QUALITY-AWARE SERVICE-ORIENTED SOFTWARE
PRODUCT LINES: FEATURE-DRIVEN PROCESS
CONFIGURATION AND OPTIMIZATION

by

Bardia Mohabbati
M.Sc. (Computer Science), Amirkabir University of Technology, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in the
School of Interactive Arts and Technology
Faculty of Applied Sciences

© Bardia Mohabbati 2013
SIMON FRASER UNIVERSITY
Fall 2013

All rights reserved.
However, in accordance with the Copyright Act of Canada, this work may be reproduced without authorization under the conditions for “Fair Dealing.” Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.
APPROVAL

Name: Bardia Mohabbati
Degree: Doctor of Philosophy
Title of Thesis: Quality-aware Service-Oriented Software Product Lines: Feature-Driven Process Configuration and Optimization
Examinig Committee: Dr. Kate Hennessy
Assistant Professor, SFU
Chair

Dr. Marek Hatala, Senior Supervisor
Professor, Information Technology, SFU

Dr. Dragan Gašević, Co-Supervisor
Associate Professor, Information Technology, SFU

Dr. Hausi A. Müller, Supervisor
Professor, Computer Science, University of Victoria

Dr. Halil Erhan, Internal Examiner
Assistant Professor, Information Technology, SFU

Dr. Markus Stumptner, External Examiner
Professor, Information Technology and Mathematical Sciences, University of South Australia

Date Approved: December 10, 2013
Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the non-exclusive, royalty-free right to include a digital copy of this thesis, project or extended essay[s] and associated supplemental files (“Work”) (title[s] below) in Summit, the Institutional Research Repository at SFU. SFU may also make copies of the Work for purposes of a scholarly or research nature; for users of the SFU Library; or in response to a request from another library, or educational institution, on SFU’s own behalf or for one of its users. Distribution may be in any form.

The author has further agreed that SFU may keep more than one copy of the Work for purposes of back-up and security; and that SFU may, without changing the content, translate, if technically possible, the Work to any medium or format for the purpose of preserving the Work and facilitating the exercise of SFU’s rights under this licence.

It is understood that copying, publication, or public performance of the Work for commercial purposes shall not be allowed without the author’s written permission.

While granting the above uses to SFU, the author retains copyright ownership and moral rights in the Work, and may deal with the copyright in the Work in any way consistent with the terms of this licence, including the right to change the Work for subsequent purposes, including editing and publishing the Work in whole or in part, and licensing the content to other parties as the author may desire.

The author represents and warrants that he/she has the right to grant the rights contained in this licence and that the Work does not, to the best of the author’s knowledge, infringe upon anyone’s copyright. The author has obtained written copyright permission, where required, for the use of any third-party copyrighted material contained in the Work. The author represents and warrants that the Work is his/her own original work and that he/she has not previously assigned or relinquished the rights conferred in this licence.

Simon Fraser University Library
Burnaby, British Columbia, Canada

revised Fall 2013
Abstract

Research initiative in Service-Oriented Computing (SOC) aims at developing adaptable and scalable distributed applications and addressing challenges such as application integration, reusability, modularity, and interoperability. Service-Oriented Architecture (SOA) as an architectural style enables organizations to offer their application functionality as a service and enhance the adaptability to changes of new requirements of stakeholders, i.e., service consumers. Nowadays enterprises and service providers face several challenges to develop SOA-based solutions. They indispensably require to effectively manage variability in both functional and non-functional (quality) requirements at the business process level to rapidly and cost-effectively develop and deploy customized services that best meet the stakeholders’ feature needs. SOAs provide the architectural underpinnings to support software reuse and enable variability at both design and run-time; however, they lack support to manage variability that promotes configurability and customization. Variability modeling and management have been the core research subjects in Software Product Line Engineering (SPLE) with the objective of addressing the issues of engineering and developing software-intensive systems. Combining SPLE and SOAs has been a subject of considerable research interest in recent years to develop highly configurable software systems.

In this thesis, we adopt a product-line approach in the service domain and hypothesize that the SPLE paradigm, enabling variability management and systematic planned reuse, can be applied orthogonally to aid Service-Oriented Software Engineering (SOSE) to yield these benefits and construct Service-Oriented Software Product Lines (SOSPLs). We propose the Configurable Process Models as the realization of SOSPLs, where services are the building blocks for the implementation of software features, which provide support for variation among members of a product line configured based on stakeholders’ requirements. Our proposed approach provides scalable and efficient automated decision-making support in the course of configuration helping to create tailored software services according to stakeholders’ preferences.

The key contributions of this thesis are: (i) a systematic analysis of the state-of-the-art in SOSPL research; (ii) a methodology to support variability modeling and management for the development of an SOSPL by extending a conventional SPLE life-cycle; (iii) a quality model and evaluation method for aggregation of quality properties in SOSPLs; (iv) a framework supporting automatic quality-aware process configuration; and (v) an empirical evaluation of performance and scalability of quality aggregation and process configuration.
To my family
“Do not be satisfied with the stories that come before you. Unfold your own myth.”

“You are not a drop in the ocean. You are the entire ocean in a drop.”

— Rumi
Acknowledgments

The Ph.D. dissertation is not the result of only one person's work. I am sincerely appreciative of the very many individuals who have supported my work and continually encouraged me through the writing of this dissertation. It would not have been possible without the help and support I received from all my friends and mentors.

This dissertation is a report on the results of my four-year-long endeavours through which I had a great opportunity to collaborate with lots of people. I would like to acknowledge everybody but I think it would then require all the manuscript pages in this thesis, for brevity, here I only go over a concise list of people who played a major role during my Ph.D.

First and foremost, I would like to sincerely thank my doctoral committee members, Prof. Dr. Marek Hatala, Prof. Dr. Hausi A. Müller, and Prof. Dr. Dragan Gašević for their continuous support, valuable advice, and enthusiasm regarding all aspects of this dissertation. Additionally, I owe my deepest gratitude to Prof. Dr. Dragan Gašević for his encouragement, guidance, understanding, patience, and most importantly, his friendship during my graduate studies.

I am grateful to my colleagues in our research group for many useful discussions and for providing a sympathetic and enthusiastic research atmosphere. Especially I would like to thank the core group of people, Prof. Dr. Ebrahim Bagheri, Dr. Marko Bošković, Dr. Mohsen Asadi, Dr. Gerd Gröner, and Dr. Ivana Ognjanović. In particular, I would like to thank Dr. Mohsen Asadi, for all his contributions, insights, and great friendship.

I express my heart-felt gratitude to my wonderful friends for their unlimited kindness, patience, and support, for me to focus on the important areas in my life. I wish I could name all of you individually, but there is truly too many! I appreciate you all, and your insights and criticisms and support have been invaluable to my growth and success in this endeavor. You all hold a special place in my heart. I would like to highlight a few who helped me out so much along the way; however, if you have the feeling that you are one of the people I should acknowledge, you absolutely can feel confident that deep in my heart that is the case!

I wish to thank Dr. Nima Kaviani for being a true friend and a great brother. We started the long journey together and spiced up life in our base camp at Discovery place (I) in Vancouver. It was a great time working together on various projects, papers, hacks and other things. I appreciate all his contributions, initial ideas, and many inspiring conversations for me to embark on this research. We were taking up new challenges every day, tackling them
with all our grit and determination, always thriving to come out victorious. Forging ahead we did and still do all we can to explore the unknowns since one, in my belief, must trade the comfort for triumph. I look forward to continuing our new adventures.

Family and friends’ support has been continuously energizing me during my long journey to finish this doctoral program. Particularly, I would like to extend my thanks to those who have also been amazingly supportive all along with their constant warm-hearted encouragement and companion, to name a few, Dr. Shahin Navardi, Dr. Mohammad Mirdamadi, Dr. Aida Moeini, Dr. Mohammad Tayebi, Dr. Pooyan Fazli, Dr. Melody Siadaty, Farahnaz Shokouhi, Hooman Tahbaz, Amir Madjd, Behrouz Shokrizadeh, Radka Peterová, and Dr. Loretta YW Teng. Also, special thanks to Blair Jovellar and Arash Khoshghadamnia for their thoughtful feedback and editing advice for this work in the final days. Indeed, there are many of you, who do come to my mind and stand out, who have been absolutely awesome to know, but it would be a long list to those who I am indebted to. I gratefully thank all of you.

My deepest and sincere thanks go to my beloved angel mother, Jaleh Dolatshahi, who has always supported me throughout my whole life, encouraging me in all of my pursuits and inspiring me to follow my dreams. Words never can express gratitude and appreciation for her endless patience and countless sacrifice. I also express my heartfelt gratitude to my sisters, Bita and Annahita Mohabbati, who have always been on my side even when I was far away from home. I am exceedingly grateful for their gracious support, light-heartedness, and unrivalled belief in me. I also want to grant a special acknowledgement to Annahita, whose strength, unwavering tenderness, and patience always provide deep sources of emotional renewal for me when I am away from family. Without you, I would not have had the peace of mind needed to undertake this journey.

Last but not least, I would like to thank the most important person in my life, my father Jafar Mohabbati, who is not anymore among us. Turning points in ones life coincide with inspiration. As for me, the inspiration did not happen overnight but happened through the protracted touch with my father. He raised me with a love for science. All things I have experienced in my life with you will never be forgotten. This work is dedicated to you!

Bardia Mohabbati
Vancouver, Canada
December 2013
Publications

Parts of the work presented in this dissertation have been published in the following journal, conference, and workshop papers and book chapter.


