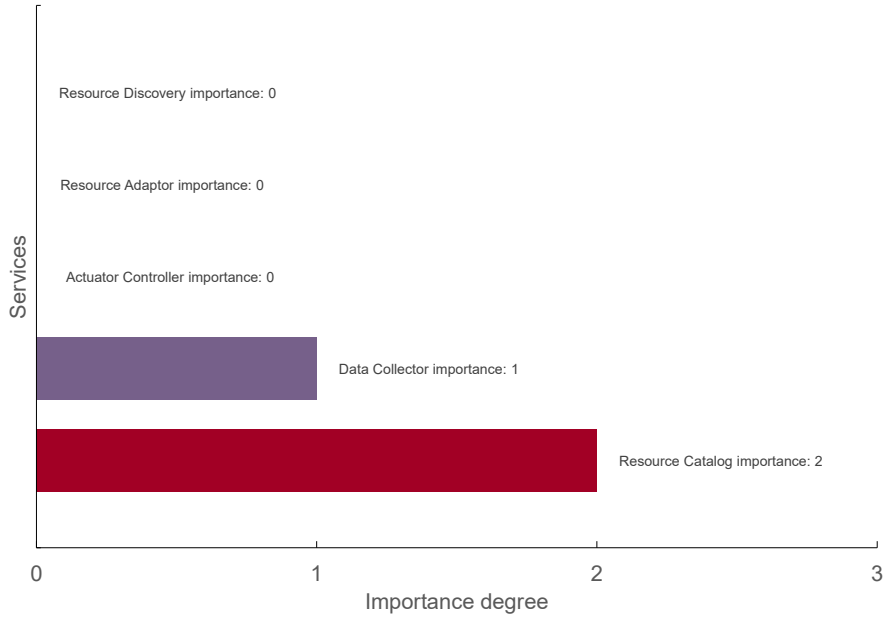


Figure E.5: *Synchronous importance degree of each InterSCity service.*

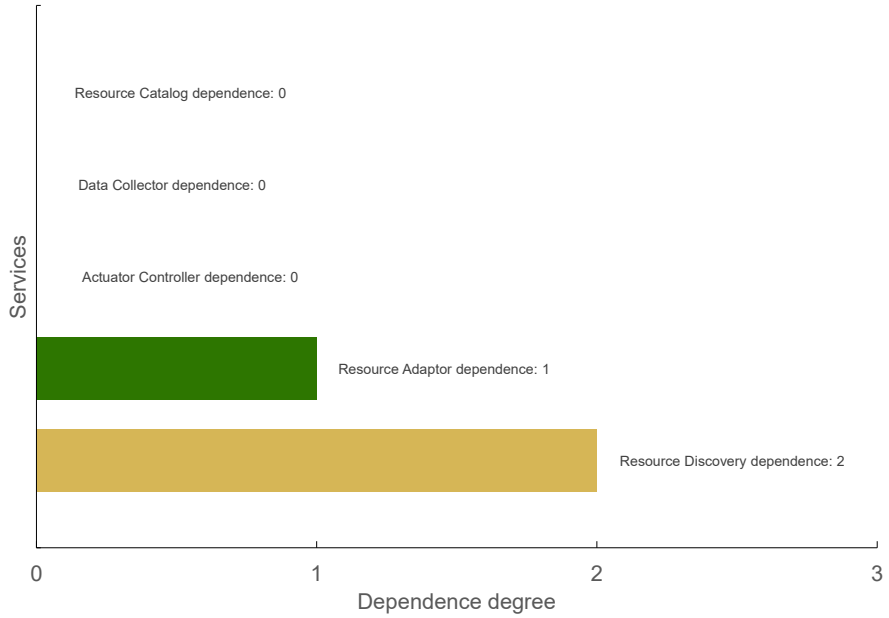


Figure E.6: *Synchronous dependence degree of each InterSCity service.*

Asynchronous Coupling

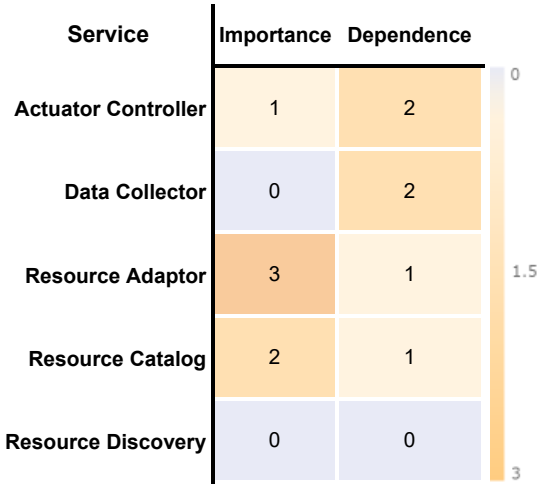


Figure E.7: *Degree of the asynchronous importance and asynchronous dependence of each service of the InterSCity.*

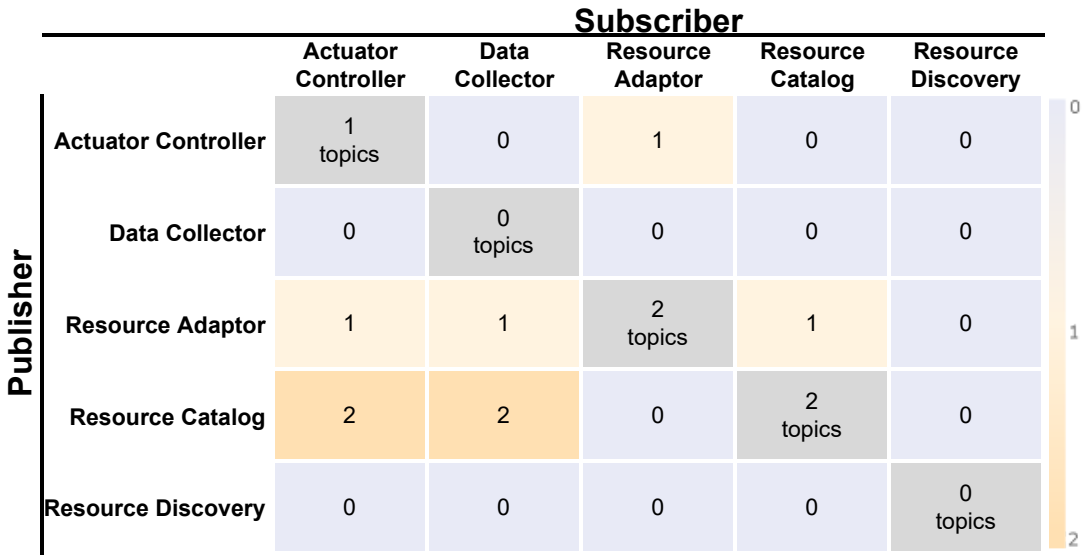


Figure E.8: *Relationship between services that publish on a message queue and services that are subscribed to receive messages from a queue in the InterSCity Platform.*

