Guiding Service-Oriented Software Engineering
- A View-based Approach

Qing Gu

2011
SIKS Dissertation Series No. 2011-20

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2008-2012 under grant agreement 215483 (S-Cube).

Promotiecommissie:
Prof.Dr. H. Akkermans (VU University Amsterdam)
Prof.Dr. W.J.A.M. van den Heuvel (Tilburg University)
Dr. E. Di Nitto (Politecnico di Milano)
Dr. I. Richardson (Lero Research Centre and University of Limerick)
Prof.Dr. R.J. Wieringa (University of Twente)

NUR 982

Cover designed by Nikki Vermeulen
Printed by Ridderprint BV, Ridderkerk, the Netherlands
Copyright ©2011 by Qing Gu
All rights reserved
VRIJE UNIVERSITEIT

Guiding Service-Oriented Software Engineering - A View-based Approach

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op donderdag 6 oktober 2011 om 13.45 uur
in de aula van de universiteit,
De Boelelaan 1105

doort
Qing Gu
geboren te Shanghai, China
promotor: prof.dr. J.C. van Vliet
copromotor: dr. P. Lago
Acknowledgments

It is a pleasure to thank those who made this thesis possible. My heartfelt thanks go to my direct supervisor Patricia Lago for your continuous encouragement, guidance, and support over the past four years. You always impress me with your immense knowledge as well as creative ideas in solving the research questions. I do not believe I could have received better guidance to complete the Ph.D. work. Special thanks to my promotor, Prof. Hans van Vliet for being always supportive with your impressive scientific experience.

I also want to express my gratitude to the reading committee, Hans Akkermans, Willem-Jan van den Heuvel, Elisabetta Di Nitto, Ita Richardson, and Roel J.Wieringa, for carefully reading my manuscript and putting forward various interesting comments.

I was happy to join the S-Cube project which provided me the chance to collaborate with many researchers from different disciplines. I really enjoyed the work with Elisabetta Di Nitto, Michael Parkin, Félix Cuadrado, Juan C. Dueñas, Stephen Lane, and Ita Richardson. Many steps towards this thesis were accompanied by you.

I thank my colleagues and fellow Ph.D. students for the support and good company I received at the Software Engineering group. Special thanks to Elly Lammer for helping me dealing with the university bureaucracy. I thank my ex-colleagues Rik Farenhorst and Remco de Boer for helping me out, reading drafts of papers, and taking time to discuss my even uncooked ideas, especially in the beginning of my Ph.D. work. Rahul Premraj, Antony Tang, Natalia Silvis Cividjian, Victor Clerc, Adam Vanya, Maryam Razavian, and Christina Manteli, it has been very pleasant working with you and I enjoyed our inspiring discussions, lunch time, and yearly multiculti party.

I want to thank Jan Baas from Hogeschool Inholland for helping me out as a foreigner when I just came to the Netherlands, for offering me a teaching position which not only brought me much working experience but also solved my financial problems during my student’s life, and for encouraging me to continue my academic life with Masters and Doctoral studies.

I would like to express my sincere appreciation to my friends (Taoying, Li, Haiyan, Annemieke, and Rocco), my family, and relatives for supporting and encouraging me during all those years. In particular, I would like to thank my mother, Xiaoli Xie, and my father, Zangzhi Gu, for your everlasting support in all choices I made in my life. Lots of thanks go to my parents-in-law, Berts and Kok Bonnet, for always caring about me and always being there for me. Most of all, I want to thank my husband Nico Bonnet, my son Jason, and soon-to-be-born baby for all the happiness you brought me.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the Ph.D. work.
# Contents

## Acknowledgments v

1 Introduction 1

1.1 Service-oriented software engineering .......................... 1
1.2 The S-Cube project ............................................. 2
1.3 Research questions and research design ........................ 2
1.4 Outline of this thesis .......................................... 4
1.5 Research methods ............................................. 9

I SOSE challenges 13

2 A literature review of SOSE challenges 15

2.1 Introduction .................................................. 15
2.2 Research method ............................................. 18
  2.2.1 Systematic review ....................................... 18
  2.2.2 Research questions ...................................... 18
  2.2.3 Review protocol ......................................... 21
2.3 Overview of the included studies ............................. 27
2.4 Results of the systematic review .............................. 29
  2.4.1 RQ1 - Claimed SOSE challenges ........................ 29
  2.4.2 RQ2.1 - Topics of SOSE challenges .................... 31
  2.4.3 RQ2.2 - Types of SOSE challenges ...................... 37
  2.4.4 RQ2.3 - On other ways to classify SOSE challenges ... 43
  2.4.5 The SOSE challenges classified along two dimensions ... 44
2.5 Discussion .................................................... 44
  2.5.1 Threats to validity ...................................... 44
  2.5.2 Quality assessment ...................................... 47
2.6 Conclusions .................................................. 47

3 On the differences between SOSE and TSE 49

3.1 Introduction .................................................. 49
3.2 Motivation and Research Approach ............................ 50
3.3 The Framework ............................................... 52
3.4 The Need For Service-oriented Viewpoints .................... 55
3.5 Conclusions .................................................. 56
CONTENTS

4 A stakeholder-driven Service Life Cycle Model for SOA 59
  4.1 Introduction ............................................. 59
  4.2 Related work ........................................... 60
  4.3 The Service aspects ..................................... 63
    4.3.1 The relevance of cross-organizational collaboration ...... 64
    4.3.2 Increased importance of the identification
        of stakeholders .......................................... 65
  4.4 The proposed life cycle model ............................... 67
    4.4.1 Service provider ....................................... 70
    4.4.2 Service broker ......................................... 73
    4.4.3 Application builder (service consumer) ................. 74
  4.5 Conclusions ............................................. 76

5 A taxonomy of SOSE stakeholder types 79
  5.1 Introduction ............................................. 79
  5.2 The S-Cube service engineering lifecycle ..................... 80
  5.3 Research method ........................................ 82
  5.4 A taxonomy of service engineering stakeholders ............. 84
  5.5 Coverage of the S-Cube service engineering lifecycle ....... 86
    5.5.1 Coverage of lifecycle phases by the stakeholder types ... 87
    5.5.2 The participation of each type of stakeholders .......... 88
    5.5.3 Comparison between the evolution and adaptation cycles .. 88
  5.6 Related work ........................................... 89
  5.7 Conclusions ............................................. 91

6 Guiding the SOSE Process: the Importance of Service Aspects 93
  6.1 Introduction ............................................. 93
  6.2 Applying service aspects to a concrete methodology ........ 94
    6.2.1 The SeCSE methodology ................................ 95
    6.2.2 The SOSE process model for the SeCSE methodology ...... 95
  6.3 Related work ........................................... 102
  6.4 Conclusions ............................................. 103

7 Guiding the selection of service-oriented software engineering
   methodologies 105
  7.1 Introduction ............................................. 105
  7.2 A service aspects-driven evaluation framework ............... 107
    7.2.1 Generic evaluation criteria ........................... 107
    7.2.2 Service-specific criteria ............................. 108
  7.3 Comparison of existing SOSE methodologies using the evaluation
      framework ................................................. 114
# CONTENTS

7.3.1 Overview of the selected SOSE methodologies 114  
7.3.2 Comparing the generic aspects of the SOSE methodologies 117  
7.3.3 Comparing the service-specific aspects of the SOSE methodologies 125  
7.4 Observations 134  
7.5 Related work 136  
7.6 Conclusions 138  

8 Guiding the selection of service identification methods 141  
8.1 Introduction and research questions 141  
8.2 Review protocol 142  
8.3 The results of the review 144  
8.3.1 RQ 1 What are the existing SIMs? 144  
8.3.2 RQ 2.a what are the different types of inputs? 145  
8.3.3 RQ 2.b what are the different types of services being produced? 147  
8.3.4 R2.c what types of strategies and techniques? 150  
8.4 An input-output matrix for the selection of SIMs 153  
8.5 Conclusions 156  

9 A template for SOA design decision making in an educational setting 157  
9.1 Introduction 157  
9.2 Background 159  
9.3 A template for documenting quality-driven SOA design decisions 160  
9.4 Results on the use of the template 162  
9.4.1 The evaluation of the software designs 164  
9.4.2 Students feedback on the use of the template 167  
9.5 Related work 169  
9.6 Conclusions 170  

10 SOA decision making - what do we need to know 173  
10.1 Introduction 173  
10.2 The service aspects 175  
10.2.1 Temporary provision-consumption relationship 175  
10.2.2 Different architecture types 176  
10.2.3 Dealing with heterogeneity 179  
10.2.4 Different perspectives of stakeholders 179  
10.2.5 A summary of identified information 180  
10.3 Two examples of SOA design issues 180  
10.3.1 Example 1 180  
10.3.2 Example 2 182
## CONTENTS

10.3.3 Foreseen benefits ........................................ 184
10.4 SOA design decisions in the literature ........................ 186
10.5 Related work .................................................. 187
10.6 Conclusions .................................................... 187

   11.1 Introduction ............................................... 189
   11.2 Related work ............................................. 191
   11.3 Motivation ................................................ 192
   11.4 The service aspects and examples of SOA Process Decisions . 194
      11.4.1 Shift of ownership of software ....................... 194
      11.4.2 The relevance of cross-organizational cooperation .... 195
   11.5 Mapping SOA process decisions
      on the core model ..................................... 197
      11.5.1 Gaps in the existing architectural
             knowledge models ................................ 197
      11.5.2 A suggestion to extend the core model ........... 200
   11.6 Conclusions ............................................. 200

12 3D Architecture Viewpoints on Service Automation 203
   12.1 Introduction ............................................. 203
   12.2 An introduction to automation ............................ 205
      12.2.1 Automation and human interaction ................. 205
      12.2.2 Automation in SBAs ............................. 206
   12.3 The need for service automation viewpoints ......... 207
      12.3.1 A study on automation of service management and deployment ........................................ 207
      12.3.2 Documenting software architecture ............ 209
   12.4 Research approach .................................... 209
   12.5 3D service automation viewpoints ...................... 211
      12.5.1 An overview of 3D service automation viewpoints . 211
      12.5.2 Documenting architecture viewpoints .......... 212
      12.5.3 A fragment of the SDCA service flow - running scenario 212
      12.5.4 Viewpoint ‘Decision on service automation’
             (Decision Viewpoint) ............................... 214
      12.5.5 Viewpoint ‘Degree of service automation’ (Degree Viewpoint) 220
      12.5.6 Viewpoint ‘Data relating to service automation’
             (Data Viewpoint) ................................ 223
      12.5.7 Summary ............................................. 228
   12.6 Discussion ................................................ 228
12.6.1 The visual support of the 3D service automation viewpoints 228
12.6.2 The applicability of the 3D VP’s 231
12.7 Conclusions 231

13 Conclusions 233
13.1 Contributions 235
13.1.1 What are the challenges in SOSE? (RQ 1) 235
13.1.2 How is the development process of SBAs different from that of traditional software systems? (RQ 2) 235
13.1.3 How is service-oriented architectural knowledge different from traditional architectural knowledge? (RQ 3) 237
13.2 Summary 238
13.3 Future research 239

Appendices 243
A The judgment scale to assess the methodology support for a feature 245
B The service lifecycle model 247
Samenvatting 249
SIKS Dissertation Series 253
References 271
1 Introduction

1.1 Service-oriented software engineering

New generation business models impose additional requirements on the IT support of business processes (Rehfeldt & Turowski, 2000). In particular, quickly responding to new business requirements, continuously reducing IT cost, and dynamically integrating new business partners and customers are highly demanded (Chen et al., 2003).

Recently, the trend in software development has shifted from developing software systems to developing service-oriented systems that are composed of ready-to-use services. The Service-Oriented Architecture (SOA) (Erl, 2005) architectural style has been widely adopted in industry thanks to its ability of providing seamless integration among software services. If the services are well-specified, loosely-coupled, and coherent, implementing a SOA can bring many benefits to an enterprise, including: increased ROI, reduced IT costs, and increased organizational agility (Erl, 2005).

SOA implementations (i.e. service-based applications or SBAs) are constructed by integrating heterogeneous services that are developed using various programming languages and running on different operating systems from a range of service providers. Services are loosely coupled entities, often designed under open-world assumptions, distributed across organizational boundaries, and executed remotely at their service providers’ environment. They require a smoother transition from development to operation than traditional applications. An SBA is different from, for example, a component-based application since the owner of the component-based application also owns and controls its components, whereas the owner of the SBA generally does not own or control all of its component services.

Since developing SBAs requires identifying, discovering, and composing services in addition to traditional software engineering (TSE) activities (such as
CHAPTER 1. INTRODUCTION

coding, testing, and deployment), existing software development methodologies no longer fulfill the needs for developing SBAs. Consequently, principles known from TSE need to be tailored to service-oriented development. Systematic, disciplined, and quantifiable approaches for designing, developing, and maintaining SBAs are needed. These approaches are generalized as Service-oriented Software Engineering (SOSE).

In the context of the European network of excellence S-Cube, providing SOSE methodologies to support the development, deployment, and evolution of SBAs is the main goal of S-Cube. This thesis is one of its results.

1.2 The S-Cube project

The S-Cube Network of Excellence (NoE) (Metzger & Pohl, 2009; Papazoglou & Pohl, 2009) is a four-year European Commission (EC) project to establish an integrated, multidisciplinary, and vibrant research community in the field of SOSE and to advance the knowledge in these areas to the wider community. The aim of S-Cube’s work is to define an integrated research framework (IRF) that has the goal of creating a coherent and holistic framework to integrate the SOSE principles, techniques, and methods developed in the S-Cube NoE.

As of Nov. 2010, the network includes 16 academic partners, an industrial advisory board with 6 industrial partners and 11 associate members from both academia and industry. Amongst other outreach activities, S-Cube organizes the annual EC Service and Software Architectures, Infrastructures and Engineering (SSAIE) Summer School. The network has approximately 200 individual researchers, and many are considered experts in their field.

1.3 Research questions and research design

SOSE - as compared to TSE - is a relatively new and immature discipline. The SOSE community has agreed that traditional software engineering principles, techniques, and methods cannot be blindly applied to SOSE (Papazoglou, 2003; Erl, 2005; Di Nitto et al., 2008). Rather, engineering SBAs requires a different mindset than TSE. Design decisions are often postponed to runtime with the aim to accommodate ever-changing requirements. Many responsibilities (e.g., maintenance) of service consumers are shifted to service providers since services are physically located at the service provider’s side; in contrast, service consumers acquire additional responsibilities such as service discovery, composition and monitoring. The relationship between business and IT becomes closer since SOA is often business-driven and deals with enterprise-level implementation.
1.3. RESEARCH QUESTIONS AND RESEARCH DESIGN

Much existing SOSE work focuses on the design and development of methods, tools, and mechanisms to support various SOSE activities. However, very little work has been done on guiding the shift of the mindset. For instance, many SOSE methodologies focus on the service life cycle from a service provider’s perspective but overlook their support for service consumption life cycle, e.g., how the services can be discovered by unknown consumers, how the services can fulfill the different quality expectations from different consumers, how to notify their consumers when consumed services are updated. Although many SOSE methodologies, process models, and architectural designs have been proposed in both academia and industry aiming at supporting the engineering of SBAs, one common question that SOSE researchers and practitioners face is to what extent they are service-specific, meaning that they are different from the traditional ones and specifically designed for SOSE. Without knowing the answer, one could design a SOSE methodology overlooking the required mindset changes and one could apply the methodology resulting in an SBA that cannot easily adapt to ever-changing requirements, which would cause the failure of SOA projects.

This thesis is about supporting service engineers and architects in managing the service development process and service-oriented architectural knowledge. The support provided is specific to SOSE in the sense that it focuses on issues that are less relevant in TSE but are of great importance in SOSE. To provide such support, it is important to first understand the state-of-the-art in SOSE and what makes SOSE different from TSE. Knowing these differences, their impact on SOSE can be identified as a set of service aspects that are specific to SOSE and require specific attention to. If these service aspects can be incorporated into a set of ‘views’, they can be particularly helpful for service engineers and architects showing what should be done differently in SOSE. As a result, the first research question is:

- **RQ 1 What are the challenges in SOSE?**

The differences between SOSE and TSE have an impact on the development process of SBAs. The software development process (Sommerville, 2004) is the process of carrying out the engineering activities, including design, development, and evolution. Process models (describing what activities a development process consists of and how they should be performed) have been often used since they visualize the development process proposed by the methodology (Cugola & Ghezzi, 1998). The SOSE community has realized that traditional software process modeling techniques are no longer directly applicable or adaptable in SOSE (Blake, 2007). The question that many service engineers face is what the service aspects are and how to model the SOSE development process so that the service aspects are highlighted. The development process has to be carried out by development roles or SOSE stakeholders. Due to the shift away from mono-
CHAPTER 1. INTRODUCTION

lithic application development to service provision and composition, many more stakeholders are involved in SOSE. It is interesting to understand what types of stakeholders are needed in SOSE and what their responsibilities are. Moreover, since many SOSE methods exist, guidelines are need to support the selection among alternative methods. Corresponding to the problem statements outlined above, the following research questions are established:

- **RQ 2** How is the development process of SBAs different from that of traditional software systems?
  - **RQ 2.1** What are the service aspects and how to make them explicit in SOSE process models?
  - **RQ 2.2** Which stakeholders may be involved in the development process of SBAs?
  - **RQ 2.3** How to select a SOSE method?

The differences between SOSE and TSE also have an impact on the software architecture. The software architecture is the result of the design activity and plays an important role, managing the complex interactions and dependences between system units. The attention of the software architecture community in recent years shifted towards architectural knowledge (Muhammad et al., 2009; Avgeriou et al., 2007) and the notion of software architecture changed to embrace design decisions (Jansen & Bosch, 2005). Accordingly, the methods for documenting software architecture are shifting from describing components and connectors to selecting and recording a set of design decisions that are made through the architecting process (Fallessi et al., 2006; Ali Babar & Lago, 2009; Van Der Ven et al., 2006). The question that many SOA architects face is which service aspects influence the SOA decision making process and decision models. Corresponding to the problem statements outlined above, the following research questions are established:

- **RQ 3** How is service-oriented architectural knowledge different from traditional architectural knowledge?
  - **RQ 3.1** Are existing decision models sufficient to document SOA design decisions?
  - **RQ 3.2** How to use architectural views to assist the documentation of service-oriented architectures?

1.4 Outline of this thesis

The results of this thesis are presented in three parts. Part I of the thesis presents a systematic literature review to explore the challenges claimed in the SOSE com-
1.4. OUTLINE OF THIS THESIS

Community. Using the same set of data, a list of differences between SOSE and TSE has been identified. These differences lead to a theoretical framework that serves as the basis for the other two parts of the thesis. Part II focuses on the process perspective and provides a list of guidelines for modeling the service-oriented development process, the classification of SOSE stakeholders, the comparison of SOSE methodologies, and the selection of service identification methods. Part III focuses on architectural knowledge and provides a list of guidelines for making and documenting SOA design decisions. In addition, to illustrate how, why, and the extent to which services can execute without human intervention (service automation), a set of service automation viewpoints is proposed.

An outline of this thesis is given in Figure 1.1, illustrating the chapters and their relation to the research questions.

Figure 1.1: An outline of this thesis: research questions related to chapters

Part I consists of the following chapters:

- Chapter 2: addresses research question RQ 1; it presents the results of a systematic literature review: an overview of the SOSE challenges, classified along two dimensions: a) based on themes (or topics) that they cover and b) based on characteristics (or types) that they reveal. It has been published previously as:

CHAPTER 1. INTRODUCTION

*My personal contribution:* The systematic literature review was conducted by the first author, where the analysis was carried out together with Patricia Lago.

- Chapter 3: presents the seven fundamental differences between SOSE and TSE that have been identified from the data collected in the systematic literature review presented in Chapter 2. These differences are the causes of the SOSE challenges and have implications on the development process and architectural knowledge management. The results of this chapter do not directly contribute to specific research questions but they serve as the basis for the rest of the chapters. It has been published previously as:

  *My personal contribution:* The elicitation of the differences from the previous literature review was conducted by the first author. The paper was written by the first author with support from Patricia Lago.

Part II consists of the following chapters:

- Chapter 4: addresses research question RQ 2.1; it discusses three service aspects and proposes a service life cycle model that illustrates these service aspects; it provides guidelines of how to model the service-oriented development process. It has been published previously as:

  *My personal contribution:* The study was conducted by the first author. The paper was written by the first author with support from Patricia Lago.

- Chapter 5: addresses research question RQ 2.2; it presents nineteen stakeholder types and five roles solicited from nine partners within the S-Cube project and it demonstrates the phases each role is involved in when building and operating SBAs; it provides guidelines of which stakeholders are expected or required in a service-oriented development process. It has been submitted as:

  *My personal contribution:* The first author carried out the study design, execution, and data collection. The analysis was mainly performed by the
1.4. OUTLINE OF THIS THESIS

first author with the support of Patricia Lago. This article was written in collaboration with Michael Parkin.

• Chapter 6: further addresses research questions RQ 2.1 and 2.2; it presents a case study in which the development process of the SeCSE\(^1\) methodology is modeled. The results of this chapter provide feedback to the service life cycle model proposed in Chapter 4 and the stakeholder types identified in Chapter 5. It serves as an example of how to use the guidelines proposed in Chapter 4 to model the development process used in a real-life project and which stakeholders are actually involved. It has been published previously as:


*My personal contribution:* The case study was mainly designed and conducted by the first author, where Elisabetta Di Nitto acted as interviewee and reviewed the results of the case study. The article has been written in collaboration.

Chapter 7 and 8 both address research question RQ 2.3, but with different scope. More specifically:

• Chapter 7: focuses on the selection of the methodologies for engineering SBAs (from design, development to evolution); it presents a framework for comparing the existing SOSE methodologies aiming at differentiating the methodologies that are truly service-oriented from those that deal marginally with service aspects. It provides guidelines on what a ‘good’ SOSE methodology entails. It has been published previously as:


*My personal contribution:* The evaluation framework was designed by the first author, where the guidelines for using the framework were developed together with Patricia Lago. The paper was written by the first author with support from Patricia Lago.

• Chapter 8: focuses on the selection of the methods for a specific SOSE activity - service identification; it presents the comparison of existing service identification methods. It provides guidelines on how to select a service identification method that not only fits the needs but also is feasible in practice. It has been published previously as:

\(^{1}\)www.secse-project.eu
CHAPTER 1. INTRODUCTION


*My personal contribution:* The study was conducted by the first author. The paper was written by the first author with support from Patricia Lago.

Part II consists of the following chapters:
Chapter 9, 10, and 11 all address research question RQ 3.1, but from different perspectives. More specifically:

- Chapter 9: discusses the use of a traditional architectural design decision model in documenting SOA design decisions; it proposes a template-driven approach for documenting quality-driven architectural design decisions in an educational setting and reports on the experience with the usage of the template in a service-oriented software design Master course given over the past three years. The results of this chapter provide guidelines on the use of a lightweight design decision model to document SOA design decisions.
  It has been published previously as:
  *My personal contribution:* The study was conducted by the first author. The paper was written by the first author with support from Patricia Lago and reviewed by Hans van Vliet.

- Chapter 10: focuses specifically on SOA design decisions and discusses four services aspects, from which it identified a set of information items that aids SOA decision making and a list of elements that should be explicitly documented in SOA design decisions. By pointing out information that should not be overlooked, this chapter provides guidelines for SOA architects on decision making process. It has been published previously as:
  *My personal contribution:* The study was conducted by the first author. The paper was written by the first author and reviewed by Hans van Vliet.

- Chapter 11: focuses on a specific type of SOA design decision - process decisions (in the SOA domain) - that considers the development process as a product. The results of this chapter show that process decisions that are less relevant in TSE are of great importance in SOSE. As such, it reminds
1.5. RESEARCH METHODS

SOA architects to consciously make process decisions as part of their design decisions. It has been published previously as:


*My personal contribution:* The study was conducted by the first author. The paper was written by the first author with support from Patricia Lago.

- Chapter 12: addresses research question RQ 3.2; it presents a set of automation viewpoints to address the concerns related to service automation and it demonstrates the use of the automation viewpoints by means of three case studies. The results of this chapter provide guidelines on documenting the service automation aspect of SBAs using architectural views. The automation views have been published previously as:


The description of the automation viewpoints has been submitted as:


*My personal contribution:* The study was designed by the first author and conducted in collaboration with Félix Cuadrado. The results of the study were achieved with support from Patricia Lago. The article was written in collaboration, and Juan C. Dueñas reviewed the work.

1.5 Research methods

In this thesis, a number of qualitative research methods that are common in the field of software engineering research have been used. These methods are briefly introduced as follows.

- **Systematic literature review.** Also called systematic review, this method is an evidence-based approach to thoroughly search studies relevant to some pre-defined research questions and critically select, appraise, and synthesize findings to answer such research questions (Kitchenham, 2007). It is particularly powerful in collecting and analyzing existing work, which is a common task in establishing background knowledge in any research. This
method has been used in exploring SOSE challenges (Chapter 2) and identifying differences between SOSE and TSE that are the causes of the SOSE challenges (Chapter 3). This method was also used to gain a holistic view of the existing service identification methods (Chapter 8).

- Literature study. Different from a systematic literature review that follows a pre-defined, structured review protocol, a literature study is much less formal in the sense that it allows more freedom in collecting relevant studies and analyzing their content. Although the results of a literature study might not be as complete and valid as those of a systematic literature review, thanks to its effectiveness and efficiency, this method is often used to gain certain knowledge or understanding on a specific topic. In this thesis, this method has been used to study some existing service life cycle models (in Chapter 4), SOA design decision making process and methods (in Chapter 10), and process decisions in the domain of SOA (in Chapter 11), in order to get familiar with these topics and carry out some analysis.

- Survey. This method aims at collecting information that is relevant to answer pre-defined research questions (Pfleeger & Kitchenham, 2001). In order to understand what types of stakeholders are involved in SOSE as well as their roles and responsibilities, a survey was carried out to make use of the collective knowledge and experience of the EC’s S-Cube NoE (Chapter 5).

- Experiment. An experiment is an empirical research method conducted under a controlled setting (Wohlin et al., 2000). Its aim is to investigate “a testable hypothesis where one or more independent variables are manipulated to measure their effect on one or more dependent variables.” (Easterbrook et al., 2008). In Chapter 9 a template for documenting SOA design decisions is proposed with the hypothesis that it aids the SOA design decision process. This template has been experimented in a SOA design Master course for three years and the design results were compared when the template was used and that when the template was not in place.

- Action research. An action research is an iterative research approach where the researcher actively participates in the case studies that he/she performs (Susman & Evered, 1978). With the goal of framing a basic set of concerns relating to service automation and providing a set of models, methods, and notations to illustrate the way in which the concerns are addressed in architecture design, a typical action research cycle with three iterations has been followed, each of which was carried out with one industry partner as a case study (Chapter 12).
1.5. RESEARCH METHODS

- Case study. This method is defined as “the collection and presentation of detailed information about a particular participant or small group, frequently including the accounts of subjects themselves” (Holz et al., 2006). In Chapter 6, a case study on the SeCSE methodology was carried out, modeling its development process using the techniques proposed in Chapter 4 and stakeholder roles identified in Chapter 5.

- Feature analysis. Feature analysis (Kitchenham, 1996), a qualitative evaluation, is one way to screen and assess a large number of software engineering methodologies by means of assessing their features (or characteristics). In Chapter 7, this method has been used to frame a set of criteria for comparing and selecting SOSE methodologies.

Table 1.1 depicts an overview of the research methods used in each chapter of the thesis.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Research methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 2, 3, 8</td>
<td>Systematic literature review</td>
</tr>
<tr>
<td>Chapter 4, 10, 11</td>
<td>Literature study</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Survey</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Case study</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>Feature analysis</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Experiment</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>Action research</td>
</tr>
</tbody>
</table>
Part I

SOSE challenges
A literature review of SOSE challenges

SOSE has drawn increasing attention since service-oriented computing was introduced in the beginning of this decade. A large number of SOSE challenges that call for special software engineering efforts have been proposed in the research community. To gain insight into the current status of SOSE research issues as published to date, this chapter reports a systematic literature review exploring SOSE challenges that have been claimed between January 2000 and July 2008. This chapter presents the results of the systematic review as well as the empirical research method that has been followed. In this review, of the 729 publications that have been examined, 51 were selected as primary studies, from which more than 400 SOSE challenges were elicited. The SOSE challenges have been classified along two dimensions: a) based on themes (or topics) that they cover and b) based on characteristics (or types) that they reveal. By analyzing the distribution of the SOSE challenges on the topics and types in the years 2000-2008, this chapter points out the trend in SOSE research activities. The findings of this review further provide empirical evidence for establishing future SOSE research agendas.

2.1 Introduction

As a relatively new field, numerous SOSE challenges have been claimed in the research community. In this chapter, we define SOSE challenges as SOSE-related problems or issues that demand research efforts.

Facing a large number of SOSE challenges, Kontogiannis et al. (Kontogiannis et al., 2007; Kontogiannis et al., 2008) and Papazoglou (Papazoglou et al., 2007) pointed out that a well focused research agenda is necessary to consolidate and stream current SOSE research efforts. Currently, several workshops, conferences, projects and initiatives are dedicated to establish SOSE research agenda, such as the International Workshop on Systems Development in SOA Environments.
CHAPTER 2. A LITERATURE REVIEW OF SOSE CHALLENGES

(SDSOA) and the Networked European Software and Services Initiative (NESSI). In addition, conferences, workshops, and journals possibly contributing to such SOSE research agenda but having a broader focus include ICSOC\(^1\) in 2003, SCC\(^2\) in 2004, SOSE\(^3\) and SOAIC\(^4\) in 2005, MD4SOA\(^5\) in 2006, SDSOA\(^6\) and SOCA\(^7\) in 2007 and IESS\(^8\) in 2010.

The commonality among these research activities is that they are all centered around a subset of SOSE challenges rather than a complete set, depending upon their research focuses. For instance, the SOA research agenda proposed by Kontogiannis et al. (Kontogiannis et al., 2008) discusses research issues related to SOA adoption for organizations. The challenges raised due to the collaboration between multiple parties in service-oriented development (Tsai et al., 2007c), for example, are not covered in this agenda. As another example, the SOC research roadmap proposed by Papazoglou et al. (Papazoglou et al., 2007) presents research issues in terms of extended SOA, whose core layers are service foundations, service compositions and service management. Cross-cutting concerns, such as SOA education (Lopez et al., 2007), are not considered in this roadmap.

The aforementioned research agenda and roadmap present a list of important challenges for service-oriented systems to reach their full potential. Without a holistic view on “all” the SOSE challenges posed by researchers, less well-known challenges are likely to be ignored. An overview of a complete set of SOSE challenges can provide a firm basis for establishing SOSE research agenda.

Brereton et al. (Brereton et al., 2005) are the pioneers who conducted a systematic review of issues that need to be addressed in engineering service-oriented systems. The results of their review include a number of issues vital for engineering service-oriented systems, for which solutions as well as associated research methods have been investigated. However, it has the following limitations.

First of all, the review conducted by Brereton et al. restricted the data sources to journals only. We understand that journal papers usually guarantee the quality of the studies due to strict review procedures, which is the main reason why six journals that are strongly related to software engineering research were chosen as primary sources. Conference and workshop proceedings (major sources for researchers to discuss open issues, immature ideas and research challenges), however, were not considered in the review. Second, informal data analysis methods have been used for data synthesis. The identified issues have been grouped but

\(^1\)International Conference on Service Oriented Computing
\(^2\)International Conference on Services Computing
\(^3\)International Workshop on Service-Oriented System Engineering
\(^4\)International Workshop on Service-oriented Applications, Integration and Collaboration
\(^5\)Modeling, Design, and Analysis for Service-oriented Architecture Workshop
\(^6\)International Workshop on Systems Development in SOA Environments
\(^7\)Journal of Service Oriented Computing and Applications
\(^8\)International Conference on Exploring Services Sciences
2.1. INTRODUCTION

no synthesis method has been described. Third, a framework that was designed for classifying component-based software engineering (CBSE) issues was adopted for classifying the identified SOSE issues as well. Although this framework is applicable to classify SOSE issues due to similarities between SOSE and CBSE, it provides a limited classification approach because the framework was pre-defined rather than emerged from the identified SOSE issues.

Last, the studies that have been analyzed in Brereton et al.’s review were published between 2000 and June 2004. After the establishment of all the aforementioned conferences/workshops and journals on service-oriented computing, it is time to perform an up-to-date systematic review of SOSE issues or challenges.

A systematic literature review (also called systematic review) is an evidence-based approach to thoroughly search studies relevant to some pre-defined research questions and critically select, appraise and synthesize findings to answer such research questions. Unlike traditional narrative reviews, systematic reviews often include focused research questions, explicit search strategies, explicit selection criteria, qualitative as well as quantitative-summary. Because systematic reviews methodologically encourage rigorous review results, we decided to perform a systematic review of all the SOSE challenges that have been proposed so far.

This chapter reports on the results of our systematic review aiming at identifying and classifying SOSE challenges. In doing that, we provide an overview of the-state-of-the-art of SOSE challenges, and a key for reading and interpreting them. In addition, this chapter presents a number of interesting findings, including SOSE emergence laws, a parallel between the SOSE challenges cycle and the Gartner Hype Cycle, and the importance of inter-relationships between SOSE challenges. Our findings also ring a bell to the research community that data-related SOSE issues require more attention in the SOSE research community.

As suggested in (Kitchenham, 2007), we organize the content of this chapter according to the structure proposed in (Khan et al., 2001). In § 2.2, we describe the process of our systematic review, including the need for a systematic review, the definition of the research questions, the description of search and selection strategy, the explanation of data extraction forms, and the introduction of data synthesis methods. The description of the process defines the review protocol. § 2.3 presents the included studies. Our answers to the research questions are presented in § 2.4. In § 2.5 we mainly discuss the threats to validity. Finally, we conclude the chapter in § 2.6.
CHAPTER 2. A LITERATURE REVIEW OF SOSE CHALLENGES

2.2 Research method

2.2.1 Systematic review

Systematic reviews are particularly powerful in collecting and analyzing existing work, which is a common task in establishing background knowledge in any research. The main reasons to perform systematic reviews, outlined in (Kitchenham, 2007), include: 1) summarizing the existing evidence about a specific software engineering issue; 2) identifying gaps in current research to suggest further research directions, and 3) positioning new research activities in a research framework.

We chose systematic review as our research method due to its methodological advantages and applicability to our research questions. From the methodological point of view, systematic reviews are methodical. They maximize the chance to retrieve complete data sets and minimize the chance of bias. From the applicability point of view, our research questions are in accordance with the first reason summarized in (Kitchenham, 2007), i.e., summarizing the existing evidence.

To conduct the systematic review, we followed the guidelines suggested in (Kitchenham, 2007) for performing systematic reviews in software engineering. According to the guidelines, a systematic review constitutes three main phases, namely 1) planning the review, 2) conducting the review, and 3) reporting the review. The main activities in the planning phase are to specify research questions and develop a review protocol. A review protocol is of critical importance to a systematic review because it specifies search, selection, data extraction, and synthesis strategies. After having specified and evaluated the protocol, its execution is the main activity in the conducting phase. Finally, the results of the review should be reported.

2.2.2 Research questions

SOSE challenges have been often addressed in different studies from different perspectives simultaneously, making challenges appear fragmented. For instance, one challenge can be addressed from the product perspective in one study and the process perspective from another study. This challenge therefore assumes a different meaning in each of these two studies. To give an example, service design (Papazoglou et al., 2007) is one of the SOSE challenges. When addressed from the product perspective, it concerns the quality (e.g., reusability (Zhu, 2005)) and characteristics (e.g., loose coupling, implementation neutrality (Huhns & Singh, 2005)) of resulting services. When addressed from the process perspective, it concerns the way service design is carried out (e.g., collaborative design (Tsai et al., 2007c)) and its supporting tasks (e.g., decide the level of coupling in service
2.2. RESEARCH METHOD

design (Lee et al., 2006)). From this example we can see that all these challenges (quality, characteristics, approaches, and tasks) are aspects of service design. If we were able to group or classify inter-related challenges, research efforts could be better focused to address all aspects of a clustering (super-)challenge.

SOSE challenges are also described in different studies at various levels of abstraction, making challenges appear fictively independent or isolated. For instance, typically challenges described at a high level (or conceptual level) are about proposing requirements for engineering activities or resulting products, including architectures or components of service-oriented systems; challenges described at a lower level (or implementation level) often advocate the need for new techniques or tools that can be used in service-oriented engineering activities. The implementation level challenges are often the “solutions” towards the conceptual level challenges. To give an example, a SOSE challenge claiming that services should be dynamically discoverable is a conceptual level challenge because it does not directly address implementation details. An implementation-level challenge could be the need for techniques to specify services not only in syntax but also in semantics because specifying services’ semantics is a way to enable service discovery. We can see that the challenge dynamically discoverable services is dependent on the challenge specifying services. If we were able to link the challenges that are inter-dependent, the direction of research efforts could be more precise.

In order to explore the links between SOSE challenges, in our early work (Gu & Lago, 2007) we proposed a framework aiming at classifying SOSE challenges and explicitly expressing the inter-dependencies between them. In that work, we realized that without a holistic view of the existing SOSE challenges, a sound framework is very hard to create. We also realized that a classification scheme for SOSE challenges is essential for such a framework due to the diverse nature of SOSE challenges. With this motivation in mind, the aim of this systematic review is not only to identify all the claimed SOSE challenges, but also to classify them so that inter-related and inter-dependent challenges can be grouped. Before conducting this systematic review, we defined two hypotheses on the classification of SOSE challenges.

- Hypothesis 1: SOSE challenges can be classified based on their topics.
  
  We formulated this hypothesis because SOSE challenges are addressed at different levels of detail. Some implementation-level challenges can be regarded as sub-challenges of a conceptual-level challenge. We may consider all these challenges as illustrating the same topic.

  We define topic as the subject or theme of a challenge. Challenges illustrating the same topic should address the same subject but with a different focus. For instance, dynamic service discovery and semantically enhanced service discovery are both about service discovery. The former focuses on
performing service discovery in a dynamic manner rather than a static manner, while the latter emphasizes enhancing the way that services are discovered by adding semantic annotations to service specifications.

The conditions for this hypothesis include: 1) each challenge should have one and only one topic; 2) the abstraction level of a challenge should be lower or equal to the abstraction level of its topic.

- **Hypothesis 2:** SOSE challenges can be classified based on their *types*.

  We formulated this hypothesis because SOSE challenges are addressed from different perspectives, such as process, product and architectural perspectives. We may regard all the challenges that are addressed from the same perspective as one type.

  We define type as a general *form* or *style* common to a number of challenges. The challenges that are of the same type must have distinguishing characteristics in common. For instance, *composability analysis*, *transaction management tools*, and *encryption* are three SOSE challenges. These challenges are common methods or tools that can be used to accomplish some tasks: composability analysis is used in service composition, transaction management tools support transaction management, and encryption is a way to improve security. Therefore, these three challenges can be regarded as of the same (general) type, e.g., technique challenges.

  The condition for this hypothesis is that each challenge should have one and only one type.

With these two hypotheses in mind, we specify the following research questions:

- **Q1:** What SOSE challenges have been claimed in the research community so far?
- **Q2:** How to classify the claimed SOSE challenges?
  - **Q2.a:** What are the characteristics of the challenges? Can we classify these challenges by types based on their characteristics?
  - **Q2.b:** What are the topics addressed by the challenges? Can we summarize a list of topics that these challenges have covered?
  - **Q2.c** Are there any other ways to classify SOSE challenges?
2.2.3 Review protocol

A review protocol is a plan specified prior to the execution of a systematic review. This plan describes how to search and select relevant studies as well as how to analyze extracted data to answer pre-defined research questions. A well-defined review protocol is essential to a systematic review since it encourages less researcher bias (Kitchenham, 2007).

The main components of the review protocol include: data sources, search strategy, study selection strategy, data extraction method, and data synthesis. The first three components define the scope of the study and explain the motivation behind it. The last two components describe how the results are concluded. All these components are essential for the readers to fully understand and appraise the review and critical for the researchers in the same domain to replicate the review or repeat the review with other datasets (Dybå & Dingsoyr, 2008). We, therefore, explicitly describe the review protocol in this subsection.

Data sources

The following electronic libraries give a reasonable confidence of covering all relevant publications. Therefore, we use these libraries as our main resources:

- IEEE Explore
- ACM Digital Library
- ISI Web of Knowledge
- SpringerLink
- ScienceDirect
- Wiley Inter Science Journal Finder

Search strategy

The research questions listed in § 2.2.2 require the same set of data collected from the literature. Research question Q1 is relatively more straightforward and question Q2 requires further data synthesis. The search strategy was therefore developed based on research question Q1.

Question Q1 contains two key terms, namely: SOSE and challenge. A list of related terms was constructed for each of these two terms, as shown in Table 2.1.

A search string was then constructed using Boolean “and” to connect the two key terms and “or” to allow synonyms.
CHAPTER 2. A LITERATURE REVIEW OF SOSE CHALLENGES

Table 2.1: Related terms for the two key terms: SOSE and challenge

<table>
<thead>
<tr>
<th>SOSE</th>
<th>challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA</td>
<td>approach</td>
</tr>
<tr>
<td>service-oriented</td>
<td>agenda</td>
</tr>
<tr>
<td>service computing</td>
<td>roadmap</td>
</tr>
<tr>
<td>service-based</td>
<td>road map</td>
</tr>
<tr>
<td>service-centric</td>
<td>overarching concern</td>
</tr>
<tr>
<td>service engineering</td>
<td>research issue</td>
</tr>
</tbody>
</table>

(SOSE or SOA or “service-oriented” or “service computing” or “service-based” or “service-centric” or “service engineering”) and (challenge or approach or agenda or roadmap or “road map” or “overarching concern” or “research issue”)

Besides the search string, the range of study dates should also be defined in the search strategy. For instance, Brereton et al. (Brereton et al., 2005) have chosen 2000 as the starting year for their review because SOAP was first submitted to W3C in 2000 and web services could be implemented from then on. Although major conferences/workshops on service-oriented systems like ICSOC started in 2003, we decided to use 2000 as the start date to eliminate any chance to overlook SOSE challenges proposed in conferences/workshops or journals which do not specifically focus on service-oriented systems.

The last decision we made was to apply search queries to titles, abstracts or full-text of publications. In our experience titles do not always correctly or completely reflect the content of publications. Searches based on titles cannot provide us with a reasonably complete set of publications relevant to our research questions. On other hand, if we search in full-text, the chance of finding irrelevant publications is too high. Abstracts usually provide brief and moderate summaries of publications. We, therefore, decided to apply search queries to the abstracts of the studies. This means a study is selected as a candidate study if its abstract contains the keywords defined in the search string.

Because different digital search engines often provide different search interfaces and require different search syntaxes, we had to adapt the search string for each search engine.

**Study selection**

Some of the studies might contain the keywords used in the search string but are irrelevant to our research questions. Study selection therefore has to be
performed to include only studies that contain useful information for answering the research questions. The selected studies constitute primary studies for a systematic review.

Accordingly, a set of inclusion and exclusion criteria are specified based on the scope of the review and the quality of the studies. A study is selected as a primary study if it satisfies all the pre-defined inclusion criteria and is eliminated if it fulfills any of the pre-defined exclusion criteria. These criteria, together with the corresponding motivation and rationale are given in Table 2.2.

Inclusion criterion I1 limits our review to studies that are strongly related to SOSE. More specifically, exclusion criteria E1 and E2 are used to eliminate studies that are irrelevant and marginally related to SOSE, respectively, and E3 indicates that SOSE in a specific application domain is beyond the range of our review.

Inclusion criterion I2 limits our review to studies that are strongly related to challenges. More specifically, exclusion criterion E4 indicates that solutions to SOSE challenges are beyond the scope of this review; and E5 points out that a study that is strongly related to SOSE but does not propose SOSE challenges is not relevant in our review.

The last criteria I3 and E6 are used to select studies that reach the quality of scientific papers.

Data extraction

The reference details of each primary study are recorded using Endnote. For quantitative analysis purposes, we also record the type of paper (conference/workshop or Journal/Book chapter) and the year of publications in a spreadsheet.

Each primary study is analyzed concentrating on identifying SOSE challenges. When a study describes a challenge regarding SOSE or proposes a SOSE research issue, we elicit SOSE challenges from this study in such a way that each SOSE challenge is the smallest unit (i.e., cannot be decomposed to multiple challenges from its original description). For instance, suppose a study describes a SOSE research issue as “services should be discoverable and composable”, we elicit two SOSE challenges as “services should be discoverable” and “services should be composable”. In this way, each elicited SOSE challenge has one focus.

All the identified challenges are documented in a spreadsheet in terms of their names, descriptions, and rationale (if present). The data model for the data extraction form is presented in Figure 2.1.

---

9http://www.endnote.com
CHAPERN 2. A LITERATURE REVIEW OF SOSE CHALLENGES

Table 2.2: Motivation and rationale for inclusion and exclusion criteria

<table>
<thead>
<tr>
<th>Inclusion criteria</th>
<th>Exclusion criteria (exceptions to inclusion criteria)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1 A study that is mainly about service-oriented systems or service-oriented</td>
<td>E1 A study that is not about service-oriented architecture.</td>
</tr>
<tr>
<td>architectures. Motivation: we are interested in challenges that are related to SOSE.</td>
<td>Rationale: since we use SOA as one of the keywords in the search strings, studies that have “SOA” in their abstracts but are completely irrelevant to service-oriented systems should be excluded. For instance, studies about Society Of Actuaries or Silicon-On-Anything will be excluded.</td>
</tr>
<tr>
<td></td>
<td>E2 A study which is marginally related to service-oriented systems.</td>
</tr>
<tr>
<td></td>
<td>Rationale: if the core of a study is about another field than service engineering and is only marginally related to service-oriented systems should be excluded. For instance, a study that is mainly about how to design and develop health information systems should be excluded.</td>
</tr>
<tr>
<td></td>
<td>E3 A study that is about SOA application in a specific domain.</td>
</tr>
<tr>
<td></td>
<td>Rationale: we are interested in SOSE-related challenges in general but not in specific application domains. The challenges that are very important to one application domain might be less relevant to another application domain. Although domain challenges are also vital to the application of service-orientation, it is not the focus of this study. For instance, a study proposing SOSE challenges that the military domain faces should not be included because these challenges might not be relevant or important to other application domains; however, a study proposes SOSE challenges in general and further elaborates how these challenges influence the military information system will be included.</td>
</tr>
<tr>
<td>I2 One of the main objectives of a study is to present service engineering research</td>
<td>E4 The objective of a study is only to present solutions to a SOSE challenge.</td>
</tr>
<tr>
<td>challenges, issues, open questions, etc.</td>
<td>Rationale: if presenting SOSE challenges is not one of the objectives of a study, most likely the challenges (if ever mentioned) are known challenges. These known challenges should be (originally) claimed by other studies.</td>
</tr>
<tr>
<td>Motivation: if one of the objectives of a study is to propose a challenge, we expect that this challenge has not been proposed before.</td>
<td>E5 A study that has other objective than proposing challenges or solutions.</td>
</tr>
<tr>
<td></td>
<td>Rationale: a study that does not propose any SOSE challenges has no value to our research questions.</td>
</tr>
<tr>
<td>I3 A study that is in form of a scientific paper.</td>
<td>E6 A study that is not in form of a scientific paper. For instance, tutorials, keynotes, presentations do not provide sufficient information.</td>
</tr>
<tr>
<td>Motivation: a scientific paper guarantees a certain level of quality and contains reasonable amount of content.</td>
<td>Rationale: a study that is not a scientific paper might not have sufficient level of quality or does not provide reasonable amount of information.</td>
</tr>
</tbody>
</table>
2.2. RESEARCH METHOD

Data synthesis

After all the SOSE challenges are identified from the primary studies and recorded in a spreadsheet, we have a pool of challenges in the form of descriptive text. In order to answer the question Q2 (how to classify SOSE challenges), we need to synthesize the extracted data as follows:

Classify the SOSE challenges by their topics. Reciprocal translation and line of argument synthesis, two strategies of Meta-ethnography especially useful for interpretive reviews, were first introduced by Noblit and Hare (Noblit & Hare, 1988).

As Kitchenham (Kitchenham, 2007) suggested, reciprocal translation can be used “when researchers need to provide an additive summary on the studies that are about similar things by translating each case into each of the other cases”. Instead line of argument synthesis can be used “when researchers are concerned about what they can infer about a topic as a whole from a set of selective studies that look at a part of the issue”.

In our systematic review, if some of the SOSE challenges are about a similar topic, reciprocal translation is applicable to translate each topic into each of the other topics. For instance, suppose we have identified three challenges, namely 1) dynamic service composition, 2) compose services at runtime 3) automatic service composition. By applying reciprocal translation, we may conclude that all these three challenges are about composing services without human intervention. Therefore, we may unify these three challenges by naming all of them dynamic service composition. We should apply reciprocal translation to all the identified SOSE challenges until no challenge can be translated into another.

After applying reciprocal translation to all the challenges, we examine whether one challenge is a sub-challenge of another. We use line of argument synthesis to determine the main theme of a set of challenges. For instance, we have identified the following challenges: 1) dynamic service
composition, 2) integrate transaction management into the service compositions, 3) service composition language, and 4) composability analysis. All these four challenges fall under the umbrella of another challenge “service composition”. We may therefore conclude that these four challenges all address the topic “service composition”. In this case, “service composition” is both the name of the SOSE challenge and of the topic.

In practice, we should apply line of argument synthesis to all the challenges in an iterative manner. In the first iteration, temporary topics should be determined for all the SOSE challenges. Afterwards, we should apply line of argument synthesis to all the temporary topics in case some of the temporary topics can be further translated. We have to repeat this process until each topic is distinguishing and none of the topic is a sub-topic of another. In the end, we generate a list of topics of the SOSE challenges.

Classify the SOSE challenges by their types. A type of the SOSE challenges can be regarded as a term that insightfully describes the characteristics of SOSE challenges. The SOSE challenges that are assigned by the same type should share the same characteristics. A common method that is used for this purpose is the grounded theory method because the theories (types) are “grounded” in the data (the SOSE challenges) (Strauss & Glaser, 1967).

Constant comparison method, one of the grounded theory techniques, has been often used in analyzing data and generating categories of data. Although constant comparison method can be used on any set of data, it is particularly suitable for the data that are context sensitive (Seaman, 1999) (i.e., data can be interpreted differently in different contexts).

To interpret an SOSE challenge correctly, one often needs to understand in which context the challenge is proposed and how it is addressed. For instance, Zhu (Zhu, 2005) claims that “to make services reusable” is a challenge. Without understanding the context of this challenge, we cannot conclude whether this challenge is about designing reusable services or about what can be reused in services. By reading the context, we understand that this challenge is about the understandability of services, the price/performance ratio of services, etc. We can therefore conclude that this challenge was addressed from the quality perspective (reusability) rather than the design principle perspective.

Miles and Huberman (Miles & Huberman, 1994) described the procedure for the constant comparison method as: coding, pattern coding, memo, refining proposition. Following this procedure, we have to label each identified SOSE challenge with emergent codes; look for patterns or trends
2.3. OVERVIEW OF THE INCLUDED STUDIES

in the codes; compare and study coded passages; record any findings in memos; and eventually, propositions can be established and refined from the memos. The iteration of these steps continues until no new proposition emerges and no modification to proposition is needed, which means theoretical saturation is reached. These propositions reflect the patterns and characteristics (types) recognized among a subset of SOSE challenges. In this way, SOSE challenges are classified by their types.

2.3 Overview of the included studies

In total we found 729 publications, whose abstracts contain the keywords defined in the search string. By carefully evaluating these publications against the inclusion and exclusion criteria, we selected 51 publications that are strongly relevant to our research questions, identified as primary studies.

Figure 2.2 shows the distribution of primary studies published by year as well as the trend-line. The positive trend-line slope indicates an increasing amount of research work being dedicated to identifying, proposing and collecting SOSE challenges.

According to the results of our review, the first three primary studies proposing SOSE challenges were published in 2003 (when ICSOC was launched). Although the number of primary studies slightly drops in 2004, it tremendously increases and reaches its first peak in 2005; it slightly decreases again in 2006 followed by the second peak in 2007, which is doubled as compared to the first peak. We interpret these two drops as shifting research efforts from identifying challenges to investigating solutions. As soon as more experiences have been accumulated in engineering service-oriented systems, researchers are keen to investigate ways to maximize the potential benefits that SOA promised, such as flexibility, agility and reusability.

It is very interesting to notice that our interpretation on the distribution of primary studies resembles the two most prominent laws of software evolution suggested by Lehman (Lehman & Perry, 1997): continuous changes and increased complexity. These two laws indicate that when software is evolving, changes to software are inevitable. Maintenance efforts therefore have to be devoted to reduce the increased complexity caused by the changes. When we look at SOSE challenges, there are continuous expectations instead of continuous changes; and there are growing understandability instead of increased complexity. Consequently, the higher the expectations, the more work is dedicated to identifying challenges; the better the challenges are understood, the more work is dedicated to overcome the challenges. These two regular patterns are called SOSE emergence laws in our review.
These two SOSE emergence laws are also revealed by the summary of primary studies in terms of sources and year, as presented in Figure 2.3. The sources of the primary studies include conference, workshop and symposium proceedings, as well as journal articles, and book chapters. Generally speaking, publications in journals and books represent more mature research results than those in conference, workshop or symposium proceedings. According to our review, the SOSE challenges elicited from journal articles or book chapters were first posed in 2005. The increasing number of SOSE challenges appearing in journal articles or book chapters in 2006 and 2007 not only indicates an incremental insight in the field, but also points out that SOSE challenges are attracting growing attention in the research community.

Table 2.3: Summary of primary studies in terms of source and year

<table>
<thead>
<tr>
<th>Source</th>
<th>2003</th>
<th>2004</th>
<th>2005</th>
<th>2006</th>
<th>2007</th>
<th>2008</th>
<th>Grand Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book chapter</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Journal</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>Subtotal</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>Conference</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>Symposium</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Workshop</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>Subtotal</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>13</td>
<td>6</td>
<td>33</td>
</tr>
<tr>
<td>Grand total</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>7</td>
<td>21</td>
<td>9</td>
<td>51</td>
</tr>
</tbody>
</table>
2.4 Results of the systematic review

2.4.1 RQ1 - Claimed SOSE challenges

A total of 413 SOSE challenges were elicited from the selected 51 primary studies. The distribution of the number of SOSE challenges proposed by year is illustrated in Figure 2.3. While Figure 2.2 is about the amount of work dedicated to identifying SOSE challenges, this distribution presents the number of SOSE challenges actually being identified. It would be reasonable to expect an increasing number of challenges with the increase in the number of publications related to challenges. Nonetheless, e.g., certain publications might identify many more challenges than others. In other words, the distribution of challenges is not necessarily the same or similar to the distribution of primary studies. For instance, we might encounter a situation in which the studies in 2003-2005 propose say 20 challenges, and the studies after 2005 propose only one challenge. In this hypothetical situation, the trend-line slope of SOSE challenges would be negative. The similarity between these two distributions (in Figures 2.2 and 2.3) points out that the actual results from research publications on SOSE challenges also reflect the SOSE emergence laws.

![Figure 2.3: Distribution of the number of SOSE challenges proposed by year](image)

It is interesting to see that the distribution line presented in Figure 2.3 reveals what we call the SOSE challenges cycle (see the dashed line). This cycle recalls the Gartner Hype Cycle (Inc., 2008) (both illustrated in Figure 2.4). While the Gartner Hype Cycle (thicker line in the Figure) graphically presents the maturity level of the application of specific technologies, the SOSE challenges cycle presents the awareness of SOSE-related research issues. These two cycles are mutually related. When SOSE faces peak of inflated expectations (see the peak
of the Gartner hype cycle in Figure 2.4), researchers are ambitious to investigate how to fulfill those expectations and therefore the awareness of SOSE challenges tremendously increases. When investigation results start to appear, new research issues keep emerging and known research issues are further decomposed. Therefore, awareness of SOSE challenges increases further until it reaches its peak (see the peak of the SOSE challenges cycle in Figure 2.4). At the same time, SOSE faces trough of disillusionment (see the base of the Gartner hype cycle in Figure 2.4) since researchers are aware that there are too many SOSE challenges to be solved before SOSE reaches the level of maturity to be widely adopted.

Figure 2.4: Gartner Inc.’s hype cycle (Source: http://en.wikipedia.org/wiki/Hype_cycle) and the SOSE challenges cycle

We observe that a) in 2007 a large number (193) of SOSE challenges have been published and b) it is unlikely that there will be more SOSE challenges published in 2008 than in 2007 since there are only 64 challenges published before July 2008 (i.e., about one third of the challenges in 2007). Further, researchers have already accumulated knowledge of SOSE-related research issues from 2003 and these issues cover a wide range of SOSE research topics. Based on these observations, we deduce that the number of SOSE challenges is reaching stability (see the decreasing part of the SOSE challenges cycle in Figure 2.4). We therefore argue that we are currently in the slope of enlightenment of the Gartner Hype Cycle.
2.4. RESULTS OF THE SYSTEMATIC REVIEW

2.4.2 RQ2.1 - Topics of SOSE challenges

By using the reciprocal translation and line of argument synthesis methods explained in § 2.2.3, we determined one topic for each elicited SOSE challenge and we summarized 45 topics in total. In this way, the SOSE challenges can be classified by their topics, which confirmed our first hypothesis. The summarized topics are presented in Figure 2.5, sorted by the descending number of challenges under each topic.