Contents

[Approval] ii

[Partial Copyright License] iii

[Abstract] iv

[Dedication] v

[Quotation] vi

[Acknowledgments] vii

[Publications] ix

[Contents] xi

[List of Tables] xvi

[List of Figures] xviii

1 Introduction

1.1 Motivation ............................................ 1
1.2 Background ........................................... 3
   1.2.1 Service-Oriented Software Engineering ................. 3
   1.2.2 Software Product Line Engineering ..................... 6
   1.2.3 Definition of Important Terms .......................... 7
1.3 Research Definition .................................... 9
   1.3.1 Key Research Challenges and Issues .................. 9
   1.3.2 Research Questions ................................. 13
1.4 Research Objectives .................................. 13
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
<td>Research Methodology and Design</td>
<td>14</td>
</tr>
<tr>
<td>1.6</td>
<td>Contributions</td>
<td>16</td>
</tr>
<tr>
<td>1.7</td>
<td>Organization of the Thesis</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>Literature Review</td>
<td>20</td>
</tr>
<tr>
<td>2.1</td>
<td>Introduction</td>
<td>20</td>
</tr>
<tr>
<td>2.2</td>
<td>Related Work</td>
<td>20</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Variability Management</td>
<td>20</td>
</tr>
<tr>
<td>2.2.1.1</td>
<td>Variability Modeling</td>
<td>23</td>
</tr>
<tr>
<td>2.2.1.2</td>
<td>Variability Description and Representation</td>
<td>25</td>
</tr>
<tr>
<td>2.2.1.3</td>
<td>Variability Realization and Implementation</td>
<td>29</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Knowledge-based Configuration</td>
<td>30</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Business Process Variability Modeling and Configuration</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3.1</td>
<td>Business Process Modeling Languages</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3.2</td>
<td>Configurable Process Models</td>
<td>36</td>
</tr>
<tr>
<td>2.2.3.3</td>
<td>SPL-based Process Model Variability Management</td>
<td>40</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Service Composition Synthesis Approaches</td>
<td>42</td>
</tr>
<tr>
<td>2.2.4.1</td>
<td>Static and Dynamic Composition</td>
<td>43</td>
</tr>
<tr>
<td>2.2.4.2</td>
<td>Manual and Automated Composition</td>
<td>43</td>
</tr>
<tr>
<td>2.2.4.3</td>
<td>Model-driven and Business Rule-based Composition</td>
<td>44</td>
</tr>
<tr>
<td>2.2.4.4</td>
<td>Ontology-driven Service Composition</td>
<td>45</td>
</tr>
<tr>
<td>2.2.4.5</td>
<td>Declarative Composition</td>
<td>46</td>
</tr>
<tr>
<td>2.2.4.6</td>
<td>Comparative Analysis</td>
<td>46</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Business Process Optimization</td>
<td>47</td>
</tr>
<tr>
<td>2.3</td>
<td>A Systematic Mapping Study of Service-Oriented Software Product Lines</td>
<td>51</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Systematic Mapping and Research Method</td>
<td>52</td>
</tr>
<tr>
<td>2.3.1.1</td>
<td>Review Protocol</td>
<td>53</td>
</tr>
<tr>
<td>2.3.1.2</td>
<td>Research Questions</td>
<td>54</td>
</tr>
<tr>
<td>2.3.1.3</td>
<td>Data Sources</td>
<td>55</td>
</tr>
<tr>
<td>2.3.1.4</td>
<td>Search Strategy</td>
<td>55</td>
</tr>
<tr>
<td>2.3.1.5</td>
<td>Inclusion and Exclusion Criteria</td>
<td>57</td>
</tr>
<tr>
<td>2.3.1.6</td>
<td>Data Extraction</td>
<td>58</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Classification Scheme</td>
<td>58</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Results</td>
<td>59</td>
</tr>
<tr>
<td>2.3.3.1</td>
<td>Demographic Data</td>
<td>59</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>2.3.3.2</td>
<td>Research Focus</td>
<td>60</td>
</tr>
<tr>
<td>2.3.3.3</td>
<td>Service Variability Modeling</td>
<td>61</td>
</tr>
<tr>
<td>2.3.3.4</td>
<td>Service Identification</td>
<td>65</td>
</tr>
<tr>
<td>2.3.3.5</td>
<td>Service Reuse</td>
<td>66</td>
</tr>
<tr>
<td>2.3.3.6</td>
<td>Service Configuration and Customization</td>
<td>67</td>
</tr>
<tr>
<td>2.3.3.7</td>
<td>Dynamic Software Product Lines</td>
<td>69</td>
</tr>
<tr>
<td>2.3.3.8</td>
<td>Adaptive Systems</td>
<td>70</td>
</tr>
<tr>
<td>2.3.3.9</td>
<td>Contribution Facets</td>
<td>72</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Threats to Validity</td>
<td>73</td>
</tr>
<tr>
<td>2.3.5</td>
<td>Discussion</td>
<td>75</td>
</tr>
</tbody>
</table>

3 Service-Oriented Software Product Line Design and Development | 77 |
| 3.1 | Introduction | 77 |
| 3.2 | Software Product Line Development | 78 |
| 3.3 | Comparison of SPL and SOA | 80 |
| 3.3.1 | Development Processes | 80 |
| 3.3.2 | Reuse in SPL versus SOA | 83 |
| 3.3.3 | Architectural aspects of SPL versus SOA | 85 |
| 3.3.4 | Variability in SPL and SOA | 85 |
| 3.4 | Case Study | 89 |
| 3.5 | Applying SPLE to Develop SOSPLs | 91 |
| 3.5.1 | Service-Domain Engineering | 91 |
| 3.5.1.1 | Product Line Requirements Analysis | 93 |
| 3.5.1.2 | Variability Analysis and Modeling | 94 |
| 3.5.1.3 | Reference Business Process Model | 96 |
| 3.5.1.4 | Feature Resolution and Mapping Model | 98 |
| 3.5.1.5 | Non-functional Specifications | 99 |
| 3.5.1.6 | Reference Business Process Model Implementation | 100 |
| 3.5.2 | Service-Application Engineering | 100 |
| 3.5.2.1 | Application Analysis | 101 |
| 3.5.2.2 | Application Design and Implementation | 102 |
| 3.5.2.3 | Application Deployment | 103 |
| 3.6 | Discussion | 103 |
| 3.7 | Summary | 105 |
# 4 QoS and Aggregation Models for Service-Oriented Software Product Line Architecture

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Illustrative Example</td>
<td>108</td>
</tr>
<tr>
<td>4.2 Preliminaries</td>
<td>111</td>
</tr>
<tr>
<td>4.2.1 Process Model and Specification</td>
<td>114</td>
</tr>
<tr>
<td>4.2.2 Process Structure Tree</td>
<td>115</td>
</tr>
<tr>
<td>4.2.3 Feature Model Specification</td>
<td>115</td>
</tr>
<tr>
<td>4.3 Quality Requirements and QoS Model</td>
<td>118</td>
</tr>
<tr>
<td>4.3.1 Quality Requirements and Criteria</td>
<td>118</td>
</tr>
<tr>
<td>4.3.2 QoS Languages</td>
<td>119</td>
</tr>
<tr>
<td>4.3.3 QoS Model</td>
<td>120</td>
</tr>
<tr>
<td>4.4 QoS Aggregation and Computation</td>
<td>126</td>
</tr>
<tr>
<td>4.4.1 Combining Structural Variability and Composition Patterns</td>
<td>129</td>
</tr>
<tr>
<td>4.4.2 Aggregation Rules Based on Structural Variability and Composition Patterns</td>
<td>132</td>
</tr>
<tr>
<td>4.4.3 A Method for QoS-range Aggregation and Computation</td>
<td>136</td>
</tr>
<tr>
<td>4.5 Discussion</td>
<td>142</td>
</tr>
<tr>
<td>4.6 Summary</td>
<td>146</td>
</tr>
</tbody>
</table>

# 5 QoS-aware Process Configuration and Optimization

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Introduction</td>
<td>147</td>
</tr>
<tr>
<td>5.2 Process Configuration and Challenges Overview</td>
<td>148</td>
</tr>
<tr>
<td>5.3 Problem Description</td>
<td>150</td>
</tr>
<tr>
<td>5.4 Approach Overview</td>
<td>152</td>
</tr>
<tr>
<td>5.5 Preference and Requirements Prioritization</td>
<td>155</td>
</tr>
<tr>
<td>5.5.1 Preference Structure</td>
<td>157</td>
</tr>
<tr>
<td>5.5.2 Conditional Stratified Analytical Hierarchy Process</td>
<td>160</td>
</tr>
<tr>
<td>5.6 Constraint Satisfaction and Optimization</td>
<td>163</td>
</tr>
<tr>
<td>5.6.1 Basic Definitions</td>
<td>164</td>
</tr>
<tr>
<td>5.6.2 Constraint Optimization Problem</td>
<td>165</td>
</tr>
<tr>
<td>5.6.3 Integer Programming</td>
<td>167</td>
</tr>
<tr>
<td>5.6.4 Mixed-Integer Linear Programming</td>
<td>167</td>
</tr>
<tr>
<td>5.7 QoS-aware Optimization</td>
<td>169</td>
</tr>
<tr>
<td>5.7.1 Formalizing Feature Model in MILP Model</td>
<td>170</td>
</tr>
<tr>
<td>5.7.2 Optimization Model</td>
<td>174</td>
</tr>
<tr>
<td>5.7.3 Solution Methods</td>
<td>182</td>
</tr>
</tbody>
</table>
6 Evaluation

6.1 Methodology and Experimental Evaluation ........................................... 188
  6.1.1 Experimental Framework ......................................................... 189
  6.1.2 Experiment Design and Method .................................................. 195

6.2 Performance Evaluation and Analysis .................................................. 195
  6.2.1 Evaluation of QoS-aware Process Configuration ............................... 196
  6.2.2 Evaluation of QoS-range Aggregation .......................................... 199

6.3 Statistical Analysis .............................................................................. 201
  6.3.1 Structural Variability Analysis ..................................................... 203
  6.3.2 Analysis of Composition Patterns ............................................... 209

6.4 Discussion ............................................................................................ 215

7 Conclusions and Future Research .............................................................. 219

  7.1 Summary and Main Contributions ..................................................... 219
  7.2 Assessment of the Research Questions .............................................. 222
  7.3 Outlook and Future Research ............................................................ 225

Appendix A Mapping Study ............................................................................ 230

Bibliography .................................................................................................. 234

Mapping Study References ............................................................................ 260
List of Tables

2.1 Research foci and references. ..................................................... 62

3.1 Comparison of the major engineering activities of software product line engineering and service orientation ......................................................... 81

3.2 Reuse in SPL and SOA .............................................................. 84

4.1 Aggregation rules based on Mandatory/Optional patterns for cost and response time. ............................................................................. 133

4.2 Aggregation rules based on Or/Alternative-feature groups variability patterns for cost and response time. .......................................................... 134

4.3 Aggregation rules based on Mandatory/Optional patterns for availability and throughput. ...................................................................... 135

4.4 Aggregation rules based on Or/Alternative-feature groups variability patterns for availability and throughput. ............................................. 136

5.1 The numerical and linguistic scales for importance [269]. ....................... 158

5.2 Mapping feature model to linear constraints for MILP ........................................ 170

5.3 QoS attributes and pattern-based aggregation functions. ....................................... 182

6.1 Characteristics of the generated models .............................................. 196

6.2 Statistics of computational cost for different process model size. ............... 198

6.3 Descriptive statistics of computational cost (optimization time) corresponding to different ratio values of variability patterns. ......................... 205

6.4 The mean ranks and test statistics of the optimization time corresponding to different ratio of variability patterns. .................................... 205

6.5 Pairwise comparison analysis of variability patterns according to groups (I-IV) including different ratio by their \( p \)-values. .............................................. 208
6.6 The mean ranks and the test statistic for group analysis of variability patterns. 208
6.7 Pairwise comparisons of variability patterns by their $p$-values. 209
6.8 Descriptive statistics of optimization time corresponding to different ratio values of composition patterns. 211
6.9 The mean ranks and the test statistics of the optimization time corresponding to different ratio of composition patterns. 211
6.10 Pairwise comparison of composition patterns according to within-groups (I-IV) representing different ratio by their $p$-values. 213
6.11 The mean ranks and the test statistic for group analysis of composition patterns 213
6.12 Pairwise comparisons of composition patterns by their $p$-values. 213

A.1 List of related search terms. 230
A.2 List of journals. 232
A.3 List of workshops. 232
A.4 List of conferences and symposiums. 233
List of Figures

1.1 Methodology of research design.................................................. 15
1.2 Layered view of SOA, software service development life-cycle hierarchy. . . 16

2.1 Variability modeling space [164].................................................. 21
2.2 Sample feature model for an SPL in Home Integration System (HIS) domain
   (adopted from [44]). Feature model is extended by quality attributes....... 27
2.3 Systematic mapping process..................................................... 53
2.4 Stages of search and study selection.......................................... 56
2.5 Distribution of publication type by publication years[1]...................... 59
2.6 Distribution of research foci of collected primary studies.................... 60
2.7 Map of research foci with respect to contribution and research types....... 72