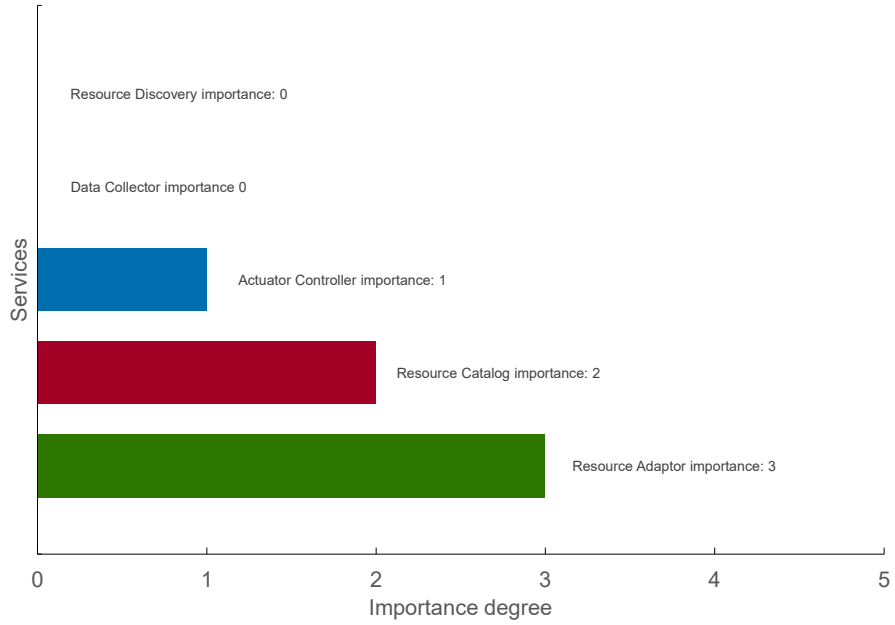


Figure E.9: *Asynchronous importance degree of each InterSCity service.*

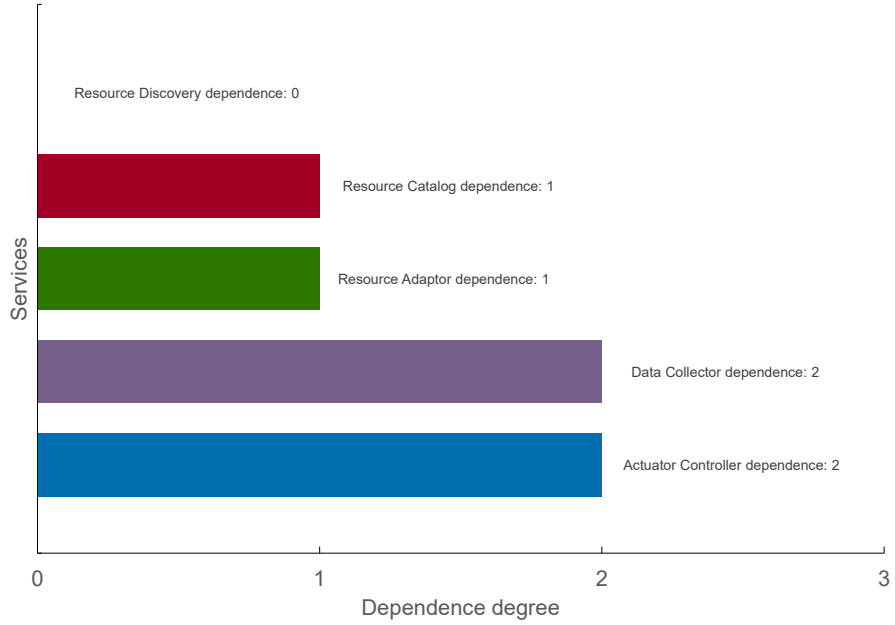


Figure E.10: *Asynchronous dependence degree of each InterSCity service.*

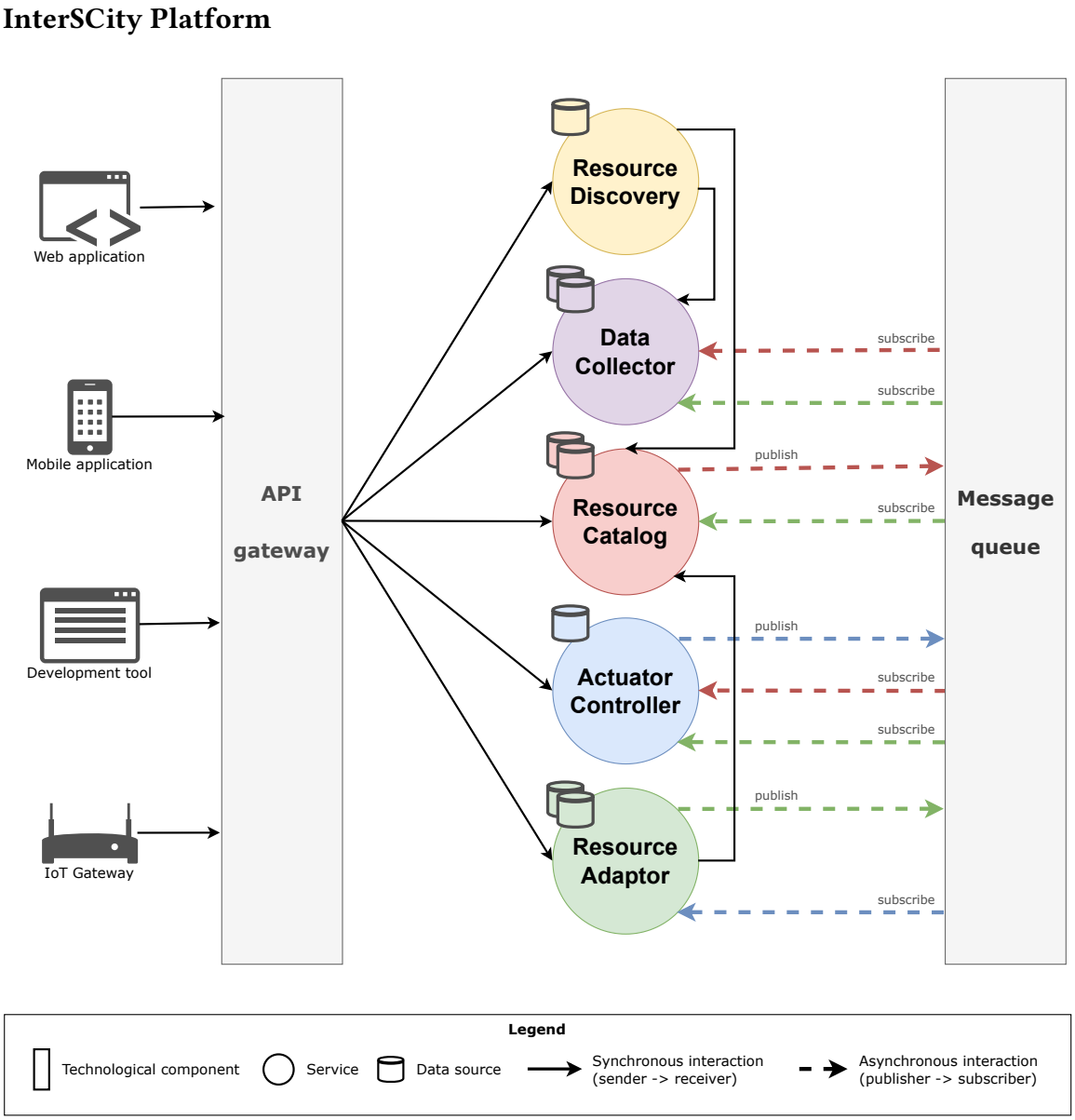


Figure E.11: *Main structural elements of the InterSCity architecture.*

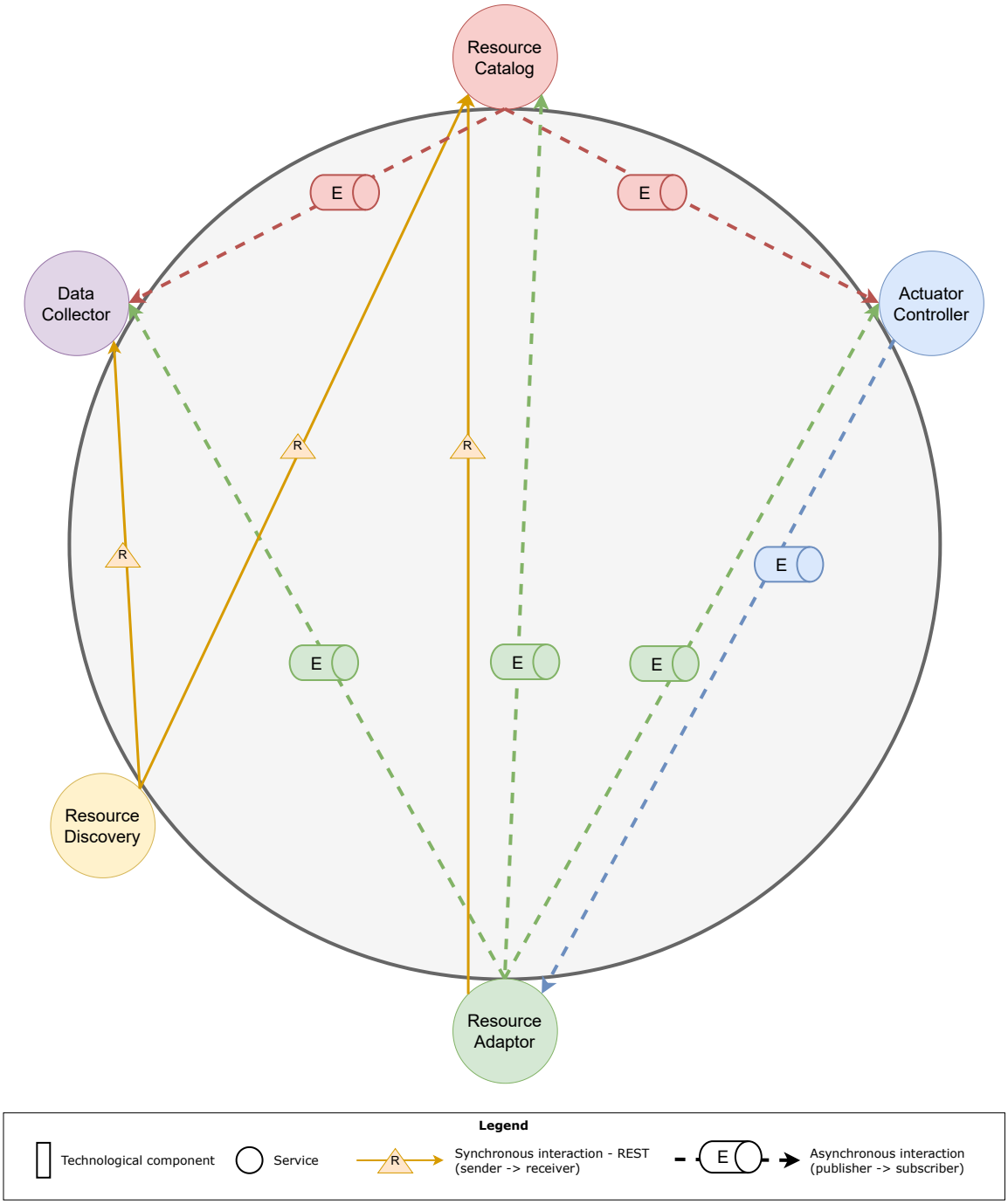


Figure E.12: *Interactions between the InterSCity services (inspired by* RICHARDSON, *2018).*

Appendix F

Complementary Visualizations of the Characterization of the Search System

In addition to the rulers, the CharM is composed of a set of other artifacts that help visualize the architectural characteristics of the systems and services analyzed. This appendix presents the architectural visualization artifacts generated from the Company A case study. Such artifacts are organized based on the four CharM dimensions.