Looking at the summarized topics, we noticed that many topics are not at the same abstraction level. Some of the topics (e.g., SOA education) are broader than the others (e.g., fault handling, transaction). The different abstraction levels are caused by the fact that the topics that have been addressed as SOSE challenges elicited from the primary studies already belong to different abstraction levels. And this becomes apparent after using the synthesis methods (explained in § 2.2.3) as well.

To get an overall idea about the coverage of these topics, we compared our results with the issues identified in the systematic review conducted by Brereton et al. (Brereton et al., 2005), which to our knowledge is the only systematic review performed on SOSE studies. Of the 20 issues concluded in (Brereton et al., 2005), 2 issues are not touched at all in our review (‘payment’ and ‘time to market’), and 2 issues partially overlap as being only indirectly related to the topics in our review (‘service replication’ and ‘statefulness in the context of replication’). Service replication is one of the approaches to improve the capability of fault tolerance or fault handling. In our review, although we do not find any SOSE challenge directly linked to service replication, we do find some challenges addressing fault handling. Therefore, we regard these two issues as indirectly related. The remaining 16 issues match with the topics summarized in our review.

We have to keep in mind that the publication duration covered by these two reviews is different. The review conducted by Brereton et al. covers the studies that are published between January 2000 and June 2004, whereas our review covers studies that are published between January 2000 and July 2008. Furthermore, the research questions addressed by these two reviews overlap but with a different focus: Brereton et al. analyzed the issues occurred in engineering service-oriented systems as well as solutions proposed to address these issues, whereas our review does not investigate solutions. Therefore, the identified SOSE research topics between these two reviews are most likely to overlap with some differences. We therefore use the comparison results for reference purposes only, rather than validation purposes.
### CHAPTER 2. A LITERATURE REVIEW OF SOSE CHALLENGES

<table>
<thead>
<tr>
<th>Topics</th>
<th>2003</th>
<th>2004</th>
<th>2005</th>
<th>2006</th>
<th>2007</th>
<th>2008</th>
<th>Grand Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>quality</td>
<td>--</td>
<td>2</td>
<td>17</td>
<td>7</td>
<td>27</td>
<td>--</td>
<td>53</td>
</tr>
<tr>
<td>service</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>12</td>
<td>19</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>data</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>19</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>testing</td>
<td>--</td>
<td>--</td>
<td>11</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>verification and validation</td>
<td>--</td>
<td>--</td>
<td>9</td>
<td>2</td>
<td>11</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>modeling</td>
<td>--</td>
<td>--</td>
<td>4</td>
<td>--</td>
<td>17</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>service composition</td>
<td>--</td>
<td>--</td>
<td>5</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>implementation</td>
<td>2</td>
<td>--</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>infrastructure</td>
<td>--</td>
<td>1</td>
<td>6</td>
<td>--</td>
<td>7</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>service specification</td>
<td>2</td>
<td>--</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>design</td>
<td>--</td>
<td>--</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>management</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>--</td>
<td>8</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>transaction</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>11</td>
<td>--</td>
<td>12</td>
</tr>
<tr>
<td>BPM-SOA</td>
<td>--</td>
<td>1</td>
<td>1</td>
<td>--</td>
<td>6</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>governance</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>4</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>monitoring</td>
<td>--</td>
<td>--</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>requirements engineering</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>evaluation</td>
<td>--</td>
<td>--</td>
<td>4</td>
<td>--</td>
<td>3</td>
<td>--</td>
<td>7</td>
</tr>
<tr>
<td>service contracts</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>2</td>
<td>--</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>engineering</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>service provisioning</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>SOA education</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>strategy</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>business-IT alignment</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>people</td>
<td>--</td>
<td>2</td>
<td>--</td>
<td>--</td>
<td>3</td>
<td>--</td>
<td>5</td>
</tr>
<tr>
<td>development</td>
<td>--</td>
<td>--</td>
<td>3</td>
<td>--</td>
<td>1</td>
<td>--</td>
<td>4</td>
</tr>
<tr>
<td>registry</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>service consumption</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>fault handling</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>--</td>
<td>2</td>
<td>--</td>
<td>3</td>
</tr>
<tr>
<td>indicators</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>maintenance</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>migration to SOA</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>3</td>
<td>--</td>
<td>--</td>
<td>3</td>
</tr>
<tr>
<td>service selection</td>
<td>--</td>
<td>--</td>
<td>2</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>workflow</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>--</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>collaboration</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>2</td>
</tr>
<tr>
<td>decomposition</td>
<td>--</td>
<td>1</td>
<td>1</td>
<td>--</td>
<td>1</td>
<td>--</td>
<td>2</td>
</tr>
<tr>
<td>iaas</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>2</td>
<td>--</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>service adoption</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>2</td>
<td>--</td>
<td>2</td>
</tr>
<tr>
<td>service discovery</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>2</td>
<td>--</td>
<td>2</td>
</tr>
<tr>
<td>service versioning</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>best practice</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>service invocation</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>service negotiation</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>social and legal Issues</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>stakeholder management</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Grand Total**: 12 10 85 49 193 64 413

Figure 2.5: An overview of the number of SOSE challenges of each topic each year (sorted by the descending number of challenges under each topic)
2.4. RESULTS OF THE SYSTEMATIC REVIEW

Top 3 topics: quality, service and data

Looking at the summarized topics in Figure 2.5, some of them are well-known in that we have seen numerous research papers relating to those topics, but some of them are new to us in that we do not know how much research efforts have been dedicated to those challenges. To gain insight into the heaviest or lightest published research areas, we specifically zoomed into the topics that cover the highest number of challenges as well as the lowest ones.

As we can see in Figure 2.5, the topic of ‘quality’ has been addressed by the highest number of challenges. More than 50 challenges elicited in our review address quality-related issues, including security, reusability, flexibility, interpretability and performance. These quality-related issues are emphasized due to the dynamic nature of service-oriented systems. When facing a number of competitive services that deliver similar functionalities, QoS is one of the few factors that a service consumer uses to decide which services to select. Since services are often cross-organizational (which means that the characteristics of hardware and platforms are unpredictable), delivering, measuring and testing the quality of the provided services are extremely challenging. Furthermore, when business functions are progressively exposed as services, the requirement for QoS turns to be more critical. Therefore, it is not surprising to see that more than one tenth of the elicited SOSE challenges address quality-related issues.

Next to the topic of ‘quality’ is the topic of ‘services’, the building blocks of service-oriented systems. The challenges relating to this topic express the properties that a good service is supposed to encompass. Such properties include loosely coupled, self-managed, discoverable, and compostable, just to name a few. The requirements of these properties come from the prospect that services are meant to be composed in order to deliver value-added business functions and that this composition can be achieved dynamically. This prospect also explains why the topics of ‘V&V’, ‘testing’, ‘service composition’, ‘infrastructure’ and ‘implementation’ are all among the top ten hottest SOSE topics.

As a surprise, we did not expect that many SOSE challenges relating to the topic of ‘data’. In order to gain a better insight into this topic, we have looked into the primary studies where data-related SOSE challenges were elicited. The first primary study posing data-related SOSE challenge was published in 2003 by Risse et al. (Risse & Knezevic, 2003). They proposed a research challenge regarding data management for service-oriented systems, in particular, how to store data to achieve reliability. Few more challenges were posed in successive years, mainly about integrated view of data, data visualization, and data mining. Most of the data-related SOSE challenges were posed in (Tsai et al., 2007c) in 2007 by Tsai et al., who also pointed out the important issues related to data provenance in service-oriented environments (Tsai et al., 2007a). Due to the link between data provenances and service composition, reliability, security, and
## CHAPTER 2. A LITERATURE REVIEW OF SOSE CHALLENGES

### Table 2.6: An overview of the number of SOSE challenges of each topic and each type (sorted in alphabetical order)

<table>
<thead>
<tr>
<th>Topics</th>
<th>business</th>
<th>service</th>
<th>operation</th>
<th>characteristic</th>
<th>cross-cutting</th>
<th>concern</th>
<th>QoS</th>
<th>artifacts</th>
<th>technique</th>
<th>SOSE activity</th>
<th>Grand Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>best practice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>BPM-SOA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>business-IT alignment</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>collaboration</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>26</td>
</tr>
<tr>
<td>decomposition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>development</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>evaluation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>fault handling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>governance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>IaaS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>implementation</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>indicators</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>infrastructure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>maintenance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>migration to SOA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>modeling</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>monitoring</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>people</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>quality</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>40</td>
<td></td>
<td></td>
<td>53</td>
</tr>
<tr>
<td>registry</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>requirements engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>service</td>
<td></td>
<td></td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>service adoption</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>service composition</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>service consumption</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>service contracts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>service discovery</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>service invocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>service negotiation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>service provisioning</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>service selection</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>service specification</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>service versioning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SOA education</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>social and legal issues</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>stakeholder management</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>strategy</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>testing</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>transaction</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>verification and validation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>workflow</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

**Grand Total:** 10 15 37 31 41 37 145 97 413
2.4. RESULTS OF THE SYSTEMATIC REVIEW

integrity of service-oriented systems, Tsai et al. (Tsai et al., 2007c) summarized a list of activities in a SOA data life cycle and corresponding techniques needed to support these activities, from which we elicited 18 SOSE challenges. We notice that two third of the data-related SOSE challenges originated there, which means that although many challenges have been elicited, the importance of the topic ‘data’ has not been commonly recognized in the community. We are reminded that more research efforts could be devoted to data handling in the context of service-oriented environments.

Topics that are less attractive

From the bottom of Figure 2.5, we looked into the topics that appear less attractive to the SOSE community. Surprisingly enough service discovery, a well known SOSE research issue, falls under one of the bottom topics. This unexpected bottom topic triggered us to investigate whether it is true that only two SOSE challenges addressed service discovery. The results of the investigation show that there are in fact a number of challenges addressing other topics that are indirectly related to service discovery. For instance, service description should provide necessary means for service discovery (Papazoglou, 2003)(topic: service), federating registries to support service discovery (Blake & Huhns, 2008)(topic: registry), one of the characteristics of services is being discoverable (Tsai et al., 2006)(topic: service), infrastructure should be context-aware to find services that are optimal for contexts (Siljee et al., 2005)(topic: infrastructure). Each of these challenges focuses on various topics like service, registry and infrastructure, but overcoming these challenges will eventually contribute to service discovery. These findings reveal that thanks to the active SOSE research efforts of the last years, researchers have gained valuable insight into some SOSE challenges. This insight allows to decompose a too broad challenge into a number of more specific or concrete challenges that often address various topics. In this way, research concerns are better organized and separated, and research directions appear clearer.

Innovative topics

We also notice that some topics located at the bottom of Figure 2.5 are quite innovative, for which researchers might have not yet precisely realized the related challenges. For instance, when enterprises start to adopt SOA, and services travel across the organizational boundaries, changes to social and legal aspects are inevitable (Contenti et al., 2003). These aspects are relatively new in the SOSE research community, where researchers usually concentrate on the technology-related issues rather than the social and legal ones.

Moreover, some topics like ‘best practice’ are typically the interests of practitioners rather than researchers. Since we did not purposely collect publications
CHAPTER 2. A LITERATURE REVIEW OF SOSE CHALLENGES

from the industry, it is reasonable that challenges with respect to SOA practice and application are less frequent in our review, as discussed in § 2.5.1 and § 2.6.

The trend of the topics

Looking at the distribution of the number of topics addressed by the SOSE challenges in each year (see Figure 2.7), we notice that it is slightly different from the distribution of the number of the SOSE challenges illustrated in Figure 2.3. Instead of the two drops in 2004 and 2006 with regard to the number of SOSE challenges, the number of topics increases each year from 2003 to 2007. Although it drops in 2008, it is still relatively high considering the drop of the number of challenges. In our opinion, the increasing number of topics in each year indicates the fact that the researchers have been keeping identifying new challenges in various SOSE disciplines.

Figure 2.7: Distribution of the number of topics by year

It is also interesting to see that the larger the number of the SOSE challenges is, the longer the corresponding topics exist. Looking at the upper part of Figure 2.5, we notice that most of the topics are addressed in all the years; while looking at the lower part of Figure 2.5, we notice that most of the topics are addressed only by the SOSE challenges identified in very recent years. This shows that new SOSE challenges emerge while known challenges remain in the SOSE research community.

More specifically, we observe that all the topics that appear between 2003 and 2006 remain in 2007 and 2008 except for three topics, namely: collaboration, decomposition and service negotiation. The SOSE challenges that address collaboration point out that business partners of a service-oriented system need to collaborate and work as a team. The SOSE challenges that address decomposition focus on how to decompose an existing application into services. In our
2.4. RESULTS OF THE SYSTEMATIC REVIEW

opinion, the reason why these two topics disappear after 2006 might be that two broader topics (stakeholder management and migration to SOA) attract more attention in recent years. Hence, collaboration and decomposition are replaced by these two broader topics.

It is very surprising to see that only one challenge proposed in 2006 addresses service negotiation. Service negotiation refers to the interaction between a service consumer and provider with the aim of reaching a service level agreement. The automation of this interaction is of great importance for a service-oriented system that dynamically discovers and composes services. Since dynamic service discovery and composition are still under research and are seldom applied in practice, service negotiation does not actively play a role in current service-oriented systems. However, we expect service negotiation to come back as a hotter research topic in near future.

We also observe several topics that emerge in 2008 including: service adoption, indicators, social, and legal issues and stakeholder management. These topics are quite innovative and often cross-cut multiple disciplines in SOSE. They concern not only technical issues but also business and organizational issues. The emergence of these topics shows that the scope of SOSE is not limited to IT-related issues; rather, it involves business, stakeholders, organizations, and the society.

2.4.3 RQ2.2 - Types of SOSE challenges

Following the constant comparison method explained in §2.3.5, we coded each elicited SOSE challenge and identified a number of distinguishing patterns among the codes. These patterns can be regarded as the properties of a group of SOSE challenges. This confirms our second hypothesis - SOSE challenges can be classified based on their types.

The definition of the eight types of SOSE challenges

We have identified eight types of SOSE challenges, namely characteristics, quality attributes, artifacts, SOSE activity, service operations, technique, business, and cross-cutting concerns. Each type of SOSE challenges is described as follows:

**Characteristic** is an unique feature, peculiarity, requirement or property of services that have been distinguished by the research community. Sometimes, characteristics are also called design principles. Being applicable to all service-oriented systems, characteristics are regarded as non-functional requirements within the scope of systems (indicated as white area in (Bass et al., 2003)), which means they are of concern for service producers including service developers, service providers, and service integrators. All the service-oriented systems should have the same set of characteristics.
CHAPTER 2. A LITERATURE REVIEW OF SOSE CHALLENGES

from the perspective of service orientation. For instance, loose coupling is a characteristic because service producers are supposed to deliver services that have the least dependencies on each other, that are composable with other services and, if changed, the impact is minimized (Papazoglou, 2003).

Quality attribute refers to the quality or capability that satisfies customer specifications. Different from characteristics, quality attributes are explicitly demanded by service consumers. Consequently, quality attributes often require various disposition on a case-by-case basis (indicated as black area in (Bass et al., 2003)). For instance, high performance might be important to one service consumer, whereas high security might be the interest of another service consumer.

Artifact is an object or unit that commonly exits in deployed service-oriented systems, such as service, infrastructure, service contract, etc. In this review we address artifacts from the product point of view. In the other words, artifacts constitute products. Therefore, objects or units not directly used in the deployed systems are not regarded as artifacts here. For instance, requirements specifications are not regarded as artifacts in our review because they are used during the design, development or even the testing phase but not directly involved in the operation of a deployed system. A service specification, however, is an artifact because it is used in service discovery, service composition, and service monitoring, which might occur after services are deployed (Papazoglou, 2003).

SOSE activity is a task carried out in engineering service-oriented systems, such as requirements engineering, service design, or testing. The challenges related to SOSE activities are often described in terms of requirements or expectations on these activities. For instance, requirements engineering in SOSE becomes a challenge because system structure, modeling and specification languages and execution environment all differ from traditional software systems (Tsai et al., 2007b).

Service operation is a special task which is often expected to be automated and executed dynamically in engineering service-oriented systems. We also notice that the objects of such a task are often services. To distinguish these tasks from the SOSE activities that are usually carried out by engineers, we classify this type of task as a service operation.

SOSE technique is a specific method, skill or tool that can be adopted in order to build a system with the required level of quality or encompasses the characteristics of service-oriented systems. A distinctive characteristic of a SOSE technique is that it can often be adopted in different contexts.
2.4. RESULTS OF THE SYSTEMATIC REVIEW

For instance, simulation is a technique that is often used to simulate new services for evaluating their runtime behavior before deployment. Simulation can also be used in requirements engineering and modeling. However, since this technique is not mature enough to simulate all the (runtime) behavior correctly and completely, research efforts are needed (Kontogiannis et al., 2007). Once simulation is mature, multiple fields (e.g., requirements engineering, testing) may benefit from it.

Cross-cutting concern is an issue relevant both to SOSE and to other domains. For instance, SOA education (Kontogiannis et al., 2007) is relevant to both SOSE and the education domain. This type of concerns can be regarded as challenges where research efforts are demanded to leverage issues from diverse domains.

Business-related challenges reflect issues that enterprises have to deal with due to SOA adoption. When enterprises start to expose their business functions as services, they may face challenges related to workload management, business process, stakeholder management, etc. Although these challenges are not directly related to SOSE, business-IT alignment requires the cooperation of the business community and the IT community (Shishkov et al., 2007). Therefore, we also regard business challenges as SOSE challenges.

Inter-relationship between types of SOSE challenges

All types of SOSE challenges are potentially inter-related. For instance, SOSE technique challenges can be regarded as a super-type which cross-cuts all the other types. This is because any proposed methods or tools, which are in support of attaining characteristics, quality attributes, required artifacts, engineering tasks, or even aspects relating to business and other domains, are regarded as technique challenges in our review. As a result, technique challenges are linked to the other seven types of challenges. When a technique challenge is resolved, its inter-related challenges are also benefited.

Another inter-relationship exists between process-related and product-related challenges. Challenges of SOSE activities and service operations are process-related challenges because they concern how to build service-oriented systems that meet the requirements of customers, whereas challenges of characteristics, quality attributes and artifacts are product-related challenges because they concern what should be built in service-oriented systems to meet the requirements of customers. Since better approaches (how) often lead to better results (what), overcoming process-related challenges are often in support of tackling product-related challenges.

To further investigate the potential inter-relationships between types, we
CHAPTER 2. A LITERATURE REVIEW OF SOSE CHALLENGES

study their distribution over time. In this way we are able to observe the emergence of each type, without entering the details of the individual challenges belonging to that type (which can be a further issue for future work).

**Technique challenges.** As presented in Figure 2.8, the distribution of the number of SOSE technique challenges is quite similar to that of the other types of SOSE challenges. We observe that publications addressing other types of SOSE challenges often address associated technique challenges. This explains why the two lines in the figure are similar. Nonetheless, this observation is purely based on quantitative aspects; no conclusion can be drawn in this review on the extent to which certain technique challenges be indeed inter-related to which other challenges. We will explore inter-relationships between individual challenges in our future work. Furthermore, we observe that the distribution of the number of SOSE technique challenges is also similar to that of the primary studies, which means the average number of technique challenges proposed in each primary study is similar in each year. In our interpretation, this observation shows that SOSE challenges are commonly recognized in the SOSE research community.

![Figure 2.8: The distribution of the number of technique SOSE challenges versus other types of SOSE challenges](image)

**Product-related SOSE challenges.** Further, we studied the distribution of the number of challenges of artifacts, characteristics and QoS, which are the product-related challenges. The graphic in Figure 2.9 reveals some interesting facts with regard to the product perspective. First of all, no QoS-related challenges were posed in 2003. This is quite reasonable in that
2.4. RESULTS OF THE SYSTEMATIC REVIEW

Functional requirements can be directly implemented while non-functional requirements often rely on other aspects, such as hardware, network, etc. When the service-oriented paradigm just emerged, it is logical that less attention is put on what QoS is demanded from customers, but more on what service-oriented systems should deliver. Second, while the numbers of the other two types of SOSE challenges drop in 2006, the characteristic challenges increased. We believe this is because increasing attention was put on quality and artifacts of service-oriented systems in 2005, researchers were able to elicit their characteristics based on those results. Third, we did not recognize any QoS and characteristics challenges in 2008 until July, which might indicate that the QoS challenges have reached their saturation, and focus is being put on engineering solutions.

![Figure 2.9: The distribution of the number of product-related SOSE challenges](image)

**Process-related SOSE challenges.** In addition to these product-related challenges, we also studied the distribution of SOSE activity and service operation challenges, which are the process-related challenges (see Figure 2.10). We notice that different from the other seven types of SOSE challenges, the number of SOSE activity challenges does not drop in 2006. Instead, it continuously grows and reaches its peak (35 SOSE activity challenges) in 2007, which is also the highest number as opposed to the other six types (excluding the super-type: technique challenges) of SOSE challenges proposed each year. The cause of such a high number, in our opinion, is due
to the fact that a large number of product-related SOSE challenges are known by the year of 2007, indicating a set of requirements for service-oriented systems. Consequently, SOSE activity challenges as one type of process-related challenges are proposed in support of these new requirements. Service operation (e.g., service discovery or service composition), another type of process-related challenges, began to play an important role due to the increasing need to provide integrated solutions, which leads to corresponding challenges.

Figure 2.10: The distribution of the number of process-related SOSE challenges

Business-related challenges and cross-cutting concerns. It is surprising to see that business-related challenges and cross-cutting concerns start to draw more attention in the research community only in recent years (as depicted in Figure 2.11). In our opinion, these challenges should have occurred or been recognized as soon as enterprises started SOA adoption in the beginning of this decade. The reason why business challenges are gaining more attention only recently, in our opinion, is the increasing need for business-IT alignment. As argued by Saugatuck Research (West et al., 2006), SOA adoption happens in three waves, namely project-based, process-based, and program-based. When enterprises just start the journey to service-orientation, the deployment of services is vertical (i.e., departmental-driven or project-based). In this case, often legacy systems are wrapped and delivered as services that are used within a department or project. After these pilot projects become mature, enterprises are ready to extend services horizontally (i.e., across-department or process-based). Even at this stage, there are not too many challenges at the business level in that services are created or wrapped on top of legacy systems. Although cooperations be-
2.4. RESULTS OF THE SYSTEMATIC REVIEW

between departments might occur, these cooperations are still under one business domain and controllable. Until SOA is adopted enterprise-wise (i.e., program-based), which starts recently, the impact of service-orientation on the business level becomes inevitable. Business process models have to be aligned with IT solutions, risk analysis has to be done in the context of SOA, and strategic decisions have to be made on which business partners to choose keeping in mind the extent to which the business partners support SOA, just to give few examples. Furthermore, enterprise-wide SOA adoption requires much more time and financial investment, as well as extra concerns relating to governance and security issues, enterprises face many more new challenges.

Next to business-related challenges, enterprise-wide SOA adoption brings more cross-cutting concerns. When SOA has been progressively adopted, challenges related to training, culture, social and legal issues also emerge. We expect an increasing number of cross-cutting challenges in the future.

![Figure 2.11: The distribution of the number of business-related and cross-cutting SOSE challenges](image)

2.4.4 RQ2.3 - On other ways to classify SOSE challenges

Instead of classifying the elicited SOSE challenges in terms of the topics that they address and the types they reveal, we found that they can also be classified in terms of the origins or the impacts. For instance, some SOSE challenges emerge due to cultural or organizational factors, and some have impact on projects or management. Based on this idea, we could propose a classification scheme like: engineering challenges, project challenges, organizational challenges,
cultural challenges, management challenges, etc. However, due to the fact that our primary studies all come from academic computer science publications, most of the elicited SOSE challenges are engineering challenges and only very few challenges could be classified into the other categories. Therefore, we do not use this classification scheme in this review. It could be used in future work, however, when we will elicit challenges from industrial SOSE practices, too.

2.4.5 The SOSE challenges classified along two dimensions

In our review, we have identified 51 primary studies where 413 SOSE challenges have been proposed spanning from January 2003 to July 2008. We studied all these SOSE challenges in the context of their corresponding primary studies. By applying data synthesis, we have further determined, for each challenge, its type and covering topic.

The SOSE challenges that address the same topic may have different types. For instance, capturing user requirements and simulation are two SOSE challenges that are both about requirements engineering, where the former is a SOSE activity while the latter is a technique. Similarly, the SOSE challenges that are of the same type may address different topics. For instance, data visualization, runtime ranking mechanism, and dynamic security analysis are three technique challenges which address the topics: data, service selection and quality respectively.

All these findings are collectively presented in a cross-referenced table (see Figure 2.6) showing under each topic, how many challenges have been proposed of each type, or vice versa. This table can be very valuable when researchers need to get an overview of the state-of-the-art research challenges on a specific topic or a specific type of SOSE challenges.

2.5 Discussion

2.5.1 Threats to validity

As an empirical research, the validity of the results from the systematic review have to be evaluated. In particular, due to the fact that subjective measurements have been involved in selecting the primary studies, data extraction and data analysis in the course of the review, the credibility of the results could be argued. Thus, in this section, we justify the validity of the results by discussing possible threats that might restrict our ability to interpret extracted data and conclude our findings.

Suggested by Perry et al. (Perry et al., 2000), at least three types of threats should be discussed. This includes a) threats to construct validity, b) threats to internal validity, and c) threats to external validity. Construct validity means to
what extent the inferences can be made correctly (i.e., consistent understanding between study designers and executors). Internal validity focuses on the degree to which the study design allows results following data (i.e., bias is eliminated). External validity refers to how far the results of the study can be generalized (i.e., environmental factors do not have impact on findings). In the following we discuss each of them.

**Construct validity.** In our review, we do not encounter threats to construct validity due to the fact that the review was carried out by the same person who designed the review. Thus, the chance of misinterpretation of theoretical concepts is minimized. As the objective of the review is to explore SOSE challenges, SOSE and challenge are two main concepts in our reviews. In the review protocol, we have explicitly defined SOSE challenges as research issues (challenges) in engineering service-oriented systems (SOSE). Such a definition helps readers to achieve the same interpretations and helps researchers to replicate the review (if needed) in the future.

**Internal validity.** With respect to the process of the systematic review, the main limitation is that the review is mainly conducted by a Ph.D. candidate rather than multiple researchers. Since the decisions that need to be made in selecting primary studies and synthesizing extracted data are quite subjective, the possibility of bias is increased. In order to minimize the chance of bias we have explicitly defined a review protocol beforehand, which includes research questions, search strategies, study selection strategies with precise inclusion and exclusion criteria, as well as a number of data synthesis methods. The review protocol has been carefully examined by three colleagues with experience in empirical software engineering as well as systematic reviews. After the review protocol was finalized, the review was executed by strictly following the protocol. Furthermore, a secondary researcher has conducted a quality assurance check by evaluating randomly selected data set. Nevertheless, with the involvement of colleagues and a secondary researcher, we can minimize but not fully prevent the subjective influence of the conductor of the systematic review.

With respect to the design of the review, we recognize two threats. First of all, to reduce the number of studies that meet the search criteria into a feasible range, we have created a list of related terms tightly related to the two keywords (SOSE and challenge) and restricted the search in abstracts only. Prior to executing the review, we have evaluated the search strategy against a number of studies known to be relevant. The result shows that all these known studies are retrieved by the search strategy. We therefore have the confidence that the search strategy can be used to find other relevant unknown studies. We, however, are not able to prevent the chance that a
publication completely relevant to our review is not selected simply because its abstract does not contain the keywords defined in the search string.

Second, we explicitly excluded publications whose objective is only to present SOSE solutions. Our underlying assumption is that if proposing a challenge is not one of the objectives of a publication, the challenge that might be addressed in the study is a known challenge, which means it should be originally proposed in other publications. However, it is possible that a publication actually does propose new SOSE challenges but this objective is not presented in a clear way. In this case, we might not identify this publication as a primary study and therefore miss the new challenge. To gain more confidence on the completeness of elicited SOSE challenges, we have checked the elicited SOSE challenges against the SOSE challenges that are addressed by the publications where proposing SOSE challenges is not one of their objectives. The results of this check show that we did not overlook any SOSE challenges addressed in those publications.

**External validity.** The scope of our review is restricted purely to the academic domain. We purposely looked for research challenges proposed in the form of scientific papers that are collected by a number of electronic databases which contain only academic research papers. We would assume that SOSE challenges appearing in the academic domain occur in SOSE practice as well since these challenges are simply issues that need to be solved to engineer service-oriented systems, regardless who poses them. However, since SOSE research challenges presented in other forms than scientific papers were not considered in our review, the completeness of the results might be threatened. Nevertheless, our results show that (as explained in § 2.4) the trend of SOSE challenges, with regard to the number of SOSE challenges posed each year on each topic and each type, is in accordance with the waves of SOA adoption indicated by Saugatuck Technology (West et al., 2006).

Evaluating to what extent the results of one review also hold in replicated reviews is one way to address both internal and external validity threats (Shull et al., 2008). With respect to internal validity, bias could be easily detected when different results are concluded in different replicated reviews; and with respect to external validity, whether results of a review hold only under specific conditions can be determined by analyzing underlying conditions of each replicated review.

Due to the feasibility issues, we did not perform replications in our review. However, given the concrete and detailed review protocol, replications can be conducted as soon as practically possible. Other researchers may also replicate the review based on our protocol or carry out an up-to-date review in the future.
2.6. CONCLUSIONS

2.5.2 Quality assessment

Differently from the other systematic reviews in software engineering, ours did not appraise the quality of the primary studies per se. In our review, we did not study the nature of SOSE; rather, we are interested in what has been proposed as SOSE challenges. Therefore, we do not enter the merit of the quality of the primary studies.

2.6 Conclusions

To gain insight into the current status of SOSE research issues, we have conducted a systematic review exploring SOSE challenges. This chapter presents the results of the systematic review as well as the empirical research method we followed.

The main contributions of this work come from an overview of all the SOSE challenges being recognized in the research community and their classification. According to the results of our review, the elicited SOSE challenges can be classified along two dimensions, namely topics and types. In this way, the SOSE challenges assuming different meanings in different studies are clustered, and the SOSE challenges appearing fictively independent are also aggregated. This means, given a specific topic or type (or both), the corresponding inter-dependencies emerge. Consequently, when research efforts are devoted to overcome one of these SOSE challenges, researchers are able to put this challenge in a context of inter-related research issues rather than focus on them in isolation.

In addition to the classification of the SOSE challenges, we also have a number of interesting findings by studying the number of challenges proposed in each year. SOSE emergence laws, one of the most important findings, point out the relationship between expectations, research work on challenges and solutions. We also expect that more business-related challenges and challenges across other domains will emerge in the coming years.

We discussed two unexpected findings. Very few challenges directly addressing service discovery make us realize there are many more challenges indirectly addressing service discovery. Thus further indicates the need, and usefulness of exploring the inter-relationships between the SOSE challenges. Our results also discover that data-related SOSE challenges require more attention in the research community.
Despite the many publications around the innovation and challenges introduced by SOSE, the ‘real’ differences with TSE are still very fuzzy. To better understand the innovative points (if any), in this chapter we identified seven fundamental differences reusing the data collected from the systematic literature review in the previous chapter, linked them to some service-relevant aspects and mapped the differences & service aspects on the well-known architecture-related concepts of ‘architectural concerns’ & ‘architectural viewpoints’. As such, we were able to identify an initial set of requirements for service-oriented viewpoints and views.

3.1 Introduction

There are two reasons that cause the innovative points of SOSE and SOA remain unclear. First, numerous differences between SOSE (resp. SOA) and TSE (resp. SA) have been discussed in the literature. However, some differences referring to the same or similar meanings are described differently (e.g., both highly dynamic environment and high uncertainty can be seen as open-world assumptions (Baresi et al., 2006)); and some differences can be derived from some other ones (e.g., due to open-world assumptions, security becomes more critical (Baresi et al., 2006)). As a result, it seems that there are many differences. However, the fundamental ones are hidden.

Second, innovative points are often considered as issues that are unique to SOSE and as such have never been studied or practiced in the engineering or architecting of traditional software systems. Although numerous SOSE challenges have been proposed both in academia and industry, the lack of common agreements on the novelty of these challenges makes it difficult to determine what are the innovative points (van den Heuvel et al., 2009) (Gu & Lago, 2007).
While our ultimate goal is to understand the ‘real’ differences and their implications on SOSE, in this particular work we focus on gaining insight into the fundamental ones (without judging whether they are innovative) as the first step towards the goal. With this objective, we continued the systematic literature review described in Chapter 2 and linked its results to the service aspects described in detail in Chapter 4 and 10.

We further constructed a framework illustrating the relationship between the fundamental differences and the service aspects. Moreover, we mapped the differences & service aspects on the well-known architecture-related concepts (ISO, 2010) of ‘architectural concerns’ & ‘architectural viewpoints’. By doing so, we were able to identify an initial set of requirements for service-oriented viewpoints. Applying this set of requirements within industrial contexts or SOA project, we would finally be able to highlight service relevant aspects, and therefore exemplify architectural differences (if any), which is one step towards our ultimate goal.

The rest of the chapter is organized as follows. In § 3.2 we discuss our motivation and the research approach we conducted. § 3.3 presents identified differences and their links to service aspects in a framework. In § 3.4 we discuss the needs for service-oriented viewpoints and we conclude the chapter in § 3.5.

3.2 Motivation and Research Approach

It is often argued, in both academia and industry, whether the existing software engineering and architecting approaches (including techniques, methods and tools) are applicable as is in the context of SOA (Papazoglou et al., 2006) (van den Heuvel et al., 2009) (Boerner & Goeken, 2009). However, since their ‘real’ differences with TSE remain fuzzy, it is unclear which existing approaches can be reused or tailored and which ones are completely invalid.

As a result, new approaches and design principles to build service-oriented systems have been continuously emerging in the past 10 years. When analyzing these new approaches and design principles, we start to wonder whether they are specific to services or yet existing concepts with some new (if any) implications.

During our work on the identification of service aspects (reported in Chapter 4 and 10), we realized that by capturing them in service process models and design decisions, we were able to pinpoint the issues that are specifically relevant to SOSE or SOA. However, the way in which we identified the service aspects were based upon studies of the literature and industrial cases as well as experience of authors. A more systematic approach is lacking. In this chapter, we aimed at establishing a sound foundation for identifying any relevant service aspects or pinpointing issues that require specific attention in SOSE and SOA.

An effective approach to identifying differences between service-oriented approaches and traditional ones claimed in the literature is to conduct a systematic
3.2. MOTIVATION AND RESEARCH APPROACH

literature review. This method has been used in our work to explore SOSE-related challenges (in Chapter 2), where a population of 51 scientific papers (published between January 2000 and July 2008) have been selected as primary studies out of a total of 729 analyzed papers. Motivated by the fact that if a primary study discusses SOSE challenges, most likely it also provides arguments on what causes those SOSE challenges. We assume that the SOSE challenges discussed in the primary studies are specific to SOSE and consequently the causes or the drivers that introduce these challenges may highly relate to certain differences between SOSE and TSE. Therefore, we revisited the 51 primary studies and paid particular attention to the context where SOSE challenges are discussed. The steps we took during the review were as follows:

- **Step 1: eliciting differences.** We first conducted a text-based search on the primary studies. The keywords specified in the search query are “tradition” and “differ”. The search results consist of a collection of sentences that contain these two keywords or their variations (e.g., “traditional” or “different”). By studying each of these sentences as well as the citations, we elicited a list of differences claimed in these studies. However, not all of these differences are directly related to the SOSE challenges. To scope down this list of differences and identify only the ones that are related to SOSE challenges, we further studied the section(s) where a certain SOSE challenge is discussed. We assessed the differences identified from each primary study, removed the ones that have no relation to the SOSE challenges, and added newly identified ones that are discussed in the particular sections (when the keywords do not appear in the text).

  The results of this step are a list of differences (causes or drivers) that introduce the SOSE challenges.

- **Step 2: synthesizing results.** Looking at the list of elicited differences, we noticed that there are redundant data in our results. By redundant, we mean data items that have overlapped meanings or can be derived from other data items. With the objective of identifying a set of differences that are exclusive to each other, we first summarized the data items that have similar meanings (e.g., both highly dynamic environment and high uncertainty can be seen as open-world assumptions (Baresi et al., 2006)). Subsequently, we clustered the data items that can be derived from the others (e.g., due to open-world assumptions, security becomes more critical (Baresi et al., 2006)). In doing so, we significantly reduced the number of differences from the initial list to seven fundamental differences.
3.3 The Framework

In our work presented in Chapter 4 and 10 we highlighted the way in which SOSE process modeling and SOA architectural decision making are different from the traditional approaches in terms of some service aspects. Since we argued that these aspects are specific to services and reveal the differences with traditional approaches, we are encouraged to verify whether those service aspects can be linked to the differences identified from the systematic literature review.

To this end, we created a framework illustrating the relation between the service aspects and the identified differences (see Figure 3.1). The boxes in the figure represent the seven fundamental differences and the ellipses represent the service aspects. While continuous arrows denote the relationships between the differences, the dashed arrows indicate the links between differences and service aspects. In this section, we explain all the elements in the framework in detail.

Figure 3.1: A framework of SOA concerns and service-oriented viewpoints.

The identified differences between service-oriented approaches and traditional ones are explained as follows:

- **Services are the building blocks**: instead of focusing on implementing a software system as a whole, SOSE and SOA focus on composing coarse-grained discoverable services acting as building blocks of service-oriented systems.

- **Services are open**: instead of not allowing any architectural changes after deployment, an architecture of a service-oriented system can be changed or even determined at runtime since services become the building blocks that can be composed at runtime.
3.3. THE FRAMEWORK

- **Additional development roles are involved in development**: developer is not the only development role. This is rather split into three essential roles: service consumer, service provider and service broker since services are building blocks that need to be published and consumed.

- **Services are consumed and executed remotely**: instead of buying and installing software locally, users of services (pay and) consume services that are executed remotely at the service provider’s side since services as building blocks are by definition used rather than owned.

- **Services are cross-organizational**: a software is often managed and owned by one organization. Instead, as services are not executed locally at the consumer’s side, the control of services is often highly distributed and crosses the trust and organizational boundaries.

- **SOA is designed under open-world assumptions**: instead of assuming stable execution environment, high uncertainty of the external environment has to be kept into consideration in SOSE and SOA.

- **Services are designed with multiple sets of (non)-functional requirements**: a software application is engineered for a single set of (non)-functional requirements. Instead, since the consumers as well as their needs are not completely known at design time (according to open-world assumptions), services are engineered with multiple sets of (non)-functional requirements to fulfill different groups of potential consumers with different quality requirements.

For the sake of space, only the way in which service aspects address the identified differences is explained here. For further discussion on service aspects, the reader is referred to Chapter 4 and 10.

- **Increased importance of the identification of stakeholders**: The additional stakeholders involved in development increase the importance of capturing them explicitly in process models.

- **Cross-organizational collaboration**: Capturing the way in which cross-organizational collaboration is carried out is crucial to highlight that the additional development roles are distributed in different organizations and services are owned by different business partners.

- **Increased runtime effort**: The high uncertainty resulted from open-world assumption demands for more runtime effort. Further, open services imply that more decisions have to be postponed at runtime. Making runtime effort explicit in a process model may highlight which uncertainty is dealt with and which decisions are postponed.
CHAPTER 3. ON THE DIFFERENCES BETWEEN SOSE AND TSE

• **Different architecture types.** The architectures of services, composite services and service-oriented systems may indicate whether services are the building blocks of a service-oriented system.

• **Temporary provision-consumption relationship.** One of the open-world assumptions is not knowing the service consumers and their provider at design time. Further, services being consumed and executed remotely at the providers’ side means that the provision-consumption relationship ends as soon as the execution of services is finished. Capturing this temporary provision-consumption relationship in making SOA design decisions means making open-world assumptions explicit and highlights the fact that services do not execute locally at service consumers.

• **Different perspectives of stakeholders.** An SOA design decision has different impact on different stakeholders. Explicitly capturing these impact in making SOA design decisions points out the additional stakeholders involved in development.

• **Dealing with heterogeneity.** Open-world assumptions suggest a heterogeneous environment where different data formats, protocols and technologies may coexist. Specifically considering heterogeneity in the SOA design means making open-world assumptions explicit.

In the field of SA, architectural concerns refer to the requirements of different stakeholders; and architectural viewpoints refer to conventions for constructing and using an architectural representation addressing these requirements (ISO, 2010).

Looking at the framework, we noticed that the fundamental differences can be seen as ‘architectural concerns’ or SOA concerns. As such, they resemble the focus or interest of architecting service-oriented systems. For instance, traditional architecting approaches, assuming e.g., the context of an application does not change or change slowly, are not completely valid in SOA design; instead the architecting of SOA focuses on making a set of design decisions so that the architecture in question is able to adapt to highly dynamic environments (Zimmermann et al., 2007a).

Similarly, the service aspects can be seen as ‘architectural viewpoints’ or service-oriented viewpoints. As such, they resemble a set of conventions for capturing these differences from both the engineering and architecture point of view. From the engineering perspective, in Chapter 6 we showed that SOSE methodologies provide better guidance on their application when three process-related service aspects are emphasized in associated process models where elements (e.g., distributed artifacts, activities requiring cross-organizational collaboration) that are missing or less stressed in traditional approaches are highlighted. From the
architecture perspective, in Chapter 10 we discussed the relevance of paying particular attention to four *product-related service aspects* that aid the making of SOA design decisions and showed the related views in two exemplified SOA design scenarios.

### 3.4 The Need For Service-oriented Viewpoints

The advantage of representing the differences & service aspects in a framework and mapping them to architectural concerns & viewpoints is that the SOA concerns become visible, which points out where service-oriented viewpoints are needed.

First, by looking at the concerns it is visible that they can be separated into two groups in principle: development- and consumption-related concerns. Although all the concerns are relevant to both development and consumption, the top four concerns illustrated in the framework are closer to the development point of view whereas the bottom three are described closer to the consumption point of view.

In the framework, there are more development-related concerns and viewpoints than consumption-related ones. Due to the fact that services are used by consumers but executed remotely, service consumers often “participate” in the development process in the sense that they play a critical role in e.g., service negotiation, service discovery and service composition. However, the existing approaches in SOSE and SOA have a much stronger focus on service provision whereas service consumption is paid less attention to. This observation is in line with the review of existing service life cycle models (presented in Chapter 4), which points out that there is a lack of service consumer views in most of the existing models.

Second, the viewpoints developed in previous work already address part of the concerns identified here, which not only confirms the relevance of these viewpoints but also, more importantly, points out the need for more viewpoints.

Looking at the framework, we can see that neither the product nor the process viewpoints established so far address the concern of *multiple sets of (non)-functional requirements* for different groups of service consumers. This indicates a gap between the viewpoints established in our previous work and the SOA concerns.

We can also see that only two SOA concerns (*open-world assumption* and *additional development roles as stakeholders*) are addressed by both process- and product-related viewpoints. Being fundamental to SOSE, all of the SOA concerns in principle should be addressed by service-oriented viewpoints from both process and product perspectives. Therefore, we may conclude that future work on service-oriented viewpoints should explore how to address *services are the*
CHAPTER 3. ON THE DIFFERENCES BETWEEN SOSE AND TSE

Building blocks and services are consumed and executed remotely from the process perspective and services are open from the product perspective.

Furthermore, the service aspects discussed so far are introduced from the perspectives of process and product only. More perspectives (e.g., business) can be studied in the future. As such, the SOA concerns can be addressed by full-scale viewpoints fulfilling needs from multiple perspectives.

To this end, we discussed several needs for SOA and identified an initial set of requirements for service-oriented viewpoints, which consists of highlighting consumption views, addressing the fundamental differences (SOA concerns), and capturing multiple perspectives. If specialized in industrial contexts and with specific SOA projects, we would finally be able to highlight aspects that are specific to services e.g., as discussed in Chapter 6. As such, we could pinpoint as well as rationalize the need for different methods, techniques or tools introduced by services, and therefore exemplify ‘real’ differences, if any. In the remainder of the thesis, we shall investigate (from two perspectives: the SOSE development process and the service-oriented architectural knowledge management) how to identify more service aspects that are relevant to SOSE and how to create viewpoints or views to address these service aspects in order to fulfill the needs discussed in this chapter.

3.5 Conclusions

With the ultimate goal of understanding the innovation of SOSE, we studied the differences between SOSE/SOA and TSE/SA claimed in the literature and linked them to service aspects discussed in our previous work, as the first step towards this goal.

By constructing a framework illustrating the seven fundamental differences identified from the literature review and the service aspects, we noticed an analogy between them and architectural concerns & viewpoints. Such an analogy further led us to observe that visualizing SOA concerns facilitates the identification of the needs for service-oriented viewpoints. This observation guides us to explore more service aspects to address all the fundamental difference in our future work.

‘Innovation’ of SOSE and SOA is often claimed but remains abstract and fuzzy. In practice, what indeed should be done differently in engineering and architecting service-oriented systems is left implicit. Instead of focusing on the ‘innovation’ in terms of new issues that do not exist in TSE, herein we observe another way of interpreting ‘innovation’, which is the extent to which service aspects are capable of addressing the fundamental differences between SOSE/SOA and TSE/SA. In perspective, this work seems to scale down ‘innovation’ of SOSE and
3.5. CONCLUSIONS

SOA, to ‘relevance’ of service-oriented aspects to engineering and architecture, which demands for further discussion in the research community.
Because of new roles and new development tasks introduced in SOSE as opposed to TSE, a new approach to service life cycle management is required. This chapter describes a number of observations on the-state-of-the-art in the field of service life cycle models, and identifies three service aspects of service life cycles: the relevance of cross-organizational collaboration, increased importance of the identification of stakeholders, and the need for increased effort at run-/change time. Observations and aspects are then addressed in the proposed stakeholder-driven service life cycle model. Horizontally, the model shows the activities that are associated with the stakeholders in SOA. Vertically, the model shows the interactions and cooperation between the stakeholders. This model facilitates researchers and practitioners to gain further insight into service-oriented development and governance.

4.1 Introduction

In traditional software development, software project management typically faces a number of problems. Examples are software systems not satisfying user requirements, software systems developed over budget or over deadline, etc. Distributed software development faces additional problems, such as operational complexity, lack of communications between stakeholders; unclear development tasks; arguable responsibility distribution; etc.

SBAs are not only quite often jointly developed by a number of organizations like distributed systems, but they also require cooperation between the SOA stakeholders. SOA stakeholders cover architectural roles including service provider, service consumer and service broker. Furthermore, producing services shared across organizations requires good understanding of the business model
and the relationship between the business partners. As a relatively new and immature development approach, service-oriented development confronts new challenges in addition to those in traditional and distributed software development. Examples of these new challenges include: how to deal with conflicting requirements, how to align the business requirements with the IT solutions, how to distribute services across organizational boundaries in a secure manner, etc.

Because the architectural roles of SOA and new development tasks are introduced in service-oriented development, the service life cycle model in TSE cannot be directly applied. In order to maximally achieve SOA’s promises and reduce disruptions that are caused by conflicts to a minimum, a systematic approach to service-oriented development is required. A concrete service life cycle model is vital in this approach. It can help the researchers to build insight into the service-oriented development processes and it can be used as a preparatory step for SOA governance. SOA governance addresses guidance and control over SOA applications (Pulier & Taylor, 2006). Because the cross-organizational nature of services, secure, reliable and high quality SBAs can be built efficiently and successfully only under the proper governance (Mitra, 2005). This governance has to be carried out through the entire service life cycle.

By studying the-state-of-the-art in the field, we notice that the models proposed by the industrial researchers are in general abstract. Although these models might be successful for their own purposes, for instance commercial, they are not suitable for academia research purposes. To the best of our knowledge, there is no commonly agreed service life cycle model in the literature. In particular, we observe that none of the proposed life cycle models has clear indication on their relationship to the SOA stakeholders and service life cycle stages (design time, runtime, and change time). In § 4.2, we give an overview on our observations of the current service life cycle models.

To propose a life cycle model that is tailored to the development process of SOSE, in this chapter we discuss three service aspects in § 4.3. In § 4.4, we propose a stakeholder-driven service life cycle model in SOA environment, which indicates for each stage the activities, their relationships and their stakeholders. By looking at the model horizontally, we can compare the roles of the stakeholders through their activities, and by looking at the model vertically, we are shown which activities belong to which life cycle stage and the interactions between the stakeholders. § 4.5 concludes this chapter.

### 4.2 Related work

In order to achieve the promise of SOA and deliver high quality service-oriented software products, vendors devote lots of efforts on service life cycle management. We studied life cycle models proposed by the leading industrial vendors in SOA
4.2. RELATED WORK

development, technique report, and academic works that are relevant to service-oriented development. Each of the models is summarized as following:

- In the service life cycle model proposed by Sun (Sun, 2006a), the art of SOA development is nicely generalized by *conception, inception, elaboration, construction* and *transition*.

- The service life cycle model proposed by IBM (McBride, 2007) emphasizes on four iterative processes, namely *model, assemble, deploy* and *manage*.

- The service life cycle model proposed by webMethod (Matsumura, 2007) consists of *plan, design, manage, run, use* and *change*. Each of these processes are associated with a stakeholder. For instance, *plan* is aligned with an *architect* and *design* aligned with a *developer*.

- The service life cycle model proposed by Systinet (Systinet, 2006) separates the life cycle of a service provider from the life cycle of a service consumer. Process *design, build, deploy* and *assure* are the concerns of the service-provider; while process *discover, bind, interact* and *monitor* are the concerns of the service consumer. In (Systinet, 2006), these two life cycles are presented as parallel but distinct.

- In (Wall, 2006b; Wall, 2006a) by Dev2Dev, a service life cycle model is presented with the separation of design time and runtime. The design time processes include *identify business process, service modeling* and *build and compose*. The runtime processes include *publish and provision, integrate and deploy, secure and manage* and *evaluate*.

- A web service life cycle hierarchy model and the service-oriented design and development methodology are proposed in (Papazoglou & van den Heuvel, 2006). The web service life cycle hierarchy separates the life cycle model into logical part and physical part. The logical part consists of the service domain, business processes and business services, while the physical part consists of infrastructure services, component-based service realizations and the operational systems. The service-oriented design and development methodology starts with a preparatory phase which is *planning* and eight main phases that concentrate on business process, namely: *analysis and design, construction and testing, provisioning, deployment, execution and monitoring*.

- A model-driven SOA development process is proposed in (Tsai et al., 2007c). This process illustrates that the SOA development process consists of *requirement, modeling & specification, design, implementation, testing and operation and maintenance*. 
• SOMA (a method for developing service-oriented solutions) (Arsanjani et al., 2008) covers both design-time and runtime governance. It is organized around the following phases: business modeling and transformation, identification & specification, implementation, deployment & monitoring, solution management and realization.

• The S-Cube service life-cycle (S-Cube Partners, 2008) (further discussed in Chapter 5) captures the iterative methodology for continuously developing, implementing, and maintaining services. It consists of two cycles: the development cycle and adaptation cycle. The development cycle addresses the classical development and deployment lifecycle phases, including requirements and design, construction, and operations & management. The adaptation lifecycle has been added to the traditional development cycle to satisfy new requirements and to meet new situations dictated by the environment. It includes three phases: identify adaptation needs; identify adaptation strategies, and enact adaptation.

These proposed models might prove to be useful in their own domains. For instance, the models proposed by the vendors might be more applicable for the marketing or commercial purposes than for the engineering purpose. In general, the works discussed above are not applicable for the researchers to gain better insight into service-oriented development.

Most of the models are made of processes that are at high level granularity. For instance, the model proposed by IBM includes only the most essential processes. Each of these processes can contain a number of sub-processes. However, these sub-processes are left implicit in the model.

Most of the models do not associate the processes with their SOA stakeholders, except for the one proposed by Systinet. However, it ignores that the nature of SOA requires tight cooperation between service consumers and service providers (Boerner & Goeken, 2009). The interaction between these two stakeholders is missing in the life cycle model. Furthermore, when the SOA deployment becomes mature, more stakeholders in the life cycle have to be included. For instance, to increase the reusability of the services and to fully enable service discovery, a service broker is required. This model overlooks this important stakeholder.

Instead of linking the life cycle processes to SOA stakeholders, the model from WebMethods links each process with a particular development role. According to (Zimmermann & Mueller, 2004), where a detail analysis on which roles in the development team are responsible for which service-oriented development tasks is well explained, this model is not complete by only linking one process with one role. For instance, during the plan process, a project manager also plays an important role in addition to an architect; and not only a developer participates...
4.3. THE SERVICE ASPECTS

the design process but also an architect.

Service life cycle management is often divided into three stages, namely design time, runtime and change time (So, 2006). Design time refers to the life cycle of a service before it is available for use. During the runtime stage, services are put into production and the implementations start to work. The change time stage comes after runtime. It focuses on the life cycle of a service when adjustments have to be made when business requirements change. When services or service-oriented systems are put into operation in the real world, there is no guarantee that everything will work as what it is expected to be. Changes have to be made in order to meet the expectation of the user. Moreover, change is inevitable because one of the requests for enterprises to stay competitive is to be dynamic and adaptive (Orriëns & Yang, 2005). Adapting to change is one of the promises of SOA. Therefore, change time has to draw extra attention. By considering the life cycle processes in three stages, a better understanding of service life cycle management can be established. Most of the models do not put the processes into service life cycle stages, except for the one proposed at Dev2Dev. However, it only covers the design time stage and the runtime stage. Due to the dynamic and adaptive nature of SOA, the stage change time is vital for successful service-oriented systems.

The web service life cycle hierarchy model and the model-driven SOA development process model in the two literatures present two different approaches. However, as a service life cycle model, the sequences of the processes in the life cycle have to be indicated in order to explain where the process should start and what can be followed. The lack of sequences between the processes does not help to build a good understanding on the service life cycle management.

A summary of the comparison of the studied models is shown in Table 4.1.

4.3 The Service aspects

The fundamental change in developing service-oriented systems as opposed to traditional software systems is that software is delivered as a service. As such, users pay for and use services instead of buying and owning software. Consequently, users do not have the control of services, which are owned and controlled by service providers instead. These changes are reflected by three service aspects identified in this chapter.

Service aspects are issues that are specifically relevant to SOSE. These aspects reveal the core distinctions between the service-oriented paradigm and the traditional ones (e.g., component-based paradigm). Accordingly, the implication of service aspects should be explicitly expressed in SOSE process models. For instance, due to the dynamic nature of service-oriented systems, service artifacts (e.g., service specifications, service level agreements) are often generated on the
Table 4.1: A comparison of the service life cycle models

<table>
<thead>
<tr>
<th>Model</th>
<th>the level of granularity</th>
<th>associate to the SOA stakeholders</th>
<th>indication on the SOA life cycle stages</th>
<th>sequences between processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>Very abstract</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IBM</td>
<td>Very high</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>WebMethod</td>
<td>High</td>
<td>Link to development roles, but incomplete</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Systinet</td>
<td>High</td>
<td>Yes, but missing service broker</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Dev2Dev</td>
<td>High</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Web service life cycle hierarchy</td>
<td>Low</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Model-driven SOA development process</td>
<td>Low</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Service-oriented design and development methodology</td>
<td>Good</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SOMA</td>
<td>High</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>S-Cube</td>
<td>High</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

fly and used dynamically, whereas artifacts in TSE (e.g., requirements specifications) are produced in a more static way, often within one single organization. Furthermore, service artifacts like service specifications are no longer limited to local use; rather, they can be published, discovered and reused across various SOSE projects (Tsai et al., 2007b) and activities scattered across multiple enterprises. As a result, SOSE pays particular attention to the way loosely related activities contribute to cross-organizational collaboration. Therefore, we argue that cross-organizational collaboration should be specifically expressed in SOSE process models to improve the guidance of applying SOSE methodologies. In this section, we specifically discuss three service aspects and stress the necessity of expressing them in a service life cycle model.

4.3.1 The relevance of cross-organizational collaboration

The focus of SOSE is shifted from applications to services that are collaboratively developed by multiple SOA roles (Tsai et al., 2007c; Colombo et al., 2005), such as service consumer, service provider, service broker. During the development process, activities like specification&modeling, design, implementation, testing, operation and maintenance are all required to be performed in a collaborative manner (Boerner & Goeken, 2009).

For instance, service-oriented systems are built through discovering and com-
4.3. THE SERVICE ASPECTS

posing existing services from multiple service providers rather than coding as in TSE. Consequently, the processes of discovering, selecting, composing services require continuous interaction (or collaboration) between the participating roles through the service development life cycle. Hence, collaboration between participating roles becomes explicit and critical in that it enters the details of a SOSE process that is now scattered across multiple roles. This makes their relationship tighter but also demanding clearer governance and agreements.

What makes it more critical is that these roles are often distributed in multiple departments or organizations. In this case, interactions between development activities associated with multiple business roles demand for collaboration between multiple organizations. We call this type of collaboration, which crosses the boundaries of the domain of each role, cross-organizational collaboration.

In TSE, and especially in concurrent software development, component-based software development, or outsourcing, cross-organizational collaboration also occurs when one organization delegates a set of tasks to the other organization(s). The main difference is that in TSE the first organization only concerns how software is developed internally, but not how delegated tasks are carried out externally. Consequently, only the results of the delegated tasks are of importance to the development process of the first organization. As a result, the cross-organizational collaboration is a purely buying (outsourcer) and selling (supplier) relationship, and its details are hidden from the perspective of the development process of the first organization. In SOSE, instead, the collaborative roles coexist in a service-oriented system rather than having an active-passive relationship.