3.1 a) A holistic view of customer and payment service family. b) Examples of
      process variants configured for different stakeholders...................... 90
3.2 Service-Domain Engineering life-cycle........................................ 92
3.3 Feature model of e-Payment (conforming stakeholders’ requirements model). 95
3.4 A part of reference business process model and associated mappings..... 99
3.5 Service-Application Engineering life-cycle.................................... 101

4.1 An online store product line (a) feature model representing the structural
      variability of the product line and (b) a business process model representing
      the design of the Payment feature from the feature model............... 109
4.2 Meta model of extended cardinality-based feature model.................... 117
4.3 QoS meta-model with proposed extension....................................... 122
4.4 QoS meta-model extended by preference........................................ 125
4.5 Non-functional specification and aggregation for evaluating quality-range sup-
      ports by product line architecture............................................. 128
A.1 Distribution of workshops series .......................... 231
A.2 Distribution of publication fora .......................... 231
Chapter 1

Introduction

1.1 Motivation

Over the last decades, enterprises have faced a challenge: to continuously adapt their business processes in a quick and flexible manner to rapid ongoing changes in the market and customer requirements. A business process model can be considered as a set of one or more activities to achieve a specific business goal. Therefore, the number and involvement of heterogeneous stakeholders (i.e., customers, companies, etc.), and the complexity of their requirements have created the major challenges ranging from configuration to customization and adaptation of applications for a particular purpose, and business agility is often hindered by lack of adaptability and flexibility. Moreover, the growth of distributed software systems poses serious challenges to both scalability and complexity of applications.

Service-Oriented Computing (SOC) has emerged as a cross-disciplinary paradigm to address the development of adaptable and complex software applications and various challenges related to integration and interoperability such as software component composition, reusability, modularity, enterprise data and applications integration [43, 246]. Moreover, SOC aims at facilitating and automating the development of highly-autonomous and loosely-coupled applications which can be composed to communicate in heterogeneous environments. One promising approach to address these challenges is embodied in the notion of services as basic blocks to seamlessly construct applications, create composite services realizing business processes and provide adaptive business solutions.

The SOC is manifested in an architectural style commonly referred to as Service-Oriented Architecture (SOA); hence, SOAs enable businesses to offer their application functionality
as a service (SaaS) which enhances the adaptability to changes in business process models and allows the rapid response and accommodation of the new requirements of stakeholders, i.e., companies or service consumers. One of the key design principles in service-orientation as a design paradigm lies in reuse of services in business processes to achieve agility and cost reduction in application development. Service reuse essentially relies on the matching between requirements of stakeholders and service providers. In a business context and enterprise application scenarios, service providers should target and serve a large number of different groups of stakeholders. Often, individual stakeholders may have common application requirements in a domain although the requirements may be different slightly from one stakeholder to the next. A noteworthy instance is the variation in either functional or non-functional (quality) requirements. Strictly speaking, stakeholders of a service will follow or require slightly different processes or features, respectively.

Nowadays enterprises and companies deal with several challenges to develop SOA-based solutions. To stay relevant with the global competition, they need to rapidly and cost-effectively develop and deploy customer-tailored services to meet a wide variety of their particular domain or targeted market sectors. These challenges thereby motivate enterprises to shift from mass software production to mass software customization. Accordingly, service providers need to effectively manage variability and reuse and support configurability at the business process level in order to provide specialized solutions.

The SOAs provide flexible foundation to achieve adaptable business processes. Nevertheless, the reusability of a service and service-based system is determined by the degree of support the variability which is required to adjust with various contexts. Thus, providing methods helping to explicitly model and manage variability in service-oriented systems can enhance and promote reusability and facilitate highly configurable composite service [111].

SOAs basically lack support to manage variability which promote configurability (high customization), and systematic managed reuse [27, 53, 108, 192, 301].

To address the aforementioned challenges, we have recognized that the current promising approach is leveraging Software Product Line Engineering (SPLE) which has emerged as one of important software development paradigms for variability management and reuse engineering in order to develop software-intensive systems [49, 80, 119, 250, 319]. SPLE has proposed approaches and techniques for traditional development of software systems by focusing on variability modeling, management, and product derivation. The concept of software product line (SPL) is introduced to improve software reuse to mass-customize and
minimize the cost of developing and evolving products with different quality levels. Success stories of using SPL-based approaches by the many high-tech companies prove that the product-lines developments improve reuse at all stages of the life-cycle, shorten development time, and generate significant quantitative and qualitative productivity and satisfaction of stakeholders. [71, 250, 318].

We apply the product line approach in the service domain and hypothesize that the SPLE paradigm – entailing the principles of variability management and systematically planned reuse – can be adopted to aid Service-Oriented Software Engineering (SOSE) to yield these benefits. In this sense, service-oriented applications can be developed and configured to produce Service-Oriented Software Product Line (SOSPL); nonetheless, one of the main challenges in the context of SPL is the realization and implementation of variation points and the method to address run-time variability and dynamic configuration. Hence, SOA can be adopted to implement product lines with services; consequently, from this perspective service behaviour can reconcile dynamic characteristic into SPL architectures to support and promote dynamic SPLs [192]. Leveraging synergies and concepts of SLP and SOA enables us to construct a conceptual structure and address challenges discussed in detail in the following section concerning the management of varying requirements, configurability and adaptability of business process models which are built based on SOAs. To achieve these goals, this thesis addresses configurable and adaptable business process models and different facets within the development and quality-aware configuration of an SOSPL. We propose configurable process model as the realization of SOSPL, where services are the building blocks for the implementation of software features which provide support for variation among members of service products. In the following, we discuss the research challenges and focus on research issues which will be addressed in this thesis.

1.2 Background

1.2.1 Service-Oriented Software Engineering

Service-Oriented Software Engineering (SOSE) has become a widely-adopted engineering paradigm that provides approaches for the design and development of dynamic and adaptable software systems [125]. SOSE is concerned with theories, principles, methods, and tools to accelerate software development and promote adaptation, reusability, and maintenance.
In SOSE, Service-Oriented Architecture (SOA) provides an architectural style, protocols, and interfaces for the design, rapid development, and delivery of applications in form of interoperable and loosely-coupled services. It utilizes the concept of service as fundamental element to compose and develop applications. A service is a reusable and loosely-coupled building block that encapsulates unit of functionality and is accessible through a well-defined, platform-independent, and publishable interface. From an enterprise perspective, individual service corresponds to a business functions and offers functionality to a large number of stakeholders. Reusability principles refer to the applicability of the service for multiple usage scenarios to support requirements in different contexts and heterogeneous environments. Service-orientation (SO), as a design paradigm, aims at loose coupling of services with operating systems and technologies underlying applications to enable flexibility in an interoperable manner. Loose coupling, as a principle in service-oriented design achieve scalability, flexibility, and fault-tolerance. The goal of loose coupling is to minimize dependencies, thus reducing the probability that changes or modifications of one part of an application force changes in other parts.

The purpose of SOA is to address various challenges such as requirements of distributed application development and integration, interoperability, and support of dynamic binding [43]. According to enterprise and non-enterprise perspectives, SOA provides various degrees of adaptation to build dynamic software architectures to address the fluctuations of execution environments and changes in the requirements. However, the adaptation of SOA stems from the fact that it is necessary to build enterprise-scale information systems and support flexible business applications in the context of rapid and often unpredictable changes in business and information technology (IT) [43, 99, 244, 296].

In SOA, the main actors are: a service provider, service registry, and service requester. The service provider defines services descriptions and supplies required functionality and quality. The description of services is published by service providers to make services discoverable. The service registry or repository includes service descriptions and reference links to service providers and provides a mechanism for publishing and discovery. The service requester can be considered as a client, either an end-user or another service. Web services and set of related specification (referred to as WS-* family) have become the preferable implementation technologies for the realization of SOAs. Prevalent at the moment, preferred
CHAPTER 1. INTRODUCTION

four primary approaches exist for realization, standardization and provision of the specification of SOAs: the WS-* family, ebXML\(^1\), semantic Web services\(^2\) and REpresentational State Transfer (REST)-ful services \(^{105}\). Nevertheless, due to heterogeneity of service specification developed and implemented by different vendors, the Web service standardization is still a unfinished task.

One of the important principles of the SOA paradigm lies in service composition in the form of the combination of different services into one or more complex services. Services basically expose operations of certain applications or components; hence, the service composition involves the integration of the underlying, corresponding components. In SOA, the integration is characterized by data model, invocation model, and service choreography and orchestration. The data model describes the data structure and objects that are exchanged using invocation model. Orchestration and Choreography are two major forms of service composition constructing business processes. From the perspective of one party, the orchestration refers to an executable business process that can interact with internal or external services to fulfill an activity, the interactions, and dependencies are described at message level, for instance, in terms of an execution order of services. In contrast, the choreography describes how services are orchestrated from multiple-parties perspective and determines a flow of interactions among all involved services from the global perspective.

SOSE has led to a large body of research in the area of service foundations, service composition, management and monitoring, and service engineering \(^{243}^{246}\). Research in service foundations aims at addressing various challenges in dynamically-reconfigurable run-time architectures, infrastructure support for data, and semantically-enhanced service discovery and composition (at functional and non-functional (QoS) levels). In addition, service foundations focus on communication and middleware technologies to realize the runtime SOA infrastructure (e.g.,, the Enterprise Service Bus) for service composition including the area of QoS-driven composition. The dynamic composition feature in SOSE faces major challenges in QoS-aware compositions, dynamic and adaptive processes, and processes conformance. Service management aims at supporting service configuration and customization, deployment, control and monitoring of service-based applications throughout life-cycle. Service engineering encompasses service development life-cycle that include service analysis, design methodologies, process modeling implementation techniques, construction and

\(^1\)http://www.ebXML.org
\(^2\)http://www.daml.org/services
testing, provisioning, deployments, execution, and monitoring [27, 100, 245].

1.2.2 Software Product Line Engineering

Software product line engineering (SPLE) is a reuse-driven development paradigm that advocates the reusability in software artifacts. The concept of Software Product Line (SPL) (or software families) refers to engineering techniques for constructing a collection of similar software systems (similar products) and enabling systematic reuse of software artifacts in the course of software development, which aims at reducing development times, cost, and complexity, and increase quality and productivity. SPL can be described in terms of four concepts: (i) software assets set includes a library of configurable software artifacts which are used to create all the products of a product line; (ii) product decisions set includes optional or variable features for the products in a product line; (iii) production mechanisms, as a means to assemble and configure products from set of software asset based on a decision model; and (iv) software products, which are a collection of all products produced for a product line, determine the scope of the product line.

SPLE addresses the issues of designing and developing of software for reuse, variability management and mass-customization. It provides systematic approaches and techniques based on domain analysis to identify and manage variability of a system and its applications, i.e., commonality and variability among similar software systems. The commonalities embody the artifacts and properties that are shared by all product-line applications. The variability comprises the variation points representing variable items (i.e., system functionality) and defines how the different applications derived from a product line can vary. This is accomplished by analyzing and distinguishing among assets common to all the family, and assets that may be different from each other. These assets require to be reused in a consistent and systematic way to build applications. Reusable assets might include any types of software artifacts (e.g., software architecture models, design models, requirements models, process models, and software components).