Size Dimension

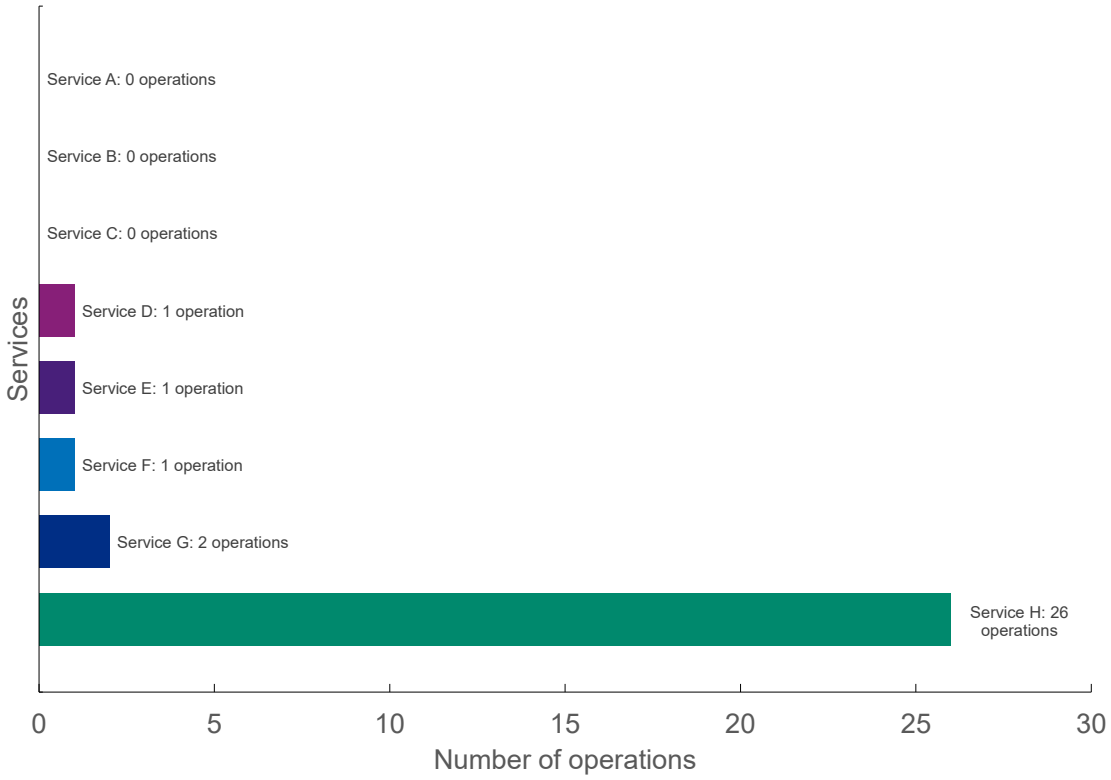


Figure F.1: *Number of operations of each Search System service.*

Data Source Coupling

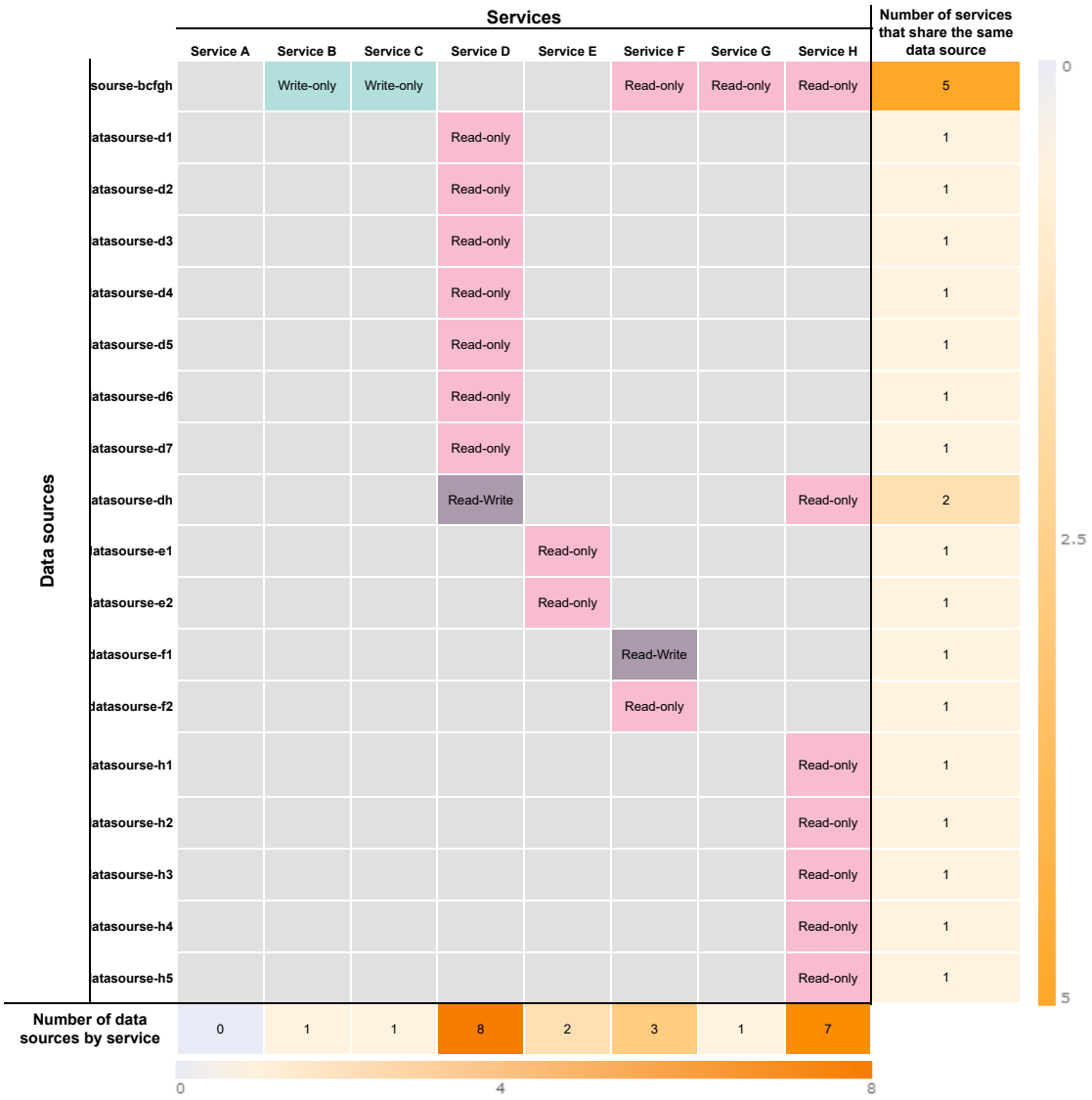


Figure F.2: *Relationship between services and data sources of the Search System.*

Synchronous Coupling

Service	Internal		External		Total	
	Importance	Dependence	Importance	Dependence	Importance	Dependence
Service A	0	0	0	0	0	0
Service B	0	0	0	0	0	0
Service C	0	0	0	0	0	0
Service D	0	0	0	0	0	0
Service E	0	0	1	0	1	0
Service F	2	0	0	0	2	0
Service G	0	1	1	0	1	1
Service H	0	1	3	1	3	2

Figure F.3: *Degree of the synchronous importance and synchronous dependence of each service of the Search System.*

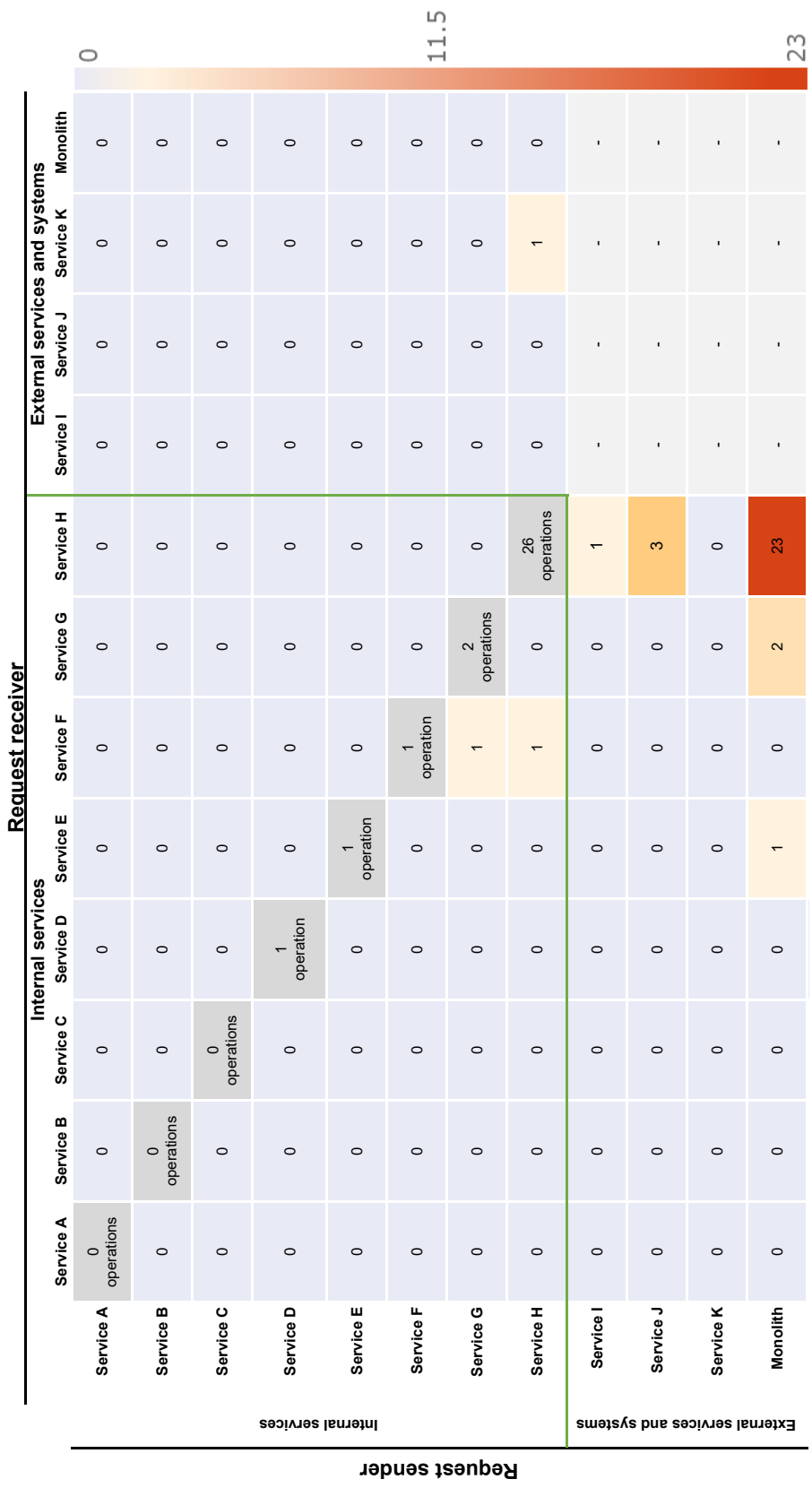


Figure F.4: *Relationship between services requesting operations and services receiving requests in the Search System.*