When collaboration crosses the boundaries of each organization, barriers (e.g., conceptual, technological barriers, and organizational barriers) to enterprise interoperability often obstruct the effectiveness of collaboration. Since collaboration between multiple roles becomes part of the SOSE process, it is of great importance to highlight this collaboration in a SOSE process model. When the collaboration becomes explicit and clear, the need for corresponding agreements or contracts becomes evident. Consequently, appropriate governance can be applied. As such, barriers to enterprise interoperability can be reduced.

4.3.2 Increased importance of the identification of stakeholders

A stakeholder can be defined as “any group or individual who can affect or is affected by the achievement of the organization’s objectives” (Freeman, 1984). In the field of software engineering, the identification of stakeholders and how to address their concerns has been of significance in the areas of requirements engineering and architectural design (Bass et al., 2003; Tarr et al., 1999). In this thesis, we have taken the notion of stakeholder and applied it to service-oriented
software engineering (SOSE) with the aim of identifying groups or individuals who can influence or who are influenced by the engineering of SBAs.

Due to the shift away from monolithic application development to service provision in SBAs, TSE methodologies (Shan & Hua, 2007; Kontogiannis et al., 2007) no longer meet the requirements for developing these service processes. The reasons for this are that the engineering, composition, continuous adaptation and consumption of services naturally involve more stakeholders. These stakeholders may take different perspectives (a service aspect discussed in § 10.2.4), such as a service provider or a service consumer. Moreover, these stakeholders may have different responsibilities as compared to those in TSE:

In SOSE, business functions are delivered in the form of both complete SBAs (analogous to software systems and applications developed using TSE) and pools of reusable software services. As a result, some TSE stakeholders such as a software designer, software architect and software developer are often split into two types of stakeholders, one focusing on services and another focusing on SBAs (e.g., service designer and SBA designer, respectively). Moreover, some types of stakeholders are specific to SOSE and are not found in TSE; for example, service modelers, service monitors or adaptation designers are only found in SOSE.

In addition, since cross-organizational collaboration (a service aspect discussed above) becomes more critical in SOSE, the importance of clearly identifying stakeholders increases accordingly. If stakeholders are identified at a too coarse granularity, the represented interaction remains not fully specified. This leads to unclear responsibilities among collaborating enterprises and thus decreases in trust and possibly in success. Because the level of details matters, the identification of stakeholders directly determines the level of detail expressed in a SOSE process model.

The decision on whether a role should be identified as a separate stakeholder in a SOSE process model depends on what type of interactions the model intends to represent. For instance, if a SOSE development approach intends to emphasize or elaborate on how service monitoring is provided, accordingly, service monitor could be selected as a stakeholder in a SOSE process model. As such, it offers the possibility to explicitly associate monitoring-related activities with the service monitor and to explicitly describe the interaction between the service monitor and other stakeholders. Of course, if service monitoring is not the main focus, then it is not necessary to select it as a separate stakeholder since the detailed interaction between a service monitor and the other stakeholders is not of interest.

As a general rule, if service monitoring (or any other activity) is performed by an independent third party, the corresponding role should better be identified as a separate stakeholder because it stands for an external organization. As such, it offers the possibility to explicitly express the responsibilities across different business domains.
In summary, identifying stakeholders with the appropriate level of detail in a SOSE development approach facilitates the establishment of a corresponding SOSE process model describing associated activities and their interactions at an appropriate level of abstraction.

### 4.3.3 The need for increased effort at run-/change time

In TSE, the main goal is to develop high quality applications that meet the requirements of the end users. Consequently, most of the effort is dedicated to design (collecting requirements, design, and implementation) and change time (maintenance). Runtime activities are hardly addressed if not in specific domains. Furthermore, change time activities are often performed off line (either with or without execution interruption).

Different from TSE, the main goal of SOSE is not only to deliver high quality but also agile and robust services which are able to meet the ever-changing business requirements. Consequently, much more development effort is shifting from design time to run-/change time. For instance, components identification is often performed at design time in TSE; the SOSE equivalent activity is service discovery, which is encouraged to be performed at runtime and it is regarded as one of the major challenges in the SOSE field.

Many existing SOSE process models do fail in emphasizing this shift. By explicitly modeling the two stages, a process model can visualize the amount of activities shifted to run-/change time, hence providing useful inputs to resource allocation.

### 4.4 The proposed life cycle model

Based on the observations of the current publications on service life cycle models, we intend to build a concrete service life cycle model. To avoid including too general service processes in the model, we take the granularity of the processes as the same level as the service-oriented design and development methodology that is proposed in (Papazoglou & van den Heuvel, 2006). In the rest of the chapter, we name the modules in the model as activities to differentiate those from other existing models.

One of the design principles of SOA is to decouple the role of service provider and service consumer (Artus, 2006). In this way, the influence between these two roles is minimized. For instance, suppose a service provider decides to change the operation system that a service currently runs on, service consumers should not be required to change correspondingly. Service provider, service consumer and service broker are three architectural roles (stakeholders) in SOA. In this
CHAPTER 4. A STAKEHOLDER-DRIVEN SERVICE LIFE CYCLE MODEL FOR SOA

Figure 4.1: A stakeholder-driven service life cycle model

Service provider
- Market scan
- Requirements engineering
- Business modeling
- Service design
- Service development
- Service testing
- Service publishing
- Service provision
- Service monitoring
- Service management

Service broker
- Registry selection
- Registry Update

(Service Consumer)

Application Provider
- Requirements engineering
- Application design
- Application implementation
- Module testing
- Service integration
- Application testing
- Application maintenance

Design time
- Service invocation
- Service orchestration/composition
- Service negotiation
- Service invocation
- Service monitoring

Runtime
- Registry selection
- Registry maintenance

Change time
4.4. THE PROPOSED LIFE CYCLE MODEL

model, we specifically present the service life cycle activities associated to these stakeholders.

Since services cannot be directly used by the end users (except for composing other services), it is usually required to have an application to provide user interfaces. When an application builder integrates a service (services) into an application, it also acts as a service consumer. In this model, we make a distinction between service-specific activities and application-specific activities by associating the former with a service consumer and the latter with an application builder. The main motivation for this is again decoupling the role to avoid dependencies. Another advantage of separating the role of the application builder and service consumer is that when a service provider produces composed services either entirely or partially by existing services, its role switches between service provider and service consumer depending on whether it is performing service discovery or service publishing. Without the separation, this switch of roles cannot be easily captured.

Although in some cases, the roles of these stakeholders can be taken by one or multiple departments or organizations, we regard these stakeholders as roles instead of physical development teams. For instance, an IT department of a company, which is responsible for building a SBA, can be both a service consumer and an application builder. However, we consider it as a service consumer when it invokes services and we consider it as an application builder when it integrates the discovered service into an application.

Moreover, being aware of which activities are carried out at which stage (design time, runtime, or change time) is of great importance for the development team. Runtime activity is in general the most critical period because the software product is available for use and any problem occurred during this period is real-time problem. Change time activity is also very important because these activities ensure adaptations. Design time activities are relatively less impending yet crucial. Any problems occurred in this period usually allow more time to deal with comparing to runtime and change time.

As depicted in Figure 4.1, the proposed model consists of three rows and three columns. The rows are partitioned by the stakeholders: Service provider, Service broker and Application builder (Service consumer). Each row comprises service life cycle activities associated with each stakeholder. The columns are segmented by Design time, Runtime and Change time. Each column comprises service life cycle activities across the stakeholders. Each of these activities is linked if the output of an activity is the input of another activity. This interaction indicates an iterative and incremental process. This means two linked activities often executed iteratively until the result is satisfactory. The horizontal links imply the interactions within a single stakeholder; the vertical links imply the interactions across the stakeholders; while the diagonal link implies the interactions
between Service consumer and Application builder. When an application builder integrates services into an application, these diagonal links can be treated as horizontal links.

This proposed life cycle model is designed to be used in a mature SOA development environment. Currently, lots of SOA applications are developed simply by wrapping existing components by service interfaces (Papazoglou et al., 2006). Although they are also advocated as SOA applications, we consider it as fake SOA applications since they do not bring main SOA characteristics, such as dynamic and adaptive, into play. The proposed model can be well adapted when SBAs are developed across organizational borders; when applications are outsourced to other IT companies; when service providers intend to deliver composed services, etc. Depending on the role of the stakeholder that a develop participant plays, this model can be applied accordingly.

In the following sections, the life cycle activities are explained in terms of design time, runtime and change time for the selected stakeholders: service provider, application builder (service consumer) and service broker.

4.4.1 Service provider

A service provider refers to the role of a development party that produces and publishes services which are ready to be executed. Service providers are the owners of services. They are responsible for implementing services as well as maintaining services.

Design time

Market scan In general, a service provider starts to produce services either under the request of a client or initiated by ideas extracted from experiences or market requests. In case of the former scenario, the service life cycle for the service provider starts with requirements engineering; while in case of the latter scenario, the service life cycle should start with a market scan. The objective of this activity is to investigate the market demands and orient the service production. A good market scan prevents producing services that either have no beneficial usage or are redundant. Inspirations from the market scan help the service provider to decide which kind of service to produce.

Requirements engineering After the initial ideas of the services are collected or requested, requirements have to be analyzed specifically. Service providers are concerned with how to achieve the reusability of the services so that they can gain investment returns as soon as possible, how to maintain services and compete with the other service providers, and so on. Unlike the traditional software producers, service providers produce services that remain with service providers instead of
4.4. THE PROPOSED LIFE CYCLE MODEL

delivering to software clients. Therefore, during the requirements engineering activity, service providers not only need to analyze the objective, functionality, interface and quality of services, but also need to consider the issues relating to their accessibility, for instance, by making service level agreements, defining policies, etc.

**Business modeling** After the requirements engineering is ready, service providers have a clear goal about what services they are going to produce and what are their target markets. The next step is to model the business process. During this phase, service providers have to capture all the requirements gathered in the requirements engineering activity and model business processes into low-level processes which can be defined without going into the technical details. Usually business analysts participate only in this activity. Since services are actually units of business functionalities, a good business modeling is therefore crucial to guarantee that services can be well implemented by IT developers without deep knowledge on business processes.

**Service design** As soon as the business models are developed, designers or architects start to design services. During this activity, they search the service registry to identify reusable service, they determine whether an existing service is applicable, they decide whether to implement a new service or modify an existing service to meet the requirements. The objective of this activity is to make a specific design of the services which complies with all the functional and non-functional requirements that are concluded from previous activities. The outputs of this activity include refined service interface, interaction style between services and their clients such as asynchronous or synchronous, invocation method such as RPC or document-based, etc.

**Service development** When the service design is ready, the service development team starts to implement new services or to modify the existing services identified during the service design activity. The development team not only has to perform the actual coding tasks, but also has to perform service testing against all the requirements defined in previous activities. A number of collaborate services might be needed in order to accomplish the test. In this case, simulation of the services might be used to perform the test. After a number of iteration of coding, testing, integration, simulation, and mapping requirements, services are ready to be handed over to a testing team.

**Service testing** After services are developed, they have to be tested before they are published for commercial usage. The purposes of testing include not only fault detection but also quality control. Reducing the errors to a minimum
is of course the essential requirement for testing; testing the services against the quality metrics defined in the service design is more significant for a service provider to gain competitive advantage against other providers that deliver similar services.

Runtime

Service publishing  Once services are developed and tested, they are ready to be published to a service registry. The objective of this activity is to make produced services available so that the service consumer can find and use them when needed.

Service provision  Although services are published, they are not yet ready to be executed. In order to guarantee that service providers are able to charge their clients for what they offer or to ensure security, usually services are only accessible by authorized users. A service level agreement (SLA), an agreement between a service provider and a service consumer regarding the promise or assurance of a service, is often used as a contract. SLAs define how well services are delivered in terms of cost, availability, performance, etc, by the service provider. The service consumer consequently needs to agree on the expectations of services as defined in SLAs and its obligations on using the services. The objective of service provision is to define rules for service consumers to invoke the published services, for instance, to specify different SLAs for different clients (if necessary). Only when SLAs are signed, or equivalent access control is performed, a service consumer is authorized to use the services deployed by the service provider.

Service monitoring  Once services are published for execution as commercial products, monitoring services to ensure SLAs or policies are met is crucial for service providers. Failing to meet those rules could cause severe financial consequences. Without proper service monitoring, service providers are not able to keep track of the behavior of the published services. By monitoring response time, quite often service providers are able to conclude the availability, reliability, and performance of these services.

Change time

Service management  In the business world, the reality shows that there is no way to avoid change. Especially when e-commerce is introduced as a new approach to perform business transactions, changes occur quickly than ever. One of the benefits of SOA is to have the capability to improve business agility through a service-oriented IT solution. Services have to be changed in order to fulfill the ever-changing business requirements. Quite often, at this point, the service
4.4. THE PROPOSED LIFE CYCLE MODEL

life cycle has to go back to the beginning of the cycle, which is requirements engineering.

The main role of service management is to manage the services so that when changes occur, the impact of change on the clients can be reduced to a minimum. In a SBA, not only services can be changed but also the associated policies or rules can be changed. For instance, a policy of the conditions under which the consumers are entitled to access the services is changed, a number of services that apply this policy might meet severe problems. Service management has to ensure that any change occurs in SOA environment is handled so that the consumers of the services can continue to use the services.

4.4.2 Service broker

A service broker acts as an intermediary role between a service provider and a service consumer. The main role of a service broker is to provide service location information which is contained in a service registry. A service registry acts as a directory for published services, such as the yellow pages for phone numbers. Service providers use the registry to publish their services and service consumers use it to look up services. Even though so far brokers are associated with registries, they might play a more advanced role, e.g., providing themselves brokering services.

Design time

Registry selection  Registries become more and more important when the number of services increases. It provides a central location for tracking and managing services. Without a service registry, the reusability of services has to be limited because services are hard to be shared across organizations. Moreover, a service registry can also help service providers avoid wasting time to develop services that already exist. Currently, there are two standard registry solutions, one is web service registry (OASIS, 2002), which is focused on service description registrations; another is ebXML Registry (OASIS, 2005), which not only supports service description but also other SOA metadata. Other extended registry based on these two standards include Sun’s Service Registry (Sun, 2006b), BEA AquaLogic Service Registry (BEA, 2007), IBM WebSphere Service Registry and Repository (Dudley et al., 2007), etc. As a service broker, it is important to select a proper service registry tool to manage the service registry.

Runtime

Registry update  Since a service registry contains the service descriptions and all the location information of published services, it has to be updated when
there are new services to be published. If we consider a service registry as a
database, services can be discovered only if their entry are created and updated
with correct information. When a service consumer discovers a service, the service
broker returns the address of the service. In this way, the service consumer can
use this information to invoke the service later on.

Change time

Registry maintenance  When more and more services contained in the service
registry expose to the public, the maintenance of a service registry is of great
importance. A good registry maintenance ensures that an update of an already
in use service does not break the invocation from existing service consumers,
the services that are never been discovered or hardly been invoked should be
considered to be removed from the registry, etc. Ideally, the service registry can
also maintain a rating function to the services so that it enhances the efficiency
of service discovery. For instance, by collecting the feedbacks from the service
consumers, the registry is able to give scores to the services that provide similar
functionalities. These scores can facilitate the service consumers to decide on the
best service to choose and also ensure a healthy market competition.

4.4.3 Application builder (service consumer)

An application builder integrates services into an application which eventually
fulfill the requirements of the end user. An application builder is also called a
service consumer when it tries to find services in the service registry and execute
the services. Although these two roles are often tightly coupled, we purposely
make a distinction between the service consumer and the application builder in
the proposed model. As shown in Figure 4.1, a service consumer focuses on the
service-level development such as service discovery, service negotiation, etc; while
a service provider concentrates on the application-level development including
integrating services to the application.

Design time

Requirements engineering  Because service consumers and service providers
have different concerns in their own business domains, the requirements are obvi-
ously in different directions. For instance, service consumers target at accomplish
a particular task through the usage of a service. Concerns of service consumers
include whether the functionalities matches the requirements, whether the per-
formance of the service meets the demands, whether the usage of the service can
improve the business efficiency, and so on.
4.4. THE PROPOSED LIFE CYCLE MODEL

**Application design, implementation and module testing** Application design, implementation and module testing are not too much different as the approach in TSE. Although the services that a system is going to invoke are still unknown, interface descriptions can be used for future integration. This approach is quite similar with component-based development.

**Runtime**

**Service discovery** After the requirements are defined, service consumers can start to look up published service candidates which comply with the requirements in service registries. Service discovery is a key activity in the whole SOA environment. Since services are self-described, they can be discovered either manually or dynamically by means of matching the requirements with service interfaces description or service contracts (Vitvar *et al.*, 2007).

**Service orchestration/composition** After the application is implemented, the next step is to integrate services to complete the functionality of the application. Service orchestration happens when services are discovered or composed and can be successfully invoked. In this activity, the application builder maps each discovered services or composed services to business activities. All these services should be able to cooperate each other and they have to be integrated into an application. In other words, service orchestration guarantees that services in an application can be well assembled to work together to fulfill business requirements.

**Service negotiation** As soon as an application has decided which discovered services to consume, it needs invoke the services for execution. Services are not always free to be invoked. Typically, the non-commercial services that are not required to guarantee the QoS or internal services that do not across organization boundary do not really require the service negotiation activity. In the business world, economically supported services can be a success only if the service providers can benefit from what they offer. Therefore, access control policies are usually made by service providers during the service provision activity. During the service negotiation activity, a service consumer exchanges a number of contract messages with a service provider in order to reach an agreement (Elfatatry & Layzell, 2004). The result of the negotiation might be a SLA.

**Service invocation** As soon as a service, which can fulfill the user requirements, is discovered, the next step is to invoke the service for execution. Services can be invoked either statically or dynamically (Vinoski, 2003). Each of the ser-
CHAPTER 4. A STAKEHOLDER-DRIVEN SERVICE LIFE CYCLE MODEL FOR SOA

services that are discovered and composed in the earlier activities has to be invoked for execution.

**Application testing** Application testing faces new challenges as opposed to TSE. Integration and non-functional requirements such as performance, compatibility, security, etc, are the main focuses in this activity. It has to ensure that the application not only fulfills all the requirements defined in previous activities, but also the integrated services perform as expected.

**Service monitoring** Service monitoring is not only essential for service providers, but also for service consumers. By monitoring services that are paid for use, service consumer are able to be aware of the QoS and compare it with SLAs or policies. In case of under performance, penalty policies can be applied to guarantee a fair business.

**Change time**

**Application maintenance** In addition to the maintenance tasks defined in TSE, this activity focuses especially on the changes that might occur. If the changes come from the end user requirements, an iterative procedure might be required to start with the requirements engineering activity. If the changes come from the integrated services (e.g., a better service is available to replace an existing one, or an integrated service is updated), the application maintenance activity has to ensure that those changes can be incorporated in the application.

4.5 Conclusions

A good service life cycle model not only can facilitate the life cycle management of SBAs but also can enhance their governance. Based on the observations of the related work and the identification of the three service aspects, this chapter proposes a stakeholder-driven service life cycle model for SOA. This model refines the service life cycle activities that are commonly discussed in the literature. It extends the existing models by associating the activities with their stakeholders and service life cycle stages in an explicit way. It also clearly indicates the interactions between and across the stakeholders.

The proposed model brings three advantages for the management of service-oriented development. First, the separation of the activities associated with the stakeholders makes the duties for each stakeholder clear and explicit. The management team can focus better if they analyze activities per stakeholder. Second, the indication of activities belonging to the three life cycle stages helps in clarifying priorities to the actions required. Third, the relationships between the
4.5. CONCLUSIONS

activities provide a clear overview on the order in which these activities should be carried out.
5

A taxonomy of SOSE stakeholder types

Stakeholder support is critical to the success of any project, but it becomes much more important in SOA-related projects. Traditional software development methodologies no longer meet the requirements for developing SBAs, due to the shift away from monolithic application development to service provision and composition. This shift introduces many more types of stakeholders, each of which can take multiple roles within the lifecycle of the SBA, and who have an interest in or are influenced by the service-oriented software process.

To understand these stakeholder types and roles, this chapter presents an initial set of stakeholder types and roles we solicited from within the S-Cube NoE. By describing these stakeholder types in the context of S-Cube service engineering lifecycle, we demonstrate the lifecycle phases each stakeholder and role is involved in during the development and operation of SBAs. The stakeholder roles and types found and the methodology we describe for discovering them will aid the identification of the requirements for these stakeholders and contribute to future research in service engineering methodologies.

5.1 Introduction

Despite of the increased importance of the identification of stakeholders (a service aspect discussed in § 4.3.2), currently there is no common understanding of these stakeholder types and roles and their relationship to the service engineering lifecycle. In this chapter, the term stakeholder type defines the specific function played by a stakeholder in SOSE; the term stakeholder role defines the perspective taken by a stakeholder in SOSE.

To help find these stakeholders, determine their role(s) and show where they are involved in the service engineering lifecycle, we used the collective knowledge and experience of the EC’s S-Cube NoE (Papazoglou & Pohl, 2009), a four-year project with the aim of establishing an integrated, multidisciplinary and vibrant
CHAPTER 5. A TAXONOMY OF SOSE STAKEHOLDER TYPES

research community in the field of SOSE. S-Cube counts 16 academic partners, an industrial advisory board with 6 industrial partners and 11 industrial and academic associate members. In total the network contains approximately 200 researchers or experts in various SOSE-related fields.

Since S-Cube’s participants work on different areas of software services and systems research and development, part of S-Cube’s work is to integrate their research through the development of a methodology for the design, implementation and operation of SBAs. To ensure that methodology meets the needs of the community, this integration requires the identification of stakeholders who are involved in SOSE. This study of stakeholder types and roles are, therefore, important not only to the SOSE community for the reasons given above but also to assist the S-Cube NoE as it seeks to tailor and promote its service engineering methodology to the correct audience.

This chapter provides an initial taxonomy of these stakeholder types and roles together with a description of the research method used to collect and derive this information. The initial taxonomy contains nineteen stakeholder types and five roles we solicited from institutions within S-Cube. To demonstrate when each stakeholder and role is involved during the design, development and operation of an SBA we provide a context for our results by mapping them to S-Cube service engineering lifecycle. Due to the nature, size and relevance of the information gathered, we feel these findings are important to the community, though it should be emphasized that this chapter contains only one set of results (from the S-Cube NoE) and is not a comprehensive field study containing responses from many types of organizations. It is hoped the initial set of stakeholder roles and types found will aid the future identification of the requirements for these stakeholders and contribute to the continuing research in service engineering methodologies.

The remainder of this chapter is structured as follows: § 5.2 describes the S-Cube service engineering lifecycle we use to position and describe the stakeholders found; § 5.3 describes the four-step research method we used to determine the set of stakeholders; § 5.4 describes the taxonomy of service engineering stakeholders we found using the research method; § 5.5 shows how the taxonomy maps to the S-Cube service engineering lifecycle; § 5.6 discusses some related work in the identification of SOA stakeholders; § 5.7 provides our conclusion.

5.2 The S-Cube service engineering lifecycle

As part of their research the S-Cube NoE has developed a model to describe the lifecycle of SBAs, shown in Figure 5.1, which captures the iterative methodology for continuously developing, implementing, and maintaining services. The development (right-hand) cycle addresses the classical development and deployment lifecycle phases, including requirements and design, construction and operations
and management. The entry-point to the lifecycle (the point where the SBA is conceived) is through the Early Requirements Engineering phase. The second (left-hand) cycle extends the classical lifecycle by explicitly defining phases addressing changes and adaptations. The adaptation lifecycle has been added to the traditional development cycle because, as described earlier, one of the distinguishing features SBAs is that they are modified as time progresses in order to satisfy new requirements and to meet new situations dictated by the environment. The adaptation lifecycle can be broken into three phases: deciding if an adaptation to the SBA is needed (Identify Adaptation Needs); deciding how the system should be adapted (Identify Adaptation Strategies); and performing the adaptation (Enact Adaptation).

Note that other service engineering lifecycles, including SOMA (Arsanjani et al., 2008), SOAD (Zimmermann et al., 2004a) and ASTRO (Marconi et al., 2008), were considered by S-Cube as a foundation for their engineering lifecycle. However, as (Di Nitto & (Editor), 2008) documents, other lifecycles cannot adequately explain the evolution and adaptation required by SBAs and is why a new lifecycle was created. In this work, we use the lifecycle to determine each stakeholder’s relationship to the service engineering lifecycle and provide another view of the connections between stakeholders.

![Figure 5.1: The S-Cube Service Engineering Lifecycle](image)

In summary, the S-Cube service lifecycle allows the continuous detection of new problems, changes and requirements for adaptation, the identification of possible adaptation strategies and the enactment of adaptation strategies. Once SBAs (or parts thereof) have been adapted, they will be re-deployed and re-provisioned and put into operation. Further details on the S-Cube service engineering lifecycle can be found in (S-Cube Partners, 2008).
5.3 Research method

As described above, part of the work of S-Cube is to develop a methodology for the design, implementation and operation of SBAs. Our motivation for identifying stakeholders involved in the service engineering process was to provide this methodology with an initial set of stakeholders involved in the design, execution and consumption of SBAs. To do this, we applied state-of-the-art research methods: a questionnaire survey (Pfleeger & Kitchenham, 2001) for step 1, content analysis (Miles & Huberman, 1994) for steps 2 & 3, and a case-specific mapping study (Kitchenham et al., 2010) for step 4. Due to space limitations this section provides an overview of how we used these methods to collect, collate and compile the stakeholder specifications gathered from researchers working within the S-Cube NoE.¹

**Step 1: Soliciting stakeholder types.** Since each S-Cube partner works on one or more service engineering research topics (e.g., service composition, service engineering, service design, etc.) to find which stakeholders are relevant for S-Cube’s engineering methodology we designed a questionnaire that allowed S-Cube researchers to describe types of stakeholder, record which activities the stakeholder participates in, how the stakeholder relates to the S-Cube engineering lifecycle and why this stakeholder is particular to service engineering.

We asked researchers working in S-Cube to use the questionnaire to identify service engineering stakeholders from their own work and the literature and to provide justifications for the identification of those stakeholders. We feel the data we collected is highly representative of the SOSE research community and their responses are highly relevant for other researchers, educators and industrial practitioners. In total 50 completed questionnaires were returned back to us.

**Step 2: Classification of the stakeholder types.** The information in each completed questionnaire was used to classify the stakeholders based on the roles they take. The motivation for classification is to find which roles are played by different types of stakeholders and which types of stakeholders have multiple roles.

Our classification schema was initially based on the four roles described in Chapter 4: **Service Provider**, **Service Broker**, **Service Consumer**, and **Application Builder**. Note that within these roles an **Application Builder** can be a **Service Consumer** when locating services (via a service registry) and executing services. When analyzing the data collected in Step 1 of the method, we identified another type of service consumer, the **Service Composer**. This role is different to the...
5.3. RESEARCH METHOD

Application Builder (who builds applications using services) as a Service Composer builds (or composes) composite services and provides them to other consumers. Since building applications requires different skills, expertise and knowledge to building services (e.g., the former activity collects requirements from the end-user while the latter collects requirements from its potential consumers) we explicitly differentiate between service composers an application builders. As a result, we split the role of Service Consumer (discussed in Chapter 4) into two specific consumer roles: Application Builder and Service Composer.

When categorizing the stakeholder types, we noticed that some types cannot be classified using the roles introduced above. These types were either end-users (the initiators of SBA interactions) or experts that provide support during the service engineering lifecycle. As a result, we identified two more roles, the former as Application Clients and the latter as Supporting Roles in order to classify these types.

Finally, in this step we also found that none of the stakeholder types collected from the questionnaires acts as a Service Broker, an intermediary role between a service provider and service consumer that provides service location information from a service registry. We discuss this finding in more detail in § 5.4.

The result of this step is the classification of stakeholder types according to the following five roles:

1. **Service Providers**: design, implement, own, publish and maintain services that can be invoked/executed.

2. **Application Builders**: integrate services into an application that meets the requirements of its end users.

3. **Service Composers**: provide composite services for internal or external usage and compose existing services or SBAs to achieve business goals.

4. **Application Clients**: use the application to achieve their goals.

5. **Supporting Roles**: are indirectly involved with the service lifecycle and provide auxiliary functions such as project management, technical advice and consultancy.

**Step 3: Consolidation of stakeholder types.** From the stakeholder questionnaires completed by the S-cube partners, we noticed that stakeholders with similar descriptions were often named differently. For example, in the questionnaires returned to us the stakeholder responsible for modeling the business process was called Business Modeler, Business Analyst and Domain Expert by different partners. To consolidate the types of stakeholders gathered and to obtain a list of unique types, in this step of the method we re-named the types of stakeholders when they had the same meaning but were named differently.
To do this we first determined the context and function for each classified stakeholder. For instance, the stakeholder Business Process Architect has the context Business Process and the function Architect. By analyzing the context and function of all stakeholders we identified synonyms that were used to rename stakeholders. For example, the Analyst function was treated as a synonym for Modeler, Designer was considered a synonym for Architect, Enterprise was judged a synonym for Business and SBA was used as a synonym for System, End-point Service-Based System and Service Network.

We then compared the descriptions of stakeholders with the synonyms. If their descriptions were the same or very similar, the stakeholders types were given the same name. After renaming, we obtained the list of unique types of stakeholders (described in detail in the next section).

**Step 4: Map stakeholder types to S-Cube lifecycle phases.** In the previous step, stakeholder types with similar descriptions were given the same name. However, these stakeholders may participate in different phases of the service engineering lifecycle, therefore to provide information on where the consolidated stakeholder types mapped to S-Cube service engineering lifecycle, we inspected the lifecycles of each of the submitted stakeholder types and for each type recorded the lifecycle phase(s) they apply to. If multiple specifications define the same stakeholder type but assert they belonged to different phases of the S-Cube lifecycle, we merge and record the coverage of the relevant S-Cube lifecycle phases for the consolidated stakeholder type. Using this method we obtained the consolidated list of the different types of stakeholders and how they participate in the S-Cube lifecycle, § 5.5.

### 5.4 A taxonomy of service engineering stakeholders

Table 5.4 shows the results of Steps 1-3 of the methodology. Taking the columns from left-to-right, the first column lists the stakeholder types found in the questionnaires returned by the S-Cube NoE participants (Step 1). Note the same type may appear more than once — this is because two partners may have returned questionnaires with the same name but not necessarily the same description. All responses received are included for completeness.

The second column of Table 5.4 lists the roles defined in Step 2 of the method and shows how multiple stakeholder types are categorized. The third and final column of Table 5.4 shows the consolidated stakeholder types found through carrying out Step 3 of the method. This column shows how similarly named stakeholder types can be consolidated using the synonyms described above.

When looking at the results in the table, the categories with the largest number of stakeholder types are Application Builders and Service Providers. With respect to the Application Builder role, although this role has the primary task of
## 5.4. A TAXONOMY OF SERVICE ENGINEERING STAKEHOLDERS

<table>
<thead>
<tr>
<th>Stakeholder Type</th>
<th>Role</th>
<th>Consolidated Stakeholder Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Architect</td>
<td>SBA Architect</td>
<td>Creates structure and design of the SBA.</td>
<td></td>
</tr>
<tr>
<td>Application Designer</td>
<td>SBA Modeler</td>
<td>Makes hypothetical descriptions of SBAs, possibly for use in testing.</td>
<td></td>
</tr>
<tr>
<td>Business Process Engineer</td>
<td>Business Analyst</td>
<td>Analyzes operations to develop a solution to its problems.</td>
<td></td>
</tr>
<tr>
<td>SBA Assembler</td>
<td>Business Process Analyst</td>
<td>Controls the configuration and operation of a process or organization.</td>
<td></td>
</tr>
<tr>
<td>SBA Developer</td>
<td>Business Process Administrator</td>
<td>Administers the operations of a process or organization.</td>
<td></td>
</tr>
<tr>
<td>SBA System Builder</td>
<td>Domain Experts</td>
<td>Subject matter expert, or person with special knowledge or skills in a particular area.</td>
<td></td>
</tr>
<tr>
<td>System Analyst</td>
<td>SBA Engineer</td>
<td>Uses their scientific knowledge to design SBAs.</td>
<td></td>
</tr>
<tr>
<td>Service Network Modeler</td>
<td>End User</td>
<td>The ultimate user for which the SBA is intended.</td>
<td></td>
</tr>
<tr>
<td>Business Process Architect</td>
<td>Application Designer</td>
<td>Concerned with facets of the application development process.</td>
<td></td>
</tr>
<tr>
<td>Application Developer</td>
<td>Business Analyst</td>
<td>Analyzes operations to develop a solution to its problems.</td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>Business Process Modeler</td>
<td>Works out the ‘form’ of the service.</td>
<td></td>
</tr>
<tr>
<td>Application Builder</td>
<td>Application Designer</td>
<td>Combines services and/or SBAs into more complex ones.</td>
<td></td>
</tr>
<tr>
<td>Developers</td>
<td>Change Manager</td>
<td>Controls resources and expenditures.</td>
<td></td>
</tr>
<tr>
<td>System Analyst</td>
<td>Change Manager</td>
<td>Controls resources and expenditures.</td>
<td></td>
</tr>
<tr>
<td>Business Analyst</td>
<td>Change Manager</td>
<td>Controls resources and expenditures.</td>
<td></td>
</tr>
<tr>
<td>Business Process Engineer</td>
<td>Change Manager</td>
<td>Controls resources and expenditures.</td>
<td></td>
</tr>
<tr>
<td>End User</td>
<td>Change Manager</td>
<td>Controls resources and expenditures.</td>
<td></td>
</tr>
<tr>
<td>Direct User</td>
<td>Change Manager</td>
<td>Controls resources and expenditures.</td>
<td></td>
</tr>
<tr>
<td>Indirect User</td>
<td>Change Manager</td>
<td>Controls resources and expenditures.</td>
<td></td>
</tr>
<tr>
<td>Service Consumer</td>
<td>Change Manager</td>
<td>Controls resources and expenditures.</td>
<td></td>
</tr>
<tr>
<td>Service Consumer</td>
<td>Change Manager</td>
<td>Controls resources and expenditures.</td>
<td></td>
</tr>
<tr>
<td>Service Consumer</td>
<td>Change Manager</td>
<td>Controls resources and expenditures.</td>
<td></td>
</tr>
<tr>
<td>Composition Designer</td>
<td>Composition Designer</td>
<td>Combines services and/or SBAs into more complex ones.</td>
<td></td>
</tr>
<tr>
<td>Service Designer</td>
<td>Service Designer</td>
<td>Works out the ‘form’ of the service.</td>
<td></td>
</tr>
<tr>
<td>Service Developer</td>
<td>Service Developer</td>
<td>Involved with the creation or improvement of services.</td>
<td></td>
</tr>
<tr>
<td>Service Developer</td>
<td>Service Developer</td>
<td>Involved with the creation or improvement of services.</td>
<td></td>
</tr>
<tr>
<td>Service Developer</td>
<td>Service Designer</td>
<td>Works out the ‘form’ of the service.</td>
<td></td>
</tr>
<tr>
<td>Service Provider</td>
<td>Service Provider</td>
<td>Supplies one or more services.</td>
<td></td>
</tr>
<tr>
<td>Service Provider</td>
<td>Service Provider</td>
<td>Supplies one or more services.</td>
<td></td>
</tr>
<tr>
<td>Project Manager</td>
<td>Supporting Expert</td>
<td>Provides assistance in a particular area.</td>
<td></td>
</tr>
<tr>
<td>Technology Consultants &amp; Suppliers</td>
<td>Supporting Roles</td>
<td>Provides assistance in a particular area.</td>
<td></td>
</tr>
<tr>
<td>Lawyers &amp; Data Privacy Officers</td>
<td>Supporting Roles</td>
<td>Provides assistance in a particular area.</td>
<td></td>
</tr>
<tr>
<td>Domain Experts</td>
<td>Supporting Roles</td>
<td>Provides assistance in a particular area.</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Classified, sorted and consolidated stakeholders
building SBAs with services it also has the functions of a service provider and/or consumer depending on the service realization strategy. For example, (Papazoglou, 2003) describes four strategies for realizing services: 1) in-house service design and implementation, 2) purchasing/leasing/paying for services, 3) outsourcing service design and implementation, and 4) using wrappers and/or adapters. In the first and fourth strategies the Application Builder functions as a Service Provider as it implements and publishes service for consumption. When taking the second and third strategies, the Application Builder acts as a Service Consumer that uses third-party services. Due to the broad scope of the Application Builder role — from design and implementation of services to integrating services to SBAs — it is natural that many types of stakeholders are required to perform this role.

The second role most present in our findings was the Service Provider, and we identified 5 types of stakeholder that perform this role. This is because in the engineering of SBAs it is not the design and development of integrated applications that are of most importance but on the engineering of services that can be reused by many, unknown service consumers. As a result, much of the research of the S-Cube NoE is focused on how to design, deploy and publish services to service consumers, all of which are a concern of the Service Provider.

As mentioned in § 5.2, what we were not expecting is the non-identification of a stakeholder playing the role of a Service Broker. The benefit of a Service Broker is that they facilitate the discovery of (possibly third-party) services through a consumer sending a description of the requested service to the Broker, who selects a service that best fits the requirement and returns this information to the client (Massuthe et al., 2005). There is a great deal of work on how service discovery can be realized, e.g., by matching between requests and services (Klusch et al., 2006) or through context awareness (Lee & Helal, 2003). However, in our experience, that work concentrates on service discovery mechanisms but not on their operation or use with service brokers. This observation is also in line with our survey on existing SOSE methodologies (presented in Chapter 7): only two of the twelve methodologies we analyzed describe service discovery as an explicit part of the engineering process, demonstrating how service discovery and brokering are underrepresented in SOSE research.

5.5 Coverage of the S-Cube service engineering lifecycle

In Step 4 of our method we mapped the consolidated stakeholder types to the phases of the S-Cube service engineering lifecycle. The results of the mapping are shown in Table 5.2. In this section we discuss the coverage of the S-Cube
5.5. COVERAGE OF THE S-CUBE SERVICE ENGINEERING LIFECYCLE

5.5.1 Coverage of lifecycle phases by the stakeholder types

With reference to Table 5.2, we can see each phase of the lifecycle is covered but some more completely than others. The last row of the table shows the coverage of each lifecycle phase as a percentage relative to the total number of consolidated stakeholder types. Reviewing the figures we can see the lifecycle phase of Early Requirements Engineering has the greatest amount of coverage with respect to stakeholder types. This is understandable as ensuring the service or SBA being developed or built is useful to its intended end-users requires the input and experience of many stakeholder types, ranging from Service Architect to the eventual End Users. Only the Negotiation Agent, Service Deployer and Service Provider types, responsible for the provision of services, are not involved in collecting requirements at an early stage of the lifecycle.

It is interesting that the phase of Identify Adaptation Needs is covered by 63% of the stakeholder types identified in this work; dynamically adapting SBAs to meet changing requirements through the identification of adaptation requirements raised by stakeholders involved in the execution of SBAs, or generated by the technological environment in which the system is running is complicated as the adaptation requirements may be conflicting (Lane et al., 2010). As more than half of the stakeholder types are involved in this phase this result shows the adaptation of SBAs is well covered by the research carried out within the S-Cube.
NoE.

The lifecycle phases of Construction, Enact Adaptation and Identify Adaptation Strategies have relatively less coverage (less than 50%). This can be interpreted as fewer stakeholder types are required here or that insufficient research has been done in these phases. We feel that these phases may involve fewer types of stakeholders because implementing SBAs without the design, provisioning or enacting a adaptation strategy requires relatively fewer stakeholder types to complete, which is what we would expect from intuition.

However, for the phase of Identify Adaptation Strategies, we expected more stakeholder types. In this phase, the ways through which the adaptation requirements are satisfied are designed and an adaptation strategy is chosen based on the specific adaptation needs. Among the stakeholder types derived, we did not see any stakeholder types that were specifically dedicated to these activities. From our survey of adaptation-related activities of sixteen service-oriented approaches (Lane et al., 2010), we found identifying adaptation strategies and selecting the most suitable strategy are generally not well supported and this finding reinforces this point.

5.5.2 The participation of each type of stakeholders

From Table 5.2 we can see how certain stakeholder types feature more prominently than others. More specifically, the SBA Architect, Service Architect and Manager types are omnipresent throughout the entire lifecycle, whilst Service Designer and SBA Engineer are present only in a few phases. It is understandable that the Manager may be involved throughout the whole lifecycle, taking care of management-related issues such as cost control and customer relationship management. We are not totally convinced the SBA Architect should be involved in all of the lifecycle phases, however, as an SBA Architect is responsible for the creation of structure and design of individual services or SBAs and should be active in the early stages of the lifecycle. This is because when a service or SBA has been implemented and deployed the operation and maintenance tasks should be dedicated to other stakeholders, such as the service monitor or adaptation designer, and should not be part of the SBA Architect’s responsibilities.

5.5.3 Comparison between the evolution and adaptation cycles

When considering the differences between the stakeholders involved in left-hand and right-hand side of Table 5.2 (i.e., at the differences between the stakeholders involved in adaptation and evolution phases of the S-Cube service engineering lifecycle), we can see that in the adaptation side of the S-Cube lifecycle a much
5.6. RELATED WORK

A broader range of roles are involved than in the evolution phases. For example, it appears roles such as Application Clients and Service Composers are more present in the adaptation phase than in the evolution phase. Conversely, we find that roles such as Application Builders are represented to a greater extent in the evolution phases rather than the adaptation phases. We think this demonstrates how the evolution phase is focused much more on the engineering and design and implementation processes rather than the adaptation lifecycle which requires the input of the SBA’s end-users, such as Application Clients and Service Consumers.

5.6 Related work

The need for determining SOSE stakeholders has been recognized and initial research has been performed into their identification. Kajko-Mattsson et al. (Kajko-Mattsson et al., 2007) proposed a framework with an initial set of IT stakeholders adapted for the development, maintenance and evolution of SBAs. Within the framework, 22 stakeholders are suggested and these are categorized into four groups: namely SOA support, SOA strategy and governance, SOA design and quality management and SOA development and evolution. Kajko-Mattsson’s framework has a different focus to our work however, as it specifically takes into account business ownership and focuses on new concerns introduced by SOA (e.g., the understanding of the individual services and their cooperation with other services within a combined business process). As a result, many stakeholders (7 out of 22) are related to business, such as Business Process Support Engineer, Business Process Assistant, Business Process Architect, etc. Since we did not focus specifically on business-related tasks and responsibilities, these business-specific stakeholder types can be considered as complementary to our taxonomy.

Zimmermann & Mueller (Zimmermann & Mueller, 2004) propose a model outlining roles in a Web Services development project. In their model, the roles are divided into three categories, existing roles are stakeholders identified by TSE, extended roles are stakeholders that take additional Web services-related responsibilities and extra roles are stakeholders that are specifically responsible for Web services-related tasks. The advantage of this model is that existing TSE stakeholders are explicitly differentiated from SOSE stakeholders. In our taxonomy, this difference is not obvious, although the description of each stakeholder and its participation in the service life cycle helps to make such a distinction. The identification of service specific aspects of the stakeholders in our taxonomy is planned as a piece of future work.

In the service delivery framework proposed by SAP research (SAP, 2009), five stakeholders have been identified; the service provider, service hoster, service

\(^2\)Note that in the literature stakeholders are often referred to as actors or roles.
CHAPTER 5. A TAXONOMY OF SOSE STAKEHOLDER TYPES

gateway, service aggregator and service broker. However, SAP’s framework just focuses on stakeholders involved in service provision, whilst our taxonomy covers the whole service engineering lifecycle.

The most important difference between the work introduced above and the work presented here is that none of the related works describe how the stakeholders they contain were identified. Although Kajko-Mattsson et al. mention that they take traditional stakeholders as a starting point and use SOSE literature for reference, a systematic approach and methodology is missing. In this chapter we have not only identified an initial set of SOSE stakeholders but also have described the structured approach we used to solicit, classify and consolidate the stakeholders found in the SOSE activities.

Another difference between the works above and ours is in how the stakeholders have been classified: for instance, in the framework developed by Kajko-Mattsson et al., the stakeholders are classified by their function with the implication that these stakeholders need to collaborate more often and solve problems together; in Zimmermann & Mueller’s model stakeholders are classified by their innovativeness, indicating that specific skills are required for performing certain roles. Our work contributes a new perspective by classifying stakeholders according to their roles played in SOSE. In service engineering it is not uncommon for these roles (e.g., service provider, service consumer, etc.) to be distributed across multiple organizations. As a result, when engineering activities are performed collaboratively across organizational boundaries, additional concerns such as trust, governance and regulatory compliance must be considered. Understanding the roles a SOSE stakeholder plays in this process aids the management of these issues.

Last but not least, in our work the identified stakeholders are associated to SOSE activities using the S-Cube service engineering lifecycle to provide them with a structure and context. This is different from the related work we reviewed, which only identifies stakeholder responsibilities and skills and does not explain at which stage of the engineering lifecycle the stakeholders should participate. The lack of an explicit link between stakeholders and SOSE engineering activities significantly reduces the applicability of SOSE methodologies, which was observed in our evaluation of a number of well-known SOSE methodologies (in Chapter 7). The mapping of the identified stakeholders to SOSE engineering activities provides an understanding of the stages where each stakeholder is expected to participate. This view is beneficial to the project management and governance of an SOSE project and is a contribution to the SOSE research community as it provides a precise instrument to identify the competencies required for specific service engineering lifecycle activities.
5.7 Conclusions

The development of SBAs requires more types of stakeholders than traditional software engineering and stakeholders may take multiple roles during the lifecycle of an SBA. In this paper we have reported our research to find information about these stakeholders from institutions within the S-Cube NoE and presented a taxonomy of nineteen stakeholder types performing five roles and a mapping between these stakeholders and the S-Cube service engineering lifecycle. However, despite the S-Cube NoE being the largest network of excellence in-the-field and hence containing broad SOSE know-how, we are aware that S-Cube does not represent the entire SOSE community. In particular, industrial practitioners have not directly participated in this survey; further investigation especially in industry is needed to validate this taxonomy and ensure generalization and completeness.

From the results it is clear that more types of stakeholders participate in the lifecycle of SBAs than in that of traditional applications and that stakeholders may play different roles depending on their focus on e.g., engineering services, SBAs, service provision or consumption. Moreover, we observed that: some types of stakeholders (such as Service Architects) are required in all of the lifecycle phases; there is a lack of stakeholders specifically dedicated to adaptation; and an absence of stakeholders playing the role of service broker. These observations will provide input for future research into these stakeholders that will concentrate on developing and tailoring service engineering methodologies for them.
This chapter presents a process model of the SeCSE methodology. SeCSE is a large European project carried out in 2004-2008. By highlighting the three service aspects discussed in Chapter 4, this chapter shows how each aspect provides guidance for engineering SBAs in practice. In this way, we are able to exemplify innovative points of SOSE by capturing these three service aspects.

6.1 Introduction

Many SOSE methodologies have been proposed in both academia and industry aiming at providing approaches, methods and (sometimes) tools for researchers and practitioners to engineer SBAs (see for instance (Papazoglou & van den Heuvel, 2006)). However, without being fully understood, a methodology is less valuable no matter how perfect it is. This is particularly relevant to SOSE methodologies as they are more complex than TSE ones, having to deal with new challenges while keeping the principles of TSE. The additional complexity results mainly from open world assumptions, co-existence of many stakeholders with conflicting requirements and the demand of adaptable systems (Baresi et al., 2006).

To improve the understandability of a methodology and its guidance, software development process models (describing what activities a development process consists of and how they should be performed) have been often used since they visualize the development process proposed by the methodology. The SOSE community has realized that traditional software process modeling techniques are no longer directly applicable or adaptable in SOSE (Blake, 2007). To overcome the mismatch between traditional software process models and SOSE, a number of service life cycle models have been proposed by both industry and academia.
CHAPTER 6. GUIDING THE SOSE PROCESS: THE IMPORTANCE OF SERVICE ASPECTS

(e.g., (McBride, 2007; Blake, 2007; Papazoglou & van den Heuvel, 2006)). However, none of the proposed models has either reached a sufficient level of maturity or been able to fully express the aspects that are peculiar to SOSE. Besides different names on the phases and on the stakeholders, one might wonder what the real difference between these models and many well defined and experimented TSE approaches is.

In Chapter 4 we identified three service aspects that are crucial to the SOSE development process, namely a) the relevance of cross-organizational collaboration, b) the importance of the identification of stakeholders, and c) the need for more effort at run- and change time. We also defined a stakeholder-driven approach that illustrates such service aspects in a SOSE process model developed from the literature.

In this chapter, we stress the necessity of expressing them in a SOSE process model. With the aim of validating the service aspects and their relevance to SOSE, we modeled the methodology developed and used in the SeCSE\textsuperscript{1} European project. This has allowed us to: 1) build on concrete examples that emphasize the relevance of the service aspects, and 2) show the applicability of the approach in practice. The results of this case study show that, by emphasizing service aspects in a SOSE process model, attention is naturally brought to those parts of the process model that are different as compared to TSE. The benefit is that guidance for applying a certain SOSE methodology is improved, and better service engineering management strategies can be put in place.

This work does not intend to propose a set of particular graphical notations for the purpose of modeling the SOSE development process. Rather, we intend to highlight what should be expressed in a SOSE process model. The graphical patterns (and associated notations) used in this chapter illustrate one possible way of describing the service aspects expressively in a concrete SOSE process model.

The remainder of the chapter is organized as follows. In § 6.2, we present the case study that we carried out and highlight the relevance of the three service aspects to the SeCSE methodology. Related work on the topic of SOSE process models is discussed in § 6.3. Finally, § 6.4 concludes the chapter.

6.2 Applying service aspects to a concrete methodology

With the aim of gaining insight in the extent to which the modeling of the three service aspects improve the guidance of a SOSE development approach,
we modeled them in a concrete and practical context, i.e. the SeCSE methodology (Di Penta et al., 2009).

In this section, we first introduce the SeCSE methodology and present its process model (with the three service aspects being highlighted), followed by a discussion of how each service aspect both emphasizes the characteristics of the SeCSE methodology itself and facilitates better SOSE guidance.

6.2.1 The SeCSE methodology

The SeCSE project is a EU-funded project. Its goal was to investigate methods, techniques and tools to develop and manage SBAs in an effective way. A large number of academic and industrial partners have been collaborating in this project. As a consequence, the resulting SeCSE methodology has both theoretical and practical value.

The SeCSE methodology describes the main development activities and tools that have been adopted by the SeCSE project. It provides ways to create service compositions where component services are discovered at runtime either on the basis of the context of usage or when a certain service fails. This focus on runtime is one step forward towards the third generation SBAs (Olsson (Editor), 2006).

Although service discovery is regarded as one of the major activities in developing SBAs, and even though techniques already exist to support service discovery, in practice service discovery is hardly adopted. Nowadays, most enterprises focus on migrating legacy systems to SBAs and implementing new services rather than discovering services from a registry (as SBAs are supposed to do). In the SeCSE project, service discovery is not only addressed by the SeCSE methodology, but also experimented in the consortium. As an advanced and relatively mature approach, the SeCSE methodology is a good candidate to be selected as the case study in this work to analyze the service aspects addressed by it.

Moreover, the design of the SeCSE methodology does not specifically or consciously take our three service aspects into consideration. This provides us the possibility to take the SeCSE methodology as such and model the service aspects addressed by the methodology. Comparing to the original process model illustrated in the documentation of the SeCSE methodology, the SeCSE process model proposed in this work provides better guidance for its users.

6.2.2 The SOSE process model for the SeCSE methodology

By focusing on service aspects, our main objective is to discuss what has to be modeled rather than how to model. For illustration purposes, we use BPMN\(^2\) as process modeling notation to be used to communicate a methodology to its

\(^2\)www.omg.org/spec/BPMN/
users. This is expressive enough to represent the various inter-dependencies and multiple stakeholders involved in the SeCSE development process.

For the purpose of modeling the service aspects of the SeCSE methodology, we illustrated the SeCSE development process by means of a process model. The decisions and assumptions that we have made to construct the SeCSE process model were verified with the SeCSE experts to check for correspondence of the model to the methodology. This analysis has also helped to elicit information that were missing or left implicit in the methodology document. After the verification from the SeCSE experts, we were able to refine and finalize the model.

The resulting model is given in Figure 6.1. In this model, we specifically modeled the stakeholders identified in the SeCSE methodology to highlight the increased importance of the identification of stakeholders; we specifically modeled their associated activities and inter-dependencies to highlight the cross-organizational collaboration. The increased effort at run-/change time becomes obvious in the model since we separated them from the design time effort.

For each service aspect we first explain its associated graphical pattern in a SOSE process model in general; and then we discuss how the SeCSE methodology addresses the service aspect by observing the SeCSE process model against the graphical pattern.

The relevance of cross-organizational collaboration

Figure 6.2 graphically illustrates the cross-organizational collaboration (COC) service aspect. The left-hand side of the figure shows three collaboration types (COC patterns): peer activities group, main-sub activities and distributed activities. These patterns are exemplified in the right-hand side of the figure.

- The **peer activities group** models same activities carried out in parallel across multiple partner enterprises. For instance, service negotiation can be carried out by a service provider and a service consumer with the common objective of reaching the agreement of service provision and consumption.

- The **main-sub activities** model the same activities carried out partially by one partner enterprise and completed by another. For instance, a service consumer may perform the user-centric part of a requirement engineering activity that the service provider is mainly responsible for.

- The **distributed activities** model inter-dependent activities carried out across multiple partner enterprises. For instance, design time service discovery can be carried out by a service developer and composition design can be carried out by a system builder. The former provides input (such as

---

3Due to the limited space, the design of the case study itself is not described in detail.
6.2. APPLYING SERVICE ASPECTS TO A CONCRETE

METHODOLOGY

Figure 6.1: The SOSE process model for the SeCSE methodology.
CHAPTER 6. GUIDING THE SOSE PROCESS: THE IMPORTANCE OF SERVICE ASPECTS

Figure 6.2: Service aspect: cross-organizational collaboration (COC).

discovered candidate services) to the latter; the latter might also provide feedback to the former when different design decisions are taken, possibly requiring different service candidates.

By observing the SeCSE process model (shown in Figure 6.1) against the three COC patterns defined in Figure 6.2, the attention is brought to specific types of cross-organizational collaboration.

- **Peer activities group**: Service negotiation occurs twice in the SeCSE development process. One is carried out by a service provider and a service consumer; another is carried out by a system builder and a service provider. By nature, each service negotiation must be performed in parallel by both its stakeholders as peers. The results of the collaborations (indicated by the data objects attached to the peers) are SLAs. Different from TSE where contracts are often established after software is built, SLAs in SOSE often precede final service products (service composition in the case of the SeCSE methodology). These SLAs are also potentially useful to other activities such as service monitoring.

- **Main-sub activities**: Service centric architecture and composition design are carried out by a system builder and service provider in a cooperative manner. A system builder has the main responsibility for this activity, whereas a service provider focuses only on a subset of its tasks. For instance, the service provider might work on the definition of the list of possible candidate services to be used at runtime; while the system builder is responsible for the overall service composition design. In this way, the subtasks that the service provider takes are of competence of the system builder.

  Service specifications are modeled as three activities with related service specification as data objects. They are carried out by a service developer, system builder and service provider independently but on related artifacts.
6.2. APPLYING SERVICE ASPECTS TO A CONCRETE METHODOLOGY

In general, a service developer creates service specifications for a component service, which influences a composite service carried out by a system builder. The system builder has to make sure that the QoS characteristics defined in the specification of the component services are compatible with those of the composite service. When a service composition or a single service is deployed, the service provider may add information to the corresponding specification known at deployment time.

- **Distributed activities**: Service centric architecture and composition design is carried out by a system builder at design time and **binding and re-binding** is carried out by a service provider at run-/change time. Cross-organizational collaboration occurs when new substituting services are discovered at run time (e.g., due to a new requirement) and service composition needs to update its bindings to accommodate the change.

Only when the collaboration is explicitly captured, the stakeholders of SBAs can gain insight on the impact between their own responsibilities and the others’. Each stakeholder has a clearer view on at what time ("when") which activity ("what") has to be carried out in cooperation with which stakeholder ("who") and in which manner ("how"). In the SOSE development process, external enterprises often continuously play important roles throughout the service life cycle. By looking for the cross-organizational collaboration patterns in a service process model, enterprises are brought to focus on the points needing strategic business agreements that should regulate such tight collaboration.

**Increased importance of the identification of stakeholders**

Figure 6.3 graphically represents the stakeholders pattern (and an example), which makes the stakeholders in a SOSE process model explicit. By observing the SeCSE process model (shown in Figure 6.1) against this stakeholder pattern, we can see that the SeCSE methodology involves mainly four stakeholders, namely: **service developer**, **system builder**, **service provider** and **service consumer**. These stakeholders, potentially representing partner enterprises, play common SOA roles from the perspective of service implementation, integration provision, and consumption.

Explicitly modeling the identified stakeholders improves the guidance of the SeCSE methodology as follows. First, by placing the SOSE activities in the corresponding swimlanes, the SeCSE process model naturally shows the responsibilities and collaborations of and among stakeholders. This is especially crucial in SOSE where cross-organizational collaboration occurs in almost all activities. In this way, the business dependencies requiring contractual/SLA agreements are made explicit, and project managers can better plan the allocation of develop-
CHAPTER 6. GUIDING THE SOSE PROCESS: THE IMPORTANCE OF SERVICE ASPECTS

Figure 6.3: Service aspect: increased importance of the identification of stakeholders.

ment activities based on the skills and responsibilities of the internal and external stakeholders.