The concept of variability is a central concept of systematic reuse in SPL [250, 326]. Variability enables software systems to be extended, changed, configured and customized for use in particular contexts. Variation points can be defined as places in the software artifacts (as conscious design decisions) that are left open at some point of the software life-cycle, and can be realized and managed at different binding times [128]. Variability can be encapsulated into abstract features which can then be used to generate concrete
and specialized components tailored for the requirements. Variability models describe and represent variabilities and constraints among them on abstract levels. Such variability models are used to describe configuration space which guide and derive a valid software product instance.

In the SPLE, the modeling and managing variability are the main activities concerned with identifying reusable assets, representing variability to achieve further software reuse, customization, and software adaptation.

In contrast to single system engineering, SPLE separates application development into two major development life-cycles that are referred to as Domain Engineering and Application Engineering. The domain engineering adopts developing-for-reuse approach that focuses on analysing the domain, identifying and modeling commonality and variability, and developing reusable software artifacts (also known as platform) such as requirements, architectural elements, or solution components. It starts with domain analysis that produces domain models that normally serve as reference to the software artifacts.

One of the main artifacts generated in the course of domain engineering phase is variability models. Feature modeling is known as an effective technique to determine SPL variability and configuration rules via features in the target domain. A feature model can capture various variability types ranging from high-level SPL variability (e.g., variation in stakeholder requirements) to low-level software variability (e.g., variation in software implementation and code).

By adopting the developing-with-reuse approach, the application engineer develops, configures, and customizes new domain-specific applications based on domain models and artifacts that are created in domain-engineering life-cycle. In feature-oriented SPLE approaches, the product configuration is performed by the selection of a set of features and validation of constraints in a feature model. The configuration is based on a decision-making mechanism which derives a specific product from the product line.

1.2.3 Definition of Important Terms

Due to the fact that some terms are often used synonymously in the literature and may cause confusion, we provide a definition of the most important terms.

A common definition of Software Product Line (SPL), also known as product family or
system family, is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way.

A feature is defined as a logical unit of behaviour of a software system that satisfies functional and non-functional (quality) requirements. In the context of SPL, we refer to a feature as a unit of functionality (increment in system functionality) by which different service products can be defined and distinguished in a product line. The concept of feature represents and encapsulates a design decision and provides a set of potential configuration options. It is also used as a useful abstraction to express commonality and variability. There are several definitions of feature from different perspectives ranging from abstract to technical. Core assets are features, software artifacts or resources built to be used in the production of multiple products within a SPL. Variability are described by variation points. A variation point represents decision in design or implementation that defines the possible variants and also constraints and dependencies on other variation points.

We define a Service-Oriented Software Product Line (SOSPL), also called service family, as a set of similar service-oriented systems that share a common, managed set of (reference) processes supporting specific domain, and are developed from a common set of core services in a well-defined way.

A configurable process model is a reference process model describing composite services which comprises commonality and variability of a (business) process in a consolidate manner and also renders a consolidated view of a service family, where services are configurable to accommodate needs of different stakeholders.

In the context of SOSPL, we consider a feature as an increment in service functionality by which different service product can be distinguished and defined within a service family; therefore, variability in the composition of a service-based system (or business processes) is further modeled and managed by treating service as a feature with different levels of granularity.

Different terms are used to describe the process of building a service product based on a product line. We define process configuration as the procedure of making decisions to select a particular set of features. Service product derivation is a synonym for service or process configuration.

---

3The terms ‘system’, ‘application’, and ‘product’ refer to outcome of product line and are used interchangeably.

4According to IEEE, a feature is a distinguishing characteristic of a software item.
configuration. *Decision* describes the variability in process models (e.g., variable services and variable QoS).

### 1.3 Research Definition

The development of service-based software solutions can be performed by developing or identifying the appropriate services, and assembling services into composite services constituting business processes at different levels of granularity. To implement service-oriented applications, a general architectural approach is the use of a layered architecture [26, 99, 245] as illustrated in Figure 1.2. The fine-grained services provide the implementation of part of business process requirements, for instance, service infrastructure and basic data access. The larger granularities are the product of the compositions of smaller grained composite services. Similarly, the coarseness of the service operations to be exposed relies on the business usage scenarios and the requirements. Therefore, several distinctive software layers are adopted and incorporated into the development life-cycle of service-oriented applications which control the development complexity and provide a logical structure of applications.

#### 1.3.1 Key Research Challenges and Issues

Flexibility in service-oriented systems can be achieved by supporting variability, adaptability, and evolution. Nevertheless, following such a layered design approach in SOA does not guarantee the provision of QoS-awareness and adaptive behaviour for a family of services because of lack of variability management and efficient configuration approaches to deal with the complexity of heterogeneous stakeholders’ requirements and cope with the dynamic nature of business processes. Therefore, SOAs not only need to be able to support dynamic business-process execution and adaptation at run-time, but also require taking into account the business process variability. Support for variable requirements for service configuration is needed. This includes the modeling and management of different process variants and configuration as well as selection of appropriate services with right level of QoS to meet given non-functional (quality) requirements from an end-to-end perspective. Accordingly, constructing a family of service-oriented applications faces several challenges which generate a number of research issues, and there is a multitude of reasons for addressing those issues.
Service Variability Modeling and Management: The configuration and customization of a process-aware or service-oriented system can inherently be achieved by changing the execution order of activities (services) or adding new activities to the process at design or runtime. Process variability can be found in many domains and requires the process to be configured or customized differently which results in process variants; consequently, support for business process variability is required, because different process variants may exist depending on the context or constraints. In practice, process variants follow resembling business objectives and share commonalities and same-core processes. They are often reused in a different application context where some services fluctuate from variant to variant. As a consequence, it results in a large number of related process model variants. Therefore, service variability imperatively demands, for instance, support for different process variants based on concrete product variant. As mentioned above, following a layered design approach in SOA does not per se imply that it addresses variability in requirements. SOA lacks support for planned or systematic managed reuse at the architectural level. Variability at the architecture level is one of the other key challenges. The architectural elements of different layers in SOA, primarily service components and service and business process layers (cf. Figure 1.2), can accommodate variation points in terms of functional and non-functional characteristics; for instance, some services can be either mandatory, or optional for the entire service family. Hence, variability causes the complexity for the configuration and customization, especially when there are a large number of variations and interdependencies among those variations. Thus, meeting business goals through a product line requires managing the variation in different layers.

Furthermore, business processes, for example, in terms of specifications and models are not often designed in such a way to accommodate and manage variability and be highly configurable and reusable in similar business domains. Thus, there is also the question of how to design for reusability and planned reuse and how to manage process variants in SOAs.

A few methodologies have been proposed for the development of service-oriented software systems (e.g., 27, 97, 100, 221, 343). However, there is a challenging downside to SOAs to support higher reusability and variability management to achieve the flexibility and adaptability for business processes owing to the fact that they do not consider product scoping and managing variability in the course of service development life-cycle.
CHAPTER 1. INTRODUCTION

QoS Integration and Evaluation: To develop a family of service-based applications, non-functional (quality) requirements should also have variability because different members of a software family may need different levels of quality. For example, they could differ in terms of cost, availability, security and reliability, etc. Stakeholders can have diverse preferences and objectives concerning quality. For instance, one stakeholder may require very high security whereas for others, high security has little importance. Non-functional variability results in service products having the same functionality but differing in quality characteristics. This is another challenge for the realization of variant services in business processes. Depending on stakeholders requirements, service implementation for a specific variant could be different. A major concern, when developing a flexible service-oriented systems, is to provide an extensible model to address the integration of non-functional aspects (i.e., QoS) on various layers (business process and service layers).

In terms of modeling service-oriented systems, there are a number of approaches for modeling service orchestration and choreography, which leverage different languages [84, 86, 93, 148, 157, 206, 238, 328]; nevertheless, the majority of approaches often do not take into account non-functional aspects at the beginning of analysis and modeling phase. Owing to the fact that the functional and non-functional variability increases the cost and time of an already complex application configuration process, the integration of non-functional properties in the early stages of designing and modeling business processes can amend the configuration of variants at later phases in the development life-cycle.

In addition, it is essential to consider constraints and quality restrictions of individual services and overall requirements to evaluate the potential quality of the developed service-oriented applications because some QoS dimensions values may vary during the service life-cycle. Accordingly, the architectural-quality evaluation becomes crucial due to the fact that it allows analyzing the potential of the architecture to meet the required quality levels and validating whether the final service product satisfies and guarantees all the ranges of quality requirements within the envisioned scope. For instance, quality-range aggregation and computation are prerequisite for the business process management to ensure that the QoS expectations of stakeholders are satisfied; furthermore, quality evaluation is an essential part of the optimization and configuration of a service family because it provides a quantitative metric for the quality of the system based on architecture specification.
QoS-Aware Process Configuration: Enterprises and companies need to support configurable process models\(^5\) that allow the combination of mass customization and the ability to offer and deploy customer-tailored services. The configurable process models enable context-specific configuration of process variants (functional variants) by considering service quality to achieve business agility, flexibility and adaptability, and adequately cope with business process evolution \([261]\). However, deriving a service product from a product line of services (service family) according to the stakeholders’ objective is a complex task which needs to resolve product line variability at different technical levels. The selection of the most suitable variants is defined as a configuration. Therefore, each configuration option (e.g., variation point) from available alternative must be determined. A service family entails multiple configurations regarding variable features and variable QoS. Tackling both forms of variability leads to a combinatorial explosion and optimization problem. In a business process layer (cf. Figure 1.2), different services with the same functionality but different non-functional properties (QoS) may also exist; hence, services could be selected based on their QoS characteristics for corresponding activity constructs in a business process. Therefore, in the context of a service family, QoS-aware process configuration involving the optimal selection of appropriate features and concrete services with respect to variable requirements and imposed constraints is not trivial with a large and complex configuration space. Moreover, various stakeholders may have a diverse range of preferences and concerns about a particular feature of the service product. Understanding and formulating the priorities and objectives of the involved stakeholder are essential to support decision-making and perform optimization in order to choose appropriate features and service during the course of configuration process.

Most existing works are related to QoS-aware service selection and composition which focus primarily on the optimization part of the problem known as NP-complete. However, the present approaches do not take into account the variability in a process (composition) models. In the context of SOSPL, the problem of quality-driven service selection within a service family comprises a much broader range of requisite tasks to solve the problem because of the presence of variable features.

\(^5\)Also, it referred to as reference models
1.3.2 Research Questions

The problems described above demand methods and a framework to effectively model and manage variable requirements to develop a family of service-oriented applications, and support configurable process models. Furthermore, the SOSPL topic is an emerging topic and has not been comprehensively investigated. This thesis investigates the following research questions:

**RQ1:** What is the available evidence regarding the integration or adoption of methods and principles of both SPLE and SOSE paradigms?

**RQ2:** How should we extend the conventional SPLE approach to model, design, and manage variability of functional and non-functional requirements for the development of an SOSPL and support configurable process models?

**RQ3:** How should we model and integrate QoS, and how can the quantifiable values of quality dimensions of an SOSPL be aggregated and computed with respect to structural and behavioral variability?