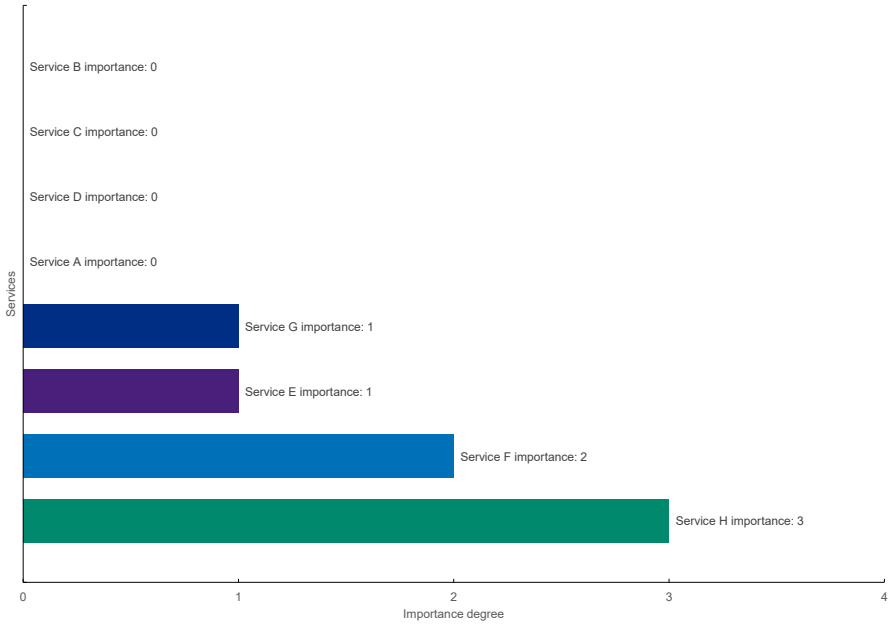


Figure F.5: *Synchronous importance degree of each Search System service.*

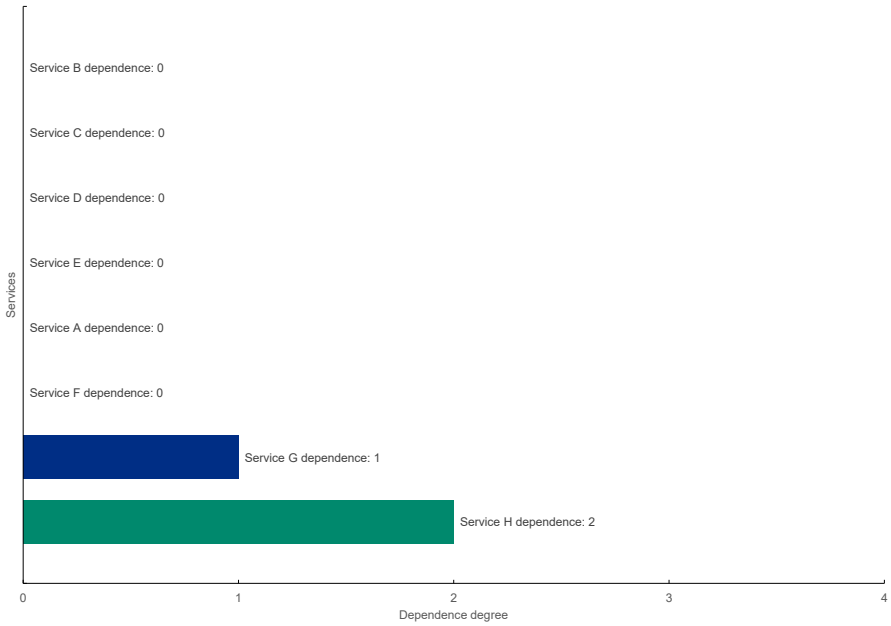


Figure F.6: *Synchronous dependence degree of each Search System service.*

Asynchronous Coupling

Service or Technological component	Internal		External		Total	
	Importance	Dependence	Importance	Dependence	Importance	Dependence
Debezium	2	0	0	0	2	0
Service A	1	1	0	0	1	1
Service B	0	1	0	0	0	1
Service C	0	1	0	0	0	1
Service D	0	1	0	0	0	1
Service E	0	0	0	0	0	0
Service F	0	0	0	0	0	0
Service G	0	0	0	0	0	0
Service H	1	0	0	0	1	0

Figure F.7: *Degree of the asynchronous importance and asynchronous dependence of each service of the Search System.*

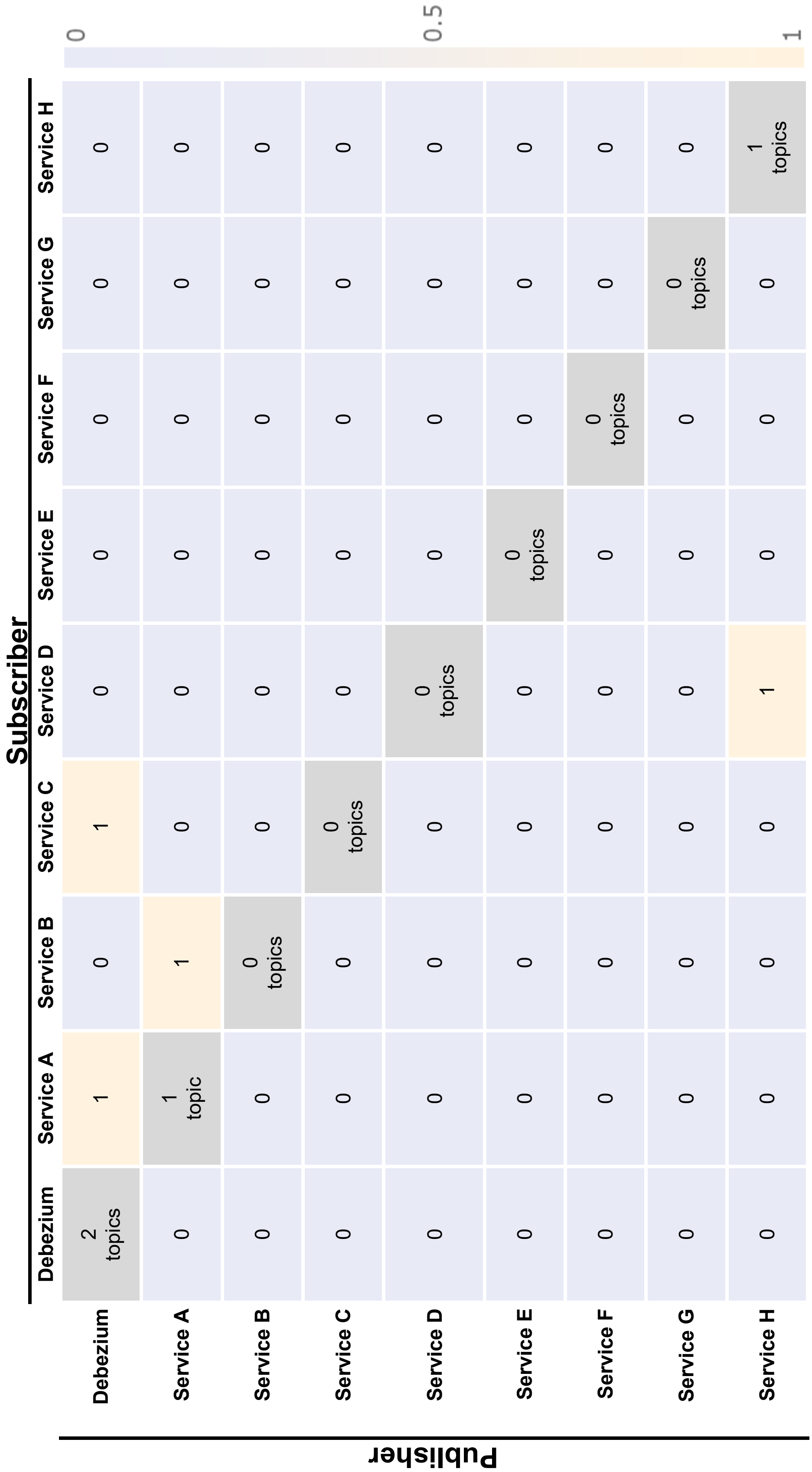


Figure F.8: *Relationship between services that publish on a message queue and services that are subscribed to receive messages from a queue in the Search System.*