Second, service composition centered characteristic of the SeCSE methodology is well captured by the SeCSE process model when identified stakeholders are explicitly modeled. Figure 6.1 shows that most of the development activities are associated to the system builder and the service provider (stakeholders that carry out service composition activities). Furthermore, the model shows that the system builder and the service provider are tightly linked; the service developer and the service consumer are instead loosely linked. Due to focus of service composition, the service consumer in the SeCSE process model is considered as the consumer of composite services, rather than the consumer of component services. Therefore, the service consumer does not have direct interaction with the service developer, and the system builder must cooperate with the service provider in multiple activities. In this way, the SeCSE process model very well captures the fact that service composition is the main focus of the SeCSE methodology and consequently provides better guidance in that the stakeholders are able to gain better understanding of the focus of the (SeCSE) methodology.

The need for increased effort at run-/change time

Our approach of separating the design and run-/change time activities in a SOSE process model is presented in Fig 6.4, where the left-hand side of the figure shows the 2-stages pattern, exemplified in the right-hand side of the figure. In this example, it is visually evident that service design is carried out at design time; while service discovery is performed at run-/change time.

By observing the SeCSE process model (shown in Figure 6.1) against the 2-stages pattern defined in Figure 6.4, we are now able to easily distinguish the design time activities (falling in the left-hand side of the figure) from the run-/change time activities (in the shadowed area at the right-hand side of the figure). Consequently, the guidance for applying the SeCSE methodology is improved in that the process model shows its support for adaptation, service composition and facilitates critical project plan decisions.
6.2. APPLYING SERVICE ASPECTS TO A CONCRETE METHODOLOGY

First, we notice that about one third of the development effort is dedicated to run-/change time activities. In particular, runtime service discovery and service negotiation are supported by the SeCSE methodology with the objective of increasing the adaptability and agility of resulting systems to meet on-the-fly requirements. Thereby, related activities such as runtime service monitoring, recovery management, and binding and re-binding are also in place.

Second, we notice that the development effort dedicated to run-/change time activities is not evenly distributed among the stakeholders in the SeCSE methodology. Instead, the service provider carries out most of the run-/change time activities; while the system builder and service developer do not perform run-/change time activities at all. We have discussed in § 6.2.2 that the roles of system builder and service provider are extensively developed due to the service composition centered approach. The process model illustrates and emphasizes further the separation of design and run-/change time activities: the system builder focuses on the design of service compositions; while the service provider focuses on the provision of service compositions.

Third, knowing which activities are executed at which stage is also crucial in SOSE. Project managers should be able to adjust project plans based on the criticality of the activities since runtime activities are directly related to executing services real-time and therefore more critical than design time activities. For instance, as shown in Figure 6.1 service negotiation is supported by the SeCSE methodology at both design time and runtime. The difference is that at design time, service negotiation occurs between a system builder and a service provider for component services that are selected for service composition; at runtime it occurs between a service provider and a service consumer for a composite service that fulfills business requirements. This difference results in different levels of business commitment. At design time, the failure of reaching service level agreements or the failure of collaborating with a service provider does not have huge business impact on the system builder; the system builder can always decide to look for an alternative service. However, at runtime if the composite service fails to execute or does not reach the quality it promises, the system builder faces risks to loose its customer and even its business market. Here, the business commitment is much higher than at design time. Being aware of this difference, a
project manager is able to decide which actions to take for activities with various levels of criticality.

Summary

As auspicated, highlighting the service aspects in the SeCSE process model allows those who have to exploit the methodology to have more clear evidence of issues (cooperation between organizations, numerous stakeholders, activities to be executed during the operation phase) that are critical from the managerial point of view. By observing the service aspects captured in the model, we may conclude (and becomes more evident) that some differences between SOSE and TSE become obvious in the applied SeCSE context. First, the fact that each of the four identified stakeholders is responsible for a common SOA role (centered on services) reflects the shifted focus from applications to service pools. Second, due to the fact that consumers of services do not have the control of them, more interactions between stakeholders (cross-organizational collaboration) occur, as shown by the many arrows crossing the various swimlanes. Third, around one third of activities are carried out at run-/change time, which shows that resulting systems are dynamic and therefore have the potential ability to adapt to ever-changing requirements.

6.3 Related work

It has been gradually recognized that traditional software process models are no longer sufficient to model the SOSE development process. To overcome the mismatch between traditional process models and the SOSE development process, a number of SOSE-specific process models have been proposed by both industry and academia. However, as we already discussed in Chapter 4, these assume that the development of SBAs is entirely internal to an organization. For instance, the model proposed by IBM in (McBride, 2007) describes four phases that are implicitly assumed to be executed by IBM itself or any organization adopting the IBM methodology. This results in the fact that the actual difference between SOSE and TSE methodologies remain unclear. In this chapter, we argue instead that interactions across the organizational boundaries require in SOSE particular attention, and should be made explicit.

While the approaches found in the literature are proposals of specific methodologies and lifecycles, our approach can be seen as a way to interpret and investigate different existing lifecycles. This has a value per se as it does not force people to adopt a specific approach for developing and managing SBAs, but it helps them to understand and make coherent all the methods they use in their common practice. In a similar line of research, Blake in (Blake, 2007) advocates the
6.4. CONCLUSIONS

distinction between two key activities: service development and service-centric system management. While the first activity follows a quite traditional iterative process, service-centric system management is seen as much more dynamic of the traditional processes associated with the development and the operation of other kinds of software. In particular, as in SeCSE, runtime (sub)activities such as re-binding are identified. The importance of stakeholders is stressed, and a number of them is identified and assigned to the various activities of the lifecycle. We differentiate from this work as we highlight not only stakeholders and their activities, but also the interaction between these stakeholders and the artifacts they produce and exchange. As we have argued in the previous sections, in fact, clarifying these aspects help all roles involved in the lifecycle in better understanding the critical aspects of the lifecycle and in properly drive it toward the achievement of the project goal.

Bell (Bell, 2008) proposes the structure of a SOSE process model, which consists of timeline, events, seasons and disciplines. As we do, he uses timelines to indicate a sequence of development activities. Design and run-/change time activities in our approach correspond to seasons in Bells structure, while our development activities can be regarded as disciplines in Bells structure. The approach is also focusing on defining and classifying those events that have an impact on the lifecycle. While a differentiation between runtime and design time activities is presented, all runtime aspects are not described in detail. Also, the approach does not seem to stress the aspects related to the interaction between the stakeholders that we consider of paramount importance.

6.4 Conclusions

In this chapter, we emphasize the importance of explicitly expressing in process models service aspects that are peculiar to SOSE. We argue that having these service aspects highlighted would provide better guidance on the SOSE development process. We have applied our service aspects on a concrete SOSE methodology. The results show that these service aspects help understanding the SOSE methodology when they are made explicit in an associated process model. Moreover, the use of the methodology and project management are also facilitated.

Further, these service aspects emphasize the SOSE support of a certain methodology. In this way, they help identifying if the methodology itself will deliver ‘real’ SBAs. For instance, by analyzing these service aspects in the SeCSE methodology, we can see that: it involves the standard SOA roles; it covers the interaction among these roles; and it pays particular attention to run-/change time activities. We therefore argue that the SBAs it delivers would potentially be dynamic, agile and have good alignment to business requirements.
Many SOSE methodologies have been proposed and practiced in both academia and industry. Some of these methodologies share common features (e.g., cover similar life cycle phases) but are presented for different purposes, ranging from project management to system modernization, and from business analysis to technical solution development. Given this diversity in the methodologies available in the literature, it is very hard for a company to decide which methodology would fit best for its specific needs. With this aim, this chapter takes a feature analysis approach and devises a framework for comparing the existing SOSE methodologies. Different from existing comparison frameworks, the proposed framework specifically highlights aspects that are specific to SOA and aims to differentiate the methodologies that are truly service-oriented from those that deal little with service aspects. As such, the criteria defined in the framework can be used as a checklist for selecting a SOSE methodology.

7.1 Introduction

Different from some service-oriented approaches for specific SOSE activities, such as service identification methods, SOSE methodologies often provide guidance on multiple SOSE activities and aim at engineering SBAs rather than only designing atomic services or service compositions. Although a mature and systematic software development methodology cannot guarantee high quality software, without such a methodology it would be even harder to trace the cause of errors when things go wrong.

A number of SOSE methodologies have been proposed by vendors, such as SOAD (Zimmermann et al., 2004a), SOMA (Arsanjani et al., 2008) and SOUP (Mittal, 2006) by IBM, a methodology for service architectures by OA-
CHAPTER 7. GUIDING THE SELECTION OF SERVICE-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

SIS (Jones & Morris, 2006), and SO process by CBDI (Allen, 2007). In the SOSE community methods and processes for developing SBAs are becoming a hot research topic. SOSE methodologies proposed by Papazoglou et al. (Papazoglou & van den Heuvel, 2006) and Chang (Chang, 2007) are examples of well known approaches from academia. Given these many methodologies, a common question for practitioners is how to select a methodology that suits their needs the best. More importantly, in SOSE community it is necessary to understand what a SOSE methodology entails to ensure the resulting SBAs being reusable and flexible.

With the objective of assisting the comparison and selection of SOSE methodologies, in this work, we have two research questions:

1. What are the characteristics of existing SOSE methodologies for engineering SBAs?

2. To what extent do they support service orientation?

The first question addresses the scope, features, attributes, or qualities of the SOSE methodologies in general. Its aim is to obtain a comprehensive overview of the existing approaches to engineering SBAs and investigate the distinctions and similarities among the SOSE methodologies. The answer to this question provides a preliminary evaluation of the methodologies, assessing their characteristics as software engineering approaches.

Complementary to the first question, the second question addresses specifically the service orientation aspects of the SOSE methodologies. Its objective is to analyze the way in which SOSE methodologies are different from the traditional ones and aspects that have been specifically designed to support service orientation. The answers to this question provide enterprises guidelines to select a methodology that best suits their needs, including adopting SOA as (part of) their IT portfolio and engineering SBAs that have the potential to achieve the benefits that SOA promises. Moreover, the answers would point out strengths and weaknesses of individual methodologies in supporting service orientation and thereby assist enterprises as well as vendors in enhancing their own SOSE methodologies.

To answer the questions above, we took a feature analysis approach and devised an evaluation framework for comparing the existing SOSE methodologies. Feature analysis (Kitchenham, 1996) is a qualitative evaluation method, offering a way to screen and assess a large number of software engineering methodologies by means of assessing their features (or characteristics). By conducting feature analysis, one can obtain an understanding of particular aspects of a methodology (Jayaratna, 1994) and can systematically select or discard a methodology to achieve a particular goal (Siau & Rossi, 1998).

Among the three alternative approaches (qualitative experiment, qualitative case study, and qualitative survey) to conduct a feature analysis (Kitchenham,
7.2. A SERVICE ASPECTS-DRIVEN EVALUATION FRAMEWORK

(1996), qualitative survey suits our work the best since it does not require the methodologies being used in practice or any experiment. As such, it is more feasible to study a large number of SOSE methodologies and gain insight in the common features they share and specific features provided by particular methodologies.

Different from existing comparison frameworks (e.g., (Ramollari et al., 2007; Kontogogos & Avgeriou, 2009)), ours specifically highlights aspects that are peculiar to SOA and aims to differentiate the methodologies that are truly service-oriented from those that deal little with service aspects. As such, this framework highlights what a SOSE methodology should entail and can be used to compare and evaluate existing SOSE methodologies.

The remainder of this chapter is structured as follows: § 7.2 presents our service aspect-driven evaluation framework, including generic criteria and service-specific criteria; § 7.3 presents the comparison of the existing SOSE methodologies using our evaluation framework and the selection guidelines; § 7.4 discusses our observations on the needs for further improving the existing SOSE methodologies; § 7.5 discusses some related work in the evaluation of SOSE methodologies. § 7.6 provides our conclusions.

7.2 A service aspects-driven evaluation framework

To develop an evaluation framework for SOSE methodologies we need to identify which features are of great importance to SOSE methodologies and relevant for comparison. For this purpose, we collected features from two perspectives, the generic and service-specific features.

The generic features are not specific to SOSE; they are properties common to software engineering methodologies in general, including SOSE methodologies. For instance, support project management and guidance in development activities & artifacts are features that are common to any software engineering methodologies. Instead, support in discriminating between service provision process and service consumption process is a service-specific feature.

In the following sections, we shall explain in detail the features that we consider relevant to SOSE methodologies and the criteria we derived from the features. Each criterion is associated with one evaluation question. The answer to the question can be used to compare and evaluate the SOSE methodologies.

7.2.1 Generic evaluation criteria

In their cataloging framework to evaluate software development methods, Karam and Casselman (Karam & Casselman, 1993) discussed 21 properties, covering
technical, usage and managerial aspects. Some of these properties are product related, in the sense that they are properties of software systems, such as reusability, maintainability, and performance engineering; some other properties are process related, regarding the development process of software systems, such as life-cycle coverage, guidelines, degree of formality. In our work, we are interested in how SBAs should be engineered but not in how certain quality attributes are obtained. Therefore, we selected only the process-related properties as the generic features of SOSE methodologies.

Based on the selected properties, we derived a list of evaluation criteria given in Table 7.1: the first column presents the categories of properties defined in Karam and Casselman’s framework, the second column presents the selected properties, the third column shows the criteria we derived from the properties. To ensure the criteria to be interpreted consistently, we also associated an evaluation question to each criterion. The answers to the evaluation question can be used to compare the SOSE methodologies. These answers can be of four types: narrative, YES/NO, scale, or multiple choices. A narrative answer is for open questions such as “what is the objective of the methodology”; a YES/NO answer can be given when a criterion is met or not; an answer with scale level can be given to a compound criterion where the degree of support provided by the methodology varies and can be judged on a scale (the reader may refer to Appendix A for an explanation of the judgment scale); when multiple choices for an answer are given, one may select one among them.

7.2.2 Service-specific criteria

Service-specific criteria are specifically designed for evaluating the service-oriented aspects of the SOSE methodologies. To derive these criteria, we used the seven differences between SOSE and TSE identified in Chapter 3 as service-specific features, motivated by the rationale that a SOSE methodology should support or provide guidelines to the aspects that are different from TSE. In the following, we shall discuss each of these differences and explain the criteria that we derived from them.

SOA is designed under open-world assumptions

Instead of assuming stable execution environment, SOSE must cope with high uncertainty in the external environment. Open-world assumptions postpone to runtime a number of design decisions that are usually made at design time (Baresi et al., 2006). For instance, in component-based development a major emphasis is that the components of a software system can be bought and assembled at design time, whereas in service-oriented design a major emphasis is on services being dynamically discovered and composed in a SBA at runtime. As a result, a SOSE
### 7.2. A SERVICE ASPECTS-DRIVEN EVALUATION FRAMEWORK

Table 7.1: Criteria for evaluating the generic aspects of SOSE methodologies

<table>
<thead>
<tr>
<th>Generic criterion</th>
<th>Evaluation question</th>
<th>Type of answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Philosophy</td>
<td>GC1 Objective</td>
<td>What are the objectives of the methodology?</td>
</tr>
<tr>
<td>Life cycle coverage</td>
<td>GC2 Lifecycle</td>
<td>How many life cycle phases does this methodology cover?</td>
</tr>
<tr>
<td>Work products &amp; notations</td>
<td>GC3.a Artifacts</td>
<td>Does the methodology specify work products as results of specific activities?</td>
</tr>
<tr>
<td></td>
<td>GC3.b Notations</td>
<td>Does the methodology specify the modeling of work products?</td>
</tr>
<tr>
<td>Procedure</td>
<td>GC4 Procedure</td>
<td>Does the methodology describe the procedures of each covered life cycle phases?</td>
</tr>
<tr>
<td>Guidelines, criteria, measures</td>
<td>GC5 Principles</td>
<td>Does the methodology provide guidelines or principles?</td>
</tr>
<tr>
<td>Degree of formality</td>
<td>GC6 Formality</td>
<td>Are the technical aspects formal? i.e. being precise, unambiguous mathematical definition and can be reasoned about mathematically (logically)?</td>
</tr>
<tr>
<td>Method specialization</td>
<td>GC7 Specialization</td>
<td>Does this methodology provide explicit support for tailoring to fit a particular organization’s needs or domain?</td>
</tr>
<tr>
<td>Automated support (tool support)</td>
<td>GC8 Tool</td>
<td>Are there tools available to support the methodology’s techniques?</td>
</tr>
<tr>
<td>Maturity/project history</td>
<td>GC9 Maturity</td>
<td>Has this methodology ever been applied in industry?</td>
</tr>
<tr>
<td>Software development organization</td>
<td>GC10 Management</td>
<td>Does this methodology support project management?</td>
</tr>
<tr>
<td>Ease of integration</td>
<td>GC11 Integration</td>
<td>What are the existing techniques that the methodology compatible with?</td>
</tr>
</tbody>
</table>
CHAPTER 7. GUIDING THE SELECTION OF SERVICE-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

methodology should provide guidance to its users in what kind of uncertainties can be expected when engineering SBAs and how to handle such uncertainties. *(Criterion SC1: Open-world assumption).*

**Services are the building blocks**

Instead of focusing on implementing a software system as a whole, SOSE focuses on composing coarse-grained discoverable services acting as building blocks of SBAs (Papazoglou, 2003). This shifted focus should be explicitly supported by SOSE methodologies. To evaluate such support, a clear and concrete definition of services *(Criterion SC2: Service definition)* is essential for SOSE methodologies. Currently, a well defined definition of services has not been commonly agreed. The definitions range from technical oriented to business oriented. A SOSE methodology, which guides the design and creation of services as well as their integration, should specify what a service means according to its own philosophy.

The second aspect relevant to services is the way that they are created *(Criterion SC3: The creation of services)*. Depending on existing resources and goals of an organization, services can be realized in many different ways, ranging from greenfield development, identifying from existing software systems or business processes, and discovering from existing services that are possibly provided by third party-providers (Papazoglou & Georgakopoulos, 2003). Accordingly, the guidance that different organizations expect from SOSE methodologies would be different. For instance, an IT company that plans to develop some services and publish them for public consumption would require guidance on greenfield development; an enterprise that intends to migrate their existing legacy software assets to SBAs, instead, would require guidance on identifying services from both existing software and business processes. A SOSE methodology should make it clear how services as building blocks of SBAs are realized.

**Additional development roles are involved in development**

A software developer is not the only development role. This is rather split into three essential participants: service consumer, service provider, and service broker. Each participant is responsible for part of the service development tasks during the service life cycle; and collaborates as a whole to accomplish the development of SBAs (Tsai, 2005). Depending on various development environments, development roles will have to be tailored. For instance, an organization that migrates existing systems to SOA requires different types of development roles from a service provider that aims at publishing reusable services to potentially unknown service consumers. The former requires a business process team for analyzing existing software and/or business assets and identifying services, while
7.2. A SERVICE ASPECTS-DRIVEN EVALUATION FRAMEWORK

the latter requires a market scan team that understands the needs of the service consumption market.

Since one cannot assume the tasks and responsibilities of various roles required in the development of SBAs, a SOSE methodology should provide guidance on what development roles (Criterion SC4: Development roles) and their tasks (Criterion SC5: Links between roles and activities) are required when it is applied.

Services are open

In traditional software development, violation of the original architectural design of the system is extremely hard to handle as soon as the software systems are implemented, deployed and in execution, whereas in service-oriented development, the architecture of SBAs is often designed to be adaptable to market changes, business demands, and customers’ needs (Bell, 2008). Since well-designed SBAs are composed of autonomous and loosely coupled services, the independence between the services increases the ability of SBAs to dynamically take ever changing requirements into account (Di Nitto et al., 2008). A SOSE methodology therefore should provide guidelines in how to design and implement services and SBAs in such a way that they can evolve without hurdlng their execution (Criterion SC6: Architectural change); moreover, a SOSE methodology should also describe what activities should be carried out at runtime in order to support dynamic evolution (Criterion SC7: Runtime activities).

Services are designed with multiple sets of (non)-functional requirements

A software application is engineered for a single set of (non)-functional requirements. In SOSE, instead, since the consumers as well as their needs are not completely known at design time (according to open-world assumptions), services are engineered with multiple sets of (non)-functional requirements, each fulfilling different groups of (potential) consumers with different quality requirements (Maximilien & Singh, 2004).

Non-functional requirements need to be considered over the entire service life cycle, not only because they play a crucial role as traditional software development, but also because they pertain to different service assets and hence have different scope. SOSE typically consists of two main processes, i.e. the development of services and the integration of services into a SBA. Different from TSE where non-functional requirements concern only software systems, in SOSE they concern individual services, service compositions, and SBAs. Quality attributes of a single service are often hard to remain the same when it is composed into service compositions and consumed in SBAs, depending on the quality attributes of the other services and resources. As a SOSE methodology, it is important that it
provides guidance on addressing non-functional requirements with different scope (Criterion SC8: Non-functional requirements).

Differently from a software application, a service might be consumed and shared among different consumers with different quality requirements depending on their application needs and business goals. To maximize reusability and manageability, a service is often realized with one implementation while provided with multiple SLAs to fulfill multiple quality requirements (Maximilien & Singh, 2004). As a result, a service should be designed and developed in such a way that it can be consumed in different contexts. For instance, a GPS service consumed by a navigator application running on a mobile phone device would pose higher requirements in terms of response time as compared to the one consumed by a hotel finder application running on a server in an enterprise. This is because the former requires real-time service while the latter requires off-line service. Accordingly, there is a strong need for SOSE methodologies to provide guidance on the engineering, provision and consumption of services with multiple sets of (non)-functional requirements (Criterion SC9: Variability).

Services are consumed and executed remotely

Instead of buying and installing software locally to the own administrative domain, users of services (pay and) consume services that are executed remotely at the service provider’s side (e.g., in the cloud) (Anand et al., 2005).

This shifted ownership brings tremendous changes in SOSE. First of all, services can be completely isolated from their applications and can be published as software products over the network. The service provision process includes designing, developing, publishing, and maintaining services. More importantly, services are physically located and executed at the service provider’s side. SOSE from the perspective of service provision means how to design and develop services that can be offered to a large number of consumers. Services can be provided either externally (e.g., to third parties), internally (e.g., own usage), or both.

Second, services are eventually composed and integrated into SBAs. After services are published by service providers, they need to be discovered, composed, and integrated into SBAs to fulfill the specific needs of service consumers. SOSE from the perspective of service consumption means how to discover services and alternatives in case of service failure, how to compose the discovered services or in-house developed services to achieve certain reusable business functions, and how to integrate services or service compositions with existing software resources.

Third, when they are published, services need to be cataloged in a service registry or repository to ensure discoverability and reusability from both known and potential consumers. SOSE from the perspective of service brokerage means create and maintain well-functional service registries, and keep the interfaces of registered services up-to-date.
7.2. A SERVICE ASPECTS-DRIVEN EVALUATION FRAMEWORK

In practice, an enterprise that needs to perform some SOSE activities may have different types of business. Taking Amazon as an example:

- Amazon provides messaging services (e.g., Amazon Simple Queue Service (SQS)) or payment & billing services (e.g., Amazon Flexible Payments Service (FPS)) to its business partners. To this purpose, it acts as a service provider performing the process of service provision.

- Amazon composes existing services to achieve a specific business goal, and provides the resulting composite services (e.g., delivery services based on shipping companies) to its business partners. To this purpose, Amazon still acts as a service provider but in this case it performs both service consumption (i.e. discover and select services provided by shipping companies) and service provision (i.e. construct composite services and publish them).

- Amazon maintains a cloud of web services from which one may select the needed services to realize its own e-business functions. To this purpose, Amazon acts as a service broker performing the process of service brokerage.

- Moreover, Amazon may also act as an application builder when it integrates all the services needed to a web application that can be directly used by the end-users. In this case, Amazon needs to perform the process of service consumption, and/or service provision, and/or service brokerage.

Because of the shift in ownership, SOSE should cover different types of process from the perspectives of a service provider, service consumer or system builder. Accordingly, it is of great importance to understand the target perspective (Criterion SC10: Perspective) that a SOSE methodology should support.

Services are cross-organizational

Multiple roles in developing, maintaining, executing, and evolving SBAs are not completely independent but rather requiring collaboration. For instance, when a service provider publishes a service through a service broker, they need to collaborate to keep the service up-to-date whenever the service description is updated. Since the development of SBAs requires more development roles as compared to traditional software systems (Kajko-Mattsson et al., 2007; Zimmermann & Mueller, 2004), the interaction between these roles becomes more complex, too.

Moreover, since services are not executed locally at the consumer’s side, control of services is often highly distributed and crosses trust and organizational boundaries. The interaction and collaboration between multiple development roles therefore cross the boundaries of the organizations as well. As a result, a
CHAPTER 7. GUIDING THE SELECTION OF SERVICE-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

SOSE methodology should provide guidance on the interactions between the development roles that are potentially distributed at multiple organizations. (\textit{SC11: Multiple organizations}).

In summary, we identified 11 evaluation criteria that are specifically relevant to SOSE methodologies. Table 7.2 summarizes the 11 service-specific evaluation criteria we identified, the corresponding evaluation questions (to be answered in evaluating a SOSE methodology) and the expected type of answer.

Table 7.2: Criteria for evaluating the service-specific aspects of SOA methodologies

<table>
<thead>
<tr>
<th>Service-specific criterion</th>
<th>Evaluation question</th>
<th>Type of answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1 Open world assumption</td>
<td>Does the methodology support open-world assumptions?</td>
<td>Scale</td>
</tr>
<tr>
<td>SC2 Service definition</td>
<td>Does the methodology give a definition of services?</td>
<td>Yes/No</td>
</tr>
<tr>
<td>SC3 The creation of services</td>
<td>Does the methodology describe how services are created?</td>
<td>Narrative</td>
</tr>
<tr>
<td>SC4 Development roles</td>
<td>Does the methodology explain roles?</td>
<td>Scale</td>
</tr>
<tr>
<td>SC5 Association of roles to activities</td>
<td>Are the activities associated to the roles?</td>
<td>Scale</td>
</tr>
<tr>
<td>SC6 Architectural change</td>
<td>Does the methodology support architectural change after services are deployed and in execution?</td>
<td>Scale</td>
</tr>
<tr>
<td>SC7 Runtime activities</td>
<td>Does the methodology support runtime activities?</td>
<td>Scale</td>
</tr>
<tr>
<td>SC8 Non-functional requirements</td>
<td>Does the methodology consider (non-)functional requirements?</td>
<td>Scale</td>
</tr>
<tr>
<td>SC9 Variability</td>
<td>Does the methodology support variability among different sets of service consumers and different contexts</td>
<td>Scale</td>
</tr>
<tr>
<td>SC10 Perspective</td>
<td>From which perspective is the methodology described from?</td>
<td>Service consumption, service provision, service brokerage</td>
</tr>
<tr>
<td>SC11 Multiple organizations</td>
<td>Does the methodology support SOA development by multiple organizations in collaboration?</td>
<td>Scale</td>
</tr>
</tbody>
</table>

7.3 Comparison of existing SOSE methodologies using the evaluation framework

7.3.1 Overview of the selected SOSE methodologies

The aim of this work is to understand what SOSE methodologies entail rather than gaining insight in the state of the art. To support our aim we selected a mix
of 12 prominent SOSE methodologies discussed and compared in already existing evaluation frameworks (discussed in detail in §7.5). The selected methodologies have been proposed from both industry and academia, and published in both journals (e.g., (Papazoglou & van den Heuvel, 2006)), conferences (e.g., (Karhunen et al., 2005)), and white papers (e.g., (Arsanjani et al., 2008)).

To show the usability of the evaluation framework, we compared the selected SOSE methodologies based on the criteria presented in §7.2. In the following, we shall briefly introduce the 12 SOSE methodologies we selected for evaluation.

The CBDI-SAE (Allen, 2007) methodology is part of the CBDI-SAE SOA Reference Framework (RF) published by the CBDI Forum. The process consists of four key discipline areas, including consume, provide, manage, and enable. To tailor the process to the needs of different audiences, multiple process views (such as flow view, existing organizational view, life-cycle stages view) can be created to assist the engineering of SBAs from different perspectives.

Chang (Chang, 2007) proposes a methodology based on SOAD, consisting of six phases. The results of each phase refer to one or more key artifacts specified in SOAD. Chang’s methodology specifically addresses concerns relating to dynamically adaptable services. To deal with service variability and mismatch, Chang’s approach considers three types of variation points in service design (workflow, service composition, and logic) and three types of service mismatch (interface, functional, and non-functional).

Steve Jones from (OASIS) (Jones & Morris, 2006) developed a methodology for service architectures, providing mechanisms for planning, managing, and delivering projects using SOA techniques. The methodology follows a broadly four step process answering four questions (what, who, why, and how). The aim of this method is to describe how a service architecture can be defined, rather then how it can be delivered.

The Web services development life-cycle methodology (SLDC) (Papazoglou & van den Heuvel, 2006) utilizes an iterative and incremental process based on several well-established process models from TSE, such as the rational unified process (RUP), component-based development (CBD), and business process modeling (BPM). The methodology consists of eight phases that cover the whole service life cycle, ranging from planning to deploying, monitoring, and managing SBAs.

The SeCSE (SeCSE, 2007) methodology is the main process adopted by the SeCSE project (a European Union funded project) in the realm of Service Centric Systems (SCS) Engineering. At the highest level the SeCSE methodology is represented by three important functional areas, service engineering functional area where the service developer develops services; service acquisition/provisioning functional area where service provider delivers the services in

\[\text{http://www.secse-project.eu/}\]
CHAPTER 7. GUIDING THE SELECTION OF SERVICE-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

the marketplace; and service-centric system engineering functional area where
the consumers can build and manage SBAs based on their choices about which
services suit their needs. Designed with service adaptation in mind, the SeCSE
methodology also supports runtime service composition and recovery manage-
ment.

SOAD (Zimmermann et al., 2004a) by IBM is a methodology based on existing
modeling disciplines such as Object-Oriented Analysis and Design (OOAD),
Enterprise Architecture (EA) frameworks, and BPM. It proposes elements, such
as domain decomposition, service categorization and aggregation, and semantic
brokering, that need to be considered in service-oriented analysis and design.

Service-oriented architecture framework (SOAF) (Erradi et al., 2006) is an
academic methodology presenting an architecture-centric framework. Its goal is
to ease the definition, design, and realization of SOA to achieve a better business
and IT alignment. To this end, the methodology uses two types of business pro-
cess models: To-be modeling: a top-down business-oriented approach describing
the candidate solution, and As-is modeling: a bottom-up approach describing
current business processes and the problem space.

Service-Oriented Development In a Unified fraMework (SODIUM) (Topouzi-
dou, 2007) is an academic project involving international research and industrial
partners. The methodology developed in this project focuses on how to define
new services based on compositions of reusable coarse-grained services. To this
end, SODIUM proposes a set of models, languages, middleware, and tools to be
adopted for engineering SBAs.

SOMA (Arsanjani et al., 2008) is an iterative and incremental methodology
developed by IBM, aiming at the identification, modeling, and design of services
and SBAs. The SOMA methodology consists of seven phases starting from busi-
ness modeling & transformation to solution management. Using the modeling
tools, SOMA breaks down the business process into a component view. The so-
lution is modeled based on SOA Reference Model, which defines a layered systems
architecture.

The SOSE (Karhunen et al., 2005) framework is a methodology aiming at
developing methods and tools to improve quality and profitability of SBA de-
velopment. It suggests that service and component design should always start
by creating a business case to justify the project implementation. Combining
component-based and service-oriented development, the framework defines three
levels of granularity: system level component (SLC), business service component
(BSC), and component level.

The Service-Oriented Unified Process (SOUP) (Mittal, 2006) was a method-
ology proposed by Knual Mittal from IBM, using the best elements from RUP
and XP. Aiming at assisting the establishment and management of SOA projects,

2http://www.atc.gr/sodium
SOUP has been specifically designed for both initial SOA deployment and ongoing SOA management.

A methodology for engineering a TRUE SOA that allows companies to flexibly adapt to changing market demands was proposed by Engels et al. (Engels et al., 2008). In this methodology, the business of an enterprise is organized in a service-oriented way (as a set of business services) and the enterprise IT architecture is structured according to those business services.

### 7.3.2 Comparing the generic aspects of the SOSE methodologies

Using the generic evaluation criteria described in § 7.2.1, we evaluated the 12 SOSE methodologies in terms of the development process. The results of the evaluation are presented in Table 7.3.2. In the following, we shall discuss the results in detail and explain how each criterion assists the selection of SOSE methodologies.

**GC1 Objective** Despite that the ultimate goal of SOSE methodologies is to delivering SBAs that meet the requirements of their users, the objective of individual SOSE methodologies often varies. Some of the SOSE methodologies are designed to provide guidance to the entire engineering process, such as CBDI-SAE, SDLC, SeCSE, and SOMA; some focus only on the analysis and design of the architecture of SBAs, such as OASIS, SOAD, SOSE, and TRUE; some concern specific engineering issues, e.g., Chang and SODIUM focus on dynamic service composition, SOAF focuses on SOA migration, and SOUP aims at assisting project management.

**Selection guideline:** Knowing the objective of the SOSE methodologies is essential for enterprises to select the one that suits their needs at best. This criterion therefore can be used as the first step to filter out relevant candidates for selection. If for instance an enterprise aims at composing existing services for internal or external uses, it may consider Chang and SODIUM as candidate methodologies.

**GC2 Lifecycle** To compare the lifecycle coverage of the SOSE methodologies, we used the service life cycle model (in Chapter 4). The description of each phase of the life cycle model is given in Appendix B. Based on this model, we analyzed each SOSE methodology (reported in Table 7.4) the coverage of each phase using the scale defined in Appendix A. It is reasonable that the SOSE methodologies aiming at providing guidance for the entire engineering process (e.g., SDLC and SeCSE) cover the largest number of life cycle phases while the methodologies that focus on the analysis and design of an architecture (e.g., SOSE and TRUE)
<table>
<thead>
<tr>
<th>Generic criterion</th>
<th>CBDI-SAE</th>
<th>SDLC</th>
<th>SeCSE</th>
<th>SOMA</th>
<th>OASIS</th>
<th>SOAD</th>
<th>SOSE</th>
<th>TRUE</th>
<th>CHANG</th>
<th>SOMO</th>
<th>SOAF</th>
<th>SOUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC1 Objective</td>
<td>Engineering process</td>
<td>Engineering process</td>
<td>Engineering process</td>
<td>Engineering process</td>
<td>A&amp;D</td>
<td>A&amp;D</td>
<td>A&amp;D</td>
<td>Service composition</td>
<td>Service composition</td>
<td>SOA migration</td>
<td>Project management</td>
<td></td>
</tr>
<tr>
<td>GC2 Lifecycle</td>
<td>See Table 7.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GC3.a Artifacts</td>
<td>+++</td>
<td>+</td>
<td>+++</td>
<td>+</td>
<td>+++</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>GC3.b Notations</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>+</td>
<td>++</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>+++</td>
<td>++</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>GC4 Procedure</td>
<td>++</td>
<td>++</td>
<td>+++</td>
<td>+</td>
<td>+++</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>++</td>
<td>+++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>GC5 Principles</td>
<td>+++</td>
<td>+++</td>
<td>-</td>
<td>+++</td>
<td>+</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>++</td>
<td>+++</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GC6 Formality</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GC7 Specialization</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>GC8 Tool</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GC9 Maturity</td>
<td>NA</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
<td>No</td>
<td>Yes</td>
<td>NA</td>
</tr>
<tr>
<td>GC10 Management</td>
<td>++</td>
<td>++</td>
<td>+++</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>+++</td>
</tr>
<tr>
<td>GC11 Integration</td>
<td>-</td>
<td>RUP, CBD, BPM</td>
<td>OOAD, RUP, BPM</td>
<td>OOAD, BPM</td>
<td>-</td>
<td>-</td>
<td>SOUP, SOMA</td>
<td>-</td>
<td>RUP, XP</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 7.3: Comparing the generic aspects of the SOSE methodologies.
### 7.3. COMPARISON OF EXISTING SOSE METHODOLOGIES USING THE EVALUATION FRAMEWORK

Table 7.4: Comparing the life cycle coverage of the SOSE methodologies

<table>
<thead>
<tr>
<th>Role</th>
<th>Life cycle activity</th>
<th>CBID-SAE</th>
<th>SDLC</th>
<th>S-CSE</th>
<th>SOMA</th>
<th>OASIS</th>
<th>SOAD</th>
<th>SOSE</th>
<th>TRUE</th>
<th>CHANG</th>
<th>SODIUM</th>
<th>SOAF</th>
<th>SOUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service provider</td>
<td>Market scan</td>
<td>-</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Requirements engineering</td>
<td>++</td>
<td>+++</td>
<td>++</td>
<td>++</td>
<td>+++</td>
<td>++</td>
<td>-</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Business modeling</td>
<td>++</td>
<td>++++++++</td>
<td>++</td>
<td>+++</td>
<td>++</td>
<td>-</td>
<td>+++</td>
<td>-</td>
<td>+++</td>
<td>+++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Service design</td>
<td>++</td>
<td>++++++++</td>
<td>++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>-</td>
<td>+++</td>
<td>+++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Service development</td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>++</td>
<td>+++</td>
<td>+++++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Service testing</td>
<td>++</td>
<td>++++++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Service publishing</td>
<td>++</td>
<td>++++++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Service provision</td>
<td>++</td>
<td>++++++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Service monitoring</td>
<td>-</td>
<td>+++++++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Service management</td>
<td>+</td>
<td>+++++++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Service broker</td>
<td>Registry selection</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Registry update</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+++++</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Registry maintenance</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Service consumer</td>
<td>Service discovery</td>
<td>-</td>
<td>+</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Service composition</td>
<td>+</td>
<td>+</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Service negotiation</td>
<td>-</td>
<td>-</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Service invocation</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Service monitoring</td>
<td>-</td>
<td>+++++++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Application builder</td>
<td>Requirements engineering</td>
<td>+++</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Application design</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Application implemen-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>tation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Module testing</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Application testing</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Application maintenance</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
only cover few life cycle phases that are related to requirements engineering and service design.

When looking at the individual life cycle phases, we can see that business modeling and service design are nearly covered by all the SOSE methodologies, while application-specific phases such as application design and implementation are not covered by any of the SOSE methodologies. The main reason for leaving out application-specific phases is that all these SOSE methodologies focus on service-specific phases including the design, development, and delivery of services (and service compositions) while regarding application-specific phases as part of TSE process (which is out of the scope of these SOSE methodologies).

Moreover, registry-related phases are only marginally covered by three SOSE methodologies (CBDI-SAE, SOAD, and SODIUM) and no detailed guidance is given. Surprisingly, phases that are specifically important to SOSE such as service monitoring, service discovery, and service composition are only covered by few methodologies.

According to the life cycle model, requirements engineering is often performed by two roles: a service provider and an application builder. Some SOSE methodologies (e.g., SDLC) support the collection of requirements from the market: in this case the role played in using the methodology is that of a service provider that delivers services to its external clients. Some other methodologies (e.g., SODIUM) support the collection of requirements from its internal users: in this case the role played in using the methodology is that of a service provider, too, but services are delivered to internal clients instead. Some SOSE methodologies (e.g., SeCSE) support the collection of system requirements from its end users: in this case the role played is that of an application builder.

Selection guideline:
Some enterprises look for guidance for specific activities rather than fully-fledged methodologies. In this case knowing the phases covered by the SOSE methodologies is crucial for selection. An enterprise that aims at building SBAs, for instance, should select a SOSE methodology and apply it with a TSE methodology since obviously the analyzed SOSE methodologies do not provide any support for application-related activities.

**GC3.a Artifacts** Most of the methodologies provide detailed description of the outcomes of each engineering activity. Only SOAD does not give any information about its produced artifacts.
7.3. COMPARISON OF EXISTING SOSE METHODOLOGIES USING THE EVALUATION FRAMEWORK

Selection guideline:
If the goal of an enterprise is specifically to produce certain outputs, this criterion should be used to drive selection, and obviously SOAD would not be a candidate for such goal.

GC3.b Notations Most of the methodologies that specify their artifacts also provide some guidance on the modeling techniques, except for CBDI-SAI, SOAF, and SOUP. Chang’s approach explicitly indicates when traditional modeling techniques (e.g., commonality and variability modeling) can be used and when service-specific modeling techniques (e.g., service decision model) are needed. Some of the methodologies not only specify the modeling of their artifacts but also give some examples (e.g., service component specification and goal-service model in SOMA, UML domain model in TRUE, business process model in SDL) (C). Many other methodologies only name some models (e.g., business process map in OASIS, service interaction model in SeCSE, abstract composition model in SODIUM) that need to be created but do not provide further guidance on how to create them.

Selection guideline:
As enterprises have tools (and their supported notations) already in place, this criterion is relevant to select methodologies that could be potentially supported by the existing tools, and (if not) to explicitly assess necessary investments. Though, excluding TRUE that uses UML, all other methodologies only use types of models, which could be then checked against supporting notations.

GC4 Procedure Most of the methodologies describe the procedures (i.e. a specified series of actions or operations which have to be executed) of each covered life cycle phase, except for SOAD. As indicated by the scale, some methodologies (e.g., OASIS, SeCSE, SOAF) provide very detailed guidance, describing comprehensively how to carry out each engineering activity and its sub-activities, while many other methodologies only explain briefly what the engineering activities (and their sub-activities) are and what they entail, but no step-by-step guidance is given.

Selection guideline:
This criterion is extremely useful for enterprises that do not have experience in developing SBAs and need (procedural) guidance on the way each life cycle phase should be carried out.

GC5 Principles Most of the methodologies discuss guidelines or principles
CHAPTER 7. GUIDING THE SELECTION OF SERVICE-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

for the development process they support. However, only CBDI-SAE, SDLC, SOAF, and SOMA explain in detail the design principles they used, among which separation of concerns, loose coupling, and business-IT alignment are the principles commonly considered. SOAD and SOUP only briefly mention the engineering activities they support but do not enter the details.

**Selection guideline:**
Similar to criterion GC4 Procedure, this criterion is useful for enterprises that lack experience and need guidance on design principles.

**GC6 Formality** None of the methodologies supports formal reasoning.

**Selection guideline:**
If an enterprise is seeking for SOSE methodologies that use mathematical definitions and can be mathematically reasoned about, obviously none of the analyzed methodologies is suitable.

**GC7 Specialization** All analyzed methodologies are domain-independent and they can be used in any organizations. However, none of the methodologies provides explicit support for how to tailor them to fit a particular organization’s needs or domain.

**Selection guideline:**
Enterprises that plan to apply SOA in specific business domains should check for SOSE methodologies yielding any domain-specific features (n/a in the SOSE methodologies we analyzed so far).

**GC8 Tool** Only three methodologies are (partially) tool supported. Thanks to their industrial applications, the two methodologies developed within European projects, SeCSE and SODIUM, are very well supported by the tools specifically developed for them. In SeCSE, tools used for supporting engineering activities and producing work products are well explained. In SODIUM, three tools (the composition studio, the SODIUM runtime environment, and the SODIUM composition repository) are introduced and their use in engineering activities (such as modeling, service discovery, service selection, web service wrapper generation, and service deployment) is explained. Different from these two industrial methodologies, SDLC does not develop its own tools; instead, it recommends some existing tools available in the market to support some engineering activities. For instance, when building business architecture SDLC suggests to use IBM Rational Portfolio Manager to gain insight into the business benefits and costs of the SOA services portfolio; and for design and developing business processes.
7.3. COMPARISON OF EXISTING SOSE METHODOLOGIES USING
THE EVALUATION FRAMEWORK

SDLC suggests to use some automated tools such as IBM's WebSphere Business
Modeller.

Selection guideline:
This criterion is crucial for selection if an enterprise is seeking for methodologies
that are tool supported. From our analysis the SOSE methodologies
that are tool supported are from industry and consequently these tools are
not publicly available. Only SDLC introduces some tools that are available
on the market.

GC9 Maturity The maturity of the SOSE methodologies relies on their application
in real-life projects. Only when applied, the advantages and shortcomings of the SOSE
methodologies become apparent and therefore can be further evolved to improve its maturity. Thanks to their industrial collaboration, most
of the SOSE methodologies proposed in industry have already been applied, such
as OASIS, SeCSE, SOAD, and SOMA. Only CBDI-SAE does not report any
real-life case studies. Among the methodologies proposed in academy, SOAF and
SOSE have been applied in Securities Trading and Electricity Market domain,
respectively. TRUE has been applied and validated in many large-scale industrial
projects. The other methodologies do not provide information about their application.

Selection guideline:
Some enterprises are open to try newly developed SOSE methodologies, testing
their strengths or weaknesses and distilling new ideas or techniques from them. Some other enterprises are not flexible to experiment with immature
methodologies due to the risks intrinsic of their business domain (e.g., banking
domain), and require tested and validated ones. Knowing the level of
maturity of the methodologies is useful for selection.

GC10 Management SOA project management refers to a business activity
(often performed by a project manager) that plans, starts, controls, and ends
the engineering of a SBA to achieve a particular aim. It is of great importance
to any SBA development when putting into industrial practices. OASIS and
SOUP are designed specifically from the perspective of project management. In
OASIS, deliverables required in each stage of the development process are defined
in terms of their description and expected duration, the required team members
are suggested, and an example of the project plan is given; in SOUP, not only key
deriverables are defined but also the development process for an initial project
(when an SBA has not been built) is differentiated from an ongoing project
(when an SBA has been built but requires maintenance). Other methodologies,
although not designed for project management, consider project management as part of their engineering process. For instance, in CBDI-SAE two terms (project and project profile) that are dedicated to project management are embraced by the CBDI-SAI SOA reference framework; in SeCSE the actions on the working products such as read-only, create or modify are defined, which assists the control of the working products; SOAF supports project management in the sense that it suggests a list of key deliverables for planning the projects but their descriptions are missing; in SOMA, initiating project management activities are suggested in the phase of solution management but their descriptions are also missing. The remaining methodologies do not include any guidance for project management.

Selection guideline:
This criterion is extremely useful for enterprises that lack experience in SOA project management. From our analysis the supported management activities are: guidance for project deliverables, team setup, project planning, setup of greenfield projects, and managing pre-existing projects.

**GC11 Integration** The ease of integration of a SOSE methodology refers to its compatibility with existing techniques. Among the studied SOSE methodologies, OOAD, CBD, BPM, IBM RUP, XP, and BPM are techniques from TSE. OOAD, an approach that models systems as a group of interacting objects, is used by SOAD and SOMA; CBD, built upon OOAD principles and focused on the separation of concerns of software systems, is explicitly used by SDLC; RUP, an iterative software development process framework created by the Rational Software Corporation, is used by SDLC, SOMA, and SOUP; XP, a type of agile software development that aims at improving the productivity of software by adopting customer requirements or feedback in frequent software releases, is used by SOUP; BPM, a management approach aiming at aligning different aspects of an organization with the requirements of its clients, is used by SDLC, SOMA, and SOAD. Chang is the only one that is built upon existing SOSE methodologies, including SOUP and SOMA.

Selection guideline:
Some enterprises have already been working with existing techniques. For these enterprises, it would be easier to apply a SOSE methodology that is compatible with the familiar techniques. This criterion is useful for enterprises to be aware of the methodologies that allow them to reuse their own integration knowledge and experience.
7.3. COMPARISON OF EXISTING SOSE METHODOLOGIES USING
THE EVALUATION FRAMEWORK

7.3.3 Comparing the service-specific aspects of the SOSE methodologies

Using the service-specific criteria described in § 7.2.2, we evaluated the 12 SOSE methodologies from the perspective of service orientation. The results of the evaluation are presented in Table 7.5 (sorted by their objectives). In the following, we shall discuss the results in detail and explain how each criterion assists the selection of SOSE methodologies.

**SC1 Open world assumption** This criterion is used to evaluate if a SOSE methodology provides guidance in handling the uncertainty (e.g., unforeseen service consumers, execution context and service usage) during the development of SBAs. Based on our evaluation, there are only two methodologies (SOMA and SDLC) embrace open world assumptions. SOMA is composed of capability patterns, executed in all phases with different degrees of elaboration and precision. For instance, exposure decisions are first made based on the information about the services known at the identification phase and these decisions are then elaborated and refined in the specification phase when more information about the non-functional requirements of the services are known. To give another example, SOMA supports the recombination of services in unanticipated service context by developing a service context diagram depicting the service ecosystem. In such a service ecosystem, a group of related services as well as their providers and consumers, input and output, and underlying systems that implement them are all illustrated.

SDLC emphasizes the importance of different types of coupling in service design to address uncertainties. For instance, it suggests that business processes should not depend on specific representational or implementation details (representational coupling); connection channels between services should be unaware of who is providing the service (identity coupling); and a sender of a message should rely only on those effects necessary to achieve effective communication (communication protocol coupling). In the rest of the SOSE methodologies, although loose coupling is often considered in service design, they do not explicitly consider any open-world assumptions.

**Selection guideline:**
If an enterprise faces many uncertainties during the development of SBAs, this criterion is important for selecting a methodology that is capable of dealing with these uncertainties. From our analysis uncertainties of non-functional requirements, unanticipated service context, unknown implementation details, unknown service providers, and unknown communication protocols are currently addressed by some of the methodologies.
<table>
<thead>
<tr>
<th>Service-specific criterion</th>
<th>CBDA-SAE</th>
<th>SDLC</th>
<th>SeCSE</th>
<th>SOMA</th>
<th>OASIS</th>
<th>SOAD</th>
<th>SOSE</th>
<th>TRUE</th>
<th>CHANG</th>
<th>SOUDM</th>
<th>SOAF</th>
<th>SOUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1 Open world assumption</td>
<td>-+++</td>
<td>-</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SC2 Service definition</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SC3 The creation of services</td>
<td>c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC4 Development roles</td>
<td>-</td>
<td>-</td>
<td>+++</td>
<td>-</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SC5 Association of roles to activities</td>
<td>-</td>
<td>-</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>+++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SC6 Architectural change</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>+</td>
<td>-</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td>SC7 Runtime activities</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SC8 Non-functional requirements</td>
<td>-</td>
<td>+++</td>
<td>+++</td>
<td>++</td>
<td>-</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>+++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>SC9 Variability</td>
<td>SP</td>
<td>SP</td>
<td>SP, SC</td>
<td>SP</td>
<td>SP</td>
<td>SP</td>
<td>SP</td>
<td>SP</td>
<td>SP</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>SC10 Perspective</td>
<td>SC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC11 Multiple organizations</td>
<td>++</td>
<td>+</td>
<td>+++</td>
<td>-</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>+++</td>
<td>-</td>
</tr>
</tbody>
</table>

Note: c1 - No service identification; Legacy transition and greenfield development.
c2 - Identified by decomposing business process; created by discovery, legacy transition and greenfield development.
c3 - No service identification; Created by discovery and composition.
c4 - Identified from organizational resources, domain decomposition, and existing asset analysis; created by legacy transition and greenfield development.
c5 - Identified by analyzing the broad “what” of the enterprise; no realization.
c6 - Identified from existing legacy systems, business processes and rules; no realization.
c7 - Identified by analyzing business domain; no realization.
c8 - Identified from enterprise architecture; no realization.
c9 - Identified from requirements; created by discovery and composition.
c10 - Identified by decomposing business processes; created by discovery and composition.
c11 - Identified from business decomposition and existing IT portfolio; created by legacy transition and greenfield development.
c12 - Identified from requirements; created by legacy transition and greenfield development.
7.3. COMPARISON OF EXISTING SOSE METHODOLOGIES USING THE EVALUATION FRAMEWORK

**SC2 Service definition** Surprisingly, only half of the SOSE methodologies provide a definition of services. Some methodologies define services from a technical perspective. For instance, SOAD defines services as “logical groupings of operations”; SOSE defines services as “components with published interfaces”; and in SOMA, services are “first-class constructs of service orientation”. OASIS and TRUE, instead, define services from a business perspective. In OASIS, services represent “what the business does” and in TRUE, a service is the output of a service provider towards a service consumer. Among all the definitions, Chang provides the most comprehensive one, which defines services from both service consumer (“a unit of functionality with a certain service level agreement expected by consumers”) and service provider (“a unit of deploying functionality which is publishable in WSDL standard”) point of view.

**Selection guideline:**
Each enterprise has its own understanding of what services entail. To ensure the resulting services or SBAs do fulfill the goals of enterprises, being aware of how services are defined in a SOSE methodology is crucial for them to select the right ones.

**SC3 The creation of services** Services as building blocks of SBAs are designed and developed differently from methodology to methodology. In general, the methodologies address the creation of services in terms of two tasks: service identification and service realization. Except for CBDI-SAE and SeCSE that do not suggest any service identification methods, all the other methodologies provide guidelines in identifying (abstract, not implemented) services from different resources, such as requirements from service consumers and stakeholders (e.g., Chang, SOUP), enterprise knowledge (e.g., OASIS and TRUE), business domain (e.g., SDLC, SOAF, SODIUM, SOSE), legacy systems (e.g., SOAD), or a mix of these resources (e.g., SOMA).

In terms of service realization, many SOSE methodologies (CBDI-SAE, SDLC, SOAF, SOMA, SOUP) support transforming legacy system into services by either wrapping existing components or re-engineering and re-factoring legacy applications. Interestingly, the methodologies that support legacy transmission often suggest greenfield development as well, by developing new services from scratch. This is quite understandable in that migrating legacy system does not address new business requirements and hence services with new business functions have to be developed.

SeCSE, SODIUM, and Chang support the development of SBAs through composing services that are discovered from existing services. In SeCSE, services are discovered in three phases: in requirements definition phase services are discovered as candidate services from the service registry if they fit the business model
CHAPTER 7. GUIDING THE SELECTION OF SERVICE-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

and system requirements, in design time services are further discovered from candidate services if they are compatible with specific system architectures and workflows, and in runtime SeCSE supports the discovery of alternatives for replacing services that become unavailable or fail to meet certain requirements. In SODIUM, heterogeneous services (i.e. web, p2p and grid services) are discovered from heterogeneous registries and networks with the use of the USQL Engine if the semantic descriptions and QoS requirements are met. In Chang, published services which correspond to the requirements are discovered.

OASIS, SOAD, SOSE, and TRUE specifically focus on the design and analysis part of the SOSE process, providing guidance only on service identification but not on service realization.

Selection guideline:
Some enterprises may have already had a clear vision on the services needed for reaching their business goals or have already applied some methods to identify services. Some other enterprises do need support of identifying services from their existing resources. In either case, knowing the different levels of support of service creation provided by SOSE methodologies facilitates the selection of SOSE methodologies.

SC4 Development roles Despite of the importance of development roles required during any software development process, many SOSE methodologies do not provide any guidelines on these roles. Only SeCSE, OASIS, and SOSE explicitly explain development roles or actors. In SeCSE, for each engineering activity, a list of actors as well as their responsibilities is described. As such, SeCSE provides a complete list of development roles or actors. In OASIS, for each engineering activity, a list of actors as well as their responsibilities is described. As such, OASIS provides a complete list of development roles required when it is applied in practice. OASIS does not provide a complete list of the development roles; instead, it only lists the roles required in the planning phase of the project, in which the key stakeholders who are required to create the service architecture should be identified. In SOSE, the development process consists of two parts, business case and service & component design. Only in the business case, the required roles and their responsibilities are described.

Selection guideline:
Enterprises, especially the ones that have little experience with developing SBAs, often do not know what kind of expertise and personnel are required before a SOA project starts. This criterion is especially useful for enterprises when they have such doubts. In addition, enterprises may select the ones that fit their needs based on the existing human resources to minimize the need for recruitment.
7.3. COMPARISON OF EXISTING SOSE METHODOLOGIES USING THE EVALUATION FRAMEWORK

SC5 Association of roles to activities Among the three SOSE methodologies (SeCSE, OASIS, and SOSE) that do explicitly explain the development roles, OASIS does not associate the development roles to any engineering activities. Rather, OASIS only briefly lists some key roles required in preparing and planning a SOA project. In both SeCSE and SOSE, the development roles are explicitly associated to their engineering activities.

Selection guideline:
Similar to criterion SC4 Development roles, this criterion is useful for enterprises that have little experience with developing SBAs to select SOSE methodologies that clearly explain responsibilities of required team members.

SC6 Architectural change Despite that adaptability is of great importance to SBAs, about half of the SOSE methodologies do not explicitly support making architectural changes after services are deployed and in execution. Among the SOSE methodologies that do provide some support for changing or even determining the service architecture at runtime, SeCSE allows the highest flexibility since it supports creating on-the-fly service compositions addressing specific runtime user requirements by directly interacting with runtime discovery process. Chang also supports adaptable services through dynamic service discovery and composition but detailed guidance is missing; moreover, Chang provides guidance of how to handle mismatches between requirements and discovered services. SDLC, SOAM, SOAF, and TRUE do not directly suggest any techniques to handle changes but they, to certain extent, support making changes by emphasizing the importance of designing for adaptability.

Selection guideline:
Some enterprises adopt SOA because its potential ability of building dynamically adaptable SBAs. For these enterprises, whether a SOSE methodology provides strong support of handling changing requirements after services are published is of great importance. From our analysis the levels of such support range from design for adaptability to the determination of service architecture at runtime. Enterprises may use this criterion to select the one that meets their adaptability needs.