**RQ4:** How should we support automating and optimizing variant selection in a configurable process model with respect to quality requirements and the constraints boundaries specified by the system and stakeholders?

The first research question (RQ1) is explored by a mapping study and analyzing the state-of-the-art approaches (cf. Chapter 2). The second research question (RQ2) is addressed by introducing a methodological foundation for modeling and developing variant-rich SOA-solutions by incorporating the principles of SPLE into the SOA development life-cycle (cf. Chapter 3). The third research question (RQ3) is answered by proposing a quality model for configurable business process models and a method for evaluation of quality-range (cf. Chapter 4), and the fourth research question (RQ4) is addressed by formulating and solving the problem as an optimization problem (cf. Chapter 5).

1.4 Research Objectives

We have studied current proposals to combine the synergies of SPL and SOA (cf. Chapter 2). Current research directions mainly focus on exploring how these two paradigms can benefit
from each other. However, the problems described above demand a set of methods and a framework for effectively modeling and managing variable (non-) functional requirements and supporting configurability to develop product lines. Configuration of service-oriented product lines has not comprehensively investigated and uncovered challenging research issues. The core objectives of this research are:

1) To devise a model-driven method to manage variable functional and non-functional requirements in the context of service-oriented software product lines. This objective is the result of a need for a methodology to provide a systematic way to support variability modeling and management for the development of a family of software services by applying SPL and SOC principles.

2) To model, integrate, and evaluate of QoS for SOSPL architecture. It is requisite to support and provide a scalable approach to model quality aspects that operate across the full development life-cycle (modeling, developing and configuration).

3) To propose an efficient and scalable approach to support decisions for configuration, also known as service product derivation. This objective is motivated by addressing the underlying challenge in configuration problem of composing the right set of features and further selecting appropriate services which implement given features with respect to stakeholders objectives and preferences.

1.5 Research Methodology and Design

In this thesis, our research methodology consists of three main phases. We follow the design methodology to conduct research as described in software engineering and computer science: problem analysis, solution design and solution validation. The design cycle includes five major steps (cf. Figure 1.1): (1) research problem awareness; (2) suggestion of solution for the problem; (3) solution development and implementation; (4) solution validation and evaluation; and (5) conclusion.

The design cycle is an iterative process which starts with awareness about what problem exists and what research gaps are through the literature review. The problem is formulated and accompanied by research questions. A tentative plans and design for solutions are formed based on the state-of-the-art literature review and ideas which constitute the basis
of the developments of solution artifacts. Suggestions for a problem solution are abductively drawn from the existing knowledge/theory base for the described problem area. For this purpose, the most relevant approaches are surveyed and studied. In the evaluation phase, the proposed method and approaches will be discussed, evaluated and conclusion will be drawn.

According to Adrion [9], software engineering research methodologies are categorized into the scientific, the engineering, the empirical, and analytical methods. The scientific methods involve observing the real world that leads to a method, model, or theory of behavior is based. The prepositions are measured and analyzed, hypotheses of the prepositions are validated, and, if possible the procedure is repeated. The engineering methods observe existing solutions and attempt to develop and propose better solutions. The propositions are measured and analyzed until no further improvement can be achieved. The empirical method proposes model and measures, analyzes, and validates it by developing statistical or other methods. The model is validated and the process is repeated. The analytical method based on proposing a formal theory or a set of axiom from which a method is developed; further, the results are derived and compared with possible empirical observations.

Based on the nature of this research, we have conducted our research mainly using the combination of scientific and engineering approaches. We have studied the existing problems and solutions have been evaluated and proposed. We employ empirical methods to validate our research. In software engineering, the three most-often-used validation methods are controlled experiments, case studies, and surveys [173, 331]. Our research triangulates these methods to some extent: we conducted a systematic mapping study and literature review of current approaches, and we also provided case study and discussion. We also

Figure 1.1: Methodology of research design.
built an experimental research methodology to evaluate proposed approach and algorithms. Moreover, we use simulation for the theoretical-framework testing and evaluation.

Figure 1.2: Layered view of SOA, software service development life-cycle hierarchy.

### 1.6 Contributions

In this section, we summarize the main contributions of the research reported in this thesis. This thesis contributes to the general research areas of SPLE and SOSE. In regard to the aforementioned research challenges and problems, the contributions of this thesis are summarized by presenting research and incorporating SOA architecture to interrelate the contributions to manifest the “holistic picture” of this thesis as illustrated in Figure 1.2. Annotations indicate the contributions to corresponding layer and their scope. The key contributions of this thesis are:
1) State-of-the-art analysis: A systematic mapping study for SOSPL and the state-of-the-art approaches in SOSE by focusing on configurable process models and business process optimization process. The combination and integration of SOSE and SPL have been an emerging research topic and a subject of considerable research interest in recent years. We conducted a mapping study to provide a broad overview of existing research works that have applied or investigated the integration of SO and SPL principles and approaches. The results can help other researchers in the field: (i) to have a quick overview of the classification of research objectives and existing adopted approaches, (ii) to understand the necessity of a systematic approach for the literature reviews and mapping study, and (iii) to realize how such a review can be conducted for further studies.

2) A method for design and development of an SOSPL: By applying SPL orthogonally in service-oriented development, it is expected to derive new software engineering approaches that will help: (i) to develop configurable process models in order to respond effectively to stakeholders’ variable non-functional and functional requirements and (ii) facilitate software architecture that could be reused in different instances. We have introduced a methodological foundation for the modeling and development of a configurable process model by incorporating the principles of SPL into the SOA development life-cycle. Our method produces new knowledge in the SOSE context to support modeling and managing variability in process models and further decision-making for service configuration and customization. We have proposed a novel feature-oriented approach as a key design driver to identify, model, and manage behavioural variation points of business process models.

3) QoS model and evaluation method: We have introduced an extensible multidimensional QoS model to capture non-functional properties that are inherent to an SOSPL. We have developed a quality model framework for holistic architectural quality evaluation. Moreover, we provided the formalization of a computational model for architectural quality evaluation which takes into account both variability and composition patterns and allows for trade-off analysis and architectural decision-making among options that provide similar functional properties but different quality levels.

4) QoS-aware business process configuration framework: We proposed a preference-based framework for quality-aware configuration and optimization of process models. We
have formulated the configuration problem as a constraints satisfaction optimization problem and proposed a formal mathematical model. We have devised efficient user-centered business-process-model configuration algorithm to support automatic variant selection (feature and service selection) with respect to the specified constraints in the system, stakeholder’s objectives and preferences, and quality requirements.

1.7 Organization of the Thesis

The rest of this thesis is organized as follows:

- Chapter 2 presents the systematic mapping study and literature review of the related work classified into the core areas summarized earlier as a part of the contributions. It is noteworthy that some related works are also discussed in other chapters where the comparison analysis of approaches is performed.

- Chapter 3 describes the proposed methodology for systematic design and development of a service family (configurable process model) in the context of SOSPL. We first present a comparison of SPL and SOA by focusing on reuse, architecture and variability aspects of the two paradigms. Then we introduce a methodology for SOSPL development by focusing on the main engineering activities.

- Chapter 4 first presents the related conceptual modeling and formalism for configurable business process model. We introduce the proposed QoS model and present the formalization of a computational model for architectural quality evaluation. This is followed by the detailed description of the framework developed for QoS aggregation and computation of configurable process model.

- Chapter 5 describes a formal foundation and theoretical background to define and provide solutions for a configurable process model. This chapter presents the configuration framework for automated decision support of variants in a business process model. We illustrate how a configuration problem is modeled and solved as a constraints optimization problem.

- Chapter 6 shows the evaluation of the proposal in terms of performance and scalability of the presented approach and algorithms for QoS-aware configuration. The evaluation presents the performance, computational complexity, and scalability of this approach.
with different model characteristics and discuss the impact analysis of structural and behavioural variability patterns on computational cost.

- Chapter 7 concludes this thesis, outlines open issues, and discusses future research directions in connection to the limitations of this work.
Chapter 2

Literature Review

2.1 Introduction

The present thesis has been influenced by many research contributions in the areas of variability modeling and management in SPL, business process management, and business process optimization. This chapter provides an overview of the most related research work according to the main area aligned with the contributions of this thesis.

In the first section, we describe existing related work and show how the suggested approaches are tied into current work. Subsequently, we present our systematic mapping study. The results of which provide a broad overview of topics related to how SPLE and SOSE are combined and leveraged.

2.2 Related Work

2.2.1 Variability Management

Software variability refers to the “ability of a software system or artifact to be efficiently extended, changed, customized or configured for use in a particular context” [303]. Variability management has been widely studied in the field of Software Product Line Engineering (SPLE). In essence, the variability of a system can be viewed in terms of a set of alternative configurations that the system can acquire; hence, variability can exist in software assets encompassing all artifacts relevant throughout software developments (e.g., requirements, documents, architecture, software components and codes, test cases). Therefore, variability
requires to be identified and modeled, implemented, represented, and managed in order to enable assets to be reused and allow the software system to be configured for different application scenarios. In SPLE, variability management encompasses the activities of explicitly representing variability in software artifacts throughout the life-cycle ranging from domain analysis and scoping to implementation and testing, managing dependencies among different variabilities, supporting configuration of variants.

SPL approaches basically include domain analysis and systematic processes for understanding commonality and variability among existing systems of a domain in contrast to traditional requirements-based approaches that merely concentrate on the variability which may exist in the requirement specifications. To model and manage variability, SPLE basically separates two processes: (1) in domain engineering analysis, variability is explicitly modeled, and reusable artifacts are developed (2) in application engineering, applications are derived from the domain artifacts the variability of which is utilized by binding (resolving) variability to configure or customize system according to the requirements defined for particular application.

Figure 2.1: Variability modeling space [164].
Variability in SPL context can also be distinguished in two dimensions: space (e.g., software artifacts) and time (i.e., software artifacts changing over time) [250]. Variability can have different types: functional variability which occurs when a particular function is likely exist in some products and not in others; non-functional variability which happens when there is similar functionality in all products but vary in quality; and data and control-flow variability which occur when particular patterns of interaction or data vary from one product to another.

Based on the domain problem, variability and commonality in a product line can be modeled for both problem space and solution space in various ways with respect to different perspectives [77, 164]. Figure 2.1 illustrates the variability space, where the problem space comprises the concepts describing the requirements of software systems and its intended behavior, i.e., the stakeholders goals or objectives, quality requirements, and the context in which the software products are used. Goals or objectives are modeled to capture and express the high-level abstraction of functional and quality requirements which need to be addressed by product line. Different goals or usage contexts demands different functionalities and/or quality characteristics.

The solution space describes how the requirements are satisfied and how the intended behavior is implemented, including functional capabilities and operational aspects and implementations (i.e., services, components, and system architecture), the operating environmental aspects (such as software platform and operating system), and design features addressing stakeholders requirements. Techniques for features implementation are modeled as “design features” [164]. The features are defined and specified by set of functional and non-functional(quality) properties at different level of abstraction. The concept of feature may serve as a means of domain modeling, variability modeling and management, encapsulating functionality or system requirements, decision-making and further planning, and communication among system stakeholders [290].