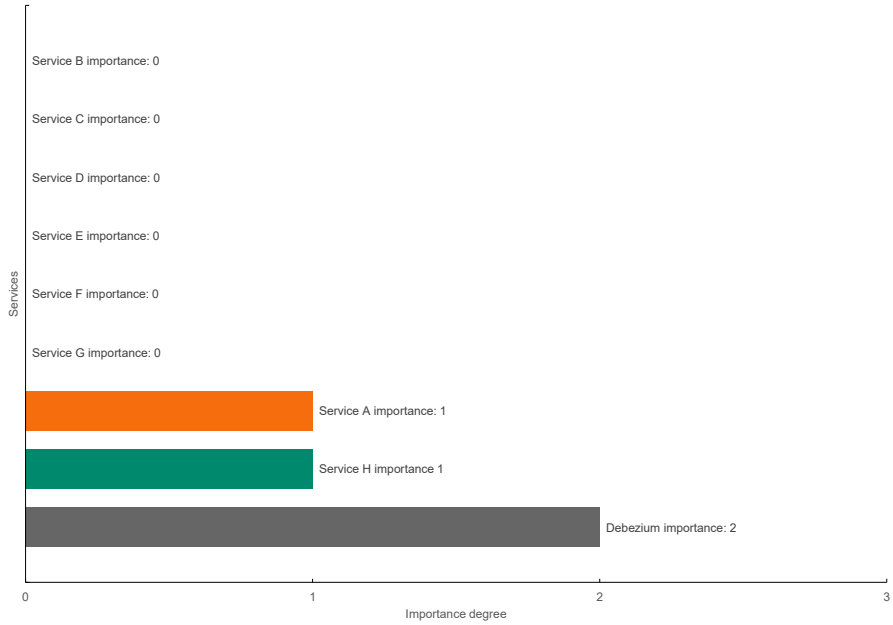


Figure F.9: *Asynchronous importance degree of each Search System service.*

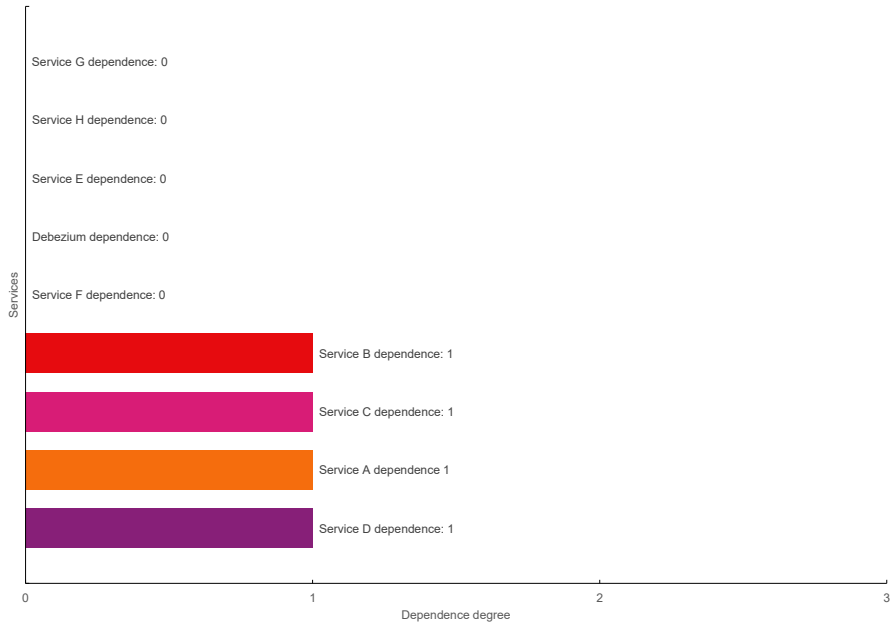


Figure F.10: *Asynchronous dependence degree of each Search System service.*

Company A's Search System

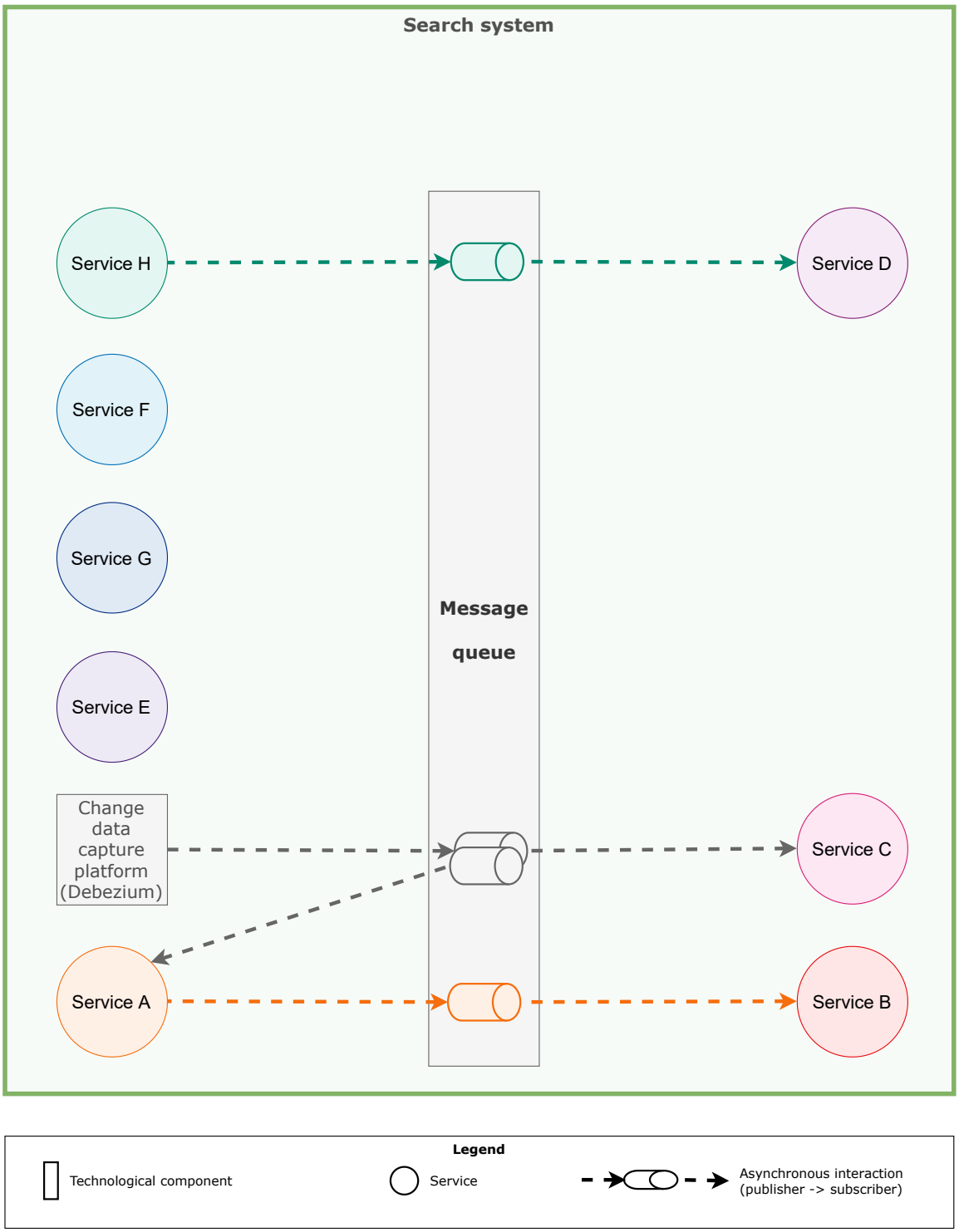


Figure F.11: *Interactions between the services and the message queue.*

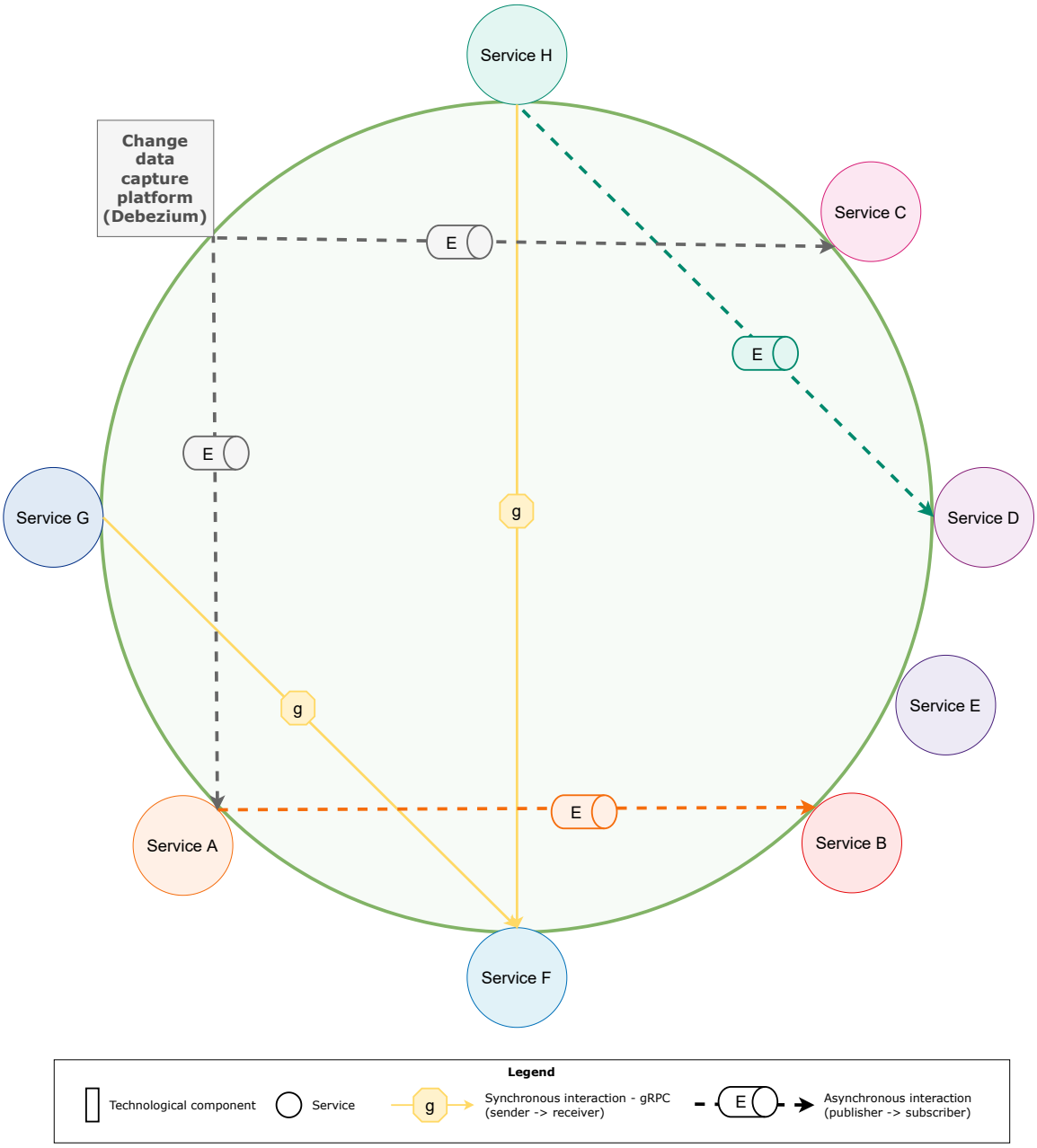


Figure F.12: *Interactions between the services of the Company A's Search System (inspired by* Richardson, 2018*).*

References

[Alshuqayran, Ali, et al. 2016] Nuha Alshuqayran, Nour Ali, and Roger Evans. "A Systematic Mapping Study in Microservice Architecture". In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. 2016, pp. 44–51. doi: 10.1109/SOCA.2016.15 (cit. on pp. 13–16).

[Alshuqayran, Ali, et al. 2018] Nuha Alshuqayran, Nour Ali, and Roger Evans. "Towards Micro Service Architecture Recovery: An Empirical Study". In: *2018 IEEE International Conference on Software Architecture (ICSA)*. 2018, pp. 47–4709. doi: 10.1109/ICSA.2018.00014 (cit. on pp. 77, 97).

[Acevedo et al. 2017] Cesar Augusto Jaramillo Acevedo, Juan Pablo Gomez y Jorge, and Ivan Rios Patino. "Methodology to transform a monolithic software into a microservice architecture". In: *2017 6th International Conference on Software Process Improvement (CIMPS)*. 2017, pp. 1–6. doi: 10.1109/CIMPS.2017.8169955 (cit. on p. 10).

[Alshuqayran 2020] Nuha Alshuqayran. "Static Microservice Architecture Recovery Using Model-driven Engineering". PhD thesis. University of Brighton, 2020 (cit. on pp. 5, 92, 93, 95, 97, 101).

[Auer et al. 2021] Florian Auer, Valentina Lenarduzzi, Michael Felderer, and Davide Taibi. "From monolithic systems to Microservices: An assessment framework". In: *Information and Software Technology* 137 (2021), p. 106600. doi: https://doi.org/10.1016/j.infsof.2021.106600 (cit. on pp. 5, 92, 94, 95, 100, 101).

[Balalaie, Heydarnoori, Jamshidi, et al. 2018] Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi, Damian A. Tamburri, and Theo Lynn. "Microservices migration patterns". In: *Software: Practice and Experience* 48.11 (2018), pp. 2019–2042. doi: 10.1002/spe.2608 (cit. on pp. 114, 116, 119, 120).

[Batista et al. 2016] Daniel Macêdo Batista et al. "InterSCity: Addressing Future Internet research challenges for Smart Cities". In: *2016 7th International Conference on the Network of the Future (NOF)*. 2016, pp. 1–6. doi: 10.1109/NOF.2016.7810114 (cit. on p. 36).

[Bass et al. 2012] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Third. Addison-Wesley Professional, 2012, p. 1169 (cit. on pp. 93, 94).

[Bushong, Das, et al. 2022] Vincent Bushong, Dipta Das, and Tomas Cerny. "Reconstructing the Holistic Architecture of Microservice Systems using Static Analysis." In: *CLOSER 2022 - Proceedings of the 12th International Conference on Cloud Computing and Services Science*. 2022, pp. 149–157. DOI: 10.5220/0011032100003200 (cit. on pp. 98, 101).

[Bernard 2011] Harvey Russell Bernard. *Research Methods in Anthropology: Qualitative and Quantitative Approaches*. Fifth. AltaMira Press, 2011, p. 680 (cit. on pp. 38, 56).

[Baresi and Garriga 2020] Luciano Baresi and Martin Garriga. "Microservices: The Evolution and Extinction of Web Services?" In: *Microservices: Science and Engineering*. Springer International Publishing, 2020, pp. 3–28. ISBN: 978-3-030-31646-4. DOI: 10.1007/978-3-030-31646-4_1 (cit. on p. 12).

[Balalaie, Heydarnoori, and Jamshidi 2016] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture". In: *IEEE Software* 33.3 (2016), pp. 42–52. DOI: 10.1109/MS.2016.64 (cit. on pp. 13–16).

[Bianco et al. 2007] Phil Bianco, Rick Kotermanski, and Paulo Merson. *Evaluating a Service-Oriented Architecture*. Tech. rep. Carnegie Mellon University, 2007 (cit. on p. 11).

[Bogner, Schlinger, et al. 2019] Justus Bogner, Steffen Schlinger, Stefan Wagner, and Alfred Zimmermann. "A Modular Approach to Calculate Service-Based Maintainability Metrics from Runtime Data of Microservices". In: *Product-Focused Software Process Improvement*. 2019, pp. 489–496. DOI: 10.1007/978-3-030-35333-9_34 (cit. on p. 98).

[Bonér 2016] Jonas Bonér. *Reactive Microservices Architecture: Design Principles for Distributed Systems*. O'Reilly, 2016, p. 54 (cit. on pp. 10, 12, 15, 16, 22).