SC7 Runtime activities Most of the SOSE methodologies focus on providing guidance on design time activities, only five methodologies explicitly describe activities that need to be performed at runtime. When looking at the objectivities (GC1 Objective) of the methodologies, we noticed that the ones that aim at design & analysis (such as OASIS and SOAD) naturally contain only design time...
activities, whereas the ones intend to provide guidance for the entire engineering process often take runtime activities into account. For instance, SeCSE, SDLC, and SOMA cover both design time and runtime activities; different from SDLC and SOMA that only briefly mention some runtime activities (e.g., dynamic binding and monitoring in SLDC and monitoring and management in SOMA), SeCSE describes its runtime activities (e.g., runtime service discovery, composition, re-binding, monitoring) in detail.

The other two SOSE methodologies that describe runtime activities are Chang and SODIUM, both aiming at (dynamic) service composition. In Chang, a dynamic composition handler (DCH) is briefly introduced, which enables runtime service invocation, service adaptation, and binding. However, detailed information about how a DCH actually works and how to use it is missing. In SODIUM, runtime service discovery and composition are described in detail; in addition, tools that support these runtime activities are also introduced.

**Selection guideline:**
Similar to criterion SC6 Architectural change, this criterion can be used to evaluate the runtime support provided by SOSE methodologies with the purpose of achieving dynamically adaptable SBAs.

**SC8 Non-functional requirements**
NFRs, including quality requirements (e.g., performance, security) and business requirements (e.g., business goals and vision), are of great importance to services. Most of the SOSE methodologies we evaluated (except for CBDL-SAE and OASIS) consider NFRs in their process, providing guidelines on collecting, specifying or achieving NFRs, with different levels of detail. In SOAD, some general principles or quality factors are identified and act as its baseline for design; however, how to collect, specify, and satisfy NFRs from the users of services is not discussed. In Chang, SOAF, SOUP, SOMA, SOSE, and TRUE, collecting and specifying of NFRs are mentioned as part of their engineering tasks; however, details about how to carry out these tasks are missing.

In SeCSE, SDLC, and SODIUM, NFRs are explicitly considered and discussed in detail. In SeCSE, binding rules and quality of service (QoS) constraints and objectives (that will be used at runtime by the binding and re-binding process) are first defined at design time. After that, in requirements-based service discovery process mechanisms are provided for matching functional and quality of service requirements with published service requirements.

SDLC discusses how to capture NFRs as well as how to monitor and enforce them by means of service-level agreements (SLAs). SLAs are used to formalize the usage conditions and quality-level guarantees during the process of service
7.3. COMPARISON OF EXISTING SOSE METHODOLOGIES USING THE EVALUATION FRAMEWORK

negotiation process; and finally whether the behavior of services abides by the agreed SLAs are determined by the process of service monitoring.

In SODIUM, QoS are first specified in two parts: one part contains the absolute QoS constraints that are used to exclude services, and another part contains the optimization criteria that are used to rank the services; and then QoS are used to sort and select a ranked list of candidate services for further composition.

<table>
<thead>
<tr>
<th>Selection guideline:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some enterprises have higher NFRs than the others. For these enterprises, it is even more important to capture and specify NFRs collected in a systematic way and carry out specific activities to ensure the NFRs delivered by services or SBAs match the needs of their users.</td>
</tr>
</tbody>
</table>

SC9 Variability To increase the reusability of services, a service should be provided with different quality attributes to fulfill the needs of different groups of service consumers. Unfortunately, our evaluation shows that only Chang, SDLC, and SeCSE provide guidelines of designing and/or delivering services for specific service consumers according to their needs. In Chang, commonality and variability among different service consumers and context are analyzed and modeled. Non-functional mismatch is identified as a variation point in service design and this information is used in mapping the NFRs collected from various service consumers and stakeholders to the candidate services.

Both SDLC and SeCSE describe the use of SLAs in specifying different requirements of service consumers and in reaching agreements with service providers on service consumption conditions. Further, SDLC emphasizes the importance of the design for service reuse and suggests making services more generic, abstracting away from differences in requirements between service consumers and attempting to provide the generic service in different context where it is applicable. SODIUM supports the variability of NFRs by allowing the users to specify individually the required QoS that must be offered by services. This information is used when selecting and prioritizing candidate services for service composition.

<table>
<thead>
<tr>
<th>Selection guideline:</th>
</tr>
</thead>
<tbody>
<tr>
<td>For enterprises that act as a service provider and aim at providing their services to different groups of users with different requirements, this criterion is crucial for selection.</td>
</tr>
</tbody>
</table>

SC10 Perspective The engineering of SBAs is concerned with both service provision and service consumption processes. However, most of the SOSE methodologies, including OASIS, SDLC, SOAD, SOAF, SOMA, SOSE, SOUP, and TRUE, are designed from the perspective of service provision. These method-
CHAPTER 7. GUIDING THE SELECTION OF SERVICE-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

ologies focus on collecting requirements from their clients, designing services and service-oriented architecture to fulfill these requirements, and publishing and delivering the services to service consumers. Most of them do not provide any guidelines of how to consume the published services, except for SDLC that briefly explains the discovery, composition, invocation, and monitoring of the published services.

Two SOSE methodologies, Chang and SODIUM, are designed from the perspective of service consumption. They focus on collecting requirements from end users, discovering and selecting services to fulfill these requirements, and composing the selected services into service compositions. Despite that they both focus on service consumption, SODIUM also mentioned briefly how to publish these service compositions, whereas Chang does not provide any guidelines in this regard.

CBDI-SAE and SeCSE, are designed from the perspective of both service provision and consumption. CBDI-SAE consists of four disciplines, among which the provide and consume disciplines are considered equally important. Nevertheless, the consume discipline focuses on addressing business requirements and much effort has been put in collecting business requirements, improving business using services, and solution assembly. Guidance specifically on consumption-related activities (such as service discovery and service negotiation) is missing. In SeCSE, how to perform both the consumption and provision process has been explained in detail.

<table>
<thead>
<tr>
<th>Selection guideline:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowing the perspective that a SOSE methodology is designed from is of great importance for enterprises to select the one that matches their business at best. Enterprises that provide services to market users may be interested in the ones that focus on service provision; enterprises that integrate existing services to service compositions may be interested in the ones that focus on service consumption; and enterprises that aim at building SBAs with both internally developed and externally provided services may be interested in the ones that focus on both service provision and consumption.</td>
</tr>
</tbody>
</table>

SC11 Multiple organizations The SOSE process requires interaction and collaboration between multiple development roles, which might be distributed in multiple organizations. Unfortunately, most of the SOSE methodologies we evaluated do not provide sufficient guidance of how to handle the interaction, especially the collaboration between multiple organizations. CBDI-SAE, OASIS, and SDLC have mentioned the need for collaboration between multiple organizations, but details are missing. In CBDI-SAE, an organizational view of the SOSE process is suggested, illustrating the relationship between some example
7.3. COMPARISON OF EXISTING SOSE METHODOLOGIES USING THE EVALUATION FRAMEWORK

organization units (which could be located in multiple organizations as well). In this view, the tasks associated with each organization units (or organizations) are made explicit and their interaction with the tasks of other organization units (or organizations) are also given. For instance, an application delivery unit is responsible for solution assembly, which requires input from business architecture unit that is responsible for business requirements planning and improvement. OASIS emphasizes that creating service architectures is “all about creating a common dialogue between the various different groups and deciding upon a boundaries that work across the business” and suggests a collaborative working with e.g., an intensive session or a conference. SDLC indicates that business domain, is a functional domain comprising a set of current and future business processes that can collaborate with each other to accomplish a higher-level business objective, implying that collaboration is required between business domains.

SeCSE and SOAF provide relatively better guidance in this respect. In SeCSE, each engineering activity is associated to at least one development role. Depending on the description of the development role, it is easy to understand which roles might be in the same organization or different ones. For instance, a service developer and service architect might be both in an organization responsible for service provision, whereas an SBA architect might be in another organization responsible for service consumption. As the interactions between the engineering activities are described in detail, naturally the potential collaboration between the organizations becomes clear as well.

In SOAF, high level collaboration models as well as the interactions, the sequence of activities and the work-products exchanged between internal and external participants suggested to be well documented in the modeling phase; in the mapping and assessing phase, external entities need to be captured as part of the portfolio data. By explicitly distinguishing internal and external participants, work-products, and business entities, the interactions needed between multiple organizations become explicit too.

Selection guideline:
Some enterprises develop services and SBAs for internal uses. Some other enterprises have wider business scope and often share services with their business partners (consuming services provided by the others or providing service to the others). Consequently, multiple parties have to collaborate to accomplish the engineering of SBAs. For these enterprises this criterion is useful for selection.
CHAPTER 7. GUIDING THE SELECTION OF SERVICE-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

7.4 Observations

The evaluation and analysis of the methodologies highlighted some weaknesses calling for further improvement. In the following, we shall discuss our observations on where the SOSE methodologies could be improved.

- Many service life cycle activities are not well supported.
  
  By evaluating the life cycle coverage (GC2 Lifecycle) of the SOSE methodologies, we noticed that some service life cycle activities have been supported by most of the SOSE methodologies, whereas some activities have received much less support. The lack of support for these activities shows that the existing SOSE methodologies focus mainly on the design and analysis part of the SOSE process, but pay little or not sufficient attention to the constructing, delivering, and management part.

When constructing an SBA, ideally services can be dynamically discovered and composed at runtime, which often requires a well-functional service registry. As shown in Table 7.4 registry-related activities (selection, update, maintenance) are hardly supported by any of the SOSE methodologies. Without such support, an enterprise that intends to publish their services in a dedicated registry to increase service reusability and discoverability could face the risk of building up a service repository that is often outdated and does not function as expected.

When delivering an SBA, both service providers and their consumers would benefit from a service negotiation that ensures that both of them reach an agreement on the behavior of the services and on the price paid for using the services. As shown in Table 7.4, service negotiation is supported only by SeCSE. Without sufficient guidelines on service negotiation, both service providers and consumers face the difficulties when the services do not behave as expected or are not paid as agreed.

When managing an SBA, it is important that the behavior of the services can be monitored and the services can be adapted when required. Unfortunately, as shown in Table 7.4, many SOSE methodologies do not provide any support in service monitoring and service management. Without keeping track of the behavior of published or consumed services, the manager of a SBA would not be aware of runtime changes in terms of e.g., quality of services. As a result, it would be difficult to identify the needs for substituting services when failure occurs. The quality of a SBA as a whole therefore cannot be ensured.

In summary, the existing SOSE methodologies provide very weak support to aid the dynamicity of SBAs. In practice, services are often developed and
7.4. OBSERVATIONS

shared only internally to an enterprise, which heavily hinders their reusability to a wide range of consumers; services are often discovered manually at design time, which significantly decreases the efficiency of engineering SBAs; and services are often not able to dynamically adapt to new requirements or environmental changes, which obviously reduces the benefits that SOA promises.

• Development roles are significantly under-addressed. Surprisingly, many SOSE methodologies we evaluated do not provide any guidance on the development roles required in a SOSE process. As shown in Table 7.5, only three methodologies mentioned their development roles, among which only two methodologies associate the roles with the activities. Generally speaking, a methodology is a description of a process, explaining what and how (and who) to carry out a set of procedures. Especially when applying a methodology, who becomes one of the top questions from its users.

In TSE, the development roles like software architect, designer, developer, and tester are well-known and their responsibilities are often clear to practitioners. In SOSE, the process is more complex than that of TSE, involving engineering, composition, continuous adaptation, and consumption of services. Naturally, more development roles are required in that traditional development roles often split into two types of roles, one focusing on services and another focusing on SBAs (e.g., service designer and SBA designer, respectively). Moreover, some development roles are specific to SOSE and are not found in TSE; for example, service modelers, service monitors or adaptation designers are only found in SOSE. What makes it more complicated is that due to the separation of service consumption and provision and the shift in ownership of services, a development role may act from multiple perspectives simultaneously, each with different goals and competencies. For instance, a service designer with the perspective of a service provider is concerned with the identification of services to provide to others whereas the service designer with the perspective of a service consumer is concerned with the identification of services for integration purposes.

The complexity introduced in SOSE development roles results many different roles proposed in the literature. A common understanding of what types of SOSE roles are required and what their responsibilities are is currently missing. As a result, to increase the understandability and usability of a SOSE methodology, it is important that it explains in detail which roles (with what skill) should be involved and associates the roles with the SOSE activities within the proposed SOSE process.

• The support of service consumption is significantly missing.
CHAPTER 7. GUIDING THE SELECTION OF SERVICE-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

The engineering of SBAs means both service provision and service consumption. From our evaluation of the existing SOSE methodologies, we noticed that most of them focus on the provision of services, aiming at designing, developing, and delivering services that meet the requirements of their users. Service consumption is often paid little attention to. However, in order to discover, select, and invoke these services, service consumers need guidance of how to carry out these tasks. As shown in Table 7.4 the activities carried out by a service consumer is only partially covered by less than half of the SOSE methodologies. Similarly, as shown in Table 7.5, there are only two SOSE methodologies describing their process from the service consumption perspective and two from both consumption and provision perspective, the other eight methodologies all consider the SOSE process as a service provision process.

Not only service consumers need guidance on service consumption-related procedures, but also service providers need guidance of how to design and develop services that are can be consumed by many different groups of service consumers with different requirements. However, many SOSE methodologies designed from the perspective of a service provider only cover the service provision process but completely leave out the aspects related to service consumption. For instance, as shown in Table 7.5, only SDLC and SeCSE (designed from the perspective of service provision) provide some guidance on service consumption by addressing the variability of non-functional requirements from different groups of service consumers.

Regardless of the perspective that a SOSE methodology takes, service consumption is part of the SOSE process and should be considered as equally important as service provision. A SOSE methodology that specifically addresses service provision should not consider the provision process in isolation, but take into account service consumption aspects, such as discoverability, composability, negotiability, and reusability.

7.5 Related work

The need for comparing and evaluating existing SOSE methodologies has been recognized and initial research has been performed. In the survey of SOSE methodologies conducted by Ramollari et al. (Ramollari et al., 2007), a set of characteristics of SOSE methodologies has been used as their evaluation criteria. All of these criteria have been directly or indirectly covered in our evaluation framework. Criteria such as lifecycle coverage, existing process&techniques, and applied in industry can be directly mapped to our criteria: GC2 Lifecycle, GC11 Integration, and GC9 Maturity. Some other criteria such as delivery strategy
and UML are addressed by SC3 The creation of services and GC3.b Notations in our framework. There is one difference in interpreting between criteria consumer view & provider view in Ramollari et al.'s survey and our criterion SC10 Perspective: in the survey, a consumer view refers to declarative and business process oriented development through service composition and a provider view refers to programmatic and component oriented development; in our work we differentiate the consumption and provision process by 'the direction of operations' on services, either in (i.e. use) or out (i.e. delivery). The main reason is that a service provider may carry out business process oriented development and provide composite services to consumers, and a service consumer may carry out programmatic development process to integrate consumed services to service compositions or SBAs. Our interpretation of service consumption and provision is of wider scope than that of Ramollari et al.

An overview of SOSE methodologies is given by Kontogogos and Avgeriou (Kontogogos & Avgeriou, 2009). In their overview, six criteria have been used for analysis. All of them have been addressed in our evaluation framework. Criteria Lifecycle, Adaptability, and Industry are directly covered by GC2 Lifecycle, SC6 Architectural change, and Maturity. Criterion detail is addressed by multiple criteria (such as GC3.a Artifacts, GC4 Procedure, GC5 Principles, GC10 Management) in our framework to assess the degree of detail including outputs, tasks, guidelines, deliverables provided by SOSE methodologies. Although criteria Service description and Behaviour specification are addressed by our criteria GC3.a Artifacts and GC3.b Notations, they can be used to assess in detail the specification of services (as part of the artifacts). Due to the important role played by services, we consider these two criteria as being complementary to our framework.

Both works discussed above do not analyze and evaluate the service-specific features that SOSE methodologies should entail. Similar to our approach, a criteria-based evaluation framework for SOSE methodologies proposed by Gholami et al (Gholami et al., 2010) consists of generic and service-specific criteria. The generic criteria are in general similar to the ones that we proposed. The major difference is that we assess the coverage of life cycle phases more precisely by looking into the activities of each life cycle phase, whereas Gholami et al. only provide an overall score showing the degree of methodology support.

The service-specific criteria proposed by Gholami et al.'s framework and ours are significantly different. In Gholami et al.'s framework the criteria are derived from two sources: one is service-related activities (e.g., business modeling, service-oriented analysis & design, service testing, SLA monitoring), another is quality requirements (e.g., service agility, adaptable with legacy systems, process agility). Assessing the support of service-related activities and quality requirements are definitively relevant. Nevertheless, Gholami et al.'s framework does not include...
CHAPTER 7. GUIDING THE SELECTION OF SERVICE-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

all the service-related activities. For instance, service negotiation, requirements engineering from external and internal users and service registry-related activities are missing. Restricting service-specific criteria only on these two aspects limits the identification of other important features that SOSE methodologies should support, such as dealing with uncertainties, explaining development roles and their responsibilities, collaboration between multiple organizations, and the awareness of provision and consumption process. In our work, instead, we derived the service-specific criteria from a list of differences between SOSE and TSE identified from a systematic literature review. As such, we are confident that our service-specific criteria are a set of must-have service-oriented features for SOSE methodologies.

7.6 Conclusions

An enterprise that decides to adopt SOA often needs to choose a SOSE methodology to follow or to tailor for its own usage. Given many SOSE methodologies available in the literature, the question is how to select one that is both beneficial and feasible to the enterprise. The evaluation framework proposed in this work helps the enterprise in deciding how to choose a SOSE methodology and tailor it for its own usage.

For understanding the characteristics of SOSE methodologies and gaining insight in their support for service orientation, in this work we took a feature analysis approach and derived a set of generic- and service-specific criteria for comparing the existing SOA methodologies. Enterprises usually select a SOSE methodology to fulfill some specific needs. To support selection, these needs can be associated with the criteria of our framework.

The framework has been validated by analyzing 12 SOSE methodologies. Next to their comparison, this exercise helps differentiating the methodologies that are truly service-oriented (e.g., SeCSE and SDL) from those that deal little with service aspects (e.g., SOAD and SOUP). As an additional result, we extracted some guidelines that can further support selection.

The analysis of the SOSE methodologies also highlighted some weaknesses calling for further improvement: increasing the coverage of service life cycle activities, improving the support for development roles, and emphasizing the guidance on service consumption-related issues.

Lastly, no one SOSE methodology covered two criteria, GC6 Formality and GC7 Specialization. This can be due to various reasons that could be argued upon. This lack of coverage may identify either a gap in the state-of-the-art, or that our choice of criteria is too demanding, calling for further investigation in the future.
7.6. CONCLUSIONS

Whereas no one SOSE methodology we analyzed fulfills the complete set of criteria, we cannot objectively assign higher or lower priorities to any of them. In principle, an “ideal” SOSE methodology would cover all features we enlist in our evaluation framework. However, in practice enterprises that need to select SOSE methodologies can use (a subset of) these criteria (depending on their needs) to evaluate SOSE methodologies available for use.
Guiding the selection of service identification methods

In the design of services, service identification (SI) is a significant task aiming at determining (based on available resources) services that are appropriate for use in an SOA. Many service identification methods (SIMs) have been proposed to support the determination of services that are appropriate for use in an SOA. However, these SIMs vary in terms of analysis objectives, identification procedures and service hierarchies. Due to the heterogeneity of the SIMs, practitioners often face the difficulty of choosing a SIM that copes with available resources and fits their needs. To gain a holistic view of existing SIMs and to support the selection of the right SIM, this chapter presents the results of a systematic literature review. A total number of 237 studies were examined, of which 30 studies were selected as primary studies. From these studies, different types of inputs, outputs, and processes used by the existing SIMs were identified. Based on these results, a matrix that can be used in three different ways to select among alternative SIMs has been proposed.

8.1 Introduction and research questions

In the design of services, service identification (SI) is a significant task aiming at determining (based on available resources) services that are appropriate for use in an SOA. Many service identification methods (SIMs) have been proposed from both academia and industry. However, these SIMs differ significantly, ranging from source code extraction to business domain analysis, from bottom-up to top-down strategy and from ontology-based process to guideline-driven process. Accordingly, the inputs and outputs of these approaches vary as well. Some approaches start with business process whereas some others start with domain knowledge (e.g., goals and strategies); some approaches focus on business services
whereas others focus on software services.

Given many SIMs, a common question that practitioners face is how to select the most appropriate SIM that copes with the available resources and fits their needs (Erl, 2005). Some enterprises, for instance, not only have well defined business process models in place but also well documented goals, business partners and enterprise capabilities (e.g., IT resources) to support its business process. For them, a SIM that takes all of this information into account will be more suitable than those identifying services only based on e.g., business processes.

Despite the comparisons (e.g., (Kohlborn et al., 2009; Boerner & Goeken, 2009)) presented so far, none systematically searches for all the existing SIMs. As a result, a holistic overview of extant SIMs is missing. Moreover, the criteria used in the existing comparison frameworks cover many aspects, including economic, business, and technical aspects. However, a comparison of the basic elements (such as the inputs, outputs and processes) of the SIMs is currently missing. When selecting SIMs one often starts questioning what is required for using certain SIMs, what can be expected from them and how to carry them out, before addressing other requirements. Without such an overview of the basic elements of SIMs, the selection of SIMs becomes more complicated. Accordingly, the following research questions arise:

- **RQ1**: What are the existing SIMs?
- **RQ2**: How do the SIMs differ? This can be elaborated into:
  - **RQ2.a** what types of inputs do the SIMs start from?
  - **RQ2.b** what types of services do the SIMs produce?
  - **RQ2.c** what types of strategies and techniques are used by the SIMs?

To answer these research questions, we decided to conduct a systematic literature review, which are particularly powerful in collecting and analyzing existing work. It can maximize the chance to retrieve complete data sets and minimize the chance of bias and summarizing the existing evidence (Kitchenham, 2007). In the remainder of the chapter, § 8.2 describes the review protocol that we defined; § 8.3 reports on the review results; § 8.4 presents an input-output matrix that aids the selection of SIMs; and § 8.5 concludes the chapter.

### 8.2 Review protocol

A review protocol is of critical importance to a systematic review because it specifies both search, selection, data extraction and synthesis strategies. Such a protocol is essential for the reader to fully understand and appraise the review.
8.2. REVIEW PROTOCOL

It is also critical for researchers to replicate or repeat the review with other datasets (Dybå & Dingsøyr, 2008). We therefore shortly describe the review protocol in the following.

**Search strategy** The objective of this review is to explore the state of the art of SIMs. Accordingly, studies that describe in detail SI approaches should be selected as primary studies. Many studies mention SI in text but often do not propose any SIMs. To significantly reduce the number of irrelevant studies, we decided to narrow down our search to the titles and abstracts of studies, rather than full text search.

To construct a search query in titles, we use two key terms, namely *service* and *identification* (and *identify* as its synonym). The rationale behind this query is that an article is likely to have these two terms in the title if it is mainly about proposing a new SIM. To construct a search query in abstract, we noticed that there are too many articles having *service* and *identification* (or *identify*) in their abstracts, but are not about SIMs. Moreover, there are also many articles that are about SIMs but do not have the keyword *method* (or *approach, way, process* as synonyms) in their abstracts. Therefore, to narrow down our search results while preventing exclusion of relevant articles, we decided to use *service identification* as well as its two synonyms ‘*identification of services*’ and ‘*identify services*’ as the keywords. In summary, we constructed the following query string:

```
(Title:service) and (Title:identification, or Title:identify) or (Ab-
stract:“service identification” or Abstract:“identify services” or Abstract:-
“identification of services”)
```

The relevant electronic libraries are: IEEE Explore, ACM digital guide, ISI Web of Knowledge, SpringerLink and ScienceDirect, all well-known digital libraries in the field of software and service engineering.

**Study selection** After applying the query string to the search engine provided by the data sources, a study selection should be performed since articles that are irrelevant to our study should be excluded from the search results. For this purpose, we defined a set of selection criteria.

First of all, studies that will be included in our review should be written in English. Second, studies should be presented in the context of SOA. For instance, identification of social services is not of our interest. Third, studies should focus on SI. Some studies that do use the term *service identification* but actually refer to *service discovery* should be excluded. Lastly, only the studies that do provide a detailed description about the process of SI will be selected to our review. In summary, the selection criteria include: 1) English, 2) SOA, 3) *service identification*, 4) *method*. Only articles that meet all these four criteria will be selected as primary studies for review.

Among the selected primary studies, we also conduct a cross-reference check
by reviewing the related work sections. The motivation is that a study that proposes a SIM is very likely to discuss other SIMs as related work. By conducting a cross-reference check, we may be able to identify more articles that are relevant but do not appear in the search results. Moreover, we may identify articles that are relevant but not collected by the data source, such as SIMs presented as a white paper or web page.

**Data extraction and data synthesis** After identifying primary studies, we review each SIM and elicit a set of critical data elements that characterize the method. To answer *RQ2 (How do the SIM differ from each other?)* and its sub-questions, we decided to elicit data about the *input*, *output*, *strategy* and *technique* of each SIM. Moreover, the applicability and usability of the methods is also relevant for their comparison and we thereby also study the *validation* of the methods. All the elicited data is documented in a spreadsheet.

In the data synthesis step, we classify the data sets collected from the data extraction step based on their similarities. For instance, business process models in BPMN and business process diagrams in UML can be classified as belonging to the same type since they are both about business process.

Data extraction and synthesis are first performed by two researchers individually and then compared and discussed. The consolidated results are then validated by a third researcher.

### 8.3 The results of the review

To conduct the review, we followed the guidelines suggested in (Kitchenham, 2007)\(^1\). In the first step, we obtained 237 articles whose titles or abstracts contain the keywords specified in our search query. After applying the selection criteria, 38 articles were identified as primary studies that are relevant for review. By further reviewing their related work (motivated by the fact that an article presenting a SIM most likely discusses other SIMs as related work), we identified 11 more primary studies. Among these two lists of primary studies, 19 articles are duplicates and hence resulting 30 articles as identical primary studies.

#### 8.3.1 RQ 1 What are the existing SIMs?

Each primary study presents one SIM and we have identified 30 SIMs altogether. An overview of the SIMs is given in Table 8.1. As we can see, SIM S1 and SIM S2 (hereafter labeled as S1 and S2) are the pioneering SIMs presented in 2004. The

\(^1\)Due to the space limitation, we do not present the review protocol that we followed. Interested readers are referred to www.few.vu.nl/~qgu/SIMReviewProtocol.pdf for details, including the specified search query, selected electronic libraries, and strategies for data extraction data synthesis.
increasing number of SIMs being proposed in the last three years (6 in 2007, 9 in 2008 and 9 in 2009) reveals its increasing importance in the service engineering field.

Most of the SIMs have certain form of validation as shown in Table 8.1. The best way to validate a method is to put it into practice. Two primary studies describe the experience in using their methods in real life projects. Another way of validation is to experiment a SIM in case studies, which was adopted by 13 studies. In order to improve their usability, 6 primary studies provide examples in explaining how to use the proposed SIMs. For judging their quality, 4 primary studies evaluate the method in terms of e.g., survey or comparison. Only five primary studies do not discuss any validation of the proposed SIMs.

8.3.2 RQ 2.a what are the different types of inputs?

We examined each primary study and extracted information about the input of the SIMs. We found that many SIMs start from the same type of information. For instance, legacy source code and existing software application or components are both existing software assets but in different forms. In the same vein, a collection of business models, requirements, strategies and organizational structures are all about the domain knowledge of an enterprise but describing specific enterprise elements. By comparing the inputs and classifying the ones that share the same type of information, we identified seven types of inputs. These types of inputs and the SIMs that use them are summarized in Table 8.2.

The classification of the inputs shows that the resources used by SIMs often have a different scope. For instance, types data, feature and use case are more specific than application domain or business process in that the former can be derived from the latter. The number of SIMs using each type of input shows that fewer methods start with more specific or technical information.

Most of the SIMs start from business processes and enterprise level information, taking both the business and its context into consideration. This is in line with the fact that service-oriented design intends to realize software re-use through large grained services especially meant to create business value.

Legacy system is another type of input and is used by four SIMs, as shown in Table 8.2. These SIMs take a bottom-up approach to examine the architecture (e.g., S5) or source code (e.g., S27) of the existing systems for identifying services.

As we can see, most of the SIMs are either based on domain knowledge at the enterprise level (top) or existing systems (bottom). When adopting SOA, enterprises rarely start from scratch; instead, very often they need to migrate their existing systems while creating additional services to address new business requirements. Only four SIMs take a meet-in-middle approach to identify services based on a combination of legacy systems and other information type Mix, such
Table 8.1: An overview of the existing SIMs

<table>
<thead>
<tr>
<th>SIM</th>
<th>Year</th>
<th>Input type</th>
<th>Strategy</th>
<th>Output format</th>
<th>Output type</th>
<th>Technique</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 (Jain et al., 2004)</td>
<td>2004</td>
<td>Application domain</td>
<td>W2</td>
<td>WS</td>
<td>List</td>
<td>Algo</td>
<td>Evaluated</td>
</tr>
<tr>
<td>S2 (Zhang &amp; Yang, 2004)</td>
<td>2004</td>
<td>Mix</td>
<td>W2&amp;7</td>
<td>WS</td>
<td>FSP</td>
<td>Algo</td>
<td>Evaluated</td>
</tr>
<tr>
<td>S3 (Wang et al., 2005b)</td>
<td>2005</td>
<td>Legacy system</td>
<td>W7</td>
<td>WS</td>
<td>SM</td>
<td>Algo</td>
<td>No</td>
</tr>
<tr>
<td>S4 (Chen et al., 2005)</td>
<td>2005</td>
<td>Legacy system</td>
<td>W7</td>
<td>WS</td>
<td>SI</td>
<td>Anal</td>
<td>Case study</td>
</tr>
<tr>
<td>S5 (Chen et al., 2006)</td>
<td>2006</td>
<td>Data</td>
<td>W3</td>
<td>DS</td>
<td>ISP</td>
<td>Patt</td>
<td>No</td>
</tr>
<tr>
<td>S6 (Baghdadi, 2007)</td>
<td>2007</td>
<td>Application domain</td>
<td>W1&amp;4</td>
<td>BS+DS+</td>
<td>SM</td>
<td>Algo</td>
<td>No</td>
</tr>
<tr>
<td>S7 (Klose et al., 2007)</td>
<td>2007</td>
<td>Application domain</td>
<td>W2&amp;5</td>
<td>BS+DS+</td>
<td>SM</td>
<td>Algo</td>
<td>No</td>
</tr>
<tr>
<td>S8 (Chaari et al., 2007)</td>
<td>2007</td>
<td>Use case process</td>
<td>W2&amp;4</td>
<td>BS+CS</td>
<td>ISP</td>
<td>Algo</td>
<td>Example</td>
</tr>
<tr>
<td>S9 (Kohlmann &amp; Alt, 2007)</td>
<td>2007</td>
<td>Application domain</td>
<td>W2</td>
<td>BS+DS</td>
<td>ISP</td>
<td>Gline</td>
<td>Project</td>
</tr>
<tr>
<td>S10 (Inaganti &amp; Behara, 2007)</td>
<td>2007</td>
<td>Business process</td>
<td>W1</td>
<td>BS+IS</td>
<td>SM</td>
<td>Gline</td>
<td>No</td>
</tr>
<tr>
<td>S12 (Amsden, 2007)</td>
<td>2007</td>
<td>Business process</td>
<td>W2&amp;4</td>
<td>BS+CS</td>
<td>ISP</td>
<td>Algo</td>
<td>Example</td>
</tr>
<tr>
<td>S11 (Kim &amp; Doh, 2008)</td>
<td>2008</td>
<td>Mix</td>
<td>W5&amp;7</td>
<td>BS+DS+</td>
<td>SM</td>
<td>Anal</td>
<td>Case study</td>
</tr>
<tr>
<td>S13 (Kim et al., 2008)</td>
<td>2008</td>
<td>Application domain</td>
<td>W5</td>
<td>BS+CS</td>
<td>SM</td>
<td>Gline</td>
<td>Case study</td>
</tr>
<tr>
<td>S14 (Kim et al., 2008)</td>
<td>2008</td>
<td>Application domain</td>
<td>W1&amp;11</td>
<td>BS+DS+</td>
<td>SI</td>
<td>Algo</td>
<td>Case study</td>
</tr>
<tr>
<td>S15 (Kohlmann &amp; Alt, 2008)</td>
<td>2008</td>
<td>Business process</td>
<td>W1&amp;3</td>
<td>PS+DS</td>
<td>SM</td>
<td>Gline</td>
<td>Evaluated</td>
</tr>
<tr>
<td>S16 (Jamshidi et al., 2008)</td>
<td>2008</td>
<td>Business process</td>
<td>W1&amp;5</td>
<td>PS+BS+</td>
<td>List</td>
<td>Algo</td>
<td>Example</td>
</tr>
<tr>
<td>S17 (Dwivedi &amp; Kakarni, 2008)</td>
<td>2008</td>
<td>Business process</td>
<td>Feature</td>
<td>W2</td>
<td>BS</td>
<td>ISP</td>
<td>Anal</td>
</tr>
<tr>
<td>S18 (Lee et al., 2008)</td>
<td>2008</td>
<td>Feature</td>
<td>W2</td>
<td>BS</td>
<td>ISP</td>
<td>Onto</td>
<td>Case study</td>
</tr>
<tr>
<td>S19 (Lee et al., 2008)</td>
<td>2008</td>
<td>Legacy system</td>
<td>W7</td>
<td>WS</td>
<td>SI</td>
<td>Info</td>
<td>Project</td>
</tr>
<tr>
<td>S20 (Kohlmann et al., 2008)</td>
<td>2008</td>
<td>Mix</td>
<td>W1&amp;5</td>
<td>BS+DS+</td>
<td>ISP</td>
<td>Anal</td>
<td>Example</td>
</tr>
<tr>
<td>S21 (Kohlmann et al., 2008)</td>
<td>2008</td>
<td>Application domain</td>
<td>W2&amp;5</td>
<td>BS+DS+</td>
<td>ISP</td>
<td>Anal</td>
<td>No</td>
</tr>
<tr>
<td>S22 (Kohlmann et al., 2008)</td>
<td>2008</td>
<td>Application domain</td>
<td>W1</td>
<td>BS</td>
<td>List</td>
<td>Onto</td>
<td>Case study</td>
</tr>
<tr>
<td>S23 (Yousef et al., 2009)</td>
<td>2009</td>
<td>Business process</td>
<td>W1</td>
<td>BS+DS+</td>
<td>SM</td>
<td>Onto</td>
<td>Case study</td>
</tr>
<tr>
<td>S24 (Kim et al., 2009)</td>
<td>2009</td>
<td>Business process</td>
<td>W1</td>
<td>BS+DS+</td>
<td>ISP</td>
<td>Algo</td>
<td>Case study</td>
</tr>
<tr>
<td>S25 (Kim et al., 2009)</td>
<td>2009</td>
<td>Business process</td>
<td>W1</td>
<td>BS+DS+</td>
<td>ISP</td>
<td>Algo</td>
<td>Case study</td>
</tr>
<tr>
<td>S26 (Kim et al., 2009)</td>
<td>2009</td>
<td>Business process</td>
<td>W1</td>
<td>BS+DS+</td>
<td>ISP</td>
<td>Algo</td>
<td>Case study</td>
</tr>
<tr>
<td>S27 (Kim et al., 2009)</td>
<td>2009</td>
<td>Legacy system</td>
<td>W7</td>
<td>WS</td>
<td>SI</td>
<td>Onto</td>
<td>Case study</td>
</tr>
<tr>
<td>S28 (Huayou et al., 2009)</td>
<td>2009</td>
<td>Use case process</td>
<td>W2&amp;4</td>
<td>BS+CS</td>
<td>ISP</td>
<td>Gline</td>
<td>Example</td>
</tr>
<tr>
<td>S29 (Klose et al., 2009)</td>
<td>2009</td>
<td>Data</td>
<td>W3</td>
<td>BS+DS+</td>
<td>ISP</td>
<td>Gline</td>
<td>Example</td>
</tr>
<tr>
<td>S30 (Ricca &amp; Marchetto, 2009)</td>
<td>2009</td>
<td>Mix</td>
<td>W1&amp;7</td>
<td>WS</td>
<td>SI</td>
<td>Gline</td>
<td>Case study</td>
</tr>
</tbody>
</table>

### 8.3. THE RESULTS OF THE REVIEW

#### Table 8.2: Types of inputs used in the SIMs

<table>
<thead>
<tr>
<th>Type of input</th>
<th>Description</th>
<th>SIM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business process</td>
<td>A collection of related tasks or activities to fulfill a specific business requirement</td>
<td>S3, S10, S12, S15, S16, S17, S23, S24, S25, S26</td>
<td>10</td>
</tr>
<tr>
<td>Application domain</td>
<td>A collection of models or documents that describe the various aspects of an application domain, including enterprise goals or mission, business rules, business processes, business entities, organization structures, etc.</td>
<td>S1, S7, S8, S9, S14, S22</td>
<td>6</td>
</tr>
<tr>
<td>Legacy system</td>
<td>Existing software assets of an enterprise. It can be software systems, source code, or the architecture of the existing systems</td>
<td>S4, S6, S20, S27</td>
<td>4</td>
</tr>
<tr>
<td>Mix</td>
<td>A mix of type legacy system and other types</td>
<td>S2, S13, S21, S30</td>
<td>4</td>
</tr>
<tr>
<td>Data</td>
<td>The information that is processed, exchanged, or produced by business processes</td>
<td>S6, S29</td>
<td>2</td>
</tr>
<tr>
<td>Feature</td>
<td>A set of distinctive attributes or characteristics of software systems</td>
<td>S18, S19</td>
<td>2</td>
</tr>
<tr>
<td>Use case</td>
<td>A sequence of business functions that human actors benefit from</td>
<td>S11, S28</td>
<td>2</td>
</tr>
</tbody>
</table>

as domain analysis in S2 and business process models in S30. This low number is contradictory to the comparison of service analysis approaches reported in (Kohlborn et al., 2009), which pointed out that most approaches postulate a meet-in-the-middle strategy. The cause for this contradiction lies in the fact that in our review we selected only the SIMs that provide detailed description of the methods. In (Kohlborn et al., 2009) such a criterion does not apply and many approaches that only conceptually discuss their SI strategies have been selected for comparison. It has been admitted by the authors in (Kohlborn et al., 2009) that many approaches do fail to go into detail. Despite of the equal importance of existing software assets and business requirements, only few SIMs provide enough guidance on how to take into account both of them.

### 8.3.3 RQ 2.b what are the different types of services being produced?

In this section, we discuss the outputs of the SIMs, including the *types of services* produced and how these services are described (*output format*).

**Types of services** The general goal of SIMs is to identify services that are valuable, either from business or IT perspective. Each individual SIM has a specific goal in identifying specific types of services. For instance, some SIMs target at services that represent business processes whereas others focus on identifying services that bridge the gap between business services and IT infrastructure. By studying, comparing and synthesizing the data about the objectives of the services
produced by the SIMs, we identified 6 types of services that have been explicitly discussed, summarized in Table 8.3 (note that a SIM may identify multiple types of services).

Table 8.3: Types of outputs produced by the SIMs

<table>
<thead>
<tr>
<th>Type of output</th>
<th>Description</th>
<th>SIM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business process service (BS)</td>
<td>A service that has the business logic or represents a business process, including task services, process services.</td>
<td>S3,S7,S8,S9,S10,S11,S12,S13,S14,S15,S17,S18,S19,S21,S22,S23,S24,S25,S26,S28,S29</td>
<td>21</td>
</tr>
<tr>
<td>Data service (DS)</td>
<td>A service that represents business-centric entities, including information services, entity services</td>
<td>S6,S7,S8,S9,S13,S15,S16,S17,S21,S22,S24,S25</td>
<td>12</td>
</tr>
<tr>
<td>Composite Service (CS)</td>
<td>A composition of multiple services.</td>
<td>S7,S8,S11,S12,S13,S14,S15,S17,S21,S25,S28,S29</td>
<td>12</td>
</tr>
<tr>
<td>IT service (IS)</td>
<td>A service that represents technology-specific functionalities, including application services, software services, utility services and infrastructures.</td>
<td>S3,S7,S8,S9,S10,S13,S17,S22,S24</td>
<td>9</td>
</tr>
<tr>
<td>Web service (WS)</td>
<td>A service that is implemented using the web service technology. This type is orthogonal to the other types</td>
<td>S1,S2,S4,S5,S20,S30</td>
<td>7</td>
</tr>
<tr>
<td>Partner service (PS)</td>
<td>A service that is offered to external partners.</td>
<td>S3,S16,S17,S26,S29</td>
<td>5</td>
</tr>
</tbody>
</table>

From a business perspective, we can see that 21 SIMs result in business services representing business processes, and 12 result in data services for business entities. Both of these two types of services are business-related. Because of the nature of services (i.e. exposing business functions), it is quite understandable that a large number of SIMs focus on business.

On the other hand, it is worth noticing that the numbers of SIMs for identifying IT services and partner services is relatively low and lower than we expected. Business-IT alignment is recognized as a research challenge (in Chapter 2) and the need of integrating technical architecture (e.g., IT infrastructure, data models) with business architecture (e.g., business models, organizational structure) has been widely agreed (Bieberstein et al., 2005; Erl, 2005). As shown in Table 8.3, all 9 SIMs that do consider IT services also identify business services and more importantly, they pay specific attention to the integration of business and IT. This alignment should, in our opinion, be supported by all the SIMs, which points out a gap in those SIMs lacking support for IT services.

As for partner services, we see that only 5 SIMs lead to services that explicitly consider their service providers (SPs) and consumers (SCs). Services are designed, developed, provided, maintained and owned by SPs. The task of SCs is to discover and compose services fulfilling their needs. SI for SPs entails how to identify
services that are potentially useful to others. For SCs, it, instead, entails how to identify services for integration purposes. Because of this difference, a SIM should explicitly indicate which role it supports. Unfortunately, most of the SIMs fail to highlight this difference. Despite that the separation of SPs and SCs is considered as one of the main characteristics of SOA, these two roles are often not explicitly considered in service engineering in general, as found by a survey of SOA methodologies that we reported in Chapter 7.

Further, 7 SIMs explicitly aim at identifying web services without describing any of the types described above. That is why we regard web service as a special type, orthogonal to the others. Interestingly, nearly all of these SIMs (except for S1) rely on legacy systems.

**Types of output format** Different SIMs also describe the identified services in multiple ways and in multiple levels of detail. Some SIMs describe their output in terms of a list of services with short descriptions; whereas some others illustrate their output in terms of a model describing the relation between services and sometimes with other artifacts of the system. To understand better the outputs of the SIMs, we analyzed the ways that the SIMs describe their identified services. As a result, five different ways of describing services have been identified, as summarized in Table 8.4.

<table>
<thead>
<tr>
<th>Output format</th>
<th>Description</th>
<th>SIM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informal service specification</td>
<td>Specify the identified services with a list of terms, such as service description, input, output, etc</td>
<td>S2, S7, S21, S29, S6, S9, S11, S18, S19, S25, S28</td>
<td>11</td>
</tr>
<tr>
<td>Service model</td>
<td>Model the identified services in terms of diagrams, illustrating the service landscape</td>
<td>S3, S8, S10, S12, S13, S14, S16, S24</td>
<td>8</td>
</tr>
<tr>
<td>Formal service specification</td>
<td>Describe the identified services using standard language, such as WSDL</td>
<td>S2, S15, S20, S26</td>
<td>4</td>
</tr>
<tr>
<td>Service implementation</td>
<td>Implement the identified services</td>
<td>S4, S5, S27, S30</td>
<td>4</td>
</tr>
<tr>
<td>A list of services</td>
<td>List the identified services with several key elements, such as name, operation, etc</td>
<td>S1, S17, S23</td>
<td>3</td>
</tr>
</tbody>
</table>

From Table 8.4 we can see that many identified services are described in *informal service specification*. However, different SIMs often use different terms for specifying the identified services. For instance, in S11 services are specified in detail using many terms including functional and non-function description, and technique-related aspects, such as operations and standards. In S6, however, services are specified only in terms of their operation and in/out messages. Some SIMs describe their output using only a *list of services*, without entering their
CHAPTER 8. GUIDING THE SELECTION OF SERVICE IDENTIFICATION METHODS

details. While we do not enter the merit of one or the other approach, we observe that there is no unified way for describing services.

A service model is used by 8 SIMs, with the purpose of illustrating the relation between the identified services (e.g., S8) and the relation between services and their providers (e.g., S16). Thanks to its power of illustrating relations, a service model is extremely useful for describing composite services (CSs) and partner services (PSs). As shown in the overview of the SIMs (given in Table 8.1), only 5 (out of 12) SIMs identifying CSs, and 2 (out of 5) SIMs that identifying PSs use the form of a service model to describe the service landscape. In our opinion, a service model should be used more often as long as CSs and PSs are concerned.

Some other formats of output produced by the SIMs are implementation-driven. One of such formats is formal service specification, often used to describe services identified under more formal techniques (e.g., algorithms used by S15; another format is service implementation, often used when services are executable programs created by wrapping source codes of legacy systems (e.g., S30).

8.3.4 R2.c what types of strategies and techniques?

The previous two research questions (RQ2.a and b) mainly focus on what is involved in the SIMs. In this section we shall focus on how to carry out the SIMs. In this chapter, we define strategy as the style, approach or plan for identifying services; we define technique as the technical procedures or actions taken to accomplish certain tasks defined in a SIM.

Strategies Hubbers et. al (Hubbers et al., 2007) suggested ten common ways (or strategies) for identifying services. To find out if these ten ways are indeed used by the SIMs, we analyzed the data elicited from the primary studies and mapped all of the SIMs on at least one of these ways. We also identified one way (W11) that has not been discussed in (Hubbers et al., 2007). All these ways and their use by SIMs are given in Table 8.5 (note that a SIM may use multiple strategies).

Four strategies (W6, 8, 9, 10) are discussed in (Hubbers et al., 2007) but have not been used by any of the SIMs that we studied. As explained in (Hubbers et al., 2007), W6 has practical difficulties due to the different nature of component and services; W8 might be risky if existing application design is of low quality; and W9 results in services heavily coupled with infrastructure. Due to these issues, it is understandable that the SIMs avoid using these strategies. W10 points out the importance of non-functional requirements (NFRs) in SI as conflicting NFRs might cause redesign. As no SIM relies on W10, further research is required in supporting NFRs.

Interestingly, we also identified a new strategy, W11 user interface (not discussed in (Hubbers et al., 2007)). User interface (UI) design, an integral part of
## 8.3. THE RESULTS OF THE REVIEW

Table 8.5: Strategies used by the SIMs

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Description</th>
<th>SIM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1 (Business Process Decomposition)</td>
<td>Decompose business process models that depict how the work is done within an organization</td>
<td>S3,S7,S10,S12,S15,S16, S17,S21,S23,S24,S25, S26,S30</td>
<td>13</td>
</tr>
<tr>
<td>W2 (Business Functions)</td>
<td>Decompose business function models that depict what an organization does</td>
<td>S1,S2,S8,S9,S11,S18, S19,S22,S28</td>
<td>9</td>
</tr>
<tr>
<td>W3 (Business Entity Objects)</td>
<td>Model services according to business object models</td>
<td>S6,S29,S21,S16</td>
<td>4</td>
</tr>
<tr>
<td>W4 (Ownership and Responsibility)</td>
<td>Take the ownership of processes into consideration when identifying services</td>
<td>S7,S11,S28</td>
<td>3</td>
</tr>
<tr>
<td>W5 (Goal driven)</td>
<td>Decompose a company’s goals down to the level of services</td>
<td>S8,S14,S13,S22</td>
<td>4</td>
</tr>
<tr>
<td>W6 (Component-Based)</td>
<td>Identifies services based on components</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>W7 (Existing Supply)</td>
<td>Identify services from the functionality provided by existing legacy applications</td>
<td>S2,S4,S5,S13,S20,S27, S30</td>
<td>7</td>
</tr>
<tr>
<td>W8 (Front-Office Application Usage Analysis)</td>
<td>Select a set of applications that support business processes and extracts comparable functions into a single service</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>W9 (Infrastructure)</td>
<td>Keep the technical infrastructure into consideration when identifying services</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>W10 (NFRs)</td>
<td>Use non-functional requirements as the primary input to identify services</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>W11 (User interface)</td>
<td>Identify services based on the design of user interface</td>
<td>S15</td>
<td>1</td>
</tr>
</tbody>
</table>
CHAPTER 8. GUIDING THE SELECTION OF SERVICE IDENTIFICATION METHODS

the software design process, is often considered out of scope of SOA design (Mani et al., 2008). However, a UI design helps one to distinguish the purely presentation aspects from data and process aspects and hence aids the identification of services. The use of UI design in SI is regarded as an innovative approach.

Some SIMs use only one strategy (W1, 2, 3 or 7) and we call these strategies primary strategies; while the others are always used in combination with the primary strategies and we call them complementary strategies. The most popular primary strategies are W1, 2 and 7, used by 13, 9 and 7 SIMs respectively. The first two are top-down approaches by examining the business requirements while the last one represents a bottom-up approach by extracting valuable and/or reusable functions from existing applications. In most of the cases, the strategies used by a SIM are directly related to its input. E.g., all the SIMs that use business process as their input use W1 (decompose business process) as its strategy, only S15 exceptionally uses a combination of W4 and W11.

The complementary strategies are W4, 5 and W11 which are specifically aided by the information about goals, stakeholders and user interfaces. However, this information alone is often not sufficient for identifying services. For instance, in S15 user interfaces are first designed based on business process decomposition and then services are identified by extracting abstract WSDL specifications from user interface specifications. Obviously using W11 (user interface) only is not sufficient in this example. The use of these complementary strategies often results in services more business-driven as explained in e.g., (Klose et al., 2007).

Techniques After studying the data about the techniques used in the SIMs, we have identified six different types of techniques, summarized in Table 8.6.

Table 8.6: Techniques used by the SIMs

<table>
<thead>
<tr>
<th>Technique</th>
<th>Description</th>
<th>SIM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>A formal approach to problem solving, such as heuristic or formalized rules</td>
<td>S1, S2, S3, S5, S8, S11, S15, S16, S17, S25, S26</td>
<td>11</td>
</tr>
<tr>
<td>Guidelines</td>
<td>A set of pre-defined regulations, such as criteria or policies; suggested but not codified</td>
<td>S7, S9, S10, S14, S26, S29, S30</td>
<td>7</td>
</tr>
<tr>
<td>Analysis</td>
<td>A process of studying, interpreting, reasoning and synthesizing data</td>
<td>S4, S12, S13, S18, S21, S22</td>
<td>6</td>
</tr>
<tr>
<td>Ontology</td>
<td>A technique to conceptually represent (domain) knowledge</td>
<td>S24, S23, S19, S27</td>
<td>4</td>
</tr>
<tr>
<td>Pattern</td>
<td>Defined as recurring solution to recurring problems</td>
<td>S6</td>
<td>1</td>
</tr>
<tr>
<td>Information manipulation</td>
<td>A text process techniques for identifying or eliciting useful information, such as information retrieval or textural similarity analysis</td>
<td>S20</td>
<td>1</td>
</tr>
</tbody>
</table>

Some of them are more formal, in the sense that they formally codify for-
8.4. AN INPUT-OUTPUT MATRIX FOR THE SELECTION OF SIMS

Formulas or rules to specify the way that services are identified, such as *algorithm, ontology, pattern* and *information manipulation*. Nearly half of the SIMs use these formal techniques and thanks to the advantage of codification, some SIMs partially automate the SI process. For instance, in S17 a tool called SQUID was developed to automate the process of SI and in S2 executable algorithms are used to analyze the legacy code.

A less formal technique is *guidelines*, which is used by 7 SIMs as shown in Table 8.6. These SIMs provide advices like how to identify the right-grained services from goal-scenario models (S14) and how to map tasks in business process models to services (S30).

Different from these relatively formal techniques, *analysis* is a technique that is more abstract and requires its users to deeply understand the problem they face and make motivated decisions. Accordingly, the subjectivity of using the technique is relatively high and different actors may achieve different results by applying the same SIM.

8.4 An input-output matrix for the selection of SIMs

The variety in the types of inputs, outputs and processes discussed in §8.3 explains why practitioners often face difficulties to select a SIM that both fits their needs and copes with the available resources. To help compare and select among the SIMs, we use these results (summarized in Table 8.1) to created an input-output matrix. The matrix is presented in Table 8.7, where rows represent the types of outputs produced by the SIMs and columns represent the types of inputs being used. Each cell of the matrix describes a specific SIM (in terms of its output format, strategy and technique) if it uses the input and produces the output represented by the column and row respectively. For instance, a SIM that uses *Application domain* as its input (column 1) and produces *Business services as well as composite service (BS+CS)* (row 2) is S14, whose output is described in terms of a service model (SM), uses *goal driven* (W5) as its strategy and *guidelines* as technique. In the following, we shall explain how this matrix aids the selection of SIMs in three different ways.

**Selection driven by the targeted output** Sometimes, before performing the task of SI, it is expected that certain types of services are identified. For instance, an enterprise that focuses on improving the efficiency and maintainability of its business processes may be interested in business services; while an enterprise that intends to expose its business functions to other partners for business collaboration might also be interested in partner services. In our input-output matrix, the SIMs are classified horizontally, according to the types of services they produce. When the target types of services to be identified are decided, one can use the matrix to eliminate those SIMs that are irrelevant to the needs.
CHAPTER 8. GUIDING THE SELECTION OF SERVICE IDENTIFICATION METHODS

Table 8.7: Input-output matrix of the SIMs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BS</td>
<td>BS</td>
<td>AP1(AFM, W5, Onto)</td>
<td>AP2(AFM, W1, Onto)</td>
<td>AP3(AFM, W1, Onto)</td>
<td>AP4(AFM, W1, Onto)</td>
<td>AP5(AFM, W1, Onto)</td>
<td>AP6(AFM, W1, Onto)</td>
<td>AP7(AFM, W1, Onto)</td>
</tr>
<tr>
<td>BS+CS</td>
<td>BS+CS</td>
<td>AP9(AFM, W5, Onto)</td>
<td>AP10(AFM, W1, Onto)</td>
<td>AP11(AFM, W1, Onto)</td>
<td>AP12(AFM, W1, Onto)</td>
<td>AP13(AFM, W1, Onto)</td>
<td>AP14(AFM, W1, Onto)</td>
<td>AP15(AFM, W1, Onto)</td>
</tr>
<tr>
<td>BS+DS</td>
<td>BS+DS</td>
<td>AP17(AFM, W5, Onto)</td>
<td>AP18(AFM, W1, Onto)</td>
<td>AP19(AFM, W1, Onto)</td>
<td>AP20(AFM, W1, Onto)</td>
<td>AP21(AFM, W1, Onto)</td>
<td>AP22(AFM, W1, Onto)</td>
<td>AP23(AFM, W1, Onto)</td>
</tr>
<tr>
<td>BS+IS</td>
<td>BS+IS</td>
<td>AP25(AFM, W5, Onto)</td>
<td>AP26(AFM, W1, Onto)</td>
<td>AP27(AFM, W1, Onto)</td>
<td>AP28(AFM, W1, Onto)</td>
<td>AP29(AFM, W1, Onto)</td>
<td>AP30(AFM, W1, Onto)</td>
<td>AP31(AFM, W1, Onto)</td>
</tr>
<tr>
<td>DS</td>
<td>DS</td>
<td>AP33(AFM, W5, Onto)</td>
<td>AP34(AFM, W1, Onto)</td>
<td>AP35(AFM, W1, Onto)</td>
<td>AP36(AFM, W1, Onto)</td>
<td>AP37(AFM, W1, Onto)</td>
<td>AP38(AFM, W1, Onto)</td>
<td>AP39(AFM, W1, Onto)</td>
</tr>
<tr>
<td>PS+BS</td>
<td>PS+BS</td>
<td>AP41(AFM, W5, Onto)</td>
<td>AP42(AFM, W1, Onto)</td>
<td>AP43(AFM, W1, Onto)</td>
<td>AP44(AFM, W1, Onto)</td>
<td>AP45(AFM, W1, Onto)</td>
<td>AP46(AFM, W1, Onto)</td>
<td>AP47(AFM, W1, Onto)</td>
</tr>
<tr>
<td>PS+CS</td>
<td>PS+CS</td>
<td>AP49(AFM, W5, Onto)</td>
<td>AP50(AFM, W1, Onto)</td>
<td>AP51(AFM, W1, Onto)</td>
<td>AP52(AFM, W1, Onto)</td>
<td>AP53(AFM, W1, Onto)</td>
<td>AP54(AFM, W1, Onto)</td>
<td>AP55(AFM, W1, Onto)</td>
</tr>
<tr>
<td>PS+DS</td>
<td>PS+DS</td>
<td>AP57(AFM, W5, Onto)</td>
<td>AP58(AFM, W1, Onto)</td>
<td>AP59(AFM, W1, Onto)</td>
<td>AP60(AFM, W1, Onto)</td>
<td>AP61(AFM, W1, Onto)</td>
<td>AP62(AFM, W1, Onto)</td>
<td>AP63(AFM, W1, Onto)</td>
</tr>
<tr>
<td>PS+IS</td>
<td>PS+IS</td>
<td>AP65(AFM, W5, Onto)</td>
<td>AP66(AFM, W1, Onto)</td>
<td>AP67(AFM, W1, Onto)</td>
<td>AP68(AFM, W1, Onto)</td>
<td>AP69(AFM, W1, Onto)</td>
<td>AP70(AFM, W1, Onto)</td>
<td>AP71(AFM, W1, Onto)</td>
</tr>
<tr>
<td>WS</td>
<td>WS</td>
<td>AP73(AFM, W5, Onto)</td>
<td>AP74(AFM, W1, Onto)</td>
<td>AP75(AFM, W1, Onto)</td>
<td>AP76(AFM, W1, Onto)</td>
<td>AP77(AFM, W1, Onto)</td>
<td>AP78(AFM, W1, Onto)</td>
<td>AP79(AFM, W1, Onto)</td>
</tr>
</tbody>
</table>

8.4. AN INPUT-OUTPUT MATRIX FOR THE SELECTION OF SIMS

Suppose partner services (PS) are of great importance, one can see from our matrix that five SIMs (row 8 to 12) could be selected. By determining more types of services to be identified (e.g., DS), one can further narrow down the number of candidate SIMs (e.g., S17 and S16 at row 10 and 12). Further, the matrix also provides a straightforward view on what types of inputs are needed if certain SIMs are selected. For example, we can see from the matrix that to identify partner services, either business processes or business-centric entities (data) should be known. This helps one to judge the feasibility of the SIMs.