In artifact space, variability and commonality are materialized and operationalized by software architecture and components. Features are abstractions of the variability in the course of variability modeling and are realized in development artifacts; therefore, features are mapped to these artifacts. Due to the fact that variability may appear in various artifacts and at any level of abstraction, and variation point in the requirement specifications may be realized by a number of variation points dispersed across different components. Accordingly, there are mappings between elements of problem space (e.g., requirements, features) and
solution space (realized by artifact space) that specify which implementation artifact belong to which requirements or features. These mapping can establish one-to-one, one-to-many, or many-to-many relationships. The mappings can have various forms depending on the implementation method and the degree of automation, and can be simple implicit mappings based on naming conventions or complex rules \cite{77, 78}. Hence, mapping models support tractability between requirement and implementation, feature configuration, and product derivation \cite{78, 120, 274, 290}.

Kang et al. \cite{164} discuss three important dimensions for the variability modeling as follows: (i) different market segments or stakeholders who may have distinct goals and/or diverse usage context; (ii) various goals or usage contexts may need different functionalities or quality characteristics; and (iii) similar capabilities or functionalities may be implemented in different ways and have described quality properties. Variability modeling aims at modeling and managing variability space by considering these various dimensions.

### 2.2.1.1 Variability Modeling

There are several approaches for variability modeling and domain analysis. Feature-Oriented Domain Analysis (FODA) \cite{161} has been known as the baseline for variability modeling. The main focus of this method is the identification of distinctive features of software systems. In the FODA method, a “feature” is defined as “a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems” in order to efficiently abstract away concepts and functionalities at distinctive levels, support communications among various stakeholders of a product line, and maximize reusability. The feature model analysis process involves following activities: (1) collecting source and identifying features; (2) abstracting and classifying the identified features in a feature model; (3) defining the features; and (4) validating the model \cite{161}. As a part of domain analysis, FODA introduced feature modeling, a conceptual modeling technique, to identify and represent the common and variant features, express relations among features and properties of features, and provide the expression of the permissible configurations of features in a given domain.

Other methodologies are developed and extended based FODA such as ODM \cite{283}, FORM \cite{163}, FeatuRSEB \cite{123}, and Generative Programming \cite{76}. There have been several efforts to present diverse viewpoints and extension for feature modeling. Such extensions include structure, binding, configuration, operational dependency, and traceability perspectives \cite{164}.
Organization Domain Modeling (ODM) \[283\] proposed an explicit and iterative domain scoping approach to analyze and model the domain; however, the modeling process is not relied on a particular modeling representation. ODM mainly focuses on the domain engineering of legacy systems. Feature-Oriented Reuse Method (FORM) \[163\] extends the FODA method to the software design and implementation phases to address the issues of reference architecture development. The main reason for the extensions is that the features of a domain characterize each variant product in the domain and the underlying implementations characterizing features should be well-modularized, manageable, and reusable. The method prescribes and provides guidelines as how the feature modeling is employed to analyze of domain feature and using identified features to develop reusable domain artifacts (architectures and components). FORM methodology proposes to dissect a domain problem with four different feature layers: capability, operating environment, domain technology, and implementation technology which is due to the fact that variability may exist in each layers.

RSEB (Reuse-Driven Software Engineering Business) \[153\] addresses the development of product lines by taking advantage of use-case model-driven reuse method, where variability is captured by structuring use cases and object models with variation points and variants. FeatuRSEB \[123\] and its recent extension FeatuRSEB/Sys \[103\] incorporate the FODA and RESB methods by integrating use-case and feature modeling. RESB uses reference architecture which is described by use cases to provide reuse-oriented model. FeatuRSEB utilizes feature model as a catalogue to model variability and commonality, provides a high-level view of the domain architecture and the reusable assets, and yields ‘configuration roadmap’ through use-case models for further system customization. Moreover, other approaches have been proposed based on RSEB that combine use-case and feature modeling to support variability management and configuration for product lines \[98, 120, 226\]. Other approaches have been proposed based on RSEB that combine use-case modeling and feature modeling to support variability management and configuration for product line, such as DREAM \[226\], Product Line Use-case modeling for Systems and Software engineering (PLUSS) \[98\], Product Line UML-based Software Engineering (PLUS) \[120\].

Czarnecki et al. \[77\] introduced Generative Programming (GP) to construct and develop product families by adopting feature-oriented software development (FOSD) \[21, 22\], a paradigm which focuses on treating features as modular, first-class entities throughout the
entire development cycle, including requirements, design, implementation, and test. Feature modeling is an integral part of the development cycle, and the methodology describes techniques as how features are identified, modeled and implemented.

Moreover, several variability modeling approaches have been proposed and developed to support variability management and product derivation [39, 286], e.g., Cardinality-based feature modeling [76, 80], Orthogonal Variability Modeling (OVM) [250], COVAMOF [285], CONSUL /Pure::Variants [46], General Variability Modeling techniques have been discussed in [67, 286].

2.2.1.2 Variability Description and Representation

Variability in a family of systems can be described and represented in different forms independent of the type of representation, classified as integrated and orthogonal variability modeling approaches.

In integrated approaches, the variability information and detailed descriptions are part of other models, for instance, as an integral part of classes, use-case diagrams, etc. [119, 120]. To describe and represent variability, integrated variability modeling approaches can be classified into three groups using: (1) standard (modeling) languages, (2) annotation to standard languages, and (3) domain-specific languages.

Standard (modeling) languages (e.g., UML) can be employed to describe and represent domain concepts by predefined templates, components and classes, and product line can be modeled and configured by frameworks and composition of predefined components. A requirements model represents the feature model, and feature selection is expressed through requirement resolution. In this approach, variation points are modeled only by means of existing mechanisms in the modeling language, for instance, composition, specialization, template instantiation, and parameterization [49, 153]. Variation points can be applied to all elements described by the languages; for instance, use case, packages, components, classes, and collaborations in UML. In such approaches, the configuration process is not an automatic processes and driven by the requirements resolution.

Variability can also be expressed by annotating model elements of a base language, for example, UML profiles with stereotype [119, 120]. In this approach, model elements subjecting to variability are explicitly marked; however, the base models are accumulated

http://www.pure-systems.com
and adumbrated with variability specifications. At the implementation level, annotative approaches also allows fine-grained implementation of variation points and enable to include or remove code segments based on the selection of features. Nevertheless, the annotative approach increases the complexity at the implementation level (source code level); thus, SPL assets are difficult to maintain and evolve.

Domain-Specific Languages (DSLs) can be utilized or defined for variability modeling. A DSL is a language offering expressive power by focusing on a particular domain. It has a meta-model and definition of concrete syntax which may come in a wide variety of forms, for instance, textual languages, graphical and diagrammatic languages, form or grid-based languages, to name a few. Since the primary input of a DSL is the domain knowledge, this approach individually has no special means to model variations in family. However, variability is modeled through the semantics of the DSL constructs in a domain. Due to the fact that DSLs are tailored to specific domain, DSL-based approach does not have the notion of family model, and they cannot be analyzed to support automatic reasoning for product configuration and derivation. Object Management Group (OMG) has initiated an ongoing standardization effort for the Common Variability Language (CVL) defined to specify and resolve variability which can be integrated with DSLs.

In orthogonal approaches, the variability is described and represented by stand-alone (dedicated) variability models independent of the variability realization and implementation, i.e., OVM or COVAMOF. An orthogonal variability model captures the variable aspects by specifying variation points and variants and possible interdependencies between these variation elements. The specified variants in an orthogonal variability model are related (mapped) to the respective variable elements in the artifacts. The underlying idea of this approach is to consolidate the variability information from different requirements and implementation models to yield an independent and consistent variability view of the product line. The key advantages of the orthogonal variability models are reduced complexity, improved decision-making and communication of variability, and supported traceability. In Pohl et al. describe the meta-model defining the concepts of the OVM language and provide detailed introduction to the OVM-based approach.

Feature models are widely-used modeling language in SPL to describe variability and

\[^{1}\text{http://variabilitymodeling.org}\]
CHAPTER 2. LITERATURE REVIEW

considered one of the important contributions to SPL modeling. Feature models as information models which can be utilized independent of design and implementation models to explicitly describe and represent variability and commonality. FODA introduced the notion of feature model to capture and characterize software product line information about common and variant features and their dependencies at different level of abstraction. A feature model represents the configuration space of a product line, where a member of product line can be specified by selecting the desired features from the feature model within the constraints defined by the model. This process is known as feature configuration.

Feature diagrams as graphical representation of feature models offer simple and intuitive notation to represent variant and variation points, which are composed of two main elements: features and their relationships. Figure 2.2 illustrates a sample feature model, which presents the core concepts shared by many feature modeling languages.

Variability is described in terms of optional or Alternative/Or feature relationships, and commonality are expressed in terms of mandatory feature relationship. Feature dependency can also be defined in simple or complex forms. Simple dependencies are often formulated in terms of exclusion or inclusion (requires) relations. Complex dependencies may also exist and affect a large number of variation points and cannot be stated easily formally, for instance, data (input-output), conditional (precondition/effect), and quality attributes.
dependency relations. In practice, features models are extended to accommodate additional information, known as extended or attributed feature models [44, 82]. "Non-functional" or "extra-functional" features are related to feature attributes [162] which as shown in figure 2.2 can be used to specify quality characteristics or extra-functional information such as cost, speed, or any properties required to support the feature.

Different visual languages (i.e., feature diagram languages) and semantics have been proposed for feature models (e.g., Schobbens et al. [276, 277] for a detailed survey). A number of extensions and different notations of the original feature model [161] have been devised to address the ambiguity and lack of precision and expressiveness [36, 44, 76, 98, 123, 162, 319]. In [77], Czarnecki et al. argue that feature models are used to deliberately relax semantics of variability to enable the representation of various variability aspects. A summary of number of feature modeling approaches and their extensions is given in [164, 277].

Feature models have formal semantics and representation using a range of knowledge representation languages, for instance, grammars [36, 77], constraints programming [44], and propositional and description logic [36, 202, 323, 341]. The formal semantics of feature models enable variability analysis and automated reasoning support for further configuration and customization. A significant body of research is available on automated analysis of feature model that aims at extracting relevant information for decision-making in the course of SPL development. Most of proposals rely on declarative techniques and logic to automate analysis by leveraging state-of-the-art solvers such as satisfiability (SAT), Binary Decision Diagrams (BDD), and Constraint Satisfaction Problems (CSP) solvers which have varying degrees of performance and coverage with regard to analysis operations. In [82], Benavides et al. present a comprehensive survey of different approaches and operations.

In practice, feature models are employed in different stages of software development ranging from requirement [163] to implementation [77, 312] to declaratively specify commonalities and differences in the applications in terms of requirements, architectures, components, and test artefacts. Feature models have proved to be effective in hiding much of the complexity in variability specifications and can be also utilized as decision models [79]. Existing survey covers some of feature model application [144].

Decision-based approaches are also employed to describe variability using decision models [66]. In the product line context, decision models include the decisions expressed in terms
of variation points that derive a specific product from a product line. Decision are typically expressed in form of questions with a defined set of possible answers. Decision models comprise the descriptions and references to the variation points, dependencies, attributes and artifacts often represented in tabular or textual forms \[79\] \[164\]. Products are further derived through decision models by setting values to the decision via answering questions, parameterization, and following the order specified by decision dependencies \[256\].