[Baltes and Ralph 2020] Sebastian Baltes and Paul Ralph. "Sampling in Software Engineering Research: A Critical Review and Guidelines". In: *Empirical Software Engineering* 27 (2020). URL: https://arxiv.org/abs/2002.07764 (cit. on pp. 4, 38, 56, 80, 81).

[Basili and Rombach 1988] Victor R. Basili and H. Dieter Rombach. "The TAME project: towards improvement-oriented software environments". In: *IEEE Transactions on Software Engineering* 14.6 (1988), pp. 758–773. DOI: 10.1109/32.6156 (cit. on p. 98).

[Bushong, Amr S Abdelfattah, et al. 2021] Vincent Bushong, Amr S Abdelfattah, et al. "On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study". In: *Applied Sciences* 11.17 (2021). DOI: 10.3390/app11177856 (cit. on p. 21).

[Brown and Woolf 2016] Kyle Brown and Bobby Woolf. "Implementation patterns for microservices architectures". In: *Proceedings of the 23rd Conference on Pattern Languages of Programs (PLoP)*. 2016, pp. 1–35. DOI: 10.5555/3158161.3158170 (cit. on pp. 114, 116, 119, 120).

[Bogner, Wagner, et al. 2017a] Justus Bogner, Stefan Wagner, and Alfred Zimmermann. "Automatically measuring the maintainability of service- and microservice-based systems". In: *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement on - IWSM Mensura '17*. 2017, pp. 107–115. DOI: 10.1145/3143434.3143443 (cit. on pp. 1, 11, 22, 77, 101, 109, 110).

[Bogner, Wagner, et al. 2017b] Justus Bogner, Stefan Wagner, and Alfred Zimmermann. "Towards a practical maintainability quality model for service and microservice-based systems". In: *11th European Conference on Software Architecture (ECSA 2017)*. 2017, pp. 195–198. DOI: 10.1145/3129790.3129816 (cit. on pp. 77, 92, 94, 98, 109, 110).

[Bogner, Wagner, et al. 2020] Justus Bogner, Stefan Wagner, and Alfred Zimmermann. "Collecting Service-Based Maintainability Metrics from RESTful API Descriptions: Static Analysis and Threshold Derivation". In: *Software Architecture*. 2020, pp. 215–227. DOI: 10.1007/978-3-030-59155-7_16 (cit. on pp. 93, 98).

[Bogner, A. Zimmermann, et al. 2020] Justus Bogner, Alfred Zimmermann, and Stefan Wagner. "Towards an Evolvability Assurance Method for Service-Based Systems". In: *Advances in Service-Oriented and Cloud Computing*. 2020, pp. 131–139. DOI: 10.1007/978-3-030-63161-1_10 (cit. on pp. 95, 99, 101).

[Cardarelli et al. 2019] Mario Cardarelli et al. "An Extensible Data-Driven Approach for Evaluating the Quality of Microservice Architectures". In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 2019, pp. 1225–1234. DOI: 10.1145/3297280.3297400 (cit. on pp. 5, 92, 93, 95, 97).

[Cerny, Donahoo, and Pechanec 2017] Tomas Cerny, Michael J. Donahoo, and Jiri Pechanec. "Disambiguation and Comparison of SOA, Microservices and Self-Contained Systems". In: *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. 2017, pp. 228–235. DOI: 10.1145/3129676.3129682 (cit. on p. 12).

[Cerny, Donahoo, and Trnka 2018] Tomas Cerny, Michael J. Donahoo, and Michal Trnka. "Contextual Understanding of Microservice Architecture: Current and Future Directions". In: *SIGAPP Appl. Comput. Rev.* 17.4 (2018), pp. 29–45. DOI: 10.1145/3183628.3183631 (cit. on p. 12).

[Cerny, Amr S. Abdelfattah, et al. 2022] Tomas Cerny, Amr S. Abdelfattah, Vincent Bushong, Abdullah Al Maruf, and Davide Taibi. "Microvision: Static analysis-based approach to visualizing microservices in augmented reality". In:

2022 IEEE International Conference on Service-Oriented System Engineering (SOSE). 2022, pp. 49–58. DOI: 10.1109/SOSE55356.2022.00012 (cit. on pp. 5, 93, 98, 101).

[Cervantes and Kazman 2016] Humberto Cervantes and Rick Kazman. *Designing Software Architectures: A practical Approach*. Addison-Wesley, 2016 (cit. on pp. 7, 8).

[Clements, Kazman, et al. 2002] Paul Clements, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2002 (cit. on p. 7).

[Clements, Bachmann, et al. 2002] Paul Clements, Felix Bachmann, et al. *Documenting Software Architectures: Views and Beyond*. First. Addison-Wesley, 2002 (cit. on pp. 8, 9).

[Callegaro et al. 2015] Mario Callegaro, Katja Lozar Manfreda, and Vasja Vehovar. *Web survey methodology*. SAGE Publications, 2015 (cit. on pp. 4, 80).

[Corbin and A. Strauss 2015] Juliet Corbin and Anselm Strauss. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Fourth. SAGE Publications, 2015, p. 456 (cit. on pp. 5, 38, 56).

[Davis 1989] Fred D. Davis. "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology". In: *MIS Quarterly* 13.3 (1989), pp. 319–340. DOI: https://doi.org/10.2307/249008 (cit. on pp. 77, 94).

[Dreyfus et al. 1986] Hubert L. Dreyfus, Stuart E. Dreyfus, and Tom Athanasiou. *Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*. The Free Press, 1986 (cit. on pp. 70, 82).

[Dragoni et al. 2017] Nicola Dragoni et al. "Microservices: Yesterday, today, and tomorrow". In: *Present and Ulterior Software Engineering*. Springer International Publishing, 2017, pp. 195–216. DOI: 10.1007/978-3-319-67425-4_12 (cit. on pp. 12–14, 16).

[Easterbrook et al. 2008] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. "Selecting Empirical Methods for Software Engineering Research". In: *Guide to Advanced Empirical Software Engineering*. Springer London, 2008, pp. 285–311. DOI: 10.1007/978-1-84800-044-5_11 (cit. on p. 35).

[Engel et al. 2018] Thomas Engel, Melanie Langermeier, Bernhard Bauer, and Alexander Hofmann. "Evaluation of microservice architectures: A metric and tool-based approach". In: *Information Systems in the Big Data Era*. 2018, pp. 74–89. DOI: 10.1007/978-3-319-92901-9_8 (cit. on pp. 5, 77, 92, 93, 98, 101, 109, 110).

[Engström et al. 2020] Emelie Engström, Margaret-Anne Storey, Per Runeson, Martin Höst, and Maria Teresa Baldassarre. "How software engineering research

aligns with design science: a review". In: *Empirical Software Engineering* 25 (2020), pp. 2630–2660. DOI: 10.1007/s10664-020-09818-7 (cit. on p. 19).

[Esposte 2018] Arthur De Moura del Esposte. "A Scalable Microservice-based Open Source Platform for Smart Cities". MA thesis. University of São Paulo, 2018 (cit. on pp. 4, 20, 36, 37).

[Francesco et al. 2017] Paolo Di Francesco, Ivano Malavolta, and Patricia Lago. "Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption". In: *2017 IEEE International Conference on Software Architecture (ICSA)*. 2017, pp. 21–30. DOI: 10.1109/ICSA.2017.24 (cit. on p. 21).

[Ford 2018] Neal Ford. *The State of Microservices Maturity – Survey Results*. O'Reilly Media, 2018 (cit. on p. 1).

[M. Fowler 2002] Martin Fowler. *Patterns of enterprise application architecture*. First. Addison-Wesley Professional, 2002, p. 560 (cit. on p. 7).

[M. Fowler 2015] Martin Fowler. *MonolithFirst*. 2015. URL: https://martinfowler.com/bliki/MonolithFirst.html (cit. on p. 10).

[S. J. Fowler 2016] Susan J. Fowler. *Production-Ready Microservices: Building Standarized Systems Across an Engineering Organization*. O'Reilly Media, 2016 (cit. on pp. 13–16).

[Foote and Joseph Yoder 1997] Brian Foote and Joseph Yoder. "Big Ball of Mud". In: *Pattern languages of program design* 4 (1997), pp. 654–692. URL: http://www.laputan.org/mud/mud.html8/28/01http://www.laputan.org/mud/mud.html8/28/01 (cit. on p. 9).

[Garland and Anthony 2003] Jeff Garland and Richard Anthony. *Large-Scale Software Architecture: A Practical Guide Using UML*. John Wiley and Sons, Ltd, 2003, p. 260 (cit. on pp. 7, 8).

[Garlan 2014] David Garlan. "Software architecture: a travelogue". In: *Proceedings of the on Future of Software Engineering - FOSE 2014*. 2014, pp. 29–39. DOI: 10.1145/2593882.2593886 (cit. on pp. 7–9).

[Granchelli et al. 2017a] Giona Granchelli et al. "MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems". In: *IEEE International Conference on Software Architecture Workshops (ICSAW 2017)*. 2017, pp. 298–302. DOI: 10.1109/ICSAW.2017.9 (cit. on pp. 5, 77, 92–94, 97).

[Granchelli et al. 2017b] Giona Granchelli et al. "Towards recovering the software architecture of microservice-based systems". In: *IEEE International Conference on Software Architecture Workshops (ICSAW 2017)*. 2017, pp. 46–53. DOI: 10.1109/ICSAW.2017.48 (cit. on pp. 93, 97).

[Glaser and A. L. Strauss 2017] Barney G Glaser and Anselm L Strauss. *Discovery of Grounded Theory: Strategies for Qualitative Research*. Routledge, 2017 (cit. on pp. 38, 56).

[Guo et al. 2016] Dong Guo, Wei Wang, Guosun Zeng, and Zerong Wei. "Microservices Architecture Based Cloudware Deployment Platform for Service Computing". In: *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. 2016, pp. 358–363. doi: 10.1109/SOSE.2016.22 (cit. on p. 13).

[Hassan and Bahsoon 2016] Sara Hassan and Rami Bahsoon. "Microservices and their design trade-offs: A self-adaptive roadmap". In: *2016 IEEE International Conference on Services Computing (SCC 2016)*. 2016, pp. 813–818. doi: 10.1109/SCC.2016.113 (cit. on pp. 13–15, 22).

[Hirzalla et al. 2009] Mamoun Hirzalla, Jane Cleland-Huang, and Ali Arsanjani. "A Metrics Suite for Evaluating Flexibility and Complexity in Service-Oriented Architectures". In: *Service-Oriented Computing – ICSOC 2008 Workshops*. 2009, pp. 41–52. doi: 10.1007/978-3-642-01247-1_5 (cit. on pp. 109, 110).

[Hevner et al. 2004] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. "Design Science in Information Systems Research". In: *MIS Quarterly* 28.1 (2004), pp. 75–105. doi: 10.2307/25148625 (cit. on pp. 3, 19, 21).