Selection driven by the available resources Knowing what kinds of resources are available for SI, one can also choose a SIM based on its starting point. In the input-output matrix, the SIMs are classified vertically, according to the types of inputs they start with. Using the matrix, one can have an overview of what types of services can be produced given the available resources and at the same time find out the SIMs that support these resources. For instance, if the only available resource is a set of use cases (see Table 8.7 column 7) describing some business functions, one can find from our matrix that two SIMs, S11 and S28 (column 7, row 2) start SI from this resource. Accordingly, one can also expect that business services and their compositions (BS+CS) can be identified using either of these two SIMs.

Selection by comparison of alternative SIMs Some SIMs can be seen as alternative methods when they share the same type of inputs and outputs. Despite of these commonalities, these methods often differ in the way that the outputs are described, and/or the strategy and technique they use. Using the input-output matrix, one can easily pinpoint and compare these alternative methods since they are grouped and located in the same cell of the matrix. For instance, given legacy system (column 3) as starting point and web services (row 13) as output, the matrix shows four SIMs: S4, S5, S20 and S27. Comparing these four SIMs in terms of their output formats, strategies and techniques, we can see that the main difference lies in the techniques they use. Therefore, one can choose among these four SIMs based on their preference of one technique over another, depending available technologies, competencies in place, etc. To give another example, given application domain (column 1) as starting point and BS+DS+IS (row 4) as output, our matrix shows two SIMs: S9 and S22. By comparing these two SIMs, it is also easy to see that S22 uses W5 (goal driven) complementary to W2 (business functions) as its strategy; and thereby an enterprise that have clearly defined business goals might opt to select S22 over S9. However, if the enterprise prefers to follow guidelines to provide relatively objective results than to perform analysis to produce relatively subjective results, it might select S9 over S22. As such, our matrix provides a way to preliminarily select alternative SIMs before more comprehensive comparison (if needed).
8.5 Conclusions

In this chapter, we report the classification and comparison of 30 SIMs identified from a systematic literature review. The many different types of inputs, outputs and processes of the SIMs show a significant heterogeneity. Our results provide a holistic overview of the SIMs and highlight their differences. To help practitioners compare and select from these SIMs, we created an input-output matrix that aids the selection of SIMs in three different but complementary ways.

Further, the findings of this review outlines future research directions to further improve the existing SIMs and to guide the design of new SIMs. Our main observations are 1) IT services that leverage business processes and underlying IT infrastructure require more attention (§ 8.3.3); 2) Services for internal use and external consumption should be differentiated due to their different characteristics (§ 8.3.3); and 3) NFRs should be explicitly considered due to their importance through the entire service life cycle (§ 8.3.4).
A template for SOA design decision making in an educational setting

An effective software design blueprint should ensure that all the quality requirements of a system of interest are supported. As the architecture of a software system can be seen as a set of architectural design decisions (ADDs), the identification of design issues and the choice of design alternatives heavily impact the quality of software systems. This is especially challenging in service orientation for which students need also to change their mindset from system design to service-oriented design. In this chapter, we present a template-driven approach for documenting quality-driven ADDs in an educational setting. We report on our experience with the usage of the template in a service-oriented software design Master course given over the past three years. We found that the template not only offers a framework guiding the students in their design but also encourages them to make sound ADDs, which leads to a service-oriented design with higher quality.

9.1 Introduction

As architectural design decisions (ADDs) are viewed as first class entities (Lee, 2007; Tyree & Akerman, 2005), the quality of a software system is directly determined by the quality of such design decisions (Babar, 2009; Hordijk et al., 2004), i.e. by the relevance of the associated design issues and the choice of design options. Design issues are problems that need to be solved in architecture design. For instance, how to prevent the loss of data caused by potential database failure is a design issue. If important issues are overlooked, concerns of stakeholders remain unaddressed in the architectural design, which leads to software systems with low satisfaction and most likely low quality as well. Design options are alternative solutions that could solve a problem (design issue). The decided solution
CHAPTER 9. A TEMPLATE FOR SOA DESIGN DECISION MAKING IN AN EDUCATIONAL SETTING

will be implemented and directly determines the quality of software systems. The quality of a software system is measured against quality requirements or non-functional requirements (NFRs). Unlike functional requirements (FRs) which are often promised by software designers, in practice NFRs are often regarded as soft goals and postponed to late stages of the software life cycle (Xu et al., 2005; Cysneiros et al., 2003). Without being thoroughly considered in the first place, NFRs may not be met after software systems are developed. Changing architectural design to meet NFRs late in the software life cycle is costly and only increases the complexity of software design because some design decisions act as constraints for other decisions. In the SOA domain, NFRs are major drivers in the design of SBAs: they must be stated at design time in e.g., service contracts and met at runtime during service discovery and composition.

Since SOA is an emerging architectural style, a mature set of SOA design principles is still lacking in the literature (Legner & Vogel, 2007). Applying traditional software design principles (e.g., separation of concerns) in architectural decision making is already difficult for students. Moreover, reusable SOA designs with supported ADDs are limited in the literature. The few SOA design decisions reported in the literature (Zimmermann et al., 2004b; Zimmermann et al., 2006) could provide some useful examples. However, these decisions are often technology-specific (e.g., J2EE vs LAMP and EJB vs. plain Java object in (Zimmermann et al., 2006)). Although made in the context of SOA projects, those decisions are not per se SOA specific as they could appear in software systems with other architectural styles as well. SOA decision making remains an open research topic in the SOSE community. On top of that, next to their lack of experience students are taught to design and develop software systems instead of software services or SBAs. The students have to shift their mindset from system design to service-oriented design, for which they need ways to build up experience by identifying, and reasoning on, a range of possibilities.

To this aim, in service-oriented software design education we see the need to create the awareness of the link between quality attributes and ADDs as well as to provide an approach to support quality-driven software architectural design. With this goal in mind, we have defined a template for documenting quality-driven ADDs. This has been experimented in a software design Master course at our university.

This chapter presents the template and reports on its use over the last years. Based on the the feedback from the students and the resulting software designs, it is evident that the template indeed aids the students in focusing on service-oriented design as well as addressing quality requirements. In addition to a tool for documenting and representing ADDs, by enforcing explicit link between quality requirements and ADDs the template guides and inspires the students to identify design issues that are specific to SOA. These observations on using the
9.2 Background

We have been giving a service-oriented software design Master course yearly. The first part of the course provides the students with the theory about designing and documenting large and modern software systems. To this end, various design techniques are taught, including the use of software design reasoning tactics and documenting the architectural knowledge pertaining a software design solution. In the second half of the course, the students carry out a design project in teams of three.

This practical design project is the context of the experiment we describe. All teams were given the same description of a broad design assignment that each team had to develop into a service-oriented software design addressing the FRs as well as NFRs that the students elicited from the business case description. At the end of the course, each team should document their design in three sections: problem space, design space, and solution space, describing (functional and non-functional) requirements elicited from the case, ADDs made during the design, and resulted architectural design respectively.

Across the years we noticed the necessity of providing the students with a tool to document their ADDs. Without such a tool, ADDs are documented in an unstructured way. Some students missed rationale and some made ADDs that were not driven by quality requirements. The lack of a structured way of documenting ADDs not only led to incompletely documented ADDs but also created difficulties for us to correct the assignments.

As explained by Shahin et al. (Shahin et al., 2009) in their comparison of existing ADD models and tools, ADDs can be defined by multiple sets of elements. Different sets of elements might encompass the same concepts but with different terminologies. To avoid the confusion of these different terminologies and to ensure that the students use the same sets of elements to document their ADDs (for sharing purpose), we decided to provide them a tool from 2007.

In order to seamlessly support the students in focusing on learning software design and give them guidance, the tool should achieve the following goals: 1)
allow the students to record ADDs in a structured way to support structured reasoning, 2) enable explicit links between quality attributes and ADDs to support traceability, and 3) encourage explicit motivation on the identification of service-specific design issues to facilitate focus on service-oriented design.

Given these three goals, we devised a template as a tool for supporting the documentation of ADDs specifically for education purposes. This template (presented in § 9.3) is of relatively narrow scope, as compared to existing tools and templates in the literature, which are more suitable for use in business or enterprise context. We shall elaborate on this in § 9.5.

9.3 A template for documenting quality-driven SOA design decisions

In this section, we present the template we devised for the students to document their ADDs. Being a template, it naturally fulfills the requirement of providing a structured way to document ADDs (goal 1) in that each ADD is required to be described according to a pre-defined set of elements.

The underlying ADD model we used is the core model suggested by de Boer et al. (de Boer et al., 2007). The core model has been proven suitable to serve as a reference model for sharing architecture knowledge. More importantly, the core model allows to “describe quality in terms of architectural decisions and their effect on the software product”. Therefore, we created the template for documenting ADDs based on this core model.

As a reference model for sharing architecture knowledge, the core model is comprehensive and covers all the essential elements for describing an architectural design. For education purposes, the template should help students focus on essential information about ADDs (such as decision, constraint, solution, and rationale - regarded as four major ADD elements in (Shahin et al., 2009)) as well as information that improves the understandability of ADDs (such as the motivation of a design issue). To this aim, we selected a part of the core model which focuses mainly on the concerns, alternatives and decisions; but left out the part describing their relation with stakeholders and architectural design as artifacts.

According to (de Boer et al., 2007), a design topic refers to design issues or problems that need a decision to be taken; a design topic may propose multiple alternatives as the solution to the design issue or problem; a concern states an area of interest of stakeholders relating to certain design topic; based on the concern, alternatives can be ranked and chosen based on ranking; the chosen alternative is ultimately a decision for the design topic.

Based on this part of the core model, we devised a template for documenting
9.3. A TEMPLATE FOR DOCUMENTING QUALITY-DRIVEN SOA DESIGN DECISIONS

ADDs and presented it in Table 12.1. In this template, the concept of design topic in the core model is represented by two entities: design issue and context. A design issue states an architectural problem that needs to be solved; while the context of the design issue refers to circumstances that form the setting for the design issue. Although these two entities can be expressed together as a design topic, documenting the context of a design issue separately from the issue itself enforces students to motivate their selection of design issues and to facilitate them to focus on service-oriented design (goal 3). For instance, a design issue concerning secured communication can be addressed differently if the context is local or distributed across multiple service providers. By motivating the design problem caused by multiple service providers in the context of SOA, the students’ mindset is naturally shifted to service orientation.

The concept of concern in the core model is represented by the entity quality attributes. Quality attributes are the NFRs that students identified based on the business case description and regarded as important requirements to be considered in their architectural design. The quality attributes defined in this entity serve as ranking criteria for the selection of design decisions. Each quality attribute is required to be named with an identifier, whose name convention has been defined in the template.

The concept of alternatives in the core model is represented by a list of architectural options as defined in Table 12.1. Each option is described by six entities: identifier, description, relationship(s), evaluation, rationale and status. The identifier of each option is used for reference purposes. An option is explained in detail in the description entity. The relationship(s) entity can be used when the option has certain relationship with any other options or decisions of other design issues. Each option should be evaluated against the quality attributes and the results are reported in the evaluation entity. Based on this evaluation, justification on whether such an option should be accepted or not is reported in the rationale entity. The accepted option corresponds to the concept of design decision in the core model. At the end, the status entity highlights whether the option is chosen or rejected.

When quality attributes related to an ADD are explicitly listed and the evaluation of each design option against each quality attribute is also explicitly documented, the link between quality attributes and ADDs naturally becomes explicit as well (goal 2).

The proposed template should be repeatedly employed to document each design issue identified during the software design as well as its possible options and an accepted decision. To give an example of a documented ADD, Table 9.2 presents an example ADD selected from a service-oriented design made by a group of students in 2009. This ADD has been made for a E-Health system, whose business description is described in (Di Nitto et al., 2009).
CHAPTER 9. A TEMPLATE FOR SOA DESIGN DECISION MAKING IN AN EDUCATIONAL SETTING

Table 9.1: A template for documenting quality-driven architectural design decisions

<table>
<thead>
<tr>
<th>Design issue</th>
<th>D#&lt;number&gt;</th>
<th>What was the design issue that needed to be solved (by taking a decision)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td></td>
<td>Motivate why the design issue has been selected under the context of the software system.</td>
</tr>
<tr>
<td>Quality attributes</td>
<td>Cr#&lt;number&gt;</td>
<td>what quality attributes have been used to select the decision based on the available options.</td>
</tr>
<tr>
<td>Architectural options (Repeat for each option)</td>
<td>Identifier D#&lt;number&gt;-Opt#&lt;number&gt;</td>
<td>Name of the option</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Short description of the option</td>
</tr>
<tr>
<td></td>
<td>Relationship(s)</td>
<td>Indicate relationships with other options (by using their identifiers): forbids, conflicts with, enables, subsumes, is related to</td>
</tr>
<tr>
<td>Status</td>
<td>Has this option been accepted or rejected?</td>
<td></td>
</tr>
<tr>
<td>Evaluation</td>
<td>Cr#&lt;number&gt;</td>
<td>To which extent does this option support ranking criterion Cr#&lt;number&gt;?</td>
</tr>
<tr>
<td>Rationale</td>
<td>Why has this option been accepted or rejected? (use the ranking criteria identifiers in the argumentation)</td>
<td></td>
</tr>
</tbody>
</table>

Legend: D#: design issue; Cr#: criteria; D#-Opt#: architectural design option.

This example shows that an architect is concerned with handling data requests for external resources. This concern arises primarily from the fact that services of the E-Health system are distributed among multiple business partners and thereby frequent data requests from partners external to the hospital are costly. Based on two quality attributes as ranking criteria: consistency and performance, the design options (sending requested data when needed, and caching requested data) have been evaluated. As shown in Table 9.2, the second option has been decided thanks to its performance advantage. Although this option would potentially cause some inconsistency problems, the consistency requirement can be met when time interval is optimal and types of data to be cached are appropriate. As a result, the students identified two other design issues, requiring corresponding decisions.

From this example we can see that quality attributes are not only the ranking criteria for the selection of design options, but they also inspire the identification of relating design issues.

9.4 Results on the use of the template

The template presented in the previous section has been used by the students during a service-oriented design Master course over the past three years. The results on the use of the template have been collected from the following two sources.
9.4. RESULTS ON THE USE OF THE TEMPLATE

Table 9.2: How do we handle data requests for external resources?

<table>
<thead>
<tr>
<th>Design issue</th>
<th>D6: How do we handle data requests for external resources?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>The E-Health system is a service-based system which often need to use external resources to retrieve certain data that could not be supplied by services internal to the system. E.g., to retrieve information about new medications the E-Health system will need to request pharmaceutical services provided by external service providers. Requesting data from an external resource, outside the internal network, requires additional communication. An effective way of handing these external data request is needed.</td>
</tr>
<tr>
<td>Quality attributes</td>
<td>Cr1: nfr_03, Consistency, Cr2: nfr_02, Performance</td>
</tr>
<tr>
<td>Identifier</td>
<td>D6-Opt1: sending request when needed.</td>
</tr>
<tr>
<td>Description</td>
<td>The system always requests to external partners whenever data is needed.</td>
</tr>
<tr>
<td>Relationship(s)</td>
<td>-</td>
</tr>
<tr>
<td>Status</td>
<td>Rejected</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Cr1: This option ensures maximal consistency, because all data are queued on-the-fly from the data provider. Cr2: Always queuing data providers over the network could result in very bad performance of the whole system, because it needs higher network bandwidth (more data is transferred) and also data providers would have to process more requests.</td>
</tr>
<tr>
<td>Architectural options</td>
<td>-</td>
</tr>
<tr>
<td>Rationale</td>
<td>This option is rejected because in the context of the E-Health system, the same types of data (such as information about medicines) is often requested by different doctors. Most of the data provided by external providers are relatively static, which means the data does not change frequently (such as the description of medicines, profiles of medical experts). Therefore data consistency is not a big issue. However, large number of requests for same type of data is regarded as unnecessary traffic that hinders performance.</td>
</tr>
<tr>
<td>Identifier</td>
<td>D6-Opt2: Caching requested data.</td>
</tr>
<tr>
<td>Description</td>
<td>Data that has been requested within certain time interval is cached.</td>
</tr>
<tr>
<td>Relationship(s)</td>
<td>D7 (What is the most appropriate time interval for caching different types of data?) D8 (What types of data can (and cannot) be cached)</td>
</tr>
<tr>
<td>Status</td>
<td>Accepted</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Cr1: This option could cause data inconsistency, because data might change while the system is in use. Cr2: This option reduces network traffic and consequently increases performance.</td>
</tr>
<tr>
<td>Rationale</td>
<td>For the data that is relatively static, data inconsistency (Cr1) would not be a big problem. For the data that dynamically changes (such as the storage of medicines, availability of devices from partner hospitals), properly configured time intervals for caching different types of data can improve data consistency.</td>
</tr>
</tbody>
</table>
CHAPTER 9. A TEMPLATE FOR SOA DESIGN DECISION MAKING IN AN EDUCATIONAL SETTING

One source is the architectural design documentation delivered by each team. When evaluating these designs, we paid particular attention to the impact of the use of the template on the quality of the designs and the relevance to service orientation.

Another source is questionnaires filled out by the students. By the end of the course in each year, we asked the students (around 35 students each year) to fill in a questionnaire, gathering feedback on various aspects of the teamwork the students have been doing to perform their assignments. One of the aspects addressed in the questionnaire is the pros and cons of explicitly documenting design decisions and rationale using the given template.

The results collected from these two sources provide evidence that the template has met the three goals described in § 9.2. In this section, we shall discuss these results in detail.

9.4.1 The evaluation of the software designs

Goal 1: support structured reasoning

In order to assess if the template supports structured reasoning, we evaluated the relevance of the design issues to the problem space, the choice of design options and the correctness of rationale.

Before the template was in place, making sound of ADDs with structured reasoning has been very challenging. Unlike an experienced architect who has already accumulated knowledge from multiple real-life software design project, a student can only use the knowledge gained in their courses to make logical ADDs. In making SOA ADDs, a common problem that students have been facing is that the evaluation of design options against quality attributes are often very limited rather than comprehensive. For instance, inspired by duplicating database to improve data reliability (which has been taught in the database course), the students decided to duplicate services to improve the reliability of services. They argued that in case one service fails, its duplicated service could replace it to ensure its end users are not affected by the failure. However, duplicating services would lead to other efforts like version control, service deployment, maintenance, and SLAs. These efforts may have an impact on performance, maintainability, and potentially lead to new design issues, such as how to deploy duplicated services. This impact has been often completely ignored or overlooked in their evaluation and rationale.

After the template has been used, we noticed that the soundness of ADDs has been improved despite that the students’ knowledge was still limited. First of all, the template “forced” the students to evaluate each design option against quality requirements. Instead of justifying (only) why certain design option has been decided (like they did before 2007), the students had to justify also why certain
9.4. RESULTS ON THE USE OF THE TEMPLATE

design options have been rejected. When explicitly documenting such evaluation and rationale, the students were led to think both positive and negative impact of certain design options on quality requirements. This stimulates the students to think and question the knowledge they already have and encourages them to discover and search for knowledge that they do not have. As a result, the ADDs made by the students from 2007 on were much more comprehensive.

Second, we noticed that the template facilitates the discovery of more relevant design issues. Most of the quality attributes have been addressed by multiple ADDs in each design. ADDs can be both contributors to the systems quality and inhibitors to the desired level of quality (Kruchten, 2004), which means one ADD might support certain quality attributes but hinders the achievement of other qualities. To ensure that all the quality attributes could be supported by their designs, the students often made multiple ADDs addressing one quality attribute. The template guided the students in documenting positive and negative impacts of each option on each quality attribute. It also stimulated the students to make trade-off analysis and document their arguments as rationale. By reviewing the ADDs that have already been made, the students could easily identify quality attributes that were weakly supported by their designs and were encouraged to find solutions to improve such quality attributes. Consequently, more design issues that would affect the architecture could be identified. As such, although some ADDs would have a negative impact on certain quality attributes, other ADDs could compensate such an impact.

All in all, by using the template the students get help to structure their reasoning and result in a better decision making process.

Goal 2: support traceability

When assessing the traceability between requirements and designs, we assessed the link between problem space (where requirements are specified), design space (where design decisions are made) and solution space (where resulting designs are illustrated or described).

The evaluation of the design deliverables showed that traceability between requirements and designs was significantly improved. Before 2007, when the template was not available yet, we noticed that majority of the designs insufficiently address the quality requirements. Some quality requirements are completely overlooked, and some are implicitly addressed by the design.

From 2007, thanks to the explicit link between quality attributes and ADDs, it is much easier to assess whether requirements (quality attributes) defined in the problem space have been addressed in the solution space and how the design solution supports such requirements. Most of the students managed to translate all the requirements elicited from the case descriptions to software designs that fulfill such requirements. ADDs served well as a bridge between requirements
CHAPTER 9. A TEMPLATE FOR SOA DESIGN DECISION MAKING IN AN EDUCATIONAL SETTING

and the resulting architecture designs. Since each quality attribute was explicitly linked to at least one ADD, it was easy to trace which ADD(s) is(are) relevant to the quality attribute. Consequently, it was also easy to trace which part of the design (architecture views) support the quality attributes.

Goal 3: facilitate focus on service-oriented design

As explained in § 9.2, the students have been facing challenges in identifying design issues that are relevant to service-oriented design. It has been extremely difficult for them to identify design issues specific to SOA and triggered by the characteristics of SOA as compared to other architectural styles.

Before 2007, typical factors (such as services provided by multiple external providers, cross-organizational collaborations, and consumers with different quality requirements) that make SOA design challenging have not been significantly considered. For instance, the students did not take into consideration security challenges caused by exposing services to unknown (or untrusted) service consumers, or availability challenges caused by composing services provided by external business partners that might or might not achieve agreed SLAs. Rather, the students still focus on traditional challenges such as network security, authentication, encryption, etc. As a result, the designs delivered by the students are more ‘traditional’ solutions rather than SOA-specific solutions.

The students are not used to think in a service-oriented way. Still, the template stimulated the students to motivate their design issues and document the motivation as context. In principle, the students should identify SOA-specific design issues by analyzing what are the SOA-specific factors that cause such design issues. When explicitly documenting these factors, naturally the students are guided to think of solutions that accommodate them.

The improvements in addressing SOA-specific aspects became evident when the template was used. When the students had to explicitly document their motivation for the identification of certain design issues, they were stimulated to think and discover SOA-specific aspects. One of such examples can be seen in the ADD given in Table 9.2, where the students particularly consider the impact of external partners on data retrieving issues. When making SOA designs, one often makes an open-world assumption that services will be discovered and invoked by external (sometimes unknown) consumers. Being aware of this assumption and identifying design issues specifically relevant to it, the students have succeeded in making ADDs more specific to SOA.

Summary

The evaluation of the designs delivered by the students showed that when the template has been used, the reasoning is more structured, the quality require-
9.4. RESULTS ON THE USE OF THE TEMPLATE

ments are more traceable and the students focus better on service-oriented design as their design topic. Still, to make high quality software designs one also requires both domain knowledge and software architecting experience: the former to identify domain-specific issues and judge the importance of different quality attributes; the latter to anticipate how a design option could affect the architecture of the system of interest as a whole. The template obviously does not facilitate one to justify or document things that are unknown or overlooked. Nevertheless, we found that it successfully encourages the students to identify design issues relevant to quality requirements and to provide sound justifications for the selection of design options, based on their limited knowledge and experience on software architecture. Again for students with little to no experience facilitating this learning process is especially relevant.

9.4.2 Students feedback on the use of the template

In general, the students provided very positive feedback on how the template aids them in identifying design issues and focusing on quality requirements. From the feedback, we can see that the students were agreed that the three goals defined in § 9.2 has been achieved.

Goal 1: Support structured reasoning

Many students agreed that the template allows an unified and structured way to document ADDs and thereby guides them in understanding what and how to document. One student wrote “The template is very useful because it made it easier for us to know what are expected from us. When the template was filled with our ADDs I had a satisfied feeling, because it was clear what should have been done.”. Structured documentation also encourages structured reasoning. As one student wrote “… we are guided in justifying our decision by analyzing pros and cons of each design options.”

Goal 2: Support traceability

Many students indicated that the template provided them guidance in software design by enforcing explicit link between ADDs and quality requirements. One student wrote “the templates forced us to be complete, and to pay attention to aspects we previously did not think about (e.g., relating decisions to quality criteria)”. Another student wrote “Using the identifiers, we can easily check ourselves if all the quality requirements have been addressed.”
CHAPTER 9. A TEMPLATE FOR SOA DESIGN DECISION MAKING IN AN EDUCATIONAL SETTING

Goal 3: Facilitate focus on service-oriented design

In contrast to the course given before 2007 where the students faced difficulties in identifying design issues, these students were driven by quality attributes and were able to identify relevant design issues as well as make reasonable justifications for their ADDs thanks to the guidance provided by the template. One student commented “The advantage is that you can follow a certain path in working towards a complete document. It is clear what needs to be done.”

Moreover, the students also realized that they were led to focus better on service-oriented design after using the template. One student wrote “For us, making SOA design was very challenging and we were lost during discussions. As soon as we started to document some early ADDs, we were surprised that we identified more ADDs during our documenting process rather than discussion sessions.”

Challenges

In addition to the benefits, the students also reported some challenges with regards to their use of the template. One of the common complaints was that “the template forces non-complex ADDs to be described in a overly complex way”. As entailed by the template, each ADD has to be described in detail with its context, problem, quality attributives, options and rationale. These details prevented the students from making “intuitive decisions” that they “feel” right. Instead, it forced the students to document at least two options for one design issue and describe their rationale for the selection of one of the options against the quality attributes. The amount of work demanded from the students was the main cause for this complain.

Another challenge was the lack of tool support for the template. Each design issue, quality attribute and option were given an identifier for the sake of traceability. With these identifiers, it was easy for the students to create references within or across ADDs when needed. Moreover, these identifiers helped the students to verify whether all the quality attributes have been supported and to trace how many ADDs have been made for certain quality attributes. However, some students pointed out that it also caused much effort in document maintenance, especially when a new option was added or a design issue was regarded as less relevant and removed at a later stage of the design. We agree that a tool developed based on the template could be greatly helpful for the course in the future.
Summary

The students’ feedback is an indication of the support provided by the template. This support is further reflected by the quality of the design deliverables which in average increased with the use of the template. This suggests that the template does not only help the students in decision making process but also gives them a framework where to focus better on the development of the design solution.

9.5 Related work

In this chapter, we have discussed that the main driver for us to devise the template was that the students have been facing difficulties in making sound SOA ADDs due to the lack of architecting experience and particularly the difficulties to adjust to service-oriented design. Expert designers often do not generate a wide range of alternatives; they ‘know’ what option to choose (Cross, 2004). Inexperienced designers demand more guidance in software design. Tang et al. (Tang et al., 2008) for instance found that inexperienced designers are able to produce a higher quality design when guided with a design reasoning approach.

Related work also addressed the problem of architectural knowledge vaporization and how to avoid it. Many tools for documenting and (in particular) managing ADDs have been proposed. These tools have been designed mainly for professional use and for managing a large number of ADDs.

A template for documenting ADD was proposed by Tyree (Tyree & Akerman, 2005). Different from ours, this template has been designed for capturing a broad range of detailed information that is relevant for a ADD. Elements that serve the same purpose, i.e. the justification of the decision, have been explicitly differentiated to allow the analysis of their impact on ADDs.

The tool implemented by Lee and Kruchten (Lee & Kruchten, 2008) supports an ontology of ADDs and visualization of ADDs as well as their dependencies and evolution. Such a tool would be very useful for decision exploration and analysis.

AREL (Tang et al., 2007) is a UML-based tool aiming at documenting ADDs with a focus on design rationale. It allows the designers to make quantitative as well as qualitative analysis to assess the chosen decision. As such, this tool facilitates the validation on e.g., the completeness of the design.

Architectural Decision Knowledge Wiki (Zimmermann et al., 2008) is a professional tool for managing service-oriented ADDs and is available only recently on IBM alphaWorks1. Based on Web 2.0, this tool supports the cooperative decision-making process of software architects from different projects and different geographical locations. Being used for documenting ADDs from a large

1www.alphaworks.ibm.com/tech/adkwik
number of IBM SOA projects, this tool can be employed as a knowledge-base for sharing and learning SOA design decisions made in different contexts.

*Document Knowledge Client* (Jansen *et al.*, 2009) is a tool specifically designed for discovering and eliciting ADDs after a software has been designed. By means of annotating existing software architecture documents, it allows implicitly documented ADDs become explicit and thereby enables knowledge sharing and increases traceability. This tool is very useful for companies to distill ADDs “after the fact”.

After reviewing the existing ADD modeling tools, we found that most of the functions provided by the tools are valuable and powerful to perform comprehensive analysis on architectural designs, such as visualization, architectural views or annotation. While the usability of the existing tools is broader, in our particular educational setting, we need a tool (with a set of minimal requirements) that is easy to use and that supports the documentation of ADDs with explicit links to quality requirements. Our template proposed in this chapter was specifically for these purposes.

### 9.6 Conclusions

In this work we addressed the problem of teaching SOA design decision making to Master students. To this end, we must first change their mindset from system design to service-oriented software design. In addition to that, students must learn to reason about SOA design decisions as related to quality requirements.

To guide the students in their design and in structuring their ADDs, we created a template based on the core model for modeling architectural knowledge. This template is of narrow scope in the sense that it requires a minimal set of ADD elements and leaves out detailed information, such as the relation between ADDs and software artifacts. More importantly, the template allows drawing explicit links between quality requirements and design decisions, as well as corresponding rationale. Such explicit links not only enforce quality requirements being thoroughly considered for each decision but also give rise to additional ADDs that might be overlooked.

The proposed template has been employed by the students over the last three years. From the feedback of the students and the evaluation of their design documents, we learned that the template guided the students in the architectural decision making process in general and in embodying quality requirements into SOA in particular. While codifying design issues helps the students to reason about which issues are of relevance, a still open question is about identifying those that are more or less important for a design problem.

In summary, this chapter contributes to software architecture research in general as it proposes a quality-driven approach to embodying quality requirements
9.6. CONCLUSIONS

into software architecture by explicitly linking quality attributes to architectural design decisions. More specifically, it contributes to architectural knowledge management as it presents an easy to use template to document architectural design decisions and to improve the traceability between requirements and architectural designs.
SOA decision making - what do we need to know

The SOA decision making process addresses specific concerns. Currently, SOA design decisions are often documented (if at all) using generic architectural design decision models (discussed in Chapter 9). The information that is specifically relevant to SOA design decisions remains hidden in the description of the existing entities of architectural decision models. As a result, this information could be overlooked during the decision making process and result in unsound decisions. This chapter highlights a set of information items that aids SOA decision making.

10.1 Introduction

The focus of engineering SBAs shifts from developing software applications to developing pools of services that can potentially be aggregated to provide a variety of solutions for various runtime scenarios to fulfill ever-changing business requirements. As a result, specific architectural concerns arise in the design of SBAs. For instance, when client-server is chosen as the architectural style for a software system, quality attributes like performance or maintainability are typically the primary architectural concerns; in the context of SOA, the scope of service pools, the runtime behavior of services, the cooperation with external stakeholders and governance issues become major concerns that need to be dealt with.

Similarly, specific information becomes relevant to the SOA design decision making process. Typically, the reasoning for selecting one of the design alternatives to form an architectural decision is based on (non-)functional requirements, business goals and cross-dependencies (Falessi et al., 2006). When making SOA design decisions, specific information (like the knowledge of external service consumers and the ownership of a business resource) is required. For instance, suppose an architect needs to decide between creating a new service and modifying
the existing service to accommodate the new business requirements. The architect has to evaluate each alternative based on e.g., the impact on the existing service consumers and the potential cross-organizational cooperation. Therefore, the knowledge of external service consumers and the ownership of a business resource become particularly relevant.

In order to identify which information is relevant, we analyze the aspects that cause SOA concerns in the design of SBAs. Based on the study of the literature, the knowledge of the state-of-the-art, and experience from various industrial case studies, we identified four aspects that are specific to SOA, namely a) temporary provision-consumption relationship, b) different architecture types, c) dealing with heterogeneity, and d) different perspectives of stakeholders. From the analysis of these four aspects, we elicited a set of information items that are intensively involved in the SOA decision making process. From this set of information items, we identified eight knowledge entities that in our opinion should be explicitly expressed when documenting SOA design decisions.

To show the relevance of this information in SOA design decisions, we created two scenarios as tangible examples. In these scenarios, we discuss related SOA design decisions where the identified knowledge entities play important roles in the decision making process. In addition, we also studied existing SOA patterns to get some evidence from proven SOA design solutions. From the problem space of the SOA design patterns, we observe that the four aspects are drivers for many SOA design issues. We also observe that the identified information is embedded in the description of the patterns. From these observations, we corroborate the relevance of the identified information.

In summary, the SOA decision making process addresses specific concerns, where specific information becomes relevant. The main contribution of this chapter is to highlight this relevant information so that its modeling becomes possible. The modeling of this information benefits from the identification of SOA design decisions, the justification of alternatives and the classification of SOA design decisions. Accordingly, we argue that the representation of architectural decisions should be extended to leverage information relevant to SOA design decisions.

The remainder of the chapter is organized as follows. The next section discusses the four aspects that may give rise to SOA design issues and information relevant to SOA design decisions. § 10.3 describes two SOA design decisions based on two business scenarios to show the relevance of the elicited information and the benefits of being modeled as first class entities. In § 10.4 we discuss our observations of published SOA design patterns and decisions in relation to the identified aspects and information. Related work on SOA design decisions is discussed in § 10.5. We draw our conclusions in § 10.6.
10.2 The service aspects

The design of SBAs has to address specific design issues arising from the aspects that assume special relevance to service engineering. Some of these aspects are specific to service engineering, while some of them are less specific in the sense that they do share certain similarities with the traditional approaches (e.g., component-based development). All these aspects, however, are prominent and should be specifically considered in SOA design. In this section, we discuss four such aspects, where the first one is specific to SOSE and the rest three aspects are less specific.

10.2.1 Temporary provision-consumption relationship

In TSE, software applications are typically developed by software producers and installed on their customers’ computers. The execution environment, such as operating systems or databases, is known in advance (i.e. given in the installation requirements) by the customers. Further, the business relationship between the producers and their customers is usually long term and stable. As a result, the producers know how to inform their customers if new releases are available; and the end-users know how to contact the producer if faults are detected or requirements change.

In SOA, the way in which software applications (and services) are delivered is different in that service consumers use services but do not own these services (Service aspect: Shift of ownership of software discussed in § 11.4.1). A temporary provision-consumption relationship exists between a service provider and its consumer. On the one hand, temporary refers to short-term in that the relationship exists only when a service consumer requests the service and terminates as soon as consumption is finished; on the other hand, temporarily implies dynamism in that a service may be consumed by different service consumers in different business solutions. Due to this temporary provision-consumption relationship, new SOA concerns arise with regard to runtime governance and uncertainty of service consumers.

- **Runtime governance** Since services physically reside at the service providers’ premises rather than the service consumers’, the service providers have to ensure that their services are available for consumption and perform as expected. To achieve this, the service providers have to monitor the behavior of the published services and manage their execution environment (Stojanovic & Dahanayake, 2005), which is not the case for traditional software applications. For instance, consider a situation in which the server where a software application or service is running encounters a system failure and requires a reboot. In the case of the software application, the user
CHAPTER 10. SOA DECISION MAKING - WHAT DO WE NEED TO KNOW

(customer) of the application has to take action to reboot the server; while in the case of service, the service provider has to reboot the server and make sure that the service is available again. Hence, an architect should be aware of this responsibility from the perspective of a service provider in the SOA decision making process.

- **Uncertainty of service consumers** Service providers may deliver services to different groups of consumers with different expectations under different runtime scenarios (Maximilien & Singh, 2004). The consumers and their expectations are not always known by the service provider at design time. Services are often designed according to some assumptions on service consumers (e.g., the number of concurrent invocations). As different assumptions might lead to completely different design decisions, these assumptions should be made explicit in the SOA decision making process.

  Due to ever-changing business requirements, changes made to a service are inevitable (Di Nitto et al., 2008). It is very undesirable to interrupt an existing consumption or leave the improved functionalities unannounced. How to inform the existing service consumers about the changes becomes a design issue. Accordingly, assumptions on service consumers again become of great importance. For instance, a service that assumes known service consumers may send a notification to its consumers whenever a change is made; while a service that assumes unknown service consumers may have to e.g., implement multiple service contracts to avoid interrupting existing consumers.

  Hence, making assumptions on service consumers is necessary in the design of SBAs. Accordingly, these assumptions should be explicitly considered in the SOA decision making process.

10.2.2 Different architecture types

The focus of engineering SBAs is to create pools of services and to (re-)aggregate these services in such a way that different business requirements can be fulfilled (Dubey & Wagle, 2007). These services are leveraged as enterprise resources which can be shared across enterprises or even business partners. Each service in a service pool exposes a number of functionalities. Functionalities delivered by various services can be aggregated according to business logic or rules to form composite services. Both services and composite services can be grouped to form one or multiple service pools depending on their business domain, ownership or governance responsibilities (Erl, 2008).

  Hence, instead of designing one architecture for one software application, the design of SBAs has to design architectures for services, composite services and
10.2. THE SERVICE ASPECTS

services pools. Each type of these architectures faces different concerns.

- **Designing services** A service can be viewed as an independent software program that realizes a set of functionalities. As each service might have an architecture different from the others, it needs to be designed individually (Erl, 2008). The design of services includes the decisions on the functionalities it will deliver, its underlying infrastructures that support those functionalities and the design of a service contract.

  In order to decide the level of granularity and define the scope of services, architects (together with business analysts) have to deeply understand the **business process and its ownership** and available **IT support** (Jamshidi et al., 2008). For instance, to ensure the integrity of a business solution, it would be desirable to embed the related functionalities in one service. However, if one of these functionalities participates in multiple solutions, a decision has to be made between isolating this functionality to create a separate service or leaving it as a repeated functionality among multiple services. The choice is heavily dependent on the relationship between solution logic, the ownership of business processes and the cost of creating new services.

  The design of the underlying infrastructure of each service requires decisions on the physical environment within which it is deployed. These decisions require information about **enterprise resources and their ownership**, such as what resources the service needs to access and what resources are accessed by the other parts of enterprises as well (Pulier & Taylor, 2006).

  Since services are invoked through their published contracts, service contracts become part of the design of services. From the perspective of a service provider, the design of a service contract requires **assumptions on the service consumers** (e.g., interest of service consumers) (Maximilien & Singh, 2004). For instance, if the service is intended to be used by different groups of service consumers which have different requirements, multiple contracts designed specifically to each group of service consumers are preferred.

  In summary, specific information relevant to designing services includes: the concerns from **the perspective of a service provider**, **assumptions on the service consumers**, information on **business process and its ownership**, **IT support** and **enterprise resources and their ownership**.

- **Designing composite services** The ultimate goal of designing services is to aggregate them to solve business problems. Composite services are the result of the aggregation, which are typically joined functionalities from participating services glued together with solution logic (Erl, 2008). The design of the corresponding architecture of composite services has a different focus.
than e.g., a component-based architecture in that service participants are
distributed with different owners, and various composition configurations
should be designed to accommodate various runtime scenarios.

Since service participants representing units of business process are often
distributed within the enterprise owned by different stakeholders, the com-
position designer often needs to collaborate with these stakeholders to e.g.,
pose additional requirements. Therefore, being aware of business process
and its ownership is sometimes necessary in the design of composite ser-

Further, it should be noticed that a service composition can be both a
service provider and service consumer depending on its runtime behavior. It
is a service provider when it publishes its service contract for consumption;
and it becomes a service consumer when it invokes its service participants.
As discussed in § 10.2.4, these two roles have different concerns. Hence, a
composite service should be designed from the perspective of both a service
consumer and provider.

In summary, being aware of the concerns from the two perspectives and
the information of business process and its ownership are relevant to SOA
decision making.

• Designing service pools Ideally, an enterprise maintains one service pool
that comprises resources needed for a complete list of business require-
ments. However, in reality (especially in large enterprises having many
departments), it is difficult to capture all the business processes and rules
to create business models that cover the entire enterprise. The lack of
complete and concrete knowledge about these business models hinders the
success of collecting a complete list of business requirements. Hence, a
service pool for an entire enterprise is often hard to implement.

Instead of creating one service pool, another option is to build multiple
service pools, each of which is governed by a specific enterprise department
unit (Erl, 2008). The division of service pools is often driven by enterprise
resources and their ownership (e.g., business domains) and IT support(e.g.,
various IT groups supporting various business functions). In this way, an
enterprise may gradually adopt SOA depending on its goals and capabilities
by allowing more flexibility for the sake of convenience or historical
influences.

Although allowing multiple service pools with variations (in terms of imple-
mentation technologies, standardizations and transmission protocols) may
increase the efficiency of building each pool, problems may occur when a
service in one pool tries to invoke services in other pools. Therefore, the
boundaries of service pools need to be well designed such that the chance of cross-boundary invocation is minimized.

Hence, the decision to place services (and service compositions) in a certain service pool depends on multiple concerns in relation to *enterprise resources and their ownership* and *IT support*, which assume special importance in service engineering.

### 10.2.3 Dealing with heterogeneity

SOA by definition allows heterogeneity in SBAs (Papazoglou, 2003). Often, services are designed independently in different projects, by different authorities and using different technologies. Eventually, when these services are required to be aggregated to fulfill business requirements, inconsistent design, implementation, data format, protocol, service contract, business rules and policies may bring lots of transformation effort. Therefore, how to deal with the unavoidable heterogeneity and how to avoid potential heterogeneity in the design are main concerns.

Since the heterogeneity often occurs when aggregating services from different pools and different owners, it is extremely important to know which service pool each of the participating services resides. Hence service pools as an *architecture type* assumes relevance to SOA decision making.

### 10.2.4 Different perspectives of stakeholders

SBAs are often composed of services owned by other stakeholders. This naturally increases the complexity of the development process as well as the execution environment. Due to the shift in ownership of software (a service aspect discussed in § 11.4.1) and temporary provision-consumption relationship (a service aspect discussed above), a stakeholder may play multiple roles (or take multiple perspectives) simultaneously, each with different goals and competencies. For example, in TSE software designers are responsible for software design and software developers are responsible for implementation. However, in SOSE a service designer in the role of *service provider* is concerned with the identification of services to provide to others whereas when the service designer is in the role of *service consumer* she is concerned with the identification of services for integration purposes. As a result, a designer with a service provider’s perspective may be concerned with the deployment environment of the services, the privacy and security related issues, the service contract to be published and expected service consumers and focus on requirements external to the organization, whereas a designer with the service consumer’s perspective may be concerned with the development effort, the selection of services and with potential collaboration between the owner of the services
and pay particular attention to the constraints internal to the organization. Often a SOA design decision involves multiple (composite) services and accordingly multiple service providers and consumers may coexist in relation to these (composite) services. Without being aware of the roles, important concerns could be overlooked. Therefore, being aware of the different concerns from the two perspectives shows relevance to SOA decision making.

10.2.5 A summary of identified information

Up to now, we have discussed four service aspects and elicited a set of information items relevant to the SOA decision making process in relation to each aspect. From this information, we identified the eight knowledge entities (listed in Table 10.1) relevant to the SOA decision making process. The exemplified application of these entities can be found in § 10.3.

<table>
<thead>
<tr>
<th>Table 10.1: Identified knowledge entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  The type of the architecture</td>
</tr>
<tr>
<td>2  Enterprise resources and their ownership</td>
</tr>
<tr>
<td>3  Business process and its ownership</td>
</tr>
<tr>
<td>4  IT support</td>
</tr>
</tbody>
</table>

10.3 Two examples of SOA design issues

Due to the distinctive aspects that have been discussed in § 10.2, business constraints and quality attributes are not the only factors that are used in the rationale for SOA design decisions. Instead, the scope of services, the knowledge of consumers and the level of heterogeneity all influence SOA design decisions. In the following, we present two examples to show how these elements aid SOA decision making.

10.3.1 Example 1

Consider a scenario where the financial department of a retailer has to create a monthly “Profit & Loss statement” by determining the cost (e.g., cost of sales, marketing and sales expenses and general and administrative expenses) and the
10.3. TWO EXAMPLES OF SOA DESIGN ISSUES

income (e.g., revenue from sold products and sales commissions). The IT department has created two services: a Cost service and Income service. These two services are aggregated to form a composite service called P&L statement which specifies the profit or loss realized in a period. The head-office of this retailer has noticed that each month there is a huge gap between the budget (delivered yearly) and the profit reported each month. Therefore, it requests the retailer to add a variance analysis to the monthly P&L statement so that the actual results can be monitored against budget.

The architect in the IT department faces two alternatives: a) change the Profit service by adding a newly created Budget service; or b) keep the Profit service and use it as a sub-composition logic for a new composite service: P&L variance analysis (actual versus budget). These two alternatives are illustrated in (Figure 10.1.a) and (Figure 10.1.b) respectively. The straight lines in the figure represent the existing situation, while the dashed lines indicate the changed or new situation. Comparing these two alternatives, we observe that 1) the first alternative potentially delivers higher performance than the second one, but the service contract requires change; 2) the second alternative increases the reusablity of the P&L statement service but decreases performance and brings more governance effort since an additional layer of service composition is introduced.

Critical to decision making is the knowledge of the existing service consumers. If the existing service consumers are unknown or outside the boundary of the financial service pools (e.g., other departments or retailers), the second alternative is preferable because it does not influence current agreements with existing consumers. If the service consumers of the P&L statement service are within the boundary of the financial service pool, the service architect might prefer the first alternative. However, adding budget information to the P&L statement may not be desired by all service consumers. The modified representation of the statement could lead to unexpected results, e.g., a bonus service that calculates the bonus for employees based on the output of the P&L statement may encounter failure if the structure of the P&L statement is completely changed. Further,
the architect has to consider the feasibility of informing all the service consumers about the change.

In this example, besides the quality attributes (like governance effort, reusability, performance) that have been used in making the trade-off analysis, the knowledge about current agreements with consumers becomes the most important factor in choosing for one of the alternatives. Therefore, whether the service consumers are known or unknown and whether they are internal or external should be explicitly documented.

This SOA design issue has been formalized in Table 10.2, using the knowledge entities identified in § 10.2.5.

Table 10.2: Documenting SOA design decisions with identified knowledge entities:

<table>
<thead>
<tr>
<th>Knowledge entities</th>
<th>Alternative A</th>
<th>Alternative B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related service</td>
<td>P&amp;L variance analysis</td>
<td></td>
</tr>
<tr>
<td>Type of architecture</td>
<td>Composite service</td>
<td></td>
</tr>
<tr>
<td>Enterprise resources and their ownership</td>
<td>Financial database owned by the financial department</td>
<td></td>
</tr>
<tr>
<td>Business process and its ownership</td>
<td>Financial report process owned by the financial department and possibly by unknown consumers from other departments</td>
<td></td>
</tr>
<tr>
<td>IT support</td>
<td>IT group within the financial department</td>
<td></td>
</tr>
<tr>
<td>Types of service consumers</td>
<td>External</td>
<td></td>
</tr>
<tr>
<td>Assumptions on service consumers</td>
<td>This service is consumed by the head-office,</td>
<td></td>
</tr>
<tr>
<td>The perspective of a service provider</td>
<td>Less effort and cost to implement the change; service contract changes; its service consumers may encounter unexpected results</td>
<td>more effort and cost since a new composite is created; increased reusability</td>
</tr>
<tr>
<td>The perspective of a service consumer</td>
<td>Higher performance</td>
<td>Lower performance</td>
</tr>
</tbody>
</table>

10.3.2 Example 2

Suppose now a new business requirement requests the P&L variance analysis to explicitly show the cost of marketing activities. As all the data related to marketing activities is maintained by the marketing department through the Marketing activities service, the P&L variance analysis service therefore faces the challenge of composing services distributed over two service pools. The architect hence faces two alternatives: a) let Marketing activities be exposed to external consumers and directly invoked by P&L variance analysis (Figure 10.2.a); or b) create Published marketing activities as an intermediary service with a service contract specifically designed for the financial department and exposes only the functionalities that are needed by the financial department (Figure 10.2.b).
10.3. TWO EXAMPLES OF SOA DESIGN ISSUES

![Diagram](image)

Figure 10.2: Example 2: a) create an intermediary service, b) direct invocation

The first alternative faces privacy and security-related concerns since the Marketing activities service might comprise functionalities that should not be exposed to external consumers due to e.g., confidentiality reasons. Further, external service consumers may not appreciate all the functionalities when they are only after a subset of the functionalities. However, if the marketing department agrees to expose the Marketing activities service for external usage, it might be a convenient and fast way for the financial department to directly invoke the marketing activities.

From the service provider’s perspective, the second alternative provides better protection for the marketing department in terms of privacy and security since business sensitive functionalities can be shielded from external consumers. In addition, the possible changes made on service Marketing activities do not have impact on external consumers since the service Published marketing activities remains the same. However, by introducing a new service, the second alternative requires more development and governance effort. From the service consumer’s perspective, the P&L variance analysis service invokes the Published marketing activities service through a dedicated service contract designed specifically for the financial department. Without being overwhelmed by all the functionalities delivered by the Marketing activities services, the P&L variance analysis service may perform the invocation more efficiently.

In this example, besides the quality attributes (like privacy, development effort, security) that have been justified in the rationale for each alternative, the perspective of a service provider or service consumer, the ownership of business resources and the boundaries of service pools are all relevant. Therefore, we argue that this information should be explicitly documented in this SOA design
decision.

This SOA design issue has been formalized in Table 10.3, using the knowledge entities identified in § 10.2.5.

10.3.3 Foreseen benefits

From the two design issues discussed in the previous sections, we observe that by modeling the identified knowledge entities as first class entities, we can support the following objectives:

- **Facilitate the detection of SOA design issues.** Due to still emergent best practices and knowledge sharing in the context of SOA, the identification of SOA design issues is still a challenge (Zimmermann et al., 2007a). Sometimes, design issues that are relevant can be overlooked due to e.g., the lack of experience in SOA design. It has been extensively studied and proven that making explicit the information about architectural design decisions facilitate the decision making process (Avgeriou et al., 2007). In the SOA context, it can further help in acquiring insight and accelerating maturity.

For instance, the three knowledge entities: enterprise resources and their ownership, business process and its ownership, and IT support may be associated with different stakeholders participating in SOA design decisions. If this association is made explicit, potential collaboration between these stakeholders becomes more obvious. Consequently, the collaboration may introduce new design issues.

- **Highlight the relevant information for justifying alternatives.** As shown in Table 10.2 and 10.3, the identified knowledge entities play important roles in justifying the selection of design alternatives. When information relevant to SOA decision making is explicitly expressed, the chance that this information is used in the evaluation of each alternative is higher. In this way, the SOA design decisions are more sound and convincing.

- **Enable filtering specific design decisions based on certain criteria.** By explicitly documenting the relevant information for SOA design decisions, it becomes possible to select a group of SOA design decisions that have the same characteristics. These characteristics form a set of indirect relationships between SOA design decisions, which is complementary to the direct relationship that is usually documented in architectural decisions. For instance, one can have a selection of SOA design decisions that are related to services designed for external service consumers or services that require access to certain enterprise resources. By doing so, a pattern language may be established.
## 10.3. TWO EXAMPLES OF SOA DESIGN ISSUES

Table 10.3: Documenting SOA design decisions with identified knowledge entities:
Example 2 - add a new service composition

<table>
<thead>
<tr>
<th>Knowledge entities</th>
<th>Alternative A</th>
<th>Alternative B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related service</td>
<td>Marketing activities</td>
<td>Published marketing activities</td>
</tr>
<tr>
<td>Type of architecture</td>
<td>Service</td>
<td>Service</td>
</tr>
<tr>
<td>Enterprise resources and their ownership</td>
<td>marketing strategy owned by the marketing dept.</td>
<td>marketing strategy owned by the marketing dept.</td>
</tr>
<tr>
<td>Business process and its ownership</td>
<td>Marketing plan owned by the marketing dept.</td>
<td>Marketing plan owned by the marketing dept.</td>
</tr>
<tr>
<td>IT support</td>
<td>IT group within the marketing dept.</td>
<td>A separated IT group that is responsible for all the intermediary services</td>
</tr>
<tr>
<td>Types of service consumers</td>
<td>Internal, known</td>
<td>External, unknown</td>
</tr>
<tr>
<td>Assumptions on service consumers</td>
<td>This service is consumed only by the marketing dept.</td>
<td>This service is consumed only by the financial dept.</td>
</tr>
<tr>
<td>The perspective of a service provider</td>
<td>Increased privacy and security related issues</td>
<td>More effort and cost since a new service is created</td>
</tr>
<tr>
<td>The perspective of a service consumer</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
10.4 SOA design decisions in the literature

Since the aspects we identified are extracted from our observations and the relevant information is deduced from the aspects, we need to verify whether this information is indeed used in SOA design decision making in practice. Therefore, we studied two types of SOA design decisions: SOA design patterns and SOA design decisions published in projects.

Most of the SOA design patterns can be regarded as general SOA design decisions without a specific context. They become proven design solutions to common problems related to the service aspects discussed in §10.2. For instance, among the 85 SOA design patterns categorized in (Erl, 2008), 18 patterns deal with the uncertainty of service consumers (service aspect: temporary provision-consumption in §10.2.1); 13 patterns deal with defining the scope of services, composite services and service pools (service aspect: different architecture types in §10.2.2); 17 patterns deal with the inconsistency between data, contract, protocol, policies, business rules, design and implementation (service aspect: dealing with heterogeneity in §10.2.3). We observe that most of the documented patterns are (implicitly) presented from the perspective of a service provider (service aspect: different perspectives of stakeholders in §10.2.4). This aspect cross-cuts all the other aspects. In summary, more than 50% of the patterns in (Erl, 2008) address the four aspects we identified.

Another set of SOA design patterns described in (Milanovic, 2006) is extracted from a web services composition server project. All these patterns deal with how to deliver services to unknown service consumers, including the access to services, protecting services from untrusted consumers, decoupling known information from unknown information to improve response time, and so forth. These patterns correspond to the service aspect temporary provision-consumption discussed in §10.2.1.

Furthermore, we also studied some SOA design decisions published in the literature (Zimmermann et al., 2004b; Zimmermann et al., 2007b). We notice that many of these design decisions focus on very low-level issues (e.g., mechanisms for dynamic composition and transactions). While these definitely belong to SOA standards and technologies, and to some extent may influence the resulting (technical) architecture, they miss the ‘big picture’ and therewith fail to participate in the more conceptual architectural aspects. Therefore, based on the material publicly available it is very difficult to judge the extent to which they address aspects relevant to SOA.
10.5 Related work

The ways to capture, manage, and model architectural design decisions have been well-known research topics in the community of architectural knowledge management. Our work mainly focuses on design decisions in the context of SOA. Being domain-specific, SOA design decisions still share common use cases (e.g., discovery, sharing) with architectural knowledge management in general.

The information that facilitates SOA decision making is not new per se. Instead, this information appears in architectural decisions in other domains as well. The difference lies in the fact that this information actively participates in SOA decision making rather than being presented as pieces of architectural knowledge. For instance, being aware of different stakeholders with different concerns is well-known; and different viewpoints are typically used to describe the software architecture to different stakeholders to address their concerns. In SOA design decisions, stakeholders, which are associated with the owners of enterprise resource, business process and IT support, are regarded as valuable input to reason about a SOA decision (e.g., an alternative is rejected because sharing certain resources among certain stakeholders is against certain enterprise policies).

In (Zimmermann et al., 2007a), Zimmermann et al. present an approach to effectively capture design decisions and propose an architectural decision model to facilitate SOA decision making. Since our work shares the similar goal but focuses on the relevant information that should be captured, ours can be regarded as complementary to the work in (Zimmermann et al., 2007a). Further, in the model proposed in (Zimmermann et al., 2007a), some of the elements (e.g., scope, roles) are comparable to some of the identified information (e.g., architecture types, perspectives) in this chapter. However, these elements are used for administrative purposes rather than the evaluation of design alternatives.

10.6 Conclusions

The design of SBAs focuses on the design of a set of services and their compositions, as well as placing them into service pools to fulfill ever-changing business requirements. Accordingly, specific design issues emerge due to some service aspects that assume great importance to service engineering. We specifically discussed four aspects, namely: a) temporary provision-consumption relationship, b) different architecture types, c) dealing with heterogeneity, and d) different perspectives of stakeholders.