### 2.2.1.3 Variability Realization and Implementation

Various variability realization and implementation methods and mechanisms have been developed \[22\] \[50\]; in contrast, traditional SPL methods use implementation techniques such as inheritance, parameterizations, dynamic class loading, frames, reflection, and design patterns to handle variability at the code level. In \[303\], Svahnberg et al. describe a taxonomy of variability realization: there are different binding times when variability is realized, i.e., design, compilation, build, assembly, configuration, deployment, and run-time \[50\]. Various binding times respond to different adaptation demands and determine when system features can or must be configured according to the desired adaptation level.

Recent works on variability implementation in SPL rely on service-oriented foundations to address run-time variability which occur during the execution of system and the realization of which service-based systems can push to run-time mode. Variability can be bounded dynamically according to different context conditions; hence, service may need to rebind to software services dynamically or be reconfigured at run-time. Different software development paradigms and techniques can be adopted for variability implementation, namely, Component-Based Software Development (CBSD), Feature-Oriented Software Development (FOSD), and Aspect-Oriented Software Development (AOSD), which enable composition mechanisms, such as aspect weaving, mixins, feature composition, and component composition to compose software artifacts using various design patterns \[112\] and service technologies (e.g., OSGi\(^2\) Sun’s Enterprise JavaBeans (EJB), SOAP/WSDL, and REST \[95\]).

\(^2\)Open Services Gateway initiative framework: [http://www.osgi.org](http://www.osgi.org)
2.2.2 Knowledge-based Configuration

Knowledge-based configuration \[300\], also referred to as product configuration, has been a subject of considerable research interest and application area of artificial intelligence (AI) with the objective of automating the assembly or customization of a product to meet the requirements of particular stakeholder. Configuration can be defined as a “special case of design activity, where the artifact being configured is assembled from instances of a fixed set of well-defined component types which can be composed conforming to a set of constraints” \[270\]. Such constraints can be related to the production process, technical constraints, or other aspects; therefore, a configuration knowledge is composed of a formal description of the structure of the product and constraints restricting the possible feature or component combinations. System supporting the configuration of customizable products are often referred to as product configuration systems (or product/service configurator). Configurators support the acquisition of the stakeholder’s requirements and the automation of configuration process using configuration knowledge, that is, a product model and well-defined rules describing how to configure valid individual product according to individual preference and objectives.

Product configuration has been addressed in various ways. Domain and configuration knowledge are two important decision dimensions of configurators which can be constituted based on three main configuration approaches: (a) rule-based, (b) model-based, and (c) case-based approaches. AI reasoning methods (rule-based reasoning/deduction, Bayesian inference, CSP techniques, etc.) use the domain and configuration knowledge to enable reasoning and achieve automation for product derivation. In this regard, Sabin et al. \[270\] conducted a survey on different configuration approaches summarized as follows:

**Rule-based reasoning:** Rule-based configurators use production rules as uniform mechanism for the representation of the configuration knowledge \[209\]. Functional requirements are represented in terms of assertions in the working memory and are mapped to components through rules. The production-rule programming paradigms in such systems enable dynamic and run-time decision-making. A production rule is often expressed in form of sequence of conditions, and a working memory hold global information: input and results from executing actions. If the condition is satisfied, the configurator executes the rule’s consequence. The product solutions are derived in a forward-chaining manner. The major drawbacks of rule-based approach are related to the problems encountered in the course of
knowledge acquisition, consistency checking, and knowledge maintenance.

Case-based reasoning: This approach is based on a tacit assumption that similar problems may have resembling solutions. The knowledge of reasoning is stored in cases that record a set of configuration. By case-based reasoning approach, configurator tries to solve the current configuration by finding a similar, previously-solved problem and adopting it to new requirements. In this approach, reasoning is performed with predefined configuration or cases, and, thus, there is no need to have a complete model of the system; therefore, the approach is appropriate when there are only a few standard products, and when model-based adaptation methods to modify these standard products are available. The system can always propose a solution to the user if it can find a similar case in the case base.

Model-based reasoning: The main underlying motivation of the model-based configuration systems lies on the separation of problem description from the algorithm solving the problem to address the deficiencies of rule-based and case-based approaches. This approach allows to analyze the problem independent of the chosen approach and define the problem description based on the model of system to be configured. Such models include a set of decomposable components and their interactions. The problem description is required to be complete to define a closed space of possible configuration space. The most important advantages of model-based approaches are: (i) reasoning support in the presence of contextual information variation; (ii) compositionability and modularity improvement, i.e., components can be added or removed without changing the entire model; and (iii) reusability enhancement and enabling to use existing knowledge to solve related class of problem. Model-based approaches can be further classified as logic-based, resource-based, and constraint-based configuration approaches:

- Logic-based: these configurators are often based on Description Logics (DLs) which are formalisms for the knowledge representation and reasoning. The inference engines enable to reason about the intentional description of concepts and their instances and allow the creation of complex and composite descriptions. The advantages of DLs are the support of the knowledge-acquisition and problem-solving phases. Furthermore, DL systems automatically maintain consistency as new description are classified and added to the knowledge base, or as existing descriptions are modified and reclassified. One potential drawback of this approach is the trade-off between the efficiency of reasoning task and the expressiveness of the knowledge-representation.
• Resource-based reasoning: Resource-based configurators rely on producer-consumer models of the configuration task. Such models characterize each technical entity by the amount and type of resources it supplies, uses, and consumes. This approach is appropriate for configuration tasks when single components only partially satisfy required functionality.

• Constraint-based reasoning: In this approach, each component is defined by a set of properties and connection to other components. Constraints define relations among the components and determine how components can be combined to form a valid configuration. Constraint-based approach is well-suited to define clear configuration space which enables to ensure composability by treating interaction between arbitrary components and by deriving global properties of sets of components. Depending on the characteristics of the configured system, this knowledge can be formulated by any kinds of constraints models (e.g., CSP or Boolean models [267]). Constraint Programming (CP) is widely-used method which enable flexible declaratively-modeling, formulating, and solving configuration problems. We discuss this approach in details in Chapter 5.

In the SPL context, variability models defining configuration space can be compared with configuration knowledge. In SPLE, variability is expressed by variability models (for example, feature models), and the features supported by the product line and their implementation are defined in domain engineering. A concrete product based on the product line is, then, derived by resolving variability or feature configuration whose features are selected and from which the corresponding implementation can be derived (application engineering). Different approaches have been proposed and developed to address and support product configuration in the SPL context [256]. In the following, we summarize how these methods support configuration of variants and product derivation.

Krueger [182] presents a pattern for SPL methodology, referred to as the 3-Tiered Methodology, which is identified based on observations of practical reuse SPL methods. The base tire provides infrastructure for first-class variation management and supporting automated production mechanism that instantiates products from the feature models, software assets (source code, documentation, etc.) and implementation-level variation points. The middle tier focuses on assets organization and development of reusable components, and the top tire provides feature-based portfolio evolution which manages product feature profiles and feature specifications. The portfolio evolves by adding or modifying feature
requirements for common, optional, and varying features. The proposed methodology is realized in the commercial tool and framework GEAR \(^3\) The framework offers a configurator to provide automation support to create product instances using feature profiles and software assets.

Batory et al. \(^{38}\) present the AHED methodology which generalizes the concept of feature based on the concept of step-wise refinements. The feature refinements are referred to as modules encapsulating individual features, where a feature is defined as a product characteristic in a product line. Features and all corresponding artifacts (such as code, makefiles, etc) are composed to assemble the products. The approach focuses on code assets and is supported by the AHEAD Tool Suite (ATS \(^4\)) \(^{37}\) which provides a tool chain for feature-oriented programming and enables feature selection.

Hotz et al. \(^{143}\) propose the Configuration in Industrial Product Families (ConIPF) methodology for the product derivation by combining product-line engineering and knowledge-based configuration. In this method, products are configured on the level of features. In the course of product derivation, features are selected and tooling supports automatically determine the required hard- and software components based on selected features. The ConIPF Variability Modeling Framework (COVAMOF) \(^{143, 285}\) is developed to enable modeling and managing the variability in a software system over multiple layers of abstraction and support product configuration \(^{287, 301}\).

Dhungana et al. \(^{89}\) have developed a decision-oriented product-line engineering approach called DOPLER to support user-centric product configuration \(^{255}\). The tool-supported approach is designed for questionnaire-based product configuration. The approach enables to define decision models, which describe the differences between products, together with the reusable assets of a product line (such as the reusable components) and mappings between the assets and the decisions. The approach introduces decision-making into product derivation by modelling stakeholder needs, product features, architectural elements, and variability \(^{88}\).

Asikainen et al. \(^{30}\) propose Kumbang, a domain ontology to represent variability in product lines, which includes concepts for variability modelling both from the feature and architecture point of views, the interrelations between them, and configuration. Weight Constraint Rule Language (WCRL) is used as the knowledge representation to represent

\(^3\) http://www.biglever.com
\(^4\) http://www.cs.utexas.edu/~schwartz/ATS/fopdocs
configuration modelling concepts and formal semantics. The Kumbang Configurator is
developed as a tool which utilizes smodels\textsuperscript{5} logic-based inference engine operating on WCRL
to implement the reasoning needed in the configurator for the construction of valid products.

Czarnecki et al. \cite{81} introduced the notion of staged configuration (multi-level configura-
tion) based on several extension to the FODA feature model to configure and derive product
in stages, where some choices are discarded in each stage. The result of each stage provide
a specialized feature model. A configuration is derived from the most specialized feature
model, where all choices are eliminated. The underlying notion is that various groups of
stakeholder may be involved in the process of configuration through multiple stages. Hence,
staged configuration results in gradual resolution of variation points in feature model through
step-wise feature-model specialization process. In this approach, a configuration consists of
the features that are selected based on stakeholder requirements and according to the vari-
ability constraints defined by the feature model. Staged configuration is performed through
subsequent specialization of a feature model. Each stage takes a feature model and yields a
specialized feature model in such way that the set of configuration of the specialized model
is a subset of the original model. A fully-specialized feature model denotes only one con-
figuration in staged configuration process. In proposed approach, the specialization steps
allow fine-grained elimination of variability.

In \cite{329}, White et al. propose MUlti-step Software Configuration probLEm solver (MUS-
CLE) which transforms multi-step feature configuration problems into constraint satisfaction
problems (CSPs) to support automatic feature selection during configuration process. In a
recent paper \cite{127}, Guo et al. propose an approach based on Genetic Algorithms (GA) to
address this problem and find the near-optimal solution for the configuration which meets
specific objectives.

Benavides et al. \cite{45, 311} present FeAture Model Analyser (FaMa\textsuperscript{6}) framework for the
automated analysis of feature model and configuration support. The developed framework
allows integrated solvers (i.e., SAT, BDD, and CSP) in the back-end to perform analysis op-
erations that enable automated extraction of information from feature models, for example,
the number of valid products, etc. \cite{82} Such information can be utilized to support product
configuration.