[Highscalability 2020] Highscalability. *One Team At Uber Is Moving From Microservices To Macroservices*. 2020. url: http://highscalability.com/blog/2020/4/8/one-team-at-uber-is-moving-from-microservices-to-macroservic.html (cit. on pp. 1, 2, 78, 92).

[C. Hofmeister et al. 1999] Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Reading, MA: Addison-Wesley, 1999 (cit. on p. 7).

[H. Hofmeister and Wirtz 2008] Helge Hofmeister and Guido Wirtz. "Supporting Service-Oriented Design with Metrics". In: *2008 12th International IEEE Enterprise Distributed Object Computing Conference*. 2008, pp. 191–200. doi: 10.1109/EDOC.2008.13 (cit. on pp. 109, 110).

[Hutapea et al. 2018] Renny C. Amantha Hutapea, Adnan Puji Wahyudi, and Suhardi. "Design Quality Measurement for Service-Oriented Software on Service Computing System: a Systematic Literature Review". In: *2018 International Conference on Information Technology Systems and Innovation (ICITSI)*. 2018, pp. 375–380. doi: 10.1109/ICITSI.2018.8696092 (cit. on p. 109).

[InfoQ 2020] InfoQ. *To Microservices and Back Again*. 2020. url: www.infoq.com/news/2020/04/microservices-back-again/ (cit. on pp. 1, 2, 78, 92).

[innoQ 2015] innoQ. *Self-Contained Systems: Assembling Software From Independent Systems*. 2015. url: scs-architecture.org/index.html (cit. on pp. 1, 11).

[ISO/IEC/IEEE42010 2022] ISO/IEC/IEEE42010. *ISO/IEC/IEEE International Standard – Software, systems and enterprise -— Architecture description*. en. Standard. International Organization for Standardization, 2022 (cit. on p. 8).

[Jaramillo et al. 2016] David Jaramillo, Duy V Nguyen, and Robert Smart. "Leveraging microservices architecture by using Docker technology". In: *SoutheastCon 2016*. 2016, pp. 1–5. doi: 10.1109/SECON.2016.7506647 (cit. on pp. 12–16).

[Kalske 2017] Miika Kalske. "Transforming monolithic architecture towards microservice architecture". MA thesis. University of Helsinki, 2017 (cit. on p. 10).

[Krafzig et al. 2005] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Service-oriented Architecture Best Practices*. Pearson Education, 2005 (cit. on p. 11).

[Killalea 2016] Tom Killalea. "The Hidden Dividends of Microservices: Microservices Aren't for Every Company, and the Journey Isn't Easy." In: *Queue* 14.3 (2016), pp. 25–34. doi: 10.1145/2956641.2956643 (cit. on pp. 13, 14, 16).

[Krylovskiy et al. 2015] Alexandr Krylovskiy, Marco Jahn, and Edoardo Patti. "Designing a Smart City Internet of Things Platform with Microservice Architecture". In: *2015 3rd International Conference on Future Internet of Things and Cloud*. 2015, pp. 25–30. doi: 10.1109/FiCloud.2015.55 (cit. on pp. 13–16).

[Kolny 2023] Marcin Kolny. *Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%*. 2023. url: www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90 (cit. on pp. 1, 78, 92).

[Le et al. 2015] Vinh D. Le et al. "Microservice-based architecture for the NRDC". In: *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. 2015, pp. 1659–1664. doi: 10.1109/INDIN.2015.7281983 (cit. on pp. 13, 14).

[Lederer et al. 2000] Albert L. Lederer, Donna J. Maupin, Mark P. Sena, and Youlong Zhuang. "The technology acceptance model and the World Wide Web". In: *Decision Support Systems* 29.3 (2000), pp. 269–282. doi: https://doi.org/10.1016/S0167-9236(00)00076-2 (cit. on p. 94).

[Lewis and M. Fowler 2014] James Lewis and Martin Fowler. *Microservices*. 2014. url: https://martinfowler.com/articles/microservices.html (cit. on pp. 1, 10–16, 22).

[Mahmood 2007] Zaigham Mahmood. "The promise and limitations of service-oriented architecture". In: *International Journal of Computers* 1.3 (2007), pp. 74–78 (cit. on pp. 1, 22).

[Martin 2018] Robert C. Martin. *Clean Architecture: A Craftsmans Guide to Software Structure and Design*. Prentice Hall, 2018, p. 409 (cit. on pp. 8, 23).

[Mendonça et al. 2021] Nabor C. Mendonça, Craig Box, Costin Manolache, and Louis Ryan. "The Monolith Strikes Back: Why Istio Migrated From Microservices to a Monolithic Architecture". In: *IEEE Software* 38.5 (2021), pp. 17–22. doi: 10.1109/MS.2021.3080335 (cit. on pp. 1, 2, 78, 92).

[Mayer and Weinreich 2018] Benjamin Mayer and Rainer Weinreich. "An Approach to Extract the Architecture of Microservice-Based Software Systems". In: *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. 2018, pp. 21–30. doi: 10.1109/SOSE.2018.00012 (cit. on pp. 5, 92–95, 99, 101).

[Merson and Jospeh Yoder 2019] Paulo Merson and Jospeh Yoder. *SATURN 2019 Talk: Modeling Microservices with DDD*. Software Engineering Institute - Carnegie Mellon University. 2019. url: https://youtu.be/FOiiWgv81S0 (cit. on pp. 28, 45, 69).

[Nadareishvili et al. 2016] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, and Mike Amundsen. *Microservice Architecture: Aligning principles, practices, and culture*. O'Reilly Media, 2016 (cit. on pp. 2, 12–14, 16, 92).

[Newman 2015] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. First. O'Reilly Media, 2015, p. 259 (cit. on pp. 11–16).

[Newman 2019] Sam Newman. *Monolith to Microservices: Evolutionary Patterns to Transform your Monolith*. First. O'Reilly Media, 2019, p. 272 (cit. on pp. 9, 10).

[Newman 2021] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. Second. O'Reilly Media, 2021, p. 616 (cit. on pp. 1, 2, 10, 12, 22, 92, 93).

[Nitto and Rosenblum 1999] Elisabetta Di Nitto and David Rosenblum. "Exploiting ADLs to specify architectural styles induced by middleware infrastructures". In: *Proceedings of the 21st international conference on Software engineering - ICSE '99*. 1999, pp. 13–22. doi: 10.1145/302405.302406 (cit. on p. 9).

[Natis and Schulte 2003] Yefim Natis and Roy Schulte. *Introduction to Service-Oriented Architecture*. Tech. rep. Gartner Group, 2003 (cit. on pp. 1, 11).

[Ntentos, Zdun, Plakidas, Meixner, et al. 2020] Evangelos Ntentos, Uwe Zdun, Konstantinos Plakidas, Sebastian Meixner, and Sebastian Geiger. "Metrics for Assessing Architecture Conformance to Microservice Architecture Patterns and Practices". In: *Service-Oriented Computing*. 2020, pp. 580–596. doi: 10.1007/978-3-030-65310-1_42 (cit. on pp. 109, 110).

[Ntentos, Zdun, Plakidas, and Geiger 2021a] Evangelos Ntentos, Uwe Zdun, Konstantinos Plakidas, and Sebastian Geiger. "Evaluating and Improving Microservice Architecture Conformance to Architectural Design Decisions". In: *Service-Oriented Computing*. 2021, pp. 188–203. doi: 10.1007/978-3-030-91431-8_12 (cit. on pp. 93, 99, 101).

[Ntentos, Zdun, Plakidas, and Geiger 2021b] Evangelos Ntentos, Uwe Zdun, Konstantinos Plakidas, and Sebastian Geiger. "Semi-automatic Feedback for Improving Architecture Conformance to Microservice Patterns and Practices". In: *2021 IEEE 18th International Conference on Software Architecture (ICSA)*. 2021, pp. 36–46. doi: 10.1109/ICSA51549.2021.00012 (cit. on pp. 94, 99, 101).

[Oliveira Rosa, Goldman, et al. 2020] Thatiane de Oliveira Rosa, Alfredo Goldman, and Eduardo Martins Guerra. "How 'micro' are your services?" In: *IEEE International Conference on Software Architecture Companion (ICSA-C 2020)*. 2020, pp. 75–78 (cit. on pp. 19, 21).

[Oliveira Rosa, Daniel, et al. 2020] Thatiane de Oliveira Rosa, João Francisco Lino Daniel, Eduardo Martins Guerra, and Alfredo Goldman. "A Method for Architectural Trade-off Analysis Based on Patterns: Evaluating Microservices Structural Attributes". In: *Proceedings of the European Conference on Pattern Languages of Programs 2020*. 2020, p. 8. doi: 10.1145/3424771.3424809 (cit. on pp. 6, 20–22).

[Papazoglou 2003] Mike P. Papazoglou. "Service-oriented computing: concepts, characteristics and directions". In: *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.* 2003, pp. 3–12. doi: 10.1109/WISE.2003.1254461 (cit. on p. 11).

[Patton 2014] Michael Quinn Patton. *Qualitative research and evaluation methods: Integrating theory and practice*. Fourth. SAGE Publications, 2014, p. 832 (cit. on pp. 38, 56).

[Parnas et al. 1985] Lorge David Parnas, Paul C. Clements, and David M. Weiss. "The Modular Structure of Complex Systems". In: *IEEE Transactions on Software Engineering* SE-11.3 (1985), pp. 259–266. doi: 10.1109/TSE.1985.232209 (cit. on p. 11).

[Peng et al. 2022] Xin Peng et al. "Trace Analysis Based Microservice Architecture Measurement". In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2022, pp. 1589–1599. doi: 10.1145/3540250.3558951 (cit. on pp. 5, 99).

[Perepletchikov et al. 2007] Mikhail Perepletchikov, Caspar Ryan, Keith Frampton, and Zahir Tari. "Coupling metrics for predicting maintainability in service-oriented designs". In: *Australian Software Engineering Conference (ASWEC'07)*. 2007, pp. 329–338 (cit. on pp. 22, 77, 109, 110).

[Pahl and Jamshidi 2016] Claus Pahl and Pooyan Jamshidi. "Microservices: A systematic mapping study". In: *Proceedings of the 6th International Conference on Cloud Computing and Services Science*. Vol. 1. 2016, pp. 137–146. doi: 10.5220/0005785501370146 (cit. on pp. 13, 14).

[Pressman 2001] Roger S. Pressman. *Software Engineering : A Practitioner's Approach*. Fifth. McGraw-Hill, 2001 (cit. on p. 8).

[C. U. Press 2019] Cambridge University Press. *Meaning of monolithic in English*. 2019. URL: https://dictionary.cambridge.org/us/dictionary/english/monolithic (cit. on p. 9).

[C. U. Press 2022] Cambridge University Press. *Meaning of characterization in English*. 2022. URL: https://dictionary.cambridge.org/dictionary/english/characterization (cit. on p. 21).