From the discussion of these service aspects, we identified a set of information items that is relevant to SOA decision making. In addition, we presented two business examples with example SOA design decisions to show how this informa-
CHAPTER 10. SOA DECISION MAKING - WHAT DO WE NEED TO KNOW

tion provides useful input to the rationale of SOA design decisions.

The relevance of this information is not limited to its usefulness for reasoning about SOA design decisions. When this information is leveraged as first class entities in SOA design decision models, it can facilitate the identification of SOA design issues, increase the chance of making sound decisions, enable the identification of implicit relationship between decisions and provide more dimensions for the classification of SOA design decisions. Therefore, the relevance of identified information becomes clearer.
11

SOA Process Decisions: New Challenges in Architectural Knowledge Modeling

The previous two chapters discuss the documentation of architectural design decisions to avoid knowledge vaporization. Process decisions are one type of design decisions that existing models have paid little attention to. This chapter specifically addresses process decisions in the SOA domain. In doing so, we challenge architectural knowledge representations by mapping two SOA process decisions against a core model of architectural knowledge. The result suggests that existing models need to be extended in order to model SOA process decisions.

11.1 Introduction

There are different kinds of architectural design decisions. Some decisions are made by regarding the architecture as a product, such as structural decisions. For instance, choosing 3-tier architecture to increase the maintainability of the system or using redundant hardware to achieve desired availability is an example that directly relates to the architecture product. Some other decisions instead consider the development process as a product, i.e. they are process decisions (Kruchten, 2004). For instance, choosing the waterfall software development process model is a decision that does not directly relate to the architecture design, rather to the development process that leads to the resulting architecture design. In this chapter, we specifically address process decisions.

Recently, various architectural knowledge models (Tyree & Akerman, 2005; Kruchten, 2004; Zimmermann et al., 2006) have been proposed for representing design decisions. Most of the architectural knowledge that is generated during the software architecture process is well represented by using the notations that the software engineering models entail. All of them are implicitly addressing those design decisions that regard software architecture as a product. Process
decisions, however, are paid little attention to.

In TSE, software development processes have reached a high level of maturity. For instance, standard models such as the waterfall model or the spiral model serve as recognized patterns for software developers. When these models are used, software developers know exactly how to carry out development processes. Process decisions are often not modeled because they are not as significant as those design decisions on software architecture.

Such standard models do not yet exist in new paradigms like service engineering. Efforts still need to be put on systematic development methodologies. Consequently, process decisions play here a more important role than in TSE.

Service-oriented computing (delivering software functionalities as services) is an emerging approach that is gradually becoming more popular and dominant in modern software engineering. Services become the building blocks of software systems. Nowadays, organizations often approach SOA in two ways. A less innovative way uses SOA as base for re-engineering, wrapping or migrating existing software systems towards SOA. In this approach, their IT portfolio is migrated to the service landscape and used within the organization. A more innovative way to use SOA (and more auspicated) is delivering software as services where the customers of these services no longer request to buy and install applications; instead, they pay for the services usage. In this chapter, we concentrate on the latter type of use. Consequently, new characteristics (or service aspects) are introduced by the service-oriented paradigm. These service aspects directly lead to a number of process decisions that have to be made in service-oriented development and bring new challenges.

This chapter addresses two of those service aspects, namely the shift of the ownership of software and the cross-organization cooperation. Each of these two service aspects is discussed by giving an example of SOA process decisions elicited from the service centric development methodology defined in an international European project. By mapping the SOA process decisions against a core model of architectural knowledge (de Boer et al., 2007), which is regarded as a common frame of reference for architectural knowledge sharing, this chapter investigates to what extent this core model can be used to model process decisions.

This chapter provides two original contributions. First, it highlights that process decisions are important architectural concerns by describing two examples of process decisions (with options and rationales) coming from the SOA domain and from industrial experience. Second, it concretely analyzes how process decisions can be codified by using architectural knowledge models aimed at representing decisions in general. Potential gaps are discussed, as a first step towards a more articulated analysis of process decisions (as part of architectural knowledge modeling).

The rest of the chapter is organized as follows. In § 11.2 we provide an
11.2. RELATED WORK

overview of the related work. To motivate our work, § 11.3 discusses the reasons why SOA process decisions are important and need to be documented. We demonstrate the link between the service aspects and process decisions in § 11.4 by using two specific SOA process decisions. In § 11.5, we map these two process decisions on the core model and argue that specific fields or notations are necessary for representing SOA process decisions. Our conclusions are given in § 11.6.

11.2 Related work

Our work aims at investigating to what extent the current architectural knowledge models can be adapted to represent decisions on software development processes, especially in the service-oriented context. Therefore, we position ourselves on the fields of architectural knowledge modeling and service-oriented development.

The need to record the rationale behind design decisions has been underlined by Potts & Bruns already in the 80’s (Potts & Bruns, 1988). Since then, we observe a major trend towards architectural design decisions that regard architectural design as a product. A template for modeling architectural design decisions proposed by Tyree (Tyree & Akerman, 2005) is one of the most well known meta models. This template captures the architectural design decisions by recording design issues, assumptions and constrains for the resulting system, arguments for making decisions, its implications and its relationship with other decisions and artifacts. This model consists of self-explaining entities and therefore is very easy to be used for representing architectural design decisions in general.

For complex, software-intensive systems, an ontology of architectural design decisions proposed by Kruchten (Kruchten, 2004) turns out to be useful. This model methodically classifies architectural design decisions into three kinds namely: existence decisions, property decisions and executive decisions. The properties, attributes and relationship between the decisions can be completely captured by this model.

Comparing these two models, many entities in the models address same concepts but are named differently. For instance, both of these two models address the reasoning on the justification of a decision. However, this reasoning is named as argument in Tyree’s template but rationale in Kruchten’s ontology. Therefore, a core model of architectural knowledge, which can be used as a common terminology for representing architectural knowledge, is presented in (de Boer et al., 2007). Each element in the core model can be mapped onto the concepts that are used in other existing models. As stated in (de Boer et al., 2007), the current models can be “wrapped around the core model to specialize the core concept architectural design decision”.

191
As explained in § 11.3, process decisions that regard development process or methodology as a product are put much less emphasis in architectural knowledge modeling. Past research did consider process decisions by documenting the end-result of the decision process (in terms of e.g., process models or guidelines and best practices). Though, the reasoning behind such process decisions remained undocumented.

The service-oriented paradigm becomes more popular and dominant in modern software engineering. Since service engineering has not reached a high level of maturity, both industry and academia are working on the development process for SBAs. Therefore, the role of process decisions that are generated in methodologies and life cycle models for service engineering becomes increasingly important. Process decisions actually have been used more or less implicitly for years. Best practices and guidelines (Brown et al., 2005) towards service-oriented development are often addressed in the industry. Methodologies (Papazoglou & van den Heuvel, 2006) approaches (Kim & Yun, 2006) and life cycle models (Blake, 2007) for service-oriented system development are also discussed in the literature. Although these authors mention and use process decisions within their work, they do not explicitly address process decisions from an architectural knowledge point of view, which leads a partial picture.

Zimmermann et al. (Zimmermann et al., 2006) present some interesting work on SOA design decisions. They suggest to “position architectural decision modeling as a prescriptive service realization technique”. SOA process decisions are recognized as one type of SOA design decisions. However, their decision tree is mainly intended to guide the practitioner through the design process rather than as a model for representing architectural knowledge.

## 11.3 Motivation

Only when architectural knowledge is explicitly documented and stored, is it available for sharing among stakeholders. These stakeholders could be in the same development team, in the same organization or even beyond the organization. Knowledge is of great business value for a company (Djavanshir & Agresti, 2007).

Process decisions, one type of architectural decisions, are paid little attention to in architectural knowledge modeling in TSE. This is understandable because software life cycle models have reached a high level of maturity. The development processes are well known and few process decisions have to be made. The significance of documenting these process decisions is therefore relatively low.

However, process decisions play a more important role in SOA. First of all, process decisions become a must. Once standard software development process models no longer fit, more process decisions have to be made for the development methodologies (Benguria et al., 2002). Various authors have devoted their effort
11.3. MOTIVATION

to propose service life cycle models (Blake, 2007) and agile design and development methodologies (Papazoglou & van den Heuvel, 2006) for developing service centric applications. For instance, whether the process of service discovery should execute at design time or runtime is on the one hand an architectural design decision and on the other hand a process decision. When regarding the system as a product, we need to decide on whether the system will support dynamic service discovery. When regarding the process as a product, the decision is on process-related aspects. For instance, when services are discovered at design time, service engineers need to first discover available services and then decide whether to reuse the services or develop new services. In this case, the process of service discovery is carried out by a service engineer. When services are discovered at runtime instead, the process of service discovery is carried out automatically by a composition engine and under the control of a service provider. The process-related aspect in this example is the executor of the process (a service engineer or composition engine controlled by a service provider). Such types of decisions yield two dimensional aspects, the process aspect orthogonal or complementary to the architectural aspect.

Second, new characteristics of the service-oriented paradigm enforce process decisions. Examples of these new characteristics include: software as services (Papazoglou, 2003), shift of the ownership of software (Dubey & Wagle, 2007), coordination across and beyond the boundaries of an organization (Ludwig, 2003), harder requirements for runtime environment (Papazoglou et al., 2006). These characteristics require extra considerations when taking process decisions. For instance, as stated in (Lassing et al., 2001), existing architectural views are incomplete because the owner of the system as one of the stakeholders is often not addressed. And systems often depend on other systems, with possibly different owners. If you then want to change one of them, you have to come to an agreement with the owners of other systems, since they have to incorporate the same change, at the same time. As a result, we are studying process decisions in the service-oriented paradigm as a modern software engineering discipline.

Similar to design decisions in other fields, process decisions have to be documented. Here, the question we pose ourselves is: can the existing models for representing architectural knowledge in general be used to represent process decisions in the SOA context too?

To answer this question, we are in the process of eliciting a number of process decisions from an international project in the SOA domain. We analyze these decisions and document all the relevant information for each of them. Our hypothesis is that if the concepts that we have used in documenting those process decisions can be all mapped onto the existing architectural knowledge models, process decisions in the SOA domain can therefore be codified using these models. Suppose some concepts cannot be mapped onto the existing models, we conclude
CHAPTER 11. SOA PROCESS DECISIONS: NEW CHALLENGES IN ARCHITECTURAL KNOWLEDGE MODELING

that process decisions face new challenges in architectural knowledge modeling.

11.4 The service aspects and examples of SOA Process Decisions

In this chapter, we particularly focus on process decisions that occur in service-oriented development. To give some factual examples, we take the SeCSE European project (SeCSE, 2007) as a case study. In this project, the SeCSE methodology has been defined and has served as the main developing process for the SeCSE industrial and academic partners to build service-centric systems. In this section, we discuss two service aspects, namely 1) shift of the ownership of software and 2) coordination across and beyond the boundaries of an organization. Each service aspect is illustrated by a process decision elicited from the deliverable of the SeCSE methodology.

11.4.1 Shift of ownership of software

The concept of software as a service (SaaS) in the service-oriented paradigm leads to the shift of the ownership of software (Dubey & Wagle, 2007). Software is no longer required to be pre-installed on the user’s computer. Rather, software as a service is based on service level agreements (SLAs). Instead of buying (owning) the service, the user uses (rents) the service from a vendor. The user of the service is not the owner of the software anymore. Instead, the service provider (vendor) is the owner of the service. This shift of ownership not only changes the way software is delivered, but also the way software is maintained and upgraded.

Traditionally, users of software pay for maintenance contracts. When faults are detected or requirements changed, users have to ask for support from vendors. If software systems are developed in-house, the users of the systems are seeking support from the development team. This procedure is time-consuming and costly. In the service-oriented paradigm, both users and vendors benefit from the transfer of ownership. From the users’ perspective, users are freed from taking the time and initiative for requesting support from the vendors. From the vendor’s perspective, time and cost are saved because as soon as one service is improved, all the users of that service have the benefit of the improvement.

In TSE, maintenance is a standard process in the software life cycle model. In service engineering instead, more process decisions need to be made. For instance, a process decision must be made regarding how a service can be updated without creating conflicts among various service consumers who share the same service.

Moreover, in the traditional approach it is very difficult to switch to a different software system when users are not satisfied with the current anymore.
Discarding current software is a huge waste of investment. In the service-oriented paradigm, users are able to easily move to any other service as long as the functionality and quality requirements are met. Therefore, extra process decisions must be considered to ensure that the infrastructure of the system supports users to move to other services if they want.

The third difference introduced by the shift of the ownership of software lies in the control on software. Traditionally, a vendor was not allowed to modify or remove the software without the permission of the user. Instead, in the service-oriented paradigm, vendors can modify or remove services without notifying the users. Despite the SLAs between service providers and service consumers, service consumers are often concerned about the ability of the system to recover from services failure. This difference proposes process decisions to ensure that the infrastructure of the system supports system recovery.

To give an example of process decisions that need to be made due to the differences explained above, we elicited a process decision from the deliverable of the SeCSE methodology and documented it (see Table 11.1) based on the template presented in Chapter 9. In this process decision, the possibility for users to easily switch to other services and the recoverability of the system are considered as the main drivers for making the decision.

### 11.4.2 The relevance of cross-organizational cooperation

Cross-organizational cooperation is an industrial practice since several decades, accordingly companies decide to acquire non-core functions externally rather than develop the necessary technology internally (Lee et al., 2003; Franceschini et al., 2003). In this approach, tasks are delegated to specialized third parties (Kakabadse & Kakabadse, 2005). The main drivers for doing so include cost efficiency and production reorganization (Franceschini et al., 2003). Such trend is called IT outsourcing (Lee et al., 2003). As a management approach, it has been widely applied to software development.

The cross-organizational cooperation in the service-oriented paradigm is different. The stakeholders coexist in the SOA domain. Instead of an active-passive relationship (outsourcer and supplier) in the outsourcing approach, the stakeholders coexist in the SOA domain. Process decisions that are made by outsourcers have minor impact on suppliers and vice versa. For instance, a supplier choosing to use the waterfall life cycle model to develop the assigned piece of software does not impact the outsourcers.

We identify three types of cross-organization interaction in the service-oriented paradigm. First, a SBA involves a number of stakeholders and these stakeholders act by assuming various roles at the same time. For instance, services are developed by a service engineer; hosted and published by a service provider; registered

Table 11.1: A SOA process decision due to the shift of ownership of software

<table>
<thead>
<tr>
<th>Design issue</th>
<th>D1. How to support service discovery?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>The architecture of a system must support service discovery in different ways. When the service consumer fails to execute the services, the infrastructure of the system must ensure that the system can be recovered from the failure.</td>
</tr>
<tr>
<td>Ranking criteria</td>
<td>Cr1: flexibility, Cr2: reliability, Cr3: performance</td>
</tr>
<tr>
<td>Architectural decision alternative</td>
<td>D1-Opt1 Static service discovery</td>
</tr>
<tr>
<td>Description</td>
<td>In static service discovery, services are looked up during design time and their location is hard coded in the software.</td>
</tr>
<tr>
<td>Executor of the decision</td>
<td>Service engineer, service integrator</td>
</tr>
<tr>
<td>Stakeholders related to this decision</td>
<td>Service developer: The services that are discovered in this process influence the work of a service developer. The more services that are selected, the less development work the service developer has to perform.</td>
</tr>
<tr>
<td>Status</td>
<td>Discarded</td>
</tr>
<tr>
<td>Rationale</td>
<td><strong>Pros:</strong> Static service discovery has better run-time performance since no communication between service consumers and the service registry has to occur at run-time. <strong>Cons:</strong> The failure of a particular service may lead to the failure of the whole system. Often service consumers are not aware of new services that have similar functionalities but better qualities in the service registry because the location of the discovered services is hard coded in the software. Manual configuration is required when services are migrated.</td>
</tr>
</tbody>
</table>

| Architectural decision alternative | D1-Opt2 Dynamic service discovery |
| Description | In dynamic service discovery, services are looked up dynamically at runtime by service search. |
| Executor of the decision | Composition engine |
| Stakeholders related to this decision | the Service Monitor (who measures and assesses the service QoS and initiates this process indirectly) |
| Status | Discarded |
| Rationale | **Pros:** Alternative services that have similar functionalities but better qualities can be identified at runtime. It is easy to switch other alternative services because their location is not hard coded in the software. **Cons:** Communication between service consumers and the service registry at run-time impact the performance of the system. The failure of a particular service may lead to the failure of the whole system. |

| Architectural decision alternative | D1-Opt3 Combination of static and dynamic service discovery |
| Description | Adopt static service discovery in conjunction with dynamic service discovery. Perform static service discovery at design time while allowing dynamic service discovery for recovery at run-time when errors occur. |
| Executor of the decision | Service engineer, service integrator, Composition engine |
| Stakeholders related to this decision | the Service Monitor (who measures and assesses the service QoS and initiates runtime discovery indirectly) |
| Status | Approved |
| Rationale | **Pros:** Since the services that are required in the system are discovered at design time, no extra communication is needed between service consumers and the service registry. Therefore, the performance remains high. Only when services those become unavailable or fail to meet certain requirements at the run-time, dynamic service discovery is triggered for discovery of alternatives. Although the run-time performance gets lower due to extra communication, the reliability of the system is enhanced. |
by a service broker, integrated into a system by a service integrator and invoked by service consumers. All these stakeholders must cooperate in such a way that the resulting system not only fulfills the end user’s requirements but also meets the individual stakeholder’s requirements. Second, one service can be integrated by various service integrators and invoked by various service consumers. Service providers have to ensure that changes that are made to the service under the request of one service consumer do not impact the usage of the other consumers. Third, services can be selected from various service providers. Therefore, a service consumer has to deal with a number of service providers for SLAs.

The cross-organizational cooperation poses new challenges in process decisions in the context of SOA. The main reason is that process decisions not only influence the development process, but also have an impact on various stakeholders. When arguing about rationale for the decision, we have to keep in mind that other stakeholders might be involved.

To give an example of how process decisions in the context of SOA influence multiple stakeholders, a sample process decision has been elicited from the deliverable of the SeCSE methodology and documented (see Table 11.2). In this example, a choice needs to be made on which stakeholder should perform service monitoring. As illustrated in Table 11.2, choosing a third party for service monitoring impacts the tasks that service provider and service consumer should carry out. The example therefore demonstrates how cross-organizational cooperation demands process decisions in the context of SOA.

11.5 Mapping SOA process decisions on the core model

11.5.1 Gaps in the existing architectural knowledge models

Two sample process decisions in the context of SOA are presented in § 11.4.1 and 11.4.2. Each process decision is explained by means of concern, ranking criteria and a number of architectural decision alternatives. Each alternative decision is described in terms of identifier, description, executor of the decision, related stakeholders, its relationship(s) to the other decisions, status and rationale.

An architectural decision must be made when there is a concern over the resulting system and optional solutions exist to overcome the concern to some extents. These optional solutions are named as architectural decision alternatives in the examples. To judge which alternative is the best for the system, ranking criteria usually reflect the demand of the user and therefore can be used as measurement for making decisions. For the sake of clear representation, each
Table 11.2: A SOA process decision due to the cross organization cooperation

<table>
<thead>
<tr>
<th>Design issue</th>
<th>D2 Which party should be responsible for service monitoring?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Context</strong></td>
<td>Service monitoring is of great importance for a successful SBA because it observes and supervises at runtime the actual behavior of the services. The system must ensure that the monitoring data can be collected by both the service provider and consumer.</td>
</tr>
<tr>
<td><strong>Ranking criteria</strong></td>
<td>C1: complexity, C2: trustworthiness, C3: performance,</td>
</tr>
<tr>
<td><strong>Architectural decision Identifier</strong></td>
<td>D2-Opt1 Service monitoring is performed by the service provider and the service consumer.</td>
</tr>
<tr>
<td><strong>alternative</strong></td>
<td>Both of the service consumer and provider perform service monitoring against the predefined QoS or SLAs. The former focuses on monitoring the consumption of the service in order to determine whether its behavior abides by the agreed QoS or SLA. The latter is more interested in the service execution.</td>
</tr>
<tr>
<td><strong>Executor of the decision</strong></td>
<td>Service provider, service consumer</td>
</tr>
<tr>
<td><strong>Stakeholders related to this decision</strong></td>
<td>D1-Opt3</td>
</tr>
<tr>
<td><strong>Relationship(s)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Status</strong></td>
<td>Approved</td>
</tr>
</tbody>
</table>
| **Rationale** | **Pros**: Both of the service provider and consumer perform service monitoring addressing their own concerns. For instance, the service consumer does not need to collect data about how the increasing number of service consumers affect the performance of the services. The separation of the concerns makes the monitoring task less complicated and more efficient. 
- When service monitoring data shows that a service stops response or performs incorrectly, service recovery procedure including runtime service discovery can be triggered immediately. **Cons**: Monitoring for own concerns lacks of trustworthiness. For instance, when the service consumer detect that a particular service does not fulfill the performance requirements as it is defined in the SLA, the service provider might not agree with the claim. Conflict may occur between the service provider and the service consumer. |

| Architectural decision Identifier | D2-Opt2 Service monitoring is performed by a delegated trusted third party that both the service provider and consumer agree with. |
| alternative | A delegated trusted third party is agreed by both the service consumer and the service provider to monitor the services. This third party regularly reports monitoring data to the service consumer and the service provider with respect to their concerns. |
| **Executor of the decision** | A delegated trusted third party |
| **Stakeholders related to this decision** | Service provider, Service consumer |
| **Relationship(s)** | D1-Opt3 |
| **Status** | Discarded |
| **Rationale** | **Pros**: Both the service provider and consumer are freed from the task of service monitoring. The report from the third party can be used as an evidence for charging the service consumer or penalize the service provider based on the SLAs. **Cons**: Both the service provider and consumer have to agree upon the delegated trusted third party. Extra effort is required for selection and setting up contracts. 
- Extra communication between the delegated trusted third party between the service provider and consumer has to occur. Monitoring data has to be reported on a regular basis. Both of the service provider and consumer have to response (if necessary) after the receiving of the monitoring data. 
- Service monitoring is a prerequisite for runtime recovery. The involvement of a third party introduces complexity. The failure of the service monitoring and the loss of communication with the third party influence the success of the whole system. Both the service provider and consumer do not have control on any problem that might occur. |
alternative decision has an identifier, which can be used as reference. Usually an identifier consists of a sequence number and a title of the decision. The description of the alternative decisions provides more explanation on the decision than the title. The rationale of a decision is of paramount importance for reasoning. Pros and cons of the decision are typically explained there. The state of the decisions, such as discarded, approved or under discussion, is recorded in status. If the decision is taken, its relationship(s) with the other decision(s) can be indicated in relationships. The stakeholder(s) who react(s) on the decision is called the executor of the decision and the stakeholder(s) who is (are) impacted by the decision is called related stakeholders.

From architectural knowledge modeling point of view, the concepts that we use in the tables form a conceptual model for recording process decisions. The two sample process decisions are instances of the conceptual model. In order to analyze whether the current architectural knowledge models can be used to codify process decisions as well, we map the concepts that we have used for representing the process decisions onto the entities that are used in the existing architectural knowledge models. The core model, the ‘Esperanto’ of architecture knowledge, can be used as a common frame of reference (de Boer et al., 2007).

Therefore, we intend to map the concepts that are used in the sample process decisions onto the core model as our first step. In doing so, we are able to locate most of the concepts in the core model. For instance, “ranking” refers to ranking criteria; “alternative” in the core model refers to architectural decision alternatives including identifier and description; a decision with the status of approved is an “architectural design decision” in the core model; the decision loop in the core model captures the relationship(s) between the decisions, etc. The only concepts that we are not able to position in the core model are executor of the decision and related stakeholders.

It is not surprising to see that current architectural knowledge models do not explicitly address stakeholders who execute the decision and stakeholders who are impacted by the decisions. These models work well in TSE when aiming at representing decisions regarding architectural design as a product. These design decisions do not impact stakeholders but the architectural design only. For instance, choosing the client/server model is a design decision which will be executed by lower-level designers and implementers. This decision does not have direct impact on the stakeholders of the system.

However, when the ownership of software starts to shift to the service provider and the increasing cross-organizational cooperation happens in the SOA context, more stakeholders can exercise choices of architectural consequences. Decisions are not only made for the architectural design but also about the development process itself. As we can see in the sample process decisions in Tables 11.1 and 11.2, various stakeholders are involved in development process and they
play more important role in making process decisions. Therefore, architectural knowledge models need to make these executors explicit. This shows a gap in the current architectural knowledge models for representing process decisions in the SOA context.

11.5.2 A suggestion to extend the core model

Process decisions introduce more concepts than the design decisions that are commonly addressed by the existing architectural knowledge models. To overcome this gap, we suggest to extend these models. In particular, we suggest a way to extend the core model for modeling architectural design decisions including process decisions in the SOA context.

Figure 11.1 illustrates our suggested solution. First, an entity named “executor of the decision” can be added in the core model. The entity “Stakeholder” is the generation of “executor of the decision”. A link with the action “execute” between the added entity “executor of the decision” and “architectural design decision” can be created. This link points out that “executor of the decision” will react on the decision. Furthermore, a link with the action “impact” between the entity “architectural design decision” and “stakeholder” can be created to denote that architectural design decisions impact stakeholders. In this way, the core model can be extended to serve as a common language to model architectural knowledge including SOA process decisions.

The two missing entities that we identified are just a starting point. More characteristics are introduced by the service-oriented paradigm eventually causing more entities to be found missing in the current architectural knowledge models. The suggested extension of the core model serves as an example showing how to overcome the inadequacy of modeling the elicited SOA process decisions that are discussed in § 11.4. Further analysis on SOA characteristics and their related process decisions need to be done to gain a complete picture on process decisions in the SOA context.

11.6 Conclusions

Architectural knowledge must be modeled for sharing. Architectural design decisions, a key element in architectural knowledge, attract increasing attention from both the industry and academia. Much work has been done on modeling architectural design decisions and especially regarding architectural design as a product. However, when modern computing paradigms such as the service-oriented paradigm emerge, process decisions as part of architectural design decisions introduce new challenges for architectural knowledge modeling.
Figure 11.1: Extended core model for SOA process decisions
The purpose of this chapter is to: a) emphasize the necessity and importance of making process decisions in the SOA context and b) point out a gap in the existing architectural knowledge models for modeling SOA process decisions. To serve these purposes, two examples of SOA process decisions coming from industrial experience are discussed. How these two process decisions are related to the service aspects introduced by SOA is explained. We have analyzed how to use an architectural knowledge model to represent these two SOA process decisions. The two concepts we are missing in the analyzed model are both related to the stakeholders of a system. We recognize that this is in line with the importance of identifying stakeholders and concerns in most current architectural work. Finally, an extension of the core model is suggested as a first step toward the solution.
12

3D Architecture Viewpoints on Service Automation

SOA is a paradigm for the execution of business-oriented as well as technical infrastructure processes by means of services. Automating the execution of services is of paramount importance in order to fulfill the needs of companies. However, we have found that automation—although important—is seldom addressed explicitly as a concern when stating requirements or designing the software architecture of the SBAs. In this chapter, we define three architectural viewpoints framing the concerns about service automation. These three viewpoints, called 3D (Decisions, Degree, Data), respectively: express architectural decisions about automation; help to identify the level (degree) of automation required, and represent the specific data required to support automation in services. They have been applied to three industrial case studies and one academic experiment. Results show that they successfully support both technical and non-technical stakeholders in understanding how, and communicating upon, their concerns related to service automation have been addressed. The application of the 3D service automation viewpoints to different domains exhibits promising reusability.

12.1 Introduction

Automation is defined as “the execution by a machine agent (usually a computer) of a function that was previously carried out by a human.” (Parasuraman & Riley, 1997). Service automation can be defined as the degree to which the service can be executed automatically without human intervention.

On the one hand, fully automating services would bring obvious benefits to companies (Crawford et al., 2005). Automation of service execution cuts down human resource costs as less human effort is required; it minimizes personnel skill requirements and training time as part of technical decisions are taken by
services; and it prevents faults caused by human mistakes (e.g., typos). More importantly, the execution of automated SBAs (defined as SBAs where every service is automated) is independent from human performance or availability.

On the other hand, fully automating services is in practice not always technically possible or is often restricted by some domain-specific factors. For instance, the automation of service negotiation is extremely difficult as quality requirements must be quantified and formalized as processable information. On top of that, services are often distributed among multiple business partners and thereby not fully controlled by one specific organization. This way, the quality of a SBA depends on the quality of its composing services. Automating service negotiation would mean any malfunctions caused by a service from untrusted service provider would potentially lower the reliability and security of the SBA. As a result, the degree of service automation often depends on the criticality of specific domains and the need of human control.

Although service automation is specifically important for SBAs to maximize their capability of dynamically adapting to business changes, stakeholders’ concerns related to service automation are often poorly addressed in the architecture description of SBAs. For automated services, service developers are often concerned with the policies and guidelines regulating their automation; for services that cannot be automated, human actors are often concerned with their participation in the execution of services and their responsibilities. Architects are concerned with communicating their design decisions about service automation with these and other stakeholders (e.g., managers).

The lack of explicitly addressing service automation in architecture descriptions becomes evident in the study presented in this chapter on the service deployment and configuration architecture (SDCA). The SDCA offers a set of services aiming at automatically deploying new or updated business services of SBAs. The SDCA has been applied to two industrial domains, an enterprise banking domain (called BankFutura) and a personal digital home domain (called HomeFutura). Each service in the SDCA service flow can be configured in terms of the degree of automation to fulfill different requirements of different domains. Despite that service automation is clearly a concern, design decisions on which services can (or cannot) be automated and the rationale behind these decisions are completely missing in the architecture description of the SDCA and its two industrial cases. This lack leads to incomplete architecture description, hurdles sharing and reusing architectural knowledge (AK) (Jansen et al., 2009).

In this work, we take a viewpoint approach (ISO, 2010) for framing a basic set of concerns relating to service automation and providing a set of models, methods and notations to illustrate the way in which the concerns are addressed in architecture design. With this purpose, we followed a typical action research cycle (Susman & Evered, 1978; Avison et al., 1999) with three iterations, each
12.2. AN INTRODUCTION TO AUTOMATION

of which was carried out with one industry partner as a case study. The results of the action research are three service automation viewpoints, which are called 3D service automation viewpoints (for better readability we use abbreviation 3D VP’s in the rest of the chapter). The defined viewpoints are focused on capturing service automation concerns. Therefore, aspects such as detailed modeling of service compositions are out of the scope of our work. They can be provided by already existing, complementary viewpoints.

The 3D VP’s consist of three distinct yet inter-related viewpoints, each dealing with one perspective of service automation. The viewpoint ‘Decision on service automation’ deals with the decisions that have been made on the degree of service automation, including the rationale behind them and the link to domain-specific factors. The viewpoint ‘Degree of service automation’ deals with the overall degree of service automation that has been designed within a service flow. And the viewpoint ‘Data relating to service automation’ focuses on the data that enables different degrees of automation of each service.

To further validate the reusability of the 3D VP’s, we carried out an university experiment in a service-oriented design Master course asking 11 groups of students to apply the viewpoints to illustrate the service automation aspect of their design of a e-Health system. Within academic limitation, the success of this experiment further confirms the reusability of the 3D VP’s to SBAs in general.

The remainder of the chapter is organized as follows. In § 12.2 we introduce service automation as a service aspect and motivate the need for service automation viewpoints in § 12.3. Our research approach is described in § 12.4. In § 12.5 we document the description of the 3D VP’s developed in this work. We then discuss in § 12.6 the visual support that the 3D VP’s provide as well as the applicability of the 3D VP’s. § 12.7 concludes the chapter.

12.2 An introduction to automation

12.2.1 Automation and human interaction

There are many aspects of a system which can potentially be automated, and multiple possible levels of automation. In (Parasuraman et al., 2000) a framework is presented for classifying automation aspects depending on the type of human functions being automated (sensory processing, analysis, decision making, action execution) and the level of automation adopted by the system (from identifying the potential options, to suggesting one of them or automatically taking the decision). Decision making and analysis are frequently the kinds of activities where the decision about the level of automation remains more open, and the balance between human and computer is usually decided for each specific system (Nachreiner et al., 2006).
The emergence of autonomic computing signified one of the latest advances for automation in the field of system management. This paradigm was defined by IBM in 2004 in an attempt to cope with the ever increasing complexity of managing current systems (Kephart & Chess, 2003). The intent is to enable the creation of systems which manage themselves in a completely automated way, analogous to biological systems. In the recent years, this proposal has gained consensus from both industry and academia as the preferred approach for obtaining complete system automation (without human intervention).

A common way to implement autonomic behavior is through the definition of policies (Agrawal et al., 2005), reasoning about the available information and deciding how to react to the current state of the system in order to address specific concerns. Policies are central elements behind automated decision making. Sloman (Sloman, 1994) defines policies as the “rules governing the choices in behavior of a system”. They represent the goals and the operational constraints of a system, enabling the automation of decisions about aspects such as security management (authorization and access control) or network management (network routing) (Bandara et al., 2007). Policies can be specified by different policy definition languages, ranging from high level ones that are readable by human-beings (e.g., PONDER and RBAS) to low level ones that are only meant to be processed by software systems (e.g., PCIM and CORS-PR PIB) (Guerrero et al., 2005).

12.2.2 Automation in SBAs

SBAs often consist of two types of services, business services and management services. Business services implement some business functions, such as hotel booking, care renting or payment. To improve the flexibility and dynamism of SBAs, the management activities supporting the execution of SBAs can also be implemented as services, called management services. Some examples are secure service discovery service in (Czerwinski et al., 1999), a multi-agent automated negotiation service in (Cao et al., 2009), an automatic deployment service (Kecskemeti et al., 2005) and a monitoring service in (Wang et al., 2005a). Maximizing the automation capability of both of these two types of services enables highly dynamic SBAs.

However, in practice automating the execution of services and delivering dynamic and reliable SBAs is not always technically possible or even desirable. SBAs introduce additional concerns when compared to traditional software systems (in Chapter 10), that further complicating. As an example, trust concerns heavily influence automation decisions when the SBA relies on externally developed services. These factors and rationale for decisions on service automation should be explicitly documented to improve the quality of architecture descrip-
12.3. THE NEED FOR SERVICE AUTOMATION VIEWPOINTS

In § 12.3.1 we further elaborate this need using a study on automation of service management and deployment.

12.3 The need for service automation viewpoints

In this section we illustrate why the currently available documentation mechanisms are not sufficient for capturing all the aspects related to automation, using a specific example. We first introduce a study we carried out on the automation of service management and deployment. Further on, we analyze the existing alternatives for capturing that knowledge about the design of the system.

12.3.1 A study on automation of service management and deployment

Deployment and configuration of SBAs are key management activities which need to be as automated as possible in order to foster their dynamicity. Automation is addressed by the system analyzed in this work, the SDCA (Service Deployment and Configuration Architecture); a SBA which provides a deployment and configuration service for distributed applications and services (Ruiz et al., 2009). The deployment functionality is offered by eight loosely coupled services which collaborate to obtain a distributed deployment plan (containing a set of changes that will be applied to the runtime environment), starting from an initial objective. The composing services perform activities such as connecting to the instrumentation agents for retrieving the runtime information, accessing software repositories, deciding which compatible version of a service to use or choosing where each deployable artifact should be physically located. The architecture has been designed with flexibility in mind as regards all the decisions taken during the process. This way, they can either be completely automated though the definition of policies or be manually controlled by requiring human input for decision making.

Over its lifespan, the SDCA has been applied in two different domains: the delivery of services to a Digital Home (HomeFutura) and the support for deploying banking deployment processes (BankFutura). These fields of application have very different concerns: the banking domain imposes hard requirements of reliability, security and stability of the system, while these aspects are not as critical for the end users which consume the services offered at HomeFutura. Additional factors, such as the technical background of the users of the SDCA (IT administrators versus end users), or the specific characteristics in each domain of the runtime infrastructure and the deployed services also present important differences in both cases.
These differences caused variation in the degree of automation of the services of one domain when compared to the other, which was made possible by the flexibility of the architecture. As an example of that, for HomeFutura the selection of which runtime node will host each service is automated by using a load balancer policy. In this scenario, the decision is not relevant for the end user, who does not have a technical background and just wants to start consuming the service as soon as possible. On the other hand, in BankFutura the same service is manually controlled by IT administration, with a deep knowledge of the runtime domain and an understating of the criticality of the service. Because of the same factors, deployment plans are instantly executed in HomeFutura, while they must be scheduled by the administrator in the banking organization, in order to provoke as less impact to the running services as possible.

Clearly, the developed solutions for both domains did take into account all those specific concerns in order to decide upon the automation of the execution flow. However, neither the concerns, nor the rationale or decisions about it were made explicit in the complete documentation of the system. This information is vital to understand the design decisions which were taken over the process, inform system actors about its intended participation and guide the potential evolution of the SDCA from its current state. For instance, the question of which services can be executed in any application domain without any human intervention from the SOA manager of the SDCA is not answered in the documentation. Instead, the documentation only points out that the SDCA supports automatic service deployment, which could be interpreted as no human actors are required at all during the service deployment process, which is not the case. If we go to the domain-specific implementations, the SOA architect of the banking domain cannot find an answer for the question of which quality attributes have an impact on the degree of automation of the deployment process, as the link between quality attributes and service automation is missing from the documentation.

Those documentation problems are common to most developments. In practice, the usefulness of architecture description is hardly recognized by stakeholders (or readers) because architecture description becomes easily outdated, inconsistent, contains either far too much information or too abstract to understand (Lethbridge et al., 2003). The software design is often documented due to organizational policies, without having in mind the expectation of readers. Whether the readers can understand and retrieve useful information from the documentation is often neglected. A potential solution to these problems is proposed in (Clements et al., 2002a); this work suggests to adopt expressive mechanisms to convey information to stakeholders, and apply them to document the information that is relevant to each stakeholder. This way, stakeholders are more motivated to read the architecture description, ultimately improving the usefulness of architecture description.
12.3.2 Documenting software architecture

Software systems must address the concerns raised by multiple stakeholders (e.g., architect, project manager and developer). These concerns must be supported and documented by the software architecture. A popular approach for documenting these concerns consists of following the separation of concerns principle, defining multiple architecture views (Bass et al., 2003). Each view focuses on specific aspects of the system architecture, such as its structure (Selonen & Xu, 2003), or the component data flow (Liu et al., 2008), and is used only by the relevant stakeholders.

Architecture viewpoints provide the means to define and reuse views among multiple systems. Some well-known view models from industry include: Kruchten’s “4+1” View Model (Kruchten, 1995), Hofmeister’s view model (Hofmeister et al., 2005), Software Engineering Institute (SEI) set of views (Clements et al., 2002b), ISO Reference Model of Open Distributed Processing (RM-ODP) (Farooqui et al., 1995), Zachman Framework for enterprise architecture (Zachman, 1987) and the viewpoint catalog collected by Eeles and Cripps (Eeles & Cripps, 2009).

Existing viewpoints address concerns that often appear in traditional software architectures. Recent software architecture styles (like SOA), bring additional specific concerns that challenge the reusability of the existing viewpoints. The lack of available views make practitioners face difficulties to find an effective way to illustrate any new characteristics introduced by these architecture styles. As a result, viewpoints enabling the illustration of specific concerns introduced by modern software architecture styles like SOA are needed.

To our knowledge, none of the available viewpoints capture the concerns about service automation, leaving a gap to be filled. The usage of service automation viewpoints is twofold. On the one hand, the viewpoints frame a basic set of concerns that are commonly related to service automation. The identification of these concerns saves effort from architects and avoids the danger of overlooking them. On the other hand, service automation viewpoints comprise a set of conventions that facilitate the creation of architecture views illustrating all the information needed to address service automation related concerns.

12.4 Research approach

With the aim of defining a set of viewpoints framing concerns commonly relevant to service automation, we followed an action research approach. Action research is an iterative research approach where the researcher actively participates in the case studies that he/she performs (Susman & Evered, 1978). The motivation for carrying out such an approach complies with what has been defined by Avison et al. (Avison et al., 1999), which is “try out a theory with practitioners in real
situations, gain feedback from these experiences, modify the theory as a result of this theory and try it again”. In the context of this work, the theory is made of a set of viewpoints on service automation; the real situations refer to the industrial case studies that we carried out; and the result of the theory refers to a set of views on service automation that can be used for architecture description.

Susman and Evered (Susman & Evered, 1978) described the action research process as a five phase, cyclical process. These five phases (see Figure 12.1) include: 1) diagnosing, 2) action planning, 3) action taking, 4) evaluating, and 5) specifying learning. Diagnosing refers to the identification of the primary problems. Action planning specifies actions that should be undertaken to solve these problems. These actions are then implemented in the action taking phase. The outcomes of the actions are evaluated in the evaluating phase. Lastly the knowledge gained in previous phases is collected in the specifying learning phase, which may serve as valuable input to the next cycle.

We have followed the action research approach above with the goal of defining a set of reusable service automation viewpoints. We carried out three case studies (described in the following) in three associated action research cycles (see Figure 12.1): each cycle produced a set of views and viewpoints, which have been both challenged in the subsequent case study, and evaluated and refined to cross-check for consistency with the previous case study.

Figure 12.1: The research approach

The first case study was carried out with the UPM (Universidad Politécnica de Madrid) team, which designed and implemented the SDCA. In Phase 1: Diagnosing, we studied the documentation of the SDCA and identified their needs for explicitly addressing concerns about service automation. In Phase 2: action planning, we identified a list of stakeholders that might have concerns relating
12.5. 3D SERVICE AUTOMATION VIEWPOINTS

to service automation and planned interview questions. In addition, we designed a view-driven approach for developing viewpoints, deriving them from diagrams that are confirmed by stakeholders.

The main work has been carried out in Phase 3: action taking. There we first conducted interviews with the stakeholders of the SDCA with the aim of eliciting their concerns regarding service automation. From this set of concerns, we identified what information should be depicted in the diagrams to address these concerns. As some concerns might require the same type of information, analyzing this information helped us in reducing the number of necessary diagrams.

After we established what information should belong to each diagram, we created them, with special attention to stakeholder understandability. We then generalized the diagrams in an initial set of viewpoints on service automation (which is the theory of the action research approach). Using this initial set of viewpoints, we created a set of views for the SDCA. We brought back these views to the stakeholders with the question of whether these views were correct (content-wise) and aiding understanding of how the concerns relating to service automation had been addressed in the design.

The feedback that we received from the stakeholders and domain experts enabled us to refine the views and viewpoints in Phase 4: evaluating to improve their expressiveness and completeness. After several iterations of interviews and refinements, we obtained a final set of viewpoints framing all the concerns that we have elicited. This set of viewpoints as well as the lessons that we learned in this action research cycle were valuable inputs for the next action research cycle.

Our next case study (see Iteration 2 in Figure 12.1) was carried out with BankFutura, which applied the SDCA to their service configuration management. The phases that we took in this cycle were similar to the ones in the SDCA case study. The only difference was that at the end of the cycle of the BankFutura case study, we revisited the cycle of the SDCA case study to ensure that the viewpoints created in the BankFutura cycle could be successfully applied to the SDCA as well. The same applied to the third case study carried out with HomeFutura (Iteration 3). The viewpoints created in the HomeFutura case study were evaluated and refined again to fulfill the needs of both SDCA and BankFutura.

12.5 3D service automation viewpoints

12.5.1 An overview of 3D service automation viewpoints

To illustrate the aspects that are relevant to service automation in architecture design, we developed three service automation viewpoints. The ‘Degree of service automation’ viewpoint addresses the concerns about the achieved and potential degree of automation for each service. The ‘Decision on service automation’
viewpoint documents the reasoning behind the decided degrees of automation. The ‘Data relating to service automation’ viewpoint illustrates what information and actors are involved in the degrees of service automation.

The 3D viewpoints are complementary, being relevant to different types of stakeholders and documenting different concerns (Dijkman et al., 2003; Boucké et al., 2008). The degree viewpoint identifies what services form the SBA, and what is the degree of automation for each of them. This information is of great interest to all the stakeholders and thereby is exposed to the “public”. The remaining two viewpoints build upon that general information. The data viewpoint documents the data and actors that intervene over service execution automation. The decision viewpoint has its information enhanced by the remaining two views. Finally, the decision viewpoint provides to the “internal” design team (e.g., architects) the means for documenting and reusing the knowledge on automation decision making. The rationale for this viewpoint is influenced by the actors and policies from the data viewpoint and refers to the services and degrees illustrated by the degree viewpoint.

12.5.2 Documenting architecture viewpoints

In the literature, there are multiple proposals for documenting viewpoints. Among them the template proposed by Hilliard (Hilliard, 2009) is in our opinion the most comprehensive one and, most importantly, reflects the requirements on viewpoints specified in ISO/IEC 42010) (Emery & Hilliard, 2008). Therefore, we chose to use this template to document the 3D VP’s developed in this work. Table 12.1 presents and describes the required items of the template.

In the following, we shall present the description of the viewpoint ‘Decision on service automation’ (§ 12.5.4), ‘Degree of service automation (§ 12.5.5), and ‘Data relating to service automation’ (§ 12.5.6) using the items of the template in Table 12.1. For each of these viewpoints, we explain in detail its metamodel and conforming notation, which are the key elements to be used to create views, as well as an example view. To illustrate the application of the 3D VP’s, we take a fragment of the SDCA service flow as a running scenario, introduced in the following section.

12.5.3 A fragment of the SDCA service flow - running scenario

With the purpose of 1) showing how to construct views by applying the proposed 3D VP’s and 2) illustrating how the constructed views address the concerns framed by the 3D VP’s, we created three service automation views as examples based on a subset of the BankFutura SDCA service flow (introduced in § 12.3.1).
### Table 12.1: Summary of Hilliard’s template for documenting architecture viewpoints

<table>
<thead>
<tr>
<th>Template item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Viewpoint Name</strong></td>
<td>Name of the viewpoint, and accepted synonyms.</td>
</tr>
<tr>
<td><strong>Overview</strong></td>
<td>Brief overview of the viewpoints.</td>
</tr>
<tr>
<td><strong>Typical stakeholders</strong></td>
<td>The typical audience for views prepared using this viewpoint.</td>
</tr>
<tr>
<td><strong>Concerns</strong></td>
<td>List of the architecture-related concerns framed by this viewpoint.</td>
</tr>
<tr>
<td><strong>Anti-Concerns</strong></td>
<td>Kinds of issues this viewpoint is not appropriate for.</td>
</tr>
<tr>
<td><strong>Model types</strong></td>
<td>Each type of model used by the viewpoint.</td>
</tr>
<tr>
<td><strong>Model languages</strong></td>
<td>Vocabulary used for constructing the view.</td>
</tr>
<tr>
<td><strong>Viewpoint metamodels</strong></td>
<td>The conceptual entities, their attributes and the relationships that comprise the vocabulary of a type of model.</td>
</tr>
<tr>
<td><strong>Conforming notation</strong></td>
<td>An existing notation or model language to be used for the model.</td>
</tr>
<tr>
<td><strong>Model correspondence rules</strong></td>
<td>The viewpoint may specify model correspondence rules.</td>
</tr>
<tr>
<td><strong>Operations on views</strong></td>
<td>Methods which may be applied to views and their models.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>Examples for the reader.</td>
</tr>
<tr>
<td><strong>Notes</strong></td>
<td>Any additional information.</td>
</tr>
<tr>
<td><strong>Sources</strong></td>
<td>The sources for this viewpoint, if any, including author, history, literature references, prior art, etc.</td>
</tr>
</tbody>
</table>
CHAPTER 12. 3D ARCHITECTURE VIEWPOINTS ON SERVICE AUTOMATION

The subset of the SDCA service flow is presented in Figure 12.2 and shortly described as follows.

![Figure 12.2: A subset of the SDCA service flow](image)

The deployment service Select Unit selects one unit among the available ones to be deployed in the environment. The selection criteria should be provided to the service as an external input. After that, the deployment service Resolve Unit is invoked, where the logical requirements of the deployment unit containing the service are analyzed, in order to find a closed set of units satisfying all their dependencies. There might be multiple candidate units satisfying one dependency (e.g., multiple units with minor, compatible versions) and for those cases a unit satisfaction criteria should be provided as an external input. Once the complete set of units that will participate in the operation has been identified, the deployment service Obtain Possible Mappings evaluates the available resources from the runtime containers (e.g., application servers), and returns for each unit the potential nodes of the environment where those can be deployed. Having these nodes, the deployment service Map Units To Nodes decides on the final destination for each involved unit, according to external distribution criteria.

The views created based on this scenario are used as examples in the description of the 3D VP’s. For a detailed description of the 3D service automation views, please refer to (Gu et al., 2010) where we report on the views created for the complete service flow for both BankFurtura and HomeFutura.

12.5.4 Viewpoint ‘Decision on service automation’
(Decision Viewpoint)

SOA architects are often concerned with the justification of decisions on the degree of automation. The rationale behind the decisions as well as the decisions themselves are architectural knowledge valuable for organizations. A good representation of such knowledge aids the SOA architects in analyzing the design...
space and making optimized decisions (Ali Babar & Lago, 2009). It also enables
the SOA architects to communicate their decision with the other stakeholders.
The viewpoint ‘Decision on service automation’ is defined to serve this purpose.
The description of the Decision Viewpoint is presented in Table 12.2.

The metamodel of the Decision Viewpoint The metamodel presented in
Figure 12.3 illustrates the conceptual relationship between the constructs of the
decision model, which includes services with a decided degree of automation,
domain-specific factors and justifications for decisions.

Each Service (see Figure 12.3) has a name and is designed to support a certain
Degree of automation. The Degree of automation can be either Domain
dependent or Domain independent, meaning that the degree of automation of a
particular service differs when the service is applied in different application
domains. Next to classifying degrees from the perspective of domain dependency,
this viewpoint defines five degrees of automation from the perspective of human
intervention): when a service does not require additional input during its execution, it is *Fully automated* as no human intervention is needed at all; when a
service does require additional input and this input can only be provided by a
human actor, it *Has to be semi-automated* and if it can only be provided by a
policy, it *Has to be policy-driven automated*; when the service flexibility allows
the input to be provided by a human actor or policies, the service is *Either semi-
automated or policy-driven automated*; if the input is decided to be provided by
### Table 12.2: Viewpoint ‘decision on service automation’ (Decision Viewpoint)

<table>
<thead>
<tr>
<th>Viewpoint item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Viewpoint Name</strong></td>
<td>Decision on service automation viewpoint (Decision Viewpoint)</td>
</tr>
<tr>
<td><strong>Overview</strong></td>
<td>The viewpoint deals with the links between domain-specific factors and the degree of automation that can be applied to each atomic service in a business process.</td>
</tr>
<tr>
<td><strong>Typical stakeholders</strong></td>
<td>SOA architects</td>
</tr>
<tr>
<td><strong>Concerns</strong></td>
<td><strong>Justification of decisions on the degree of service automation</strong> SOA architects are concerned with what are the most appropriate decisions on the degree of service automation.</td>
</tr>
<tr>
<td></td>
<td><strong>Reconfigurability in terms of automation</strong> SOA architects are concerned with what are the possible alternatives on the degree of service automation when requirements or constraints change.</td>
</tr>
<tr>
<td><strong>Model types</strong></td>
<td>Service automation decision model (Decision model)</td>
</tr>
<tr>
<td><strong>Model languages</strong></td>
<td>This model is expressed in terms of these constructs: services, and domain-specific factors that influence or lead to the decisions on service automation, decisions that are relevant to service automation, and rationale for the decisions. The decision model can be illustrated by the graphical notation proposed below.</td>
</tr>
<tr>
<td><strong>Viewpoint metamodels</strong></td>
<td>See Figure 12.3</td>
</tr>
<tr>
<td><strong>Conforming notation</strong></td>
<td>See Figure 12.4</td>
</tr>
<tr>
<td><strong>Model correspondence rules</strong></td>
<td>All the services in the degree of service automation view should appear in this view. The decision on the degree of service automation of each service in this view should be correspondent to the degree of service automation of each service in the degree view.</td>
</tr>
<tr>
<td><strong>Operations on views</strong></td>
<td><strong>Creation methods:</strong> To create a decision view, first identify which domain-specific factors have an impact on the degrees of automation of the services and then link the architecture constraints with the services with the corresponding rationale. The rationale explains how each factor leads to a decision on the degree of service automation. <strong>Interpretive methods:</strong> In the decision view, the decision on automation is rendered by two attributes: domain-dependency and re-configurability. In particular, the border of ovals shows the domain dependency of the decision (a dashed line suggests that it is domain dependent, while a straight line suggests that it is domain independent). When applying the services to another application domain, this attribute highlights that domain independent services may keep the same degree of automation whereas domain dependent services have to be justified against the new domain-specific factors. Furthermore, the shadow of ellipses points out if an alternative degree of automation is possible and thereby the service can be re-configured if needed. <strong>Analysis methods:</strong> Model correspondence rules can help the SOA architects assess consistency with other views. The information presented in the decision view not only shows the current degree of automation but also guides future re-configurations in the degree of automation. This is supported by two aspects of the view. First, the shadow reflects open degrees of automation with the current constraints. Second, the model provides traceability between the constraints and the affected services; hence the impact of those changes in the SBA will be clearly identified easing its analysis.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>See Figure 12.5</td>
</tr>
<tr>
<td><strong>Notes</strong></td>
<td>In principle, the rationale for fully automated services should explain why they do not need external inputs during their execution. The rationale for other degrees of services must motivate why external inputs are needed, why human actors have to provide it or why it is possible to define policies to assist the execution of services. If the degree of service automation is domain dependent, at least one domain constraint has to be linked to the service.</td>
</tr>
<tr>
<td><strong>Sources</strong></td>
<td>.</td>
</tr>
</tbody>
</table>
12.5. 3D SERVICE AUTOMATION VIEWPOINTS

a human actor, a service is *Opt to be semi-automated*, whereas if the input is decided to be provided by a policy, a service is *Opt to be policy-automated*.

The decision on the degree of automation is heavily influenced by Domain-specific factors, including non-functional requirements, business goals, architecture constraints, characteristics of the infrastructure, skills of the human actors, etc. These factors lead to a set of Rationale that justifies the decisions on automation for services. In addition, when a factor impacts a set of services, its rationale is linked to a Scope encasing all the services, and influences the decision on each affected service.

Note that although services that *Have to be semi-automated* (or *Have to be policy-driven automated*) and *Opt to be semi-automated* (or *Opt to be policy-driven automated*) are designed with the same degree of service automation, they are different from a decision making perspective. The former type of decision is a “has-to” decision in that only human actors (or policies) can provide the additional inputs and thereby no other choice can be made. The latter is an “opt for” decision in that both human actors and policies can provide the additional inputs and thereby the architect can make a selection between them based on the domain-specific factors of the system of interest.

One could argue that the reasoning behind “has-to” or “opt-to” is often embraced in the rationale and wonder why they are regarded as two different degrees of automation. Indeed, the rationale for decisions on service automation should explain what are the alternatives and what are the decisions and why. However, this explanation is usually descriptive and embedded in the text. By explicitly differentiating these two types of decisions and modeling them as first class entities, the expressiveness of the decision model is enhanced and directly facilitates (potential) future re-configuration and SBA evolution.

**Conforming notation of the Decision Viewpoint** Figure 12.4 illustrates the graphical notation designed for creating decision models. Services are represented by ellipses; the three ones at left most of the figure represent three different degrees of service automation that have been decided. Moreover, they also imply that alternative degrees of service automation are not feasible or reasonable. The notations in the second column of the figure represent that a selection has been made or left open among alternative degrees of service automation. These services can be re-configured to an alternative degree of service automation if necessary.

The two notations in the third column denote the dependency between a degree of service automation and a specific domain. The notations in the last column are used to illustrate the relation between decisions, domain-specific factors, and associated rationale, as well as the scope of services where architecture constraints may have an impact on.
An example of decision view  An example of the decision view created for BankFutura (based on the service flow described in § 12.5.3) is presented in Figure 12.5. This example documents the decisions taken on the degree of service automation. In the following, we shall explain how this decision view addresses the concerns (see Table 12.2) framed by the Decision Viewpoint.

The concern about justification of decisions on the degree of automation is addressed by modeling the rationale in the decision view, explaining whether external inputs are required during the execution of services and (if so) motivating whether such inputs can be obtained automatically or need to be given by human actors. For instance, as shown in Figure 12.5 service Obtain Possible Mappings does not require any additional decisions or parameters during its execution and thereby is fully automated. In turn, service Map Units To Nodes requires distribution criteria as an external input during its execution. As explained by the attached rationale, Map Units To Nodes is decided to be semi-automated since control of distribution criteria is more feasible if manually performed by a deployer in BankFutura.

Moreover, justification of decisions on the degree of automation is also addressed by domain-specific factors. For instance, in Figure 12.5 the quality attribute “security” (at bottom left corner of the figure) impacts all services, as indicated by its link through the rationale to the scope (the big rectangle in Figure 12.5), implying that critical activities should be manually controlled if possible (explained by the attached rationale). Next to security, human actor is another domain-specific factor that has an impact on the degree of service automation. In BankFutura, as a banking deployer handling deployment tasks is available and has the ability to manually control certain critical activities, it is possible to dedicate some critical deployment tasks to this human actor if needed.