\textsuperscript{5}http://www.tcs.hut.fi/Software/smodels/
\textsuperscript{6}http://www.isa.us.es/fama
2.2.3 Business Process Variability Modeling and Configuration

As discussed in the introduction, a promising vision of SOC/SOAs is to support the seamless integration of functionality of application systems in terms of loosely-coupled services spanning across organisations and (enterprise) computing platforms. Process-driven development based on SOA is an emerging software development method that supports decoupling the business logic from the specific platform technologies and implementation languages. Processes are the key factor in realizing composite applications, because they enable link between high-level abstraction and logical, implementation-independent designs to the concrete implementation of a system (cf. Figure 1.2). Business processes can express the structure of composite applications, where a process includes a set of activities realized and implemented by services. Business process models (process models for short) specify the workflow (control-flow) execution order of these activities to achieve a specific business goal. Processes are deployed and executed by a process engine, where a process can also be exposed as a service with a standard interfaces, and therefore, can be invoked by end-users or other services in enterprise scenarios, such as business-to-business processes to name one. Business processes has been one of active research area in the field of business process management including concepts, methods, and techniques to support analysis design, development, management, and configuration (or customization) of business processes [261, 327].

In the following sections, we focus and give an overview approaches for business process variability modeling and configuration.

2.2.3.1 Business Process Modeling Languages

Various theoretical models have been proposed as the underlying foundation to describe and model processes which have roots in the business process management and workflow management (e.g., Petri-net, $\pi$-calculus, and state machines). Business process modeling aims at providing high-level specification independent from the implementation. Services need to be specified so that the specification of services is decoupled from their implementation, which facilitates the flexible configuration of services by composing services to achieve complex functionality [327]. A process modeling language provides syntax and semantics to precisely define and specify business process requirements and service composition. Several graph and rule-based languages have been emerged for business process modeling and
development, which rely on these formal backgrounds, for instance, Business Process Modeling Notation (BPMN) [238], Business Process Execution Language (BPEL)/WS-BPEL, UML Activity Diagram Extensions [93], Event-Driven Process Chains (EPC) [84], Yet Another Workflow Language (YAWL), WebSphere FlowMark Definition Language (FDL) [148], XML Process Definition Language (XPDL) [328], Java BPM Process Definition Language (jPDL) [157], and Integration Definition for Function Modeling (IDEF3) [206]. These languages focus on different level of abstraction ranging from business to technical levels and have their own weaknesses and strengths for business process modeling and execution. Mili et al. [219] surveyed the major business process modeling languages and provided a brief comparison of the languages, as well as guidelines to select such a language. In [260], Recker et al. present an overview of different business-process modeling techniques. Among the existing languages, BPMN and BPEL are widely accepted as de facto standards for business process design and execution, respectively.

2.2.3.2 Configurable Process Models

One of the fundamental challenges in the business process management is the modeling and management of business process model variants (process variants for short), for instance, multiple variants for a particular business process, such as multiple shipping services for a product. Therefore, multiple process variants may exist for a specific process, and they can be configured based on concrete requirements. Conventional process modeling languages lack of support for the explicit representation of variability and modeling of such a family of process variants (i.e., business process family). Moreover, process models used in practices and industrial settings lack of support to include configuration decisions which enable customizable process models. Over the past decade, significant research efforts in business process management have focused to address these issues. [3, 121, 187, 265, 266].

Configuration and customization (adaptation) are two general, distinctive strategies to construct and maintain process variants [40]. The former is concerned with the individualization of a reference process model, which serves as a template for a particular domain, to meet the requirements of individual stakeholders. In process configuration, a reference process model is firstly constructed as the super set of process variants which comprises the behaviour of all considered variants. A reference process model is configured, where different variants can be derived by eliminating elements not relevant to the context. Configuration is not oriented to a fully-detailed process model [40] [187]. In process adaptation, a reference
model is not necessarily created. Nevertheless, after a configuration step, an adaptation step may be needed to incorporate specific requirements not having been covered in the reference process model. Process adaptation needs different mechanisms, such as modification and specialization, to adopt process variants based on new requirements \[40\]. Thus, it often devise a set of change operations (i.e., the insertion, deletion, etc.) for variant-specific adaptations. In the rest of this section, we primarily focus on process configuration approaches. Several configurable process modeling languages and approaches have been proposed and developed to model and represent configurable business process models. A conducted survey on business process variability languages can be found in \[265\]. The concept of configurable or customizable process model refer to a single unitary model representing a family of process variants, and each variant can be derived by adding or eliminating fragments according to configuration parameters or with respect to a domain model. The general approach for these efforts is to extend a conventional process modeling with constructs to enable configurable process model by accommodating variation points which are usually associated with specific model elements, i.e., control-flow elements (activities, routing nodes, resources or objects).

Based on underlying variability mechanisms, existing language-based methods to determine variability in process models can be classified into four categories \[187\]: configurable node, blocking and hiding, model projection, and annotation.

In \[3, 121\], the Configurable Workflows approach is proposed which aims at adding configuration layers to the workflow modeling languages. For this purpose, the concept of port as configuration point is introduced to provide configuration operators, namely, activating, blocking and hiding, i.e., to enable, hide, or block a configurable workflow element, such as activity. Configurable YAWL (C-YAWL) is developed, with the meta-model of the YAWL language extended with the notion of variation points to enable configuration operators. Variation points may occur. Other related languages are extended in a similar way.

Configurable EPC (C-EPC) \[187, 266\], as the configurable extension of the EPC language, is developed to explicitly represent variation points in the EPC process models and provide support for both the specification and the configuration of (reference) process models, such as SAP R/3 reference model. C-EPC introduced two constructs: configurable nodes and configuration requirements and guidelines. Process variability is expressed by ‘configurable nodes’ as decision nodes which describe variation points associated with particular types of process model elements. Each configurable node is assigned to a set of alternatives referring to one or more process variants. Configuration requirements allow to formalize
domain constraints over the alternatives of configurable nodes to define feasible and valid configurations. Configuration guidelines can also be defined to aid the configuration process and are expressed as logical predicates and represented as tags attached to the involved nodes; as a result, in this approach, dependencies between variation points are captured via domain constraints encoded in a questionnaire model which guide the configuration process, so in the course of modeling reference process model, decision nodes can be annotated to indicate if they are mandatory or optional. Such information is employed during the (reference) process model configuration which is performed by assigning one alternative to each configurable node via a questionnaire-driven approach [187]. In [187], the C-iECP extends C-ECP notation to represent a range of variations along multiple perspectives. The Aggregate EPC (aEPC) [262] is proposed, as another extension to the EPC, to aggregate process variants in a simple model with the goal of improving the model management and maintainability of multiple similar models.

The Process Variant by Option (Provop) [129, 130] is an approach and framework developed to model process family and manage process variability during the process life-cycle. The approach allows to define a base model as a reference process model. A particular process variant is derived in a context-driven way by applying a set of pre-defined and high-level parameterizable change operations and adjusting the base model, for instance, insert, delete, move activities or process fragments, and modify attributes. Such change operations can also be grouped into options to define complex adjustments. The framework also enables to specify the relations among these options and their semantic usage. The ‘options’ drive the configuration process and constitute specific process variants after evaluating stakeholders’ requirements, that is, including possible context descriptions for which corresponding process variant are required. In [185], a template- and ruled-based approach is proposed to design configurable process model that applies business rules (such as change operations) to configure process variants from a process template (i.e., a reference process model) and support run-time adaptation.

In [40] Configurative (reference) Process Modeling is proposed based on principle of model projection as a variability mechanism to capture process variants of a reference process model. These approach rely on the assumption that a reference process model typically contains the behavior of all relevant process variants and typically includes information of multiple application scenarios; thus, the projection of a reference process model for a particular scenario can be achieved by hiding (skipping) all information and process branches that
are not related to the scenario. To derive an individual variant of process, the configuration is implemented by setting configuration parameters of models.

Annotation-based approaches have been proposed as another variability mechanism by annotating model elements to express and represent variability in process models. The Process Family Engineering in Service-Oriented Application (PESOA) project [254] presents a variability mechanism relying on stereotype annotation based on UML to model variability by defining \textit{variant-rich process models}. Process activities can be annotated to represent variation points (e.g., optional and alternative ones), and variations can be associated to these variation points in the form of other stereotyped activities. Such process models are extended with stereotype annotations which can be applied to both UML Activity Diagrams (ADs) and BPMN models to accommodate variability and provide configuration options. Similar approach is proposed in [259], where UML ADs are utilized to model and manage variability both in control-flow and data-flow of process models. Variability is represented using the decomposition property of activity. Comparable annotation-based approach can also be found in [78], where the idea of annotating model elements to represent variability has been discussed. In this approach, any control-flow elements of an UML ADs can be annotated using \textit{presence conditions} which indicate if the model elements should be present or be removed.

In [263, 264], Application-based Domain Modeling (ADOM) is proposed to specify reference process model and support the specialization in addition to configuration. ADOM proposes three-layered architecture (language, domain and application) to specify reference process models and support the configuration and validation of specific process variants against the reference model. The language layer includes the meta-model of (modeling) language (i.e., EPC [264] and BPMN [263]) that can be employed to describe business process models. Domain layer encompasses specifications of reference process model and provides the guidelines and constraints for the application models. Such constraints supports both the construction of customized models and their validation. To specify the commonality and variability, multiplicity indicators are used in form of $<\min, \max>$ to annotate reference process model elements, such as activities, events, connectors, and arcs. Such cardinality attributes describe the possible lower and upper-most numbers of variants these elements may have in a business process model as well as the number of instantiations that are available for the given element, for example, how many time an activity can be used in a customized process model; case in point, activity annotated by multiplicity indicator $<0, 1>$ represents an
optional activity and can be removed from customized model, and an activity annotated by \(<1,n>\) implies mandatory activity and can be instantiated up to \(n\) times. Thus, commonalities and variability are captured by mandatory and optional elements, respectively. The application layer consists of customized process models derived from the reference model resided in the domain layer. Process configuration is achieved by behavioural restriction through removing optional elements, and process customization is performed by extension through instantiating elements and adding application-specific elements. The formalisms of ADOM is provided for EPC \([264]\) and BPMN \([263]\) languages. In both versions, the business processes are regarded at the conceptual level, and the customized models are not executable.

In \([227]\), a Business Process Family Model is proposed based on SPL concepts in order to model and analyze commonality and variability in process models by extending UML ADs. Two-level analysis is performed, with the primary variation points (mandatory and optional activities) of a reference process model determined at first-level, and the second-level defining the detailed variation points associated to the relationships among activities.

The majority of previous work has focused on business process modeling languages and extensions to address process variability. These approaches propose mechanisms to incorporate variability into reference process models, which increases the complexity of reference process models. Moreover, there is lack of support for automatic variability analysis and configuration.

### 2.2.3.3 SPL-based Process Model Variability Management

Owing to the fact that variability modeling and management have been intensively studied in SPL, considerable attention has been paid so far to apply SPL-based approach for business process variability management and configuration \([256, 273]\).

Sun et al. \([301]\) presents a framework and tool suite developed for variability management of Web service-based systems. The framework extends the COVAMOF \([285]\) to model variability at the architectural and implementation level to address design and run-time variability and support service configuration. To explicitly model and manage variability in process model describing composite services, UML profiles are developed and UML ADs are utilized. The framework enables to configure services based on a variable service composition model. The proposed framework relies on VxBPEL to implement variability at the implementation layer. VxBPEL extends BPEL to support variation points, variants, and
service replacement in a BEPEL process [178].

Montero et al. [225] introduce an approach to model and present variability in business process models represented by BPMN language for the development of a family of business processes. Variability in process models models are defined by feature model which represents