[O. U. Press 2022] Oxford University Press. *Definition of model noun from the Oxford Advanced Learner's Dictionar*. 2022. URL: https://www.oxfordlearnersdictionaries.com/definition/english/model_1?q=model (cit. on p. 21).

[Perry and Wolf 1992] Dewayne E Perry and Alexander L Wolf. "Foundations for the study of software architecture". In: *ACM SIGSOFT Software Engineering Notes* 17.4 (Oct. 1992), pp. 40–52 (cit. on p. 7).

[Ren et al. 2018] Zhongshan Ren et al. "Migrating web applications from monolithic structure to microservices architecture". In: *Proceedings of the 10th Asia-Pacific Symposium on Internetware*. 2018. DOI: 10.1145/3275219.3275230 (cit. on p. 10).

[Richards and Ford 2020] Mark Richards and Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media, 2020 (cit. on p. 8).

[Rosa et al. 2020] Thatiane Rosa, Alfredo Goldman, and Eduardo Guerra. "Modelo para Caracterização e Evolução de Sistemas com Arquitetura Baseada em Serviços". In: *Anais Estendidos do XI Congresso Brasileiro de Software: Teoria e Prática*. 2020, pp. 38–46. DOI: 10.5753/cbsoft_estendido.2020.14607 (cit. on pp. 20, 21, 109).

[Richards 2015] Mark Richards. *Software Architecture Patterns*. O'Reilly Media, 2015 (cit. on pp. 1, 9, 11, 13–16).

[Richards 2016] Mark Richards. *Microservices vs. Service-Oriented Architecture*. O'Reilly Media, 2016, p. 44 (cit. on p. 11).

[Richardson 2018] Chris Richardson. *Microservices Patterns*. Manning Publicatins Co., 2018, p. 489 (cit. on pp. 1, 9, 10, 12–16, 23, 25, 26, 147, 159).

[Richardson 2020] Chris Richardson. *Microservice Architecture - A pattern language for microservices*. 2020. URL: microservices.io/patterns/index.html (cit. on pp. 109, 110, 114–121).

[Richardson and Smith 2016] Chris Richardson and Floyd Smith. *Microservices From Design to Deployment*. NGINX, 2016, p. 74 (cit. on pp. 10, 12–16).

[Raj and Sadam 2021] Vinay Raj and Ravichandra Sadam. "Performance and Complexity Comparison of Service-Oriented Architecture and Microservices Architecture". In: *International Journal of Communication Networks and Distributed Systems* 27.1 (2021), pp. 100–117. DOI: 10.1504/ijcnds.2021.116463 (cit. on p. 12).

[Rud et al. 2006] Dmytro Rud, Andreas Schmietendorf, and Reiner R. Dumke. "Product Metrics for Service-Oriented Infrastructures". In: *International Workshop on Software Measurement - IWSM/MetriKon*. 2006 (cit. on pp. 109, 110).

[Rademacher et al. 2017] Florian Rademacher, Sabine Sachweh, and Albert Zundorf. "Differences between Model-Driven Development of Service-Oriented and Microservice Architecture". In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. 2017, pp. 38–45. doi: 10.1109/ICSAW.2017.32 (cit. on p. 12).

[Runeson et al. 2012] Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012, p. 237 (cit. on p. 4).

[Salah et al. 2016] Tasneem Salah, M. Jamal Zemerly, Yeob Yeun Chan, Mahmoud Al-Qutayri, and Yousof Al-Hammadi. "The evolution of distributed systems towards microservices architecture". In: *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*. 2016, pp. 318–325. doi: 10.1109/ICITST.2016.7856721 (cit. on p. 12).

[E. F. Z. Santana et al. 2017] Eduardo Felipe Zambom Santana, Ana Paula Chaves, Marco Aurelio Gerosa, Fabio Kon, and Dejan S. Milojicic. "Software Platforms for Smart Cities: Concepts, Requirements, Challenges, and a Unified Reference Architecture". In: *ACM Comput. Surv.* 50.6 (2017). doi: 10.1145/3124391 (cit. on p. 37).

[E. Santana et al. 2021] Erick Santana, Thatiane Rosa, João Daniel, and Alfredo Goldman. "Desenvolvendo o Sorting Hat: uma Ferramenta para Caracterização de Arquitetura Baseada em Serviços". In: *Anais Estendidos do XII Congresso Brasileiro de Software: Teoria e Prática*. 2021, pp. 127–136. doi: 10.5753/cbsoft_estendido.2021.17298 (cit. on pp. 95, 107).

[Shaw and Garlan 1996] Mary Shaw and David Garlan. *Software architecture: perspective on an emerging discipline*. Prentice Hall, 1996, p. 242 (cit. on pp. 8, 9).

[Shim et al. 2008] Bingu Shim, Siho Choue, Suntae Kim, and Sooyong Park. "A Design Quality Model for Service-Oriented Architecture". In: *15th Asia-Pacific Software Engineering Conference*. 2008, pp. 403–410. doi: 10.1109/APSEC.2008.32 (cit. on pp. 24, 109, 110).

[Singleton 2016] Andy Singleton. "The Economics of Microservices". In: *IEEE Cloud Computing* 3.5 (2016), pp. 16–20. doi: 10.1109/MCC.2016.109 (cit. on pp. 13–16).

[Sneed 2006] Harry M. Sneed. "Integrating legacy software into a service-oriented architecture". In: *Conference on Software Maintenance and Reengineering (CSMR'06)*. 2006, 11 pp.–14. doi: 10.1109/CSMR.2006.28 (cit. on p. 22).

[Sun et al. 2016] Yuqiong Sun, Susanta Nanda, and Trent Jaeger. "Security-as-a-Service for Microservices-Based Cloud Applications". In: *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. 2016, pp. 50–57. doi: 10.1109/CloudCom.2015.93 (cit. on pp. 15, 16).

[Sommerville 2011] Ian Sommerville. *Engenharia de Software*. 9th. Pearson Prentice Hall, 2011 (cit. on p. 9).

[Stol et al. 2016] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. "Grounded Theory in Software Engineering Research: A Critical Review and Guidelines". In: *Proceedings of the 38th International Conference on Software Engineering*. 2016, pp. 120–131. doi: 10.1145/2884781.2884833 (cit. on pp. 5, 38, 56).

[Shadija et al. 2017] Dharmendra Shadija, Mo Rezai, and Richard Hill. "Towards an Understanding of Microservices". In: *2017 23rd International Conference on Automation and Computing (ICAC)*. 2017, pp. 1–6. doi: 10.23919/IConAC.2017.8082018 (cit. on p. 12).

[Savchenko et al. 2015] D. I. Savchenko, G. I. Radchenko, and O. Taipale. "Microservices validation: Mjolnirr platform case study". In: *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2015, pp. 235–240. doi: 10.1109/MIPRO.2015.7160271 (cit. on pp. 15, 16).

[Statista 2022] Statista. *Software developer gender distribution worldwide as of 2021*. 2022. url: https://www.statista.com/statistics/1126823/worldwide-developer-gender/ (cit. on p. 85).

[Soldani et al. 2018] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. "The pains and gains of microservices: A systematic grey literature review". In: *Journal of Systems and Software* 146 (2018), pp. 215–232. doi: https://doi.org/10.1016/j.jss.2018.09.082 (cit. on pp. 1, 13–16, 22).

[Thönes 2015] Johannes Thönes. "Microservices". In: *IEEE Software* 32.1 (2015), pp. 116–116. doi: 10.1109/MS.2015.11 (cit. on p. 12).

[Taibi et al. 2018] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. "Architectural patterns for microservices: A systematic mapping study". In: *CLOSER 2018 - Proceedings of the 8th International Conference on Cloud Computing and Services Science*. 2018, pp. 221–232. doi: 10.5220/0006798302210232 (cit. on pp. 13, 14, 16).

[Taylor et al. 2009] Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing, 2009 (cit. on p. 9).

[Vera-Rivera et al. 2021] Fredy H Vera-Rivera, Carlos Gaona, and Hernán Astudillo. "Defining and measuring microservice granularity—a literature overview". In: *PeerJ Computer Science* 7 (2021). doi: 10.7717/peerj-cs.695 (cit. on p. 21).

[Villamizar et al. 2015] Mario Villamizar et al. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud". In: *2015 10th Computing Colombian Conference (10CCC)*. 2015, pp. 583–590. DOI: 10.1109/ColumbianCC.2015.7333476 (cit. on pp. 10, 13, 14, 16).

[Vural et al. 2017] Hulya Vural, Murat Koyuncu, and Sinem Guney. "A Systematic Literature Review on Microservices". In: *Computational Science and Its Applications – ICCSA 2017*. 2017, pp. 203–217. DOI: 10.1007/978-3-319-62407-5_14 (cit. on pp. 13, 14).

[Wagner et al. 2020] Stefan Wagner, Daniel Mendez, Michael Felderer, Daniel Graziotin, and Marcos Kalinowski. "Challenges in Survey Research". In: *Contemporary Empirical Methods in Software Engineering*. Springer International Publishing, 2020, pp. 93–125. DOI: 10.1007/978-3-030-32489-6_4 (cit. on p. 81).

[Wasson 2017] Mike Wasson. *Design patterns for microservices*. 2017. URL: azure.microsoft.com/pt-br/blog/design-patterns-for-microservices/ (cit. on pp. 114, 120).

[Wolff 2016a] Eberhard Wolff. *Self-contained Systems: A Different Approach to Microservices*. 2016. URL: https://www.innoq.com/en/articles/2016/11/self-contained-systems-different-microservices/#self-containedsystems (cit. on p. 12).

[Wolff 2016b] Eberhard Wolff. *Services: SOA, Microservices und Self-contained Systems*. 2016. URL: https://www.innoq.com/en/articles/2016/11/services-soa-microservices-scs/#fazit (cit. on p. 12).

[Zhou et al. 2023] Xin Zhou et al. "Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry". In: *Journal of Systems and Software* 195 (2023), p. 111521. DOI: https://doi.org/10.1016/j.jss.2022.111521 (cit. on p. 2).

[O. Zimmermann 2017] Olaf Zimmermann. "Microservices tenets: Agile approach to service development and deployment". In: *Computer Science-Research and Development* 32.3 (2017), pp. 301–310. DOI: 10.1007/s00450-016-0337-0 (cit. on p. 21).

[Zdun et al. 2017] Uwe Zdun, Elena Navarro, and Frank Leymann. "Ensuring and Assessing Architecture Conformance to Microservice Decomposition Patterns". In: *Service-Oriented Computing*. 2017, pp. 411–429. DOI: 10.1007/978-3-319-69035-3_29 (cit. on pp. 5, 92, 99, 101).

[Zhang and Xinke 2009] Qingqing Zhang and Li Xinke. "Complexity Metrics for Service-Oriented Systems". In: *2009 Second International Symposium on Knowledge Acquisition and Modeling*. Vol. 3. 2009, pp. 375–378. DOI: 10.1109/KAM.2009.90 (cit. on pp. 109, 110).