Another concern framed by the Decision Viewpoint is reconfigurability in terms of automation. This concern is addressed by “opt to be” services, identi-
12.5. 3D SERVICE AUTOMATION VIEWPOINTS

fied in the view with the shadows of the ovals, as it is shown in Figure 12.5. For these services, the decisions on their degrees of automation could change if the domain-specific factors change. For instance, since Select Unit carries out critical activities and any mistakes would lead to unstable deployment configurations, BankFutura opts for semi-automating this service. However, suppose the quality requirements were not stability and criticality but agility or flexibility, the architect could decide to automate Select Unit by means of policies rather than requiring a human actor to make the selection.

![Diagram of decision view](image)

**Figure 12.5:** An example of decision view
CHAPTER 12. 3D ARCHITECTURE VIEWPOINTS ON SERVICE AUTOMATION

12.5.5 Viewpoint ‘Degree of service automation’ (Degree Viewpoint)

Since services are the building blocks of SBAs, the degrees of automation of individual services directly determine the degree of automation of the whole SBA. An overview of the extent to which each service of a SBA can be executed without human-intervention is not only of great interest to managers who are responsible for resource allocation and planning but also for human actors who participate during the execution of services. The Degree Viewpoint is developed to address concerns of these two types of stakeholders. The description of the Degree Viewpoint is presented in Table 12.3.

The metamodel of the Degree Viewpoint

The relationship between the constructs of the decision on service automation model is presented in Figure 12.6. Each Service has a name and is designed with a Degree of automation. There are three basic degrees of service automation: Fully automated (i.e. no Additional input is required), Semi-automated (i.e., a Human actor provides the Additional input required by services), and Policy-driven automated (i.e., services retrieve required Additional input from Policies). A hybrid degree reflects that the service is Either semi-automated or policy-driven automated. Each service can be linked to another service with an execution sequence.

![Figure 12.6: The metamodel of the Degree Viewpoint (in UML)](image)

The conforming notation of the Degree Viewpoint

As denoted by the graphical notation presented in Figure 12.7, the degrees of automation are graphically rendered by the darkness of the color assigned to each service: the darker
### Table 12.3: Viewpoint ‘Degree of service automation’ (Degree Viewpoint)

<table>
<thead>
<tr>
<th>Viewpoint item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Viewpoint Name</strong></td>
<td>Degree of service automation viewpoint (Degree Viewpoint)</td>
</tr>
<tr>
<td><strong>Overview</strong></td>
<td>The viewpoint shows the degrees of automation services are designed for and whether each service can be automated or not. If human intervention is required, the Degree Viewpoint highlights which actors should participate during the execution of services.</td>
</tr>
<tr>
<td><strong>Typical stakeholders</strong></td>
<td>Human actors and managers</td>
</tr>
<tr>
<td><strong>Concerns</strong></td>
<td>Degree of service automation Managers are concerned with the degree of automation that the services are expected to achieve. Accountability Human actors are concerned with which services can be executed automatically and which services require intervention from human actors.</td>
</tr>
<tr>
<td><strong>Model types</strong></td>
<td>Degree of service automation model (Degree model)</td>
</tr>
<tr>
<td><strong>Model languages</strong></td>
<td>The degree of service automation model is expressed in terms of these constructs: services with different degrees of automation, human actors that are required during the execution of services. This model can be illustrated by the graphical notation proposed below.</td>
</tr>
<tr>
<td><strong>Viewpoint metamodels</strong></td>
<td>See Figure 12.6</td>
</tr>
<tr>
<td><strong>Conforming notation</strong></td>
<td>See Figure 12.7</td>
</tr>
<tr>
<td><strong>Model correspondence rules</strong></td>
<td>Services as well as their degrees of automation should be correspondent to the data model of the data view and the automation decision view. Human actors should be correspondent to the data model of the data view.</td>
</tr>
<tr>
<td><strong>Operations on views</strong></td>
<td><strong>Creation methods:</strong> a degree view can be created by retrieving the decided degrees of automation for services (from the decision view) and adding that information to the sequence of participating services. <strong>Interpretive methods:</strong> the degree of automation is illustrated by means of the darkness of the color. The darker the color, the higher degree of automation it represents. The reader of this view may capture how many levels of automation the services are designed for. The sequence between the services highlights the interaction between services and the variation in the degrees of automation, which can point to bottlenecks in the complete service execution flow. <strong>Analysis methods:</strong> model correspondence rules can help the SOA architects assess consistency with other views. The degree model not only provides a holistic overview on the degree of service automation designed for each service, but also provides guidance on detecting possible bottlenecks during the service execution flow. In principle, each semi-automated service could introduce a bottleneck when the corresponding human actor does not provide required information in time. <strong>Implementation methods:</strong> services with different degrees of automation should be provided with different interfaces dealing with different types of input/output sources.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>See Figure 12.8</td>
</tr>
<tr>
<td><strong>Notes</strong></td>
<td>The degree view is meant to show the designed degree of automation of services, whether a service has to be or opt to certain degree of automation is irrelevant to the stakeholders that are interested in this view. A service in the decision view that either has to be semi-automated and opt to be semi-automated requires inputs directly from human actors, and therefore correspondent to semi-automated services. The same applies for policy-driven automated services.</td>
</tr>
<tr>
<td><strong>Sources</strong></td>
<td>The Degree Viewpoint is derived both from UML use case diagrams and Business Possess Modeling Notation (BPMN), both of which describe a set of actions carried out by a list of actors. Similar with use case diagrams, horizontal ellipses are used in the Degree Viewpoint to represent atomic services and actors are directly linked to services. Similar with BPMN, arrow lines are used to denote sequences between two services. On top of that, we add colors to ellipses to represent services with different degrees of automation. This visualization technique helps (in particular) non-technical stakeholders in capturing the degree of automation designed for the services.</td>
</tr>
</tbody>
</table>
the color, the higher the degree of automation. In addition, human actors can be associated to semi-automated services with the purpose of highlighting who are expected to provide input to which services (see association *provides input to*). The sequence between services indicates the order in which the deployment services are invoked (see association *executes after*). With this additional information, the period during which human intervention is (and is not) required becomes explicit. By illustrating that external inputs are expected to be provided by *whom* and *when*, this view also addresses the accountability concern.

![Figure 12.7: The conforming notation of the Degree Viewpoint](image)

**An example of degree view** An example of the degree view created for Bank-Futura (based on the service flow described in § 12.5.3) is presented in Figure 12.8. Corresponding to the example of decision view (presented in Figure 12.5), this example shows the decided degree of automation for each service. In the following, we shall explain how this degree view addresses the concerns (see Table 12.3) framed by the Degree Viewpoint.

![Figure 12.8: An example of degree view](image)
The concern about *degree of service automation* is directly reflected at the metamodel, and is graphically depicted by different shades of color blue. By looking at Figure 12.5, one can easily understand that three different degrees of automation have been designed for the four services, among which two services require human intervention. Furthermore, the vertical positioning of the services depending on their degree of automation improves its graphical expressiveness, simplifying its comprehension.

The concern about *accountability* is addressed mainly by the association of the human actor with the semi-automated services. More specifically, services *Select Unit* and *Map Units To Nodes* are directly linked to a banking deployer, indicating that this human actor should provide required information during their execution. For the banking deployer, it is obvious to see when he or she should be available to perform certain tasks and which services are particularly relevant to them in the service flow. For the manager, the degree view facilitates service management in that potential bottlenecks (e.g., waiting for human inputs) can be detected.

### 12.5.6 Viewpoint ‘Data relating to service automation’
(Data Viewpoint)

Whenever the execution of a service cannot be completely automated, during execution it requires certain data, which must be provided by external sources. In order to facilitate the execution of the service, it is important to know what kind of data is missing and how to provide such data. These questions are often the concerns of human actors (e.g., *when to input what data to which services*), architects (e.g., how to ensure the input data is understood by services) and managers (e.g., the alignment between input data with organizational regulations or goals). The Data Viewpoint is developed to frame these concerns and its description is presented in Table 12.4.

**The metamodel of the Data Viewpoint**  The relationship between the constructs of the data flow model and policy model is presented in Figure 12.9. Each Service designed to support a certain Degree of automation has a name and can be linked to another service with a sequence flow. Services that are Either semi-automated or policy-driven automated require Additional input: a Policy-driven automated service requires Policy input while a Semi-automated service requires Human input provided by a Human actor. These additional inputs are guided by associated Policies: policies guiding Human inputs are often Guidelines or rules that are less structured; whereas Policies accessed directly by services have to be machine readable and thereby documented in a more structured way (Formalized policy). Each Policy has an ID, a name and a description. It is associated to a
**Table 12.4: Viewpoint 'Data relating to service automation' (Data Viewpoint)**

<table>
<thead>
<tr>
<th>Viewpoint Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>The Data Viewpoint focuses on the generation, retrieval, processing, and organizing of data that are relevant to the support of different degrees of service automation.</td>
</tr>
<tr>
<td>Typical stakeholders</td>
<td><strong>Human actors, SOA architects, and managers</strong></td>
</tr>
<tr>
<td>Concerns</td>
<td><strong>The specification of policies</strong> SOA architects are concerned with the format in which policies should be defined so that human actors can easily understand them and services can automatically process them. Managers are concerned with what policies have to be in place before the services are ready in execution and who controls them. <strong>Compliance with organizational regulations</strong> Managers are concerned with whether the external inputs provided to non-completely automated services comply with organizational regulations or goals. <strong>Human participation.</strong> Human actors are concerned with what information needs to be provided to the services, how to provide correctly the information and when it should be provided.</td>
</tr>
<tr>
<td>Model types</td>
<td>Automation-related policy model (policy model) captures the organizational-level policies that are required during the execution of services. Automation-related data flow model (data flow model) captures the policies and human input that the services will require during their execution. In addition, it captures how the services will enforce those policies and how human input interacts with the services.</td>
</tr>
<tr>
<td>Model languages</td>
<td>Policy model: policies can be described in terms of their name, description, the associated service and their format. These elements can be captured in a table. Data flow model: it is expressed in terms of these constructs: policies, human input, actors and services. The data flow model can be illustrated by graphical notation proposed below.</td>
</tr>
<tr>
<td>Viewpoint metamodels</td>
<td>See Figure 12.9:</td>
</tr>
<tr>
<td>Conforming notation</td>
<td>For data flow model see Figure 12.10; for policy model see Table 12.5</td>
</tr>
<tr>
<td>Model correspondence rules</td>
<td>Services as well as their degrees of automation should be correspondent to the data model of the data view and the automation decision view. Human actors should be correspondent to the data model of the data view.</td>
</tr>
<tr>
<td>Operations on views</td>
<td><strong>Creation methods:</strong> Create a data flow model before a policy model. To create a data flow model, first analyze what external information requires each non-completely automated service. Next, decide which policies are required for guiding human actors and assisting policy-driven automated services. To create a policy model, first record all the policies in the data flow model. Then define each policy in terms of its name, description, controller and format. <strong>Interpretive methods:</strong> In the data flow model, the graphical color scheme applied to the degree of automation is also applied to the corresponding policies. Policies illustrated in dark color directly assist the execution of policy-driven automated services; while policies illustrated in light color guide the human actors to provide information to the semi-automated services. In the policy model, the reader may obtain an overview of each system policy related to automation. <strong>Analysis methods:</strong> Model correspondence rules can help the SOA architects assess consistency with other views. As services that are not fully automated require additional information during their execution, it is important that this additional information is illustrated in this view. <strong>Implementation methods:</strong> The policy model describes important information about the policies, such as the format of definition, which can guide the implementation of the policy-driven automated services. In a similar way, defining the required information for semi-automated services, and identifying the participating actors for each one, will also guide their implementation.</td>
</tr>
<tr>
<td>Examples</td>
<td>For data flow model see Figure 12.11; for policy model see Table 12.6</td>
</tr>
<tr>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>Sources</td>
<td></td>
</tr>
</tbody>
</table>
service that is either Semi-automated or Policy-driven automated with the purpose of assisting their execution. The assigned Human actor controls the Policy, ensuring that its content complies with organizational or architectural goal and that its format is aligned with the requirements of the architect. Each policy is specified in a certain format, defined by the architect.

Conforming notation of the Data Viewpoint The data flow model can be constructed using the graphical notation presented in Figure 12.10. Besides the notation for the three basic degrees of service automation, we distinguish the guidelines/rules from formalized policies. While both guidelines/rules and formalized policies are relevant to service automation, the former are used by the human actors to drive the decision and the latter are directly accessed by deployment services to achieve policy-driven automation.

The leftmost side of Figure 12.10 shows the graphical notation denoting the relationships between the elements related to policy-driven automated services. A formalized policy directly assists the execution of a policy-driven automated service by providing the data that it needs. The middle part of Figure 12.10 shows the graphical notation denoting the relationships between the elements related to semi-automated services. More specifically, a human actor is responsible for providing a type of information that is required by a semi-automated service. For this purpose, the human actor is guided by certain guidelines or rules.

Given these details on the relationships between policies, human actors, and services, the human actors can tell which services are expecting what information from them. Moreover, it explicitly points out which organizational guidelines or
CHAPTER 12. 3D ARCHITECTURE VIEWPOINTS ON SERVICE AUTOMATION

Figure 12.10: The notation for the data relating to service automation model

rules should this information comply with. This way, the deployment actors can be prepared to transform this organizational knowledge to their input to services, hence facilitating human participation.

The policy model can be constructed in terms of a table, listing the policies that are relevant to service automation. The template for such a table is shown in Table 12.5. This table aids the specification of policies in presenting all the information relevant to the policies in a structured manner. As such, this table also aids the preparation of the policies.

Table 12.5: The template for automation-related policy table

<table>
<thead>
<tr>
<th>Policy ID</th>
<th>Policy name</th>
<th>Policy description</th>
<th>Associated service</th>
<th>Controlled by</th>
<th>Type of format</th>
</tr>
</thead>
</table>

Model correspondence rules Each policy in the automation-related policy model should be linked to a service, which should appear in each of the other views. Each policy in the automation-related data flow model should be linked to the policies specified in the automation-related policies model.

An example of data view A data view consists of two models, a data model and a policy model. An example of the former is presented in Figure 12.11 and the latter in Table 12.6. In the following, we shall explain how this data view addresses the concerns (see Table 12.4) framed by the Data Viewpoint.

The concern about human participation is mainly addressed by the data model. Corresponding to the degree view (presented in Figure 12.8), the data
model shows in detail that the two semi-automated services Select Unit and Map Units To Nodes require information about Identification of the desired service and Selection among the possible candidate units respectively during their execution. The bank deployer can see from the data model that he or she is required to provide these two pieces of information; and to do so he or she is guided by System Requirements and Unit distribution policy.

An example of the policy model is shown in Table 12.6, listing all the policies appearing in the data model given in Figure 12.11. This model mainly addresses the concern about the specification of policies. For instance, since System Requirements and Unit distribution policy are used by a human actor, they are constructed in textual documents. Whereas Unit selection policy is directly accessed by the policy-driven automated service Resolve Unit, thereby it is created under machine readable formats.

The concern about compliance with organizational regulations is addressed by both the data model and policy model. In the data model, each required external input is linked to a policy implying that such input should conform to the policy. In particular, when the external inputs are given by human actors (like identification of the desired service provided by the banking deployer as shown in Figure 12.11), it is of critical important to the manager to have certain control over these inputs. Since the links between external inputs and policies become explicit in the data model, the manager of BankFutura understands better which policies will be used for providing the required inputs. Complementarily, the policy model (presented in Table 12.6) shows the specification of the policies in detail. In particular, it shows the stakeholders who are responsible for maintain-
CHAPTER 12. 3D ARCHITECTURE VIEWPOINTS ON SERVICE AUTOMATION

Table 12.6: An example of policy model

<table>
<thead>
<tr>
<th>Policy ID</th>
<th>Policy name</th>
<th>Policy Description</th>
<th>Associated service</th>
<th>Controlled by</th>
<th>Type of format</th>
</tr>
</thead>
<tbody>
<tr>
<td>PB01</td>
<td>System requirements</td>
<td>Describes the business processes that must be supported by the environment</td>
<td>Select Unit Managers</td>
<td>Textual Document</td>
<td></td>
</tr>
<tr>
<td>PB02</td>
<td>Unit selection policy: Latest version</td>
<td>Selects the unit with the most recent version among the potential candidates</td>
<td>Resolve Unit SOA Architects</td>
<td>Formal/SQL Sorting Query</td>
<td></td>
</tr>
<tr>
<td>PB03</td>
<td>Unit distribution policy</td>
<td>Defines the rationale behind the environment definition and the quality levels to be sustained by the deployed services</td>
<td>Map Units To Nodes SOA Architect</td>
<td>Textual Environment design / SLA Document</td>
<td></td>
</tr>
</tbody>
</table>

In summary, the degree view can be regarded as the core of the three service automation views, which can serve as a “central view”. The basic elements of the degree view, a set of services with different degree of automation (ellipses marked with different colors), are also part of the other two service automation views. Thereby, we can see the decision view and data view as two views with different levels of abstraction. More specifically, the decision view is a refinement of the degree view by adding rationale for the decision on the degree of automation; and the data view is also a refinement of the degree view in that it adds information about policies and human inputs that assist the execution of services. As such, the three views have vertical relations, meaning they relate at different levels of abstraction.

12.6 Discussion

12.6.1 The visual support of the 3D service automation viewpoints

In (Harel, 2009), David Harel reflects on his early work on Statecharts, and observed: “One of the most interesting aspects of this story is the fact that the work was not done in an academic tower, inventing something and trying to push it down the throats of real-world engineers. It was done by going into the lions den,
12.6. DISCUSSION

working with the people in industry. […] If what you come up with does not jibe with how they think, they will not use it. Its that simple.” Our approach is similar. We worked together with architects to identify their visualization requirements for architectural views. Architects use views for two main purposes (Kruchten, 2008), reasoning and communication with other stakeholders. In both cases, one fundamental factor for the usefulness of a certain viewpoint is in providing the right visual notation to present the needed information.

Visualization is a common technique to convey abstract information in intuitive ways (Chen, 2004). Representing information in terms of (a set of) graphics often more easily draws readers’ attention and improves understandability, as compared to pieces of text (Card et al., 1999). From our experience, the range of stakeholders involved in services engineering includes a broader range of expertise’s than in the development of traditional applications, which further reinforces the need to achieve an effective communication between the stakeholders through the usage of visualization techniques.

Software architecture practitioners have developed several well-accepted diagram notations for documenting the systems, such as the UML (Unified Modeling Language) (Evans et al., 2004) or the BPMN (Business Process Modeling Notation) (Ouyang et al., 2006). Although they do not capture exactly the information involved in the automation concerns, they do provide a way of representing concepts such as service execution flow, or representing the actors which interact with the system. However, these existing diagrams can be difficult to understand, limiting their usefulness for reasoning with non-technical stakeholders (such as end users or business managers), and for effective communication. Therefore, these well-accepted notations can constitute the base of the proposed viewpoints, but should be adapted to intuitively represent all the elements surrounding these concerns and their relationships, using a notation easily understandable for non-technical readers.

Visual support of viewpoints has been identified in § 12.3.2 as a fundamental characteristic for their usefulness. In this section we describe how we have consciously designed the graphical notation of the 3D VP’s to make the corresponding views intuitively understandable and to hold readers attention steady. To this aim, we followed some well-known design principles in information- and software visualization (Card et al., 1999; Chen, 2004).

First of all, we used color tones in the graphical notation, with different shades of color blue, for representing the different degrees of service automation. The motivation behind this scheme is that according to human perception, dark colored objects often feel heavier in weight (tending to sink), whereas light colored objects feel lighter (tending to float). This way, the color tone renders graphically the degree of automation of each service: the darker the color, the higher the degree of automation. This representation resembles an iceberg immersed in
CHAPTER 12. 3D ARCHITECTURE VIEWPOINTS ON SERVICE AUTOMATION

the sea, with the human actors looking at the surface: only the top (white is the lightest color) is visible (i.e. human actor gets access to the service and manually participates in its execution), while the deeper the iceberg is sunk in water, the lesser accessible it becomes (i.e. increasingly automated).

The graphical notation for policies inherits the same color scheme. As shown in Figure 12.10, policies guiding human actors are light colored; whereas policies assisting the execution of policy-driven automated services are dark colored. In this way, just from the color of the policies the reader can perceive the correspondence between the degree of service automation and policies. These visualization techniques should enable the views becoming self-explaining.

Second, in addition to color selection, we also carefully chose the shapes of the graphical notation. In service automation views, services are the core objects whereas other elements such as policies or human actors are auxiliary information. Moreover, services most likely appear in other views than 3D service automation views as well. For these reasons, we see the need to graphically differentiate services from the other elements. We used ellipses to denote services (as in SoaML (OMG, 2009)), and boxes to represent other elements.

Third, to help the reader focus on the main message that a 3D view intends to convey, we positioned the elements supporting such main message in the center of the view. For instance, when looking at a decision view like Figure 12.5, the reader’s attention is naturally brought to the layer of rationale that justifies service automation decisions. In the data view (see Fig 12.11), since the data required during the execution of services is positioned in the middle, the reader naturally pays attention to data-related issues, such as which service requires such data and where the service can obtain this data, helping the main purpose of this view.

Finally, to make the conforming views easy to understand and self-explaining, we reused known notations and concepts from UML and BPMN to represent common concepts in the architecture description. For instance, we used a person symbol to represent a human actor, and a document symbol to represent a policy. Relationships also use a familiar notation: for instance, a line with arrow denotes the invocation sequence between to services, and a line connecting a human actor and a service denotes the interaction between them.

In summary, the graphical notation defined in the 3D VP’s has been carefully designed in order to maximize the expressiveness of the conforming views. The notation’s ease of understanding simplifies both the creation and interpretation of conforming views. As such, the 3D VP’s facilitate the communication between stakeholders, especially the ones lacking technical expertise.
12.6.2 The applicability of the 3D VP’s

To validate our goal of supporting reusability and applicability of the 3D VP’s across different domains, we carried out a fourth study in a totally different context, namely an education environment. In a SOA design Master course given in our university, 11 groups of 3 Master students each were asked to design a SOA e-Health system (its functional and non-functional requirements are described in (Di Nitto et al., 2009)) using the 3D VP’s in their architecture descriptions 1. As the e-Health system consisted of many business processes, the students were allowed to select one process to illustrate its automation aspect. Some examples of the selected process are diagnosis, medical device reservation, and pharmacy management.

During this course, we asked the students to first identify concerns related to service automation in the e-Health system, and then use the 3D VP’s to illustrate these concerns. In this way, we could possibly obtain some concerns that we might have overlooked in our case studies but should be framed because of their relevance. The results show that the concerns identified by the students were either specific to E-health system or domain-independent. In both cases, we could successfully map them to the set of concerns already framed in 3D VP’s in a straightforward way.

In summary, in spite of the clearly different application domain of the e-Health system (as compared to the SDCA), the students could directly apply the 3D VP’s to illustrate the service automation aspects in their design. Moreover, the students provided very positive feedback on the use of the 3D VP’s and their expressiveness. Although not conclusive, these results provide us with further positive indication of the applicability of the 3D VP’s to SBAs in general.

12.7 Conclusions

SOA has been widely adopted in industry. One of the reasons for this adoption is its ability to dynamically respond to business changes. To maximize this ability, SBAs should have a certain degree of automation and minimize the intervention of human actors. As SBA consist of services that are distributed and owned by different business parties, the lack of control over services poses additional concerns in service-oriented design. In this work, we identify service automation as a service aspect that is specifically relevant to SBAs and thereby should be explicitly documented in architecture description.

The 3D viewpoints contribute to address the top three SOSE challenges reported in Chapter 2: quality, service and data. The decision view addressed quality-based decision making, as it illustrates the trade-off between the degree

1All the designs are available at www.few.vu.nl/~qgu/eHealthDesign
CHAPTER 12. 3D ARCHITECTURE VIEWPOINTS ON SERVICE AUTOMATION

of automation services can achieve and other quality requirements such as reliability, security, agility, and flexibility. Second, the 3D viewpoints focus on service autonomy, therefore contributing to also contribute to the support of one property of “good” services. Finally, the data view frames some data provenance concerns of SBAs. More specifically, it illustrates the creation, processing, and management of the data relevant to service automation (including policy guidelines and programmatic policies).

The study of the SDCA and its two industrial applications motivated us to develop viewpoints to address service automation concerns. This study also provided us with a source from where we could elicit concerns, model corresponding designs and verify the correctness and expressiveness of the models. Using the action research approach, we developed the 3D VP’s presented here, framing the concerns relating to automating the execution of services for coordinately accomplishing a specific (business or engineering) task.
When SOSE became a hot research topic and when many organizations started to adopt SOA as their architectural style, researchers and practitioners often believed that engineering SBAs would require (completely) different methods and techniques as compared to TSE. Accordingly, new design principles and guidelines that were considered being service-specific have been proposed, and many vendors started to sell SOA as a new product.

This thesis investigates the novelty of SOSE by identifying its differences with TSE. The results of the study show that, instead of establishing a completely new research field, SOSE differs from TSE only in seven well-scoped distinctions. These distinctions require service engineers and architects to change their mindset when engineering and architecting SBAs or service inventories.

Focusing on these seven - called fundamental - differences, this thesis identified a set of service aspects that are specifically relevant to the SOSE development process and SOA decision making. Taking some service aspects as examples, Chapter 3 discusses the approach of considering the fundamental differences as engineering or architecting concerns and addressing these concerns using a ‘view approach’ (i.e. incorporated these differences and service aspects into a number of ‘views’). By visualizing the information that is of specific importance to SOSE by means of models or tables, these views attract specific attention from service engineers and architects. As such, these views can be used as a ‘tool’ to guide the mindset of service engineers and architects.

An overview of the views proposed in this thesis is given in Figure 13.1. The central part of the figure shows the relation between the SOSE characteristics (as compared to TSE, cf. differences between SOSE and TSE) and the service aspects. The views are then associated with either characteristics or aspects, depending on what they illustrate.

This chapter discusses the answers to the research questions listed in Chapter 1 using the proposed views.
CHAPTER 13. CONCLUSIONS

SOA is designed under open-world assumptions. Services are cross-organizational and are designed with multiple sets of (non-)functional requirements. Additional development roles are involved in development. Services are consumed and executed remotely. Different architecture types are used. Services are open and consumed remotely. Different perspectives of stakeholders are involved. The need for increased effort at run-time is addressed. Different stakeholders are involved in service life cycle. The relevance of cross-organizational collaboration is highlighted. The increased importance of the identification of stakeholders is emphasized. Different perspectives of stakeholders are considered.

Services are the building blocks. Services are consumed and executed remotely. SOA is designed under open-world assumptions. Services are open and consumed remotely. Different architecture types are used. Services are cross-organizational. The need for increased effort at run-time is addressed. Different stakeholders are involved in service life cycle. The relevance of cross-organizational collaboration is highlighted. The increased importance of the identification of stakeholders is emphasized. Different perspectives of stakeholders are considered.

Evaluation framework for SOSE methodologies (Ch 7). A classification of SOSE challenges by their types and topics (Ch 2). Design decision template (Ch 9). An input-output matrix for the selection of SIMs (Ch 8). A taxonomy of stakeholder (Ch 5).

Figure 13.1: An overview of the views created in this thesis
13.1 Contributions

13.1.1 What are the challenges in SOSE? (RQ 1)

Chapter 2 presents a systematic literature review exploring SOSE challenges, which were introduced by the seven differences between SOSE and TSE (discussed in Chapter 2). As a result, over 400 SOSE challenges have been identified and classified along two dimensions: a) based on themes (or topics) that they cover and b) based on characteristics (or types) that they reveal. The classification is presented in terms of two views: an overview of the number of SOSE challenges of each topic (per year, given in Figure 2.5), and an overview of the number of SOSE challenges of each topic and each type (in Figure 2.6). These two views provide us (and other SOSE researchers) a means to analyze the distribution of SOSE challenges of different topics and types across different years and hence discover SOSE trends as well as gaps.

From these two views, there are a number of interesting findings: SOSE emergence laws point out the relationship between expectations, research work on challenges and solutions; unexpectedly, very few challenges directly address service discovery and data-related SOSE challenges require more attention in the research community. The findings of this review further provide empirical evidence for establishing future SOSE research agendas.

Moreover, the context in which the SOSE challenges were proposed has been studied and seven fundamental differences between SOSE and TSE (that are often the reasons why the SOSE challenges become issues requiring further research) have been identified. These seven differences were taken as engineering and architecting concerns.

13.1.2 How is the development process of SBAs different from that of traditional software systems? (RQ 2)

Our answer to this research question is provided by the answers to the three sub-questions (RQ 2.1 - 2.3) in the following three sub-sections.

What are the service aspects and how to make them explicit in SOSE process models? (RQ 2.1)

Chapter 4 proposes a stakeholder-driven service life cycle model (see Figure 4.1). In this model, three service aspects are explicitly highlighted, namely: increased importance of the identification of stakeholders, the relevance of cross-organizational collaboration, and the need for increased effort at run-/change time. To make the service aspects explicit in SOSE process models, each identified stakeholder was assigned with a swimlane and the SOSE activities were placed in the
CHAPTER 13. CONCLUSIONS

corresponding swimlanes to show the identification of stakeholders; the message flows and sequence flows were used within and across each swimlane to show the cross-organizational collaboration; and design time activities were placed apart from the run-/change time activities to show the increased effort at run-/change time. By highlighting these three service aspects in a SOSE process model, Chapter 6 presents a case study (in which the development process of the SeCSE methodology has been modeled) to show how each aspect provides guidance for engineering SBAs in practice.

Which stakeholders may be involved in the development process of SBAs? (RQ 2.2)

Chapter 5 presents a taxonomy of SOSE stakeholders (see Table 5.2) solicited from nine institutions within the EC’s Network of Excellence in Software Services and Systems (S-Cube). The taxonomy of SOSE stakeholders contains nineteen stakeholder types played in five roles, participating in the different phases of the S-Cube service life cycle model. This view shows what the additional development roles are required in SOSE and the perspectives they take.

How to select a SOSE method? (RQ 2.3)

To answer this research question, two views have been created to guide the selection of SOSE methodologies and service identification method, respectively. Chapter 7 presents an evaluation framework (see Table 7.1 and Table 7.2) for assessing and comparing SOSE methodologies. Taking the seven differences as the service-specific features required in SOSE methodologies, a set of criteria has been derived to assess whether the SOSE methodologies do support addressing each of the differences. In principle, an “ideal” SOSE methodology would cover all features enlisted in the evaluation framework. However, in practice, enterprises may act in a more opportunistic way by selecting a SOSE methodology that would offer the right set of features specifically required by the SOA project at hand. In other words, an enterprise could be perfectly satisfied with a SOSE methodology fulfilling just a subset of the criteria included in our framework, depending on its project- or domain-specific needs.

Since services are the building block of SBAs, the identification of services becomes extremely important in service design. Chapter 8 presents an input-output matrix for the selection of service identification methods (SIMs) (see Table 8.7). Using this matrix, one can select a SIM by the targeted output (e.g., business service or IT service), the available resources (e.g., legacy systems or business process), or by comparing alternative SIMs producing similar targeted output and using similar resources but taking different strategies or techniques.
13.1. CONTRIBUTIONS

13.1.3 How is service-oriented architectural knowledge different from traditional architectural knowledge? (RQ 3)

Our answer to this research question is provided by the answers to the two sub-questions (RQ 3.1 and 3.2) in the following two sub-sections.

Are existing decision models sufficient to document SOA design decisions? (RQ 3.1)

To answer this research question, SOA design decisions in general and SOA process decision in the specific have been analyzed. Accordingly, three views have been proposed to support the modeling and codification of SOA design decisions.

Generally speaking, existing decision models could be used to document SOA design decisions. Chapter 9 proposes a template (based on the existing decision models) for documenting SOA design decisions (see Table 12.1) and discusses the benefits of using it to support structured reasoning, improve traceability, and facilitate focus on service-oriented design. By enforcing especially the documentation of the context of design issues, the template encourages the designer to think both in a service-oriented way, and motivate what are the SOA-specific factors (e.g. the seven differences) that cause such design issues.

However, existing decision models are not sufficient to specifically support SOA design. Additional knowledge entities (e.g., architecture types, perspectives of stakeholders) need to be extended to the existing decision models to fully support SOA design. Chapter 10 takes one step forward and focuses specifically on SOA design. Four service aspects (different architecture types, temporary provision-consumption relationship, dealing with heterogeneity, and different perspectives of stakeholders) have been identified. From these service aspects, eight knowledge entities are considered specifically relevant to codify SOA design decision. Two examples of using the proposed knowledge entities to document SOA design decisions are showed in Figure 10.1 and Figure 10.2 and a number of foreseen benefits are discussed.

Further, Chapter 11 focuses on a specific type of SOA design decisions - process decisions. Two service aspects (temporary provision-consumption relationship and the relevance of cross-organizational collaboration) are discussed, from which the gaps in the existing architectural knowledge models have been identified. Accordingly, this chapter proposes an extended core model (see Figure 11.1) for SOA process decisions. Using this model, architects are encouraged to document information specifically relevant to SOA process decisions (e.g., the executer of the decision and stakeholders who are related to this decision). This information can be used complementary to the entities (e.g., concerns, ranking criteria) that are often used for reasoning about design decisions.
CHAPTER 13. CONCLUSIONS

How to use architectural views to assist the documentation of service-oriented architectures? (RQ 3.2)

Generally speaking, architectural views can be used to facilitate the communication between stakeholders, especially the ones lacking technical expertise. It is of great importance that stakeholders and their concerns that an architectural view aims to address are carefully identified. Furthermore, the graphical notation defined in the architectural views should be carefully designed in order to maximize the expressiveness of the conforming views.

In SOSE, with the increasing number of stakeholders (as discussed in Chapter 5) and specific concerns arisen in the design of SBAs (as discussed in Chapter 10), architectural views become extremely useful to assist the documentation of service-oriented architecture. This thesis has chosen the degree of automation of SBAs as the aspect under study. As services are the building blocks of SBAs, the degree of automation of SBAs is directly dependent on that of its composing services. Not only SOA architects but also managers and human actors are concerned with service automation. Further, their concerns vary from design alternatives to organizational regulations. Chapter 12 shows how to frame automation-related concerns using a set of service automation viewpoints (called 3D VP’s) (see Table 12.2, 12.3, and 12.4) and how to use these 3D VP’s to create corresponding views to assist the reasoning about architecture-level decisions, options, or risks, and communication among stakeholders.

13.2 Summary

As discussed in the previous section, some of the views provide straightforward guidance to SOSE: e.g., the taxonomy of SOSE stakeholders and the matrix for selecting SIMs. These views not only suggest a way of structuring the information useful for e.g., classifying stakeholders and categorizing SIMs, but also contain actual data that are generally valid in SOSE, such as the types of stakeholders and the comparison of existing SIMs.

Some other views, however, do not contain actual data that can be directly used for guidance purposes. Rather, these views suggest a way of modeling or structuring information such that certain service aspects or fundamental differences can be highlighted. For demonstration purposes, some examples of using these views in specific context have been provided: a SOSE process model for the SeCSE methodology using the stakeholder-driven service life cycle model, an example of SOA design decision using the proposed template, and service automation views created in industrial case studies complying with the 3D VP’s. These views can be used as a tool in any SOA project when necessary.

Put together, all the proposed views are helpful to guide the mindset of service
13.3. Future research

In this thesis, a set of views has been identified to illustrate aspects that are specific to SOSE with the aim of supporting service engineers and architects. One of the most important benefits of a view-based approach is that abstract and/or unstructured information is visually illustrated and thereby becomes easy to be understood and communicated.

Due to the service aspects (the seven differences) that SOSE introduces, service-oriented software becomes more complex and challenging. Therefore when adopting SOA many enterprises face even more difficulties in order to reach a uniform understanding of their problem- and solution space. A view-based approach helps in dealing with these difficulties.

In addition to the views we have created in this thesis, such an approach further helps facilitating two important needs characterizing the evolution of modern economics: cross-border mergers & acquisitions and new business channels (Singh & Waddell, 2004):

- Supporting the needs of integration of cross-border mergers & acquisitions

   In modern economy it is increasingly common for enterprises to expand their business internationally or globally. As a result, cross-border merger & acquisition play an increasingly important role in industrial globalization (Bertrand & Zitouna, 2006). To cope with such evolutions, enterprises have to face the challenge of reorganization and harmonization.

   Much research performed in the SOSE community focuses on providing technical solutions. Such focus is evident in many major SOSE conferences, such as ICSOC, SCC, ServiceWave and journals, such as SOCA, and IEEE Software. In managing the re-alignment of the information systems of merger or acquisition partners, enterprises need guidance on how to use IT to support their business needs in a service-oriented manner, rather than sophisticated technical solutions.

   Suppose that two enterprises are merged and hence need to share their customers, they would need to migrate their pre-existing systems to common services. To optimize the identification of services (ensuring the consistency between two enterprises, maximizing the reuse of services, and minimizing duplicated business functions), a SOA design should be made and agreed by both of the enterprises. However, due to the differences in cultures,
regulations, and policies, the two enterprises typically face difficulties to understand the design and reach an agreement.

Currently, the guidance on the governance of integrating business functions of multiple enterprises and delivering them as services is still lacking. One way to solve this problem is to use architectural views. For instance, for the specific purpose of addressing automation-related concerns, a set of service automation views (in Chapter 12) can be used. Since (a) the concerns are reflected in the viewpoints, (b) the elements of the views are explained in terms of meta-models, and (c) the graphic notations are intuitive to the stakeholders, this set of views can be used by multiple enterprises to communicate the design in terms of service automation. The same approach can be used for the purpose of integrating business functions and innovating enterprise models, where enterprises are often concerned with how to merge business processes, how to fulfill the business needs of both partners, and how to ensure the newly developed services are supported by the information systems of both partners. In other words, a view-based approach can be a powerful means to address emerging concerns, like the ones mentioned above, that actually relate to two aspects where service orientation should help: facilitate communication between different expertise’s and disciplines (e.g., IT and Business Economics) and ease integration of systems of systems (Maier, 1998).

- Supporting the needs of opening new business channels

Another way for enterprises to gain competitive advantage is to expand their existing business to new business channels (Kamogawa & Okada, 2004). Taking banking domain as an example: a bank that traditionally offers its services through physical offices may plan to develop a new business channel through web applications or mobile devices. Due to the characteristics of the new business channel (e.g., mobility, digitalization, flexibility), the bank may significantly improve its business efficiency and scalability. Since the basic business functions offered to the customers are still similar (e.g., saving, withdrawing, paying check), the bank may reuse many of its existing services through the new business channels.

So far little guidance has been given to enterprises on how to identify the services to be reused and to identify opportunities for new business channels. Chapter 8 proposes a matrix for enterprises to select a service identification method, as a starting point to guide the identification of reusable services. Taking a view-based approach, views illustrating e.g., business workflows may be used to identify similar sub-workflows required by different business channels. In this way, one may identify the overlapping functions and offer these functions through different business channels. In the same vein, one
may also identify the different requirements of business channels and assess the feasibility of implementing certain business functions in terms of services, or identify opportunities for new business channels based on existing services in strategic planning.
Appendices
The judgment scale to assess the methodology support for a feature

Table A.1: The judgment scale to assess the methodology support for a feature

<table>
<thead>
<tr>
<th>Generic Scale Point</th>
<th>Definition Scale Point</th>
<th>Scale Point Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>No support</td>
<td>Fails to recognize. The feature is not addressed in the methodology.</td>
<td>-</td>
</tr>
<tr>
<td>Implicitly support</td>
<td>The feature is addressed implicitly. It can be recognized in terms of interpretation.</td>
<td>+</td>
</tr>
<tr>
<td>Explicitly support</td>
<td>The feature is explicitly addressed in the methodology, but no detailed information is given.</td>
<td>++</td>
</tr>
<tr>
<td>Strong support</td>
<td>The feature is explicitly addressed in the methodology with detailed information or guidance.</td>
<td>+++</td>
</tr>
</tbody>
</table>
The service lifecycle model
## APPENDIX B. THE SERVICE LIFECYCLE MODEL

Table B.1: The phases of the service lifecycle model and their description (Based on the life cycle model in Chapter 4)

<table>
<thead>
<tr>
<th>Role</th>
<th>Life cycle phases</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service provider</td>
<td>Market scan</td>
<td>Investigate the market demands and orient the service production.</td>
</tr>
<tr>
<td></td>
<td>Requirements engineering</td>
<td>Analyze the objective, functionality, interface and quality of services, as well as consider the requirements of their clients to access the services.</td>
</tr>
<tr>
<td></td>
<td>Business modeling</td>
<td>Model business processes without going into the technical details.</td>
</tr>
<tr>
<td></td>
<td>Service design</td>
<td>Make a specific design of the services which complies with all the functional and non-functional requirements that are concluded from previous phases. Implement new services or modify the existing services to realize the service design.</td>
</tr>
<tr>
<td></td>
<td>Service development</td>
<td>Fault detection and quality control.</td>
</tr>
<tr>
<td></td>
<td>Service testing</td>
<td>Make produced services available by publishing them to a service registry so that the service consumer can find and use them when needed.</td>
</tr>
<tr>
<td></td>
<td>Service publishing</td>
<td>Ensure services are services are accessible only by authorized users by means of e.g. service negotiation.</td>
</tr>
<tr>
<td></td>
<td>Service monitoring</td>
<td>Keep track of the behavior of the published services.</td>
</tr>
<tr>
<td></td>
<td>Service management</td>
<td>To ensure that any change occurs in SOA environment is handled so that the consumers of the services can continue to use the services.</td>
</tr>
<tr>
<td>Service broker</td>
<td>Registry selection</td>
<td>Select a proper service registry tool to manage the service registry.</td>
</tr>
<tr>
<td></td>
<td>Registry update</td>
<td>Update the registry when new services are published or existing services are changed.</td>
</tr>
<tr>
<td></td>
<td>Registry maintenance</td>
<td>Maintain the consistency, efficiency of the service registry to assist service discovery.</td>
</tr>
<tr>
<td>Service consumer</td>
<td>Service discovery</td>
<td>Look up published service candidates that comply with the requirements.</td>
</tr>
<tr>
<td></td>
<td>Service composition</td>
<td>Ensure that services in an application can be well assembled to work together to fulfill business requirements.</td>
</tr>
<tr>
<td></td>
<td>Service negotiation</td>
<td>Exchange a number of contract messages with a service provider in order to reach an agreement to access the services.</td>
</tr>
<tr>
<td></td>
<td>Service invocation</td>
<td>Each of the services that are discovered and composed in the earlier activities has to be invoked for execution.</td>
</tr>
<tr>
<td></td>
<td>Service monitoring</td>
<td>Keep track of the behavior of the invoked services.</td>
</tr>
<tr>
<td>Application builder</td>
<td>Requirements engineering</td>
<td>Analyze the objective and functionality of SBAs, collect requirements and needs from the end users. Similar to the ones in traditional software engineering.</td>
</tr>
<tr>
<td></td>
<td>Application design, application</td>
<td>Ensure that the integrated services perform as expected.</td>
</tr>
<tr>
<td></td>
<td>implementation and module testing</td>
<td>Take care of the changes that might occur from the end user requirements or the behavior of the composed services.</td>
</tr>
<tr>
<td></td>
<td>Application testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Application maintenance</td>
<td></td>
</tr>
</tbody>
</table>
Het begeleiden van service-georiënteerde software ontwikkeling - een aanpak vanuit meerdere perspectieven

De trend in de ontwikkeling van software is verschoven van het ontwikkelen van software systemen naar de ontwikkeling van service-georiënteerde systemen die zijn samengesteld uit kant-en-klare services. Service-Oriented Architecture (SOA) wordt als architectuurstijl breed geïmplementeerd in de industrie dankzij de mogelijkheid een naadloze integratie tussen software services te leveren. Indien de services een laag niveau van koppeling kennen, goed zijn gespecificeerd, en met elkaar samenwerken, kan de uitvoering van een SOA veel voordeel opleveren voor een onderneming, zoals een hogere ROI, lagere IT-kosten en meer organisatorische flexibiliteit.

Aangezien de ontwikkeling van Service-Based Applications (SBA) de identificatie van services, het ontdekken van services, en het samenstellen van services vereist, is een aanvulling op traditionele software engineering (TSE) activiteiten (zoals het programmeren, testen en uitrollen) noodzakelijk. Een systematische aanpak voor de ontwikkeling van service-georiënteerde software-systemen (SOSE) is een noodzaak geworden.

In SOSE moeten traditionele TSE principes worden afgestemd op de service-georiënteerde ontwikkeling. Vooral het ontwikkelen van SBAs vereist een andere mindset. Anders dan bij TSE worden ontwerpbeslissingen in SOSE vaak uitgesteld tot het moment dat de service operationeel is (runtime), met als doel om met steeds veranderende eisen om te kunnen gaan. Veel verantwoordelijkheid (zoals onderhoud) verschuift van de afnemer (consumer) van een service naar de aanbieder (provider), simpelweg omdat de services fysiek uitgerold worden bij deze aanbieders. Daar staat tegenover dat andere zaken, zoals het vinden, samenstellen en monitoren van services typisch de verantwoordelijkheid wordt van de afnemers. Binnen organisaties komen business en IT hierdoor dichter bij elkaar te staan. SOA is daarom vaak businessgedreven en vereist een organisatiebrede implementatie.

Dit proefschrift beschrijft een aanpak vanuit meerdere perspectieven voor het ondersteunen van service engineers en architecten bij het beheer van het service-ontwikkelingsproces en de service-georiënteerde architectuurkennis. De bijdrage is specifiek voor SOSE in die zin dat het zich richt op onderwerpen die minder relevant zijn in TSE, maar van groot belang in SOSE. In dit proefschrift worden de trends binnen SOSE onderzocht door het identificeren van de verschillen met TSE en wordt een aantal aspecten onderzocht die specifiek zijn voor SOSE. Door het visualiseren van deze informatie in modellen of tabellen is een aanpak ontstaan die service engineers en architecten helpt bij service-georiënteerde
softwareontwikkeling door vanuit verschillende perspectieven naar voor SOSE relevante onderwerpen te kijken.

Vanuit het oogpunt van het service-georiënteerde ontwikkelingsproces zijn de volgende perspectieven gedefinieerd:

- vanuit een service levenscyclusmodel richting geven aan de modellering van het SOSE proces, met de nadruk op drie service aspecten: het toegenomen belang van de identificatie van belanghebbenden, de relevantie van organisatie-overkoepelende activiteiten, en de noodzaak van verhoogde inspanning op runtime;
- een taxonomie van SOSE belanghebbenden om te laten zien welke belanghebbenden betrokken kunnen zijn bij, en in welke fase van het ontwikkelingsproces van de SBA’s;
- een evaluatieraamwerk voor het richting geven aan de selectie van SOSE methodes, voornamelijk gebaseerd op hun mate van service-oriëntatie;
- een input-output matrix voor het richting geven aan de selectie van de dienstverleners met identificatiemethoden op basis van de beschikbare middelen van de ondernemingen en hun doelgroepdiensten.

Vanuit het oogpunt van service-georiënteerde architectuurkennis zijn de volgende perspectieven gedefinieerd:

- het documenteren van architectuurbeslissingen, hetgeen gestructureerd redeneren mogelijk maakt, verbetert de traceerbaarheid en faciliteert service-georiënteerd ontwerpen;
- gericht op het documenteren van SOA ontwerphbeslissingen door ondersteuning van het modelleren van kennisentiteiten benodigd in SOA-ontwerp; dit gezichtspunt heeft betrekking op vier aspecten van services: verschillende architectuurtypen, een tijdelijke voorzienings-verbruiksrelatie, omgaan met heterogeniteit, en de verschillende perspectieven van belanghebbenden;
- het documenteren van SOA procesbesluiten vanuit twee service-aspecten: tijdelijke voorzienings-verbruiksrelatie en de relevantie van cross-organisatorische samenwerking;
- mate van betrokkenheid van mensen bij het komen tot architectuurkeuzes met name de genomen besluiten, opties, of de risico’s en de communicatie tussen de belanghebbenden.

Tezamen kunnen alle perspectieven die in dit proefschrift worden voorgesteld gebruikt worden als een instrumentarium voor service engineers en architecten.
om richting te geven tijdens service-georiënteerde software ontwikkeling, door in te zoomen op onderwerpen die minder relevant zijn in TSE, maar vooral relevant zijn voor SOSE.
SIKS Dissertation Series

1998

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games

1998-5 E.W. Oskamp (RUL)
Computerondersteuning bij Straftoemeting

1999

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Nick J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000
2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR)
Sensitivity Analysis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

====

2001
=====

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgeni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)
Task-based User Interface Design
2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan ’t Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models,
Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and
simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VUA)
Knowledge Management: The Role of Mental Models in Business Systems Design

=====

2002
=====

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments
inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology;
Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-09 Willems-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems
2002-13 Hongjing Wu (TUE)
   A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
   Agent Interaction: Abstract Approaches to Modelling, Programming and
   Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
   Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
   The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UvA)
   Understanding, Modeling, and Improving Main-Memory Database Performance

====

2003-01 Heiner Stuckenschmidt (VU)
   Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
   Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
   Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
   Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UvA)
   Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
   Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
   Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
   Repair Based Scheduling

2003-09 Rens Kortmann (UM)
   The resolution of visually guided behaviour

2003-10 Andreas Lineke (UvT)
   Electronic Business Negotiation: Some experimental studies on the interaction
   between medium, innovation context and culture

2003-11 Simon Keizer (UT)
   Reasoning under Uncertainty in Natural Language Dialogue using
   Bayesian Networks

2003-12 Roeland Ordelman (UT)
   Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

====

2004

====

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politie gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents
2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

====

2005
====

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM)
AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Sprock (UM)
Adaptive Game AI

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 J een Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments
2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law; State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

===

2006
===

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor’s Opinion

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)
Interactivation - Towards an e-ology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change

2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)
Aptness on the Web

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval
2006-27 Stefano Bocconi (CWI)
   Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UVA)
   Focused Information Access using XML Element Retrieval

2007

2007-01 Kees Leune (UvT)
   Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)
   Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
   Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
   Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)
   Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UVA)
   Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic’ (UT)
   To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU)
   Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU)
   Agent-Based Mediated Service Negotiation

2007-10 Huib Aldewereld (UU)
   Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)
   Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN)
   Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT)
   Meetings in Smart Environments; Implications of Progressing Technology

2007-14 Nick Bergboer (UM)
   Context-Based Image Analysis

2007-15 Joyca Lacroix (UM)
   NIM: a Situated Computational Memory Model

2007-16 Davide Grossi (UU)
   Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for
Multi-agent Systems

2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice

2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations

2007-19 David Levy (UM)
Intimate relationships with artificial partners

2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network

2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005

2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns

2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramirez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

== ==

2008

== ==

2008-01 Katalin Boer-Sorbn (EUR)
Agent-Based Simulation of Financial Markets: A modular, continuous-time approach

2008-02 Alexei Sharpaniskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Kreijzer (UT)
Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference
2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective

2008-10 Wauter Bosma (UT)
Discourse oriented summarization

2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach

2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation

2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks

2008-14 Arthur van Bunningen (UT)
Context-Aware Querying: Better Answers with Less Effort

2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the
Markov Decision Process Framework in First-Order Domains.

2008-16 Henriette van Vugt (VU)
Embodied agents from a user’s perspective

2008-17 Martin Op ‘t Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises

2008-18 Guido de Croon (UM)
Adaptive Active Vision

2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused
Text Search

2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van
elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven

2008-21 Krisztian Balog (UVA)
People Search in the Enterprise

2008-22 Henk Koning (UU)
Communication of IT-Architecture

2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia

2008-24 Zharko Alekovski (VU)
Using background knowledge in ontology matching

2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using
Spender-signed Currency

2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled

2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design
2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks

2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans

2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content

2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues

2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

====

2009
====

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
operating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controlling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
"RAM: Array Database Management through Relational Mapping"

2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models
2009-29 Stanislav Pokraev (UT)  
Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)  
Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)  
A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)  
Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)  
How Does Real Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)  
Advancing in Software Product Management: An Incremental Method Engineering Approach

2009-35 Wouter Koolewij (UL)  
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN)  
Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN)  
Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU)  
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)  
Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT)  
Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT)  
Digital Analysis of Paintings

2009-42Toine Bogers  
Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT)  
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients

2009-44 Roberto Santana Tapia (UT)  
Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU)  
Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA)  
Querying XML: Benchmarks and Recursion

====

2010
2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink (UT)
Work flows in Life Science

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents

2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children

2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications

2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA)
People’s Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative

2010-21 Harold van Heerde (UT)
  Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI)
  End-user Support for Access to Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU)
  The Logical Structure of Emotions

2010-24 Dmytro Tykhonov
  Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU)
  Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI)
  XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines

2010-27 Marten Voulon (UL)
  Automatisch contracteren

2010-28 Arne Koopman (UU)
  Characteristic Relational Patterns

2010-29 Stratos Idreos (CWI)
  Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT)
  Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UvA)
  Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT)
  An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT)
  Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT)
  Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT)
  Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU)
  Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE)
  Correctness of services and their composition

2010-38 Dirk Fahland (TUE)
  From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU)
  Integrative modeling of emotions in virtual agents
2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions

2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives

2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access

2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier(UU)
Work flows in Life Science

2011-03
Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems
2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning: Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT)
Cloud Content Contention

2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity

2011-17 Jiyn He (UvA)
Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU)
The Mind’s Eye on Personal Profiles: How to inform trustworthiness assessments in virtual project teams
References


Ali Babar, Muhammad, & Lago, Patricia. 2009. Design decisions and design rationale in software architecture. *Journal of systems and software*, 82(8), 1195–1197. (Cited on pages 4 and 215.)


Chang, Soo Ho. 2007. A systematic analysis and design approach to develop adaptable services in service oriented computing. *IEEE congress on services, 375–378.* (Cited on pages 106 and 115.)


Di Penta, Massimiliano, Bastida, Leire, Sillitti, Alberto, Baresi, Luciano, Ripa, Gianluca, Melideo, Matteo, Tilly, Marcel, Spanoudakis, George, Maiden, Neil, Cruz, Jesus Gorronogoitia, & Hutchinson, John. 2009. SeCSE - service
centric system engineering: an overview. At your service: Service engineering in the information society technologies program. (Cited on page 95.)


Djavaneshir, G. Reza, & Agresti, William W. 2007. IT consulting: Communication skills are key. IT professional, 9(1), 46–46. (Cited on page 192.)


Eeles, Peter, & Cripps, Peter. 2009. The process of software architecting. Addison-Wesley Professional. (Cited on page 209.)


Erl, Thomas. 2008. SOA design patterns. The Prentice Hall. (Cited on pages 176, 177, 178, and 186.)


Holz, Hilary J., Applin, Anne, Haberman, Bruria, Joyce, Donald, Purchase, Helen, & Reed, Catherine. 2006. Research methods in computing: what are they, and how should we teach them? Sigcse bull., 38(4), 96–114. (Cited on page 11.)


Hubbers, Jan-willem, Ligthart, Art, & Terlouw, Linda. 2007. Ten ways to identify services. The SOA Magazine. (Cited on page 150.)


Inaganti, Srikanth, & Behara, Gopala Krishna. 2007. Service identification: BPM and SOA handshake. in BPTrends. (Cited on page 146.)


Kohlborn, Thomas, Korthaus, Axel, Chan, Taizan, & Rosemann, Michael. 2009. Identification and analysis of business and software services - a consolidated approach. IEEE transactions on services computing, 2(1), 50–64. (Cited on pages 142, 146, and 147.)


Lane, Stephen, Gu, Qing, Lago, Patricia, & Richardson, Ita. 2010. Adaptation of service-based applications: A maintenance process? Submitted for publication. (Cited on pages 87 and 88.)


Olsson (Editor), C.M. 2006. The software and services challenge. *In: Contribution to the preparation of the technology pillar on “software, grids, security and dependability” of the ec’s 7th framework programme*. (Cited on page 95.)

OMG. 2009. *Service oriented architecture modeling language (SoaML) - specification for the UML profile and metamodel for services (UPMS)*. (accessed 16.11.10). (Cited on page 230.)


Pulier, Eric, & Taylor, Hugh. 2006. *Understanding enterprise SOA*. Manning. (Cited on pages 60 and 177.)


Tang, Antony, Jin, Yan, & Han, Jun. 2007. A rationale-based architecture model for design traceability and reasoning. Journal of systems and software, 80(6), 918–934. (Cited on page 169.)


Tsai, Wei-Tek, Malek, Miroslaw, Chen, Yinong, & Bastani, Farokh. 2006. Perspectives on service-oriented computing and service-oriented system engineering. Pages 3–10 of: Proceedings of the 2nd IEEE international sym-
posium on service-oriented system engineering. IEEE Computer Society. (Cited on page 35.)


Tsai, Wei-Tek, Wei, Xiao, Paul, Ray, Chung, Jen-Yao, Huang, Qian, & Chen, Yinong, 2007c. Service-oriented system engineering (SOSE) and its applications to embedded system development. Service oriented computing and applications, 3–17. (Cited on pages 16, 18, 33, 35, 61, and 64.)


Van Der Ven, Jan S., Jansen, Anton, Nijhuis, Jos, & Bosch, Jan. 2006. Design decisions: The bridge between rationale and architecture. Rationale management in software engineering, 16. (Cited on page 4.)


international conference on software engineering and knowledge engineering. (Cited on page 146.)


Zimmermann, Olaf, Milinski, Sven, Craes, Michael, & Oellermann, Frank. 2004b. Second generation web services-oriented architecture in production in the finance industry. Pages 283–289 of: Companion to the 19th international conference on object-oriented programming systems, languages, and applications. ACM. (Cited on pages 158 and 186.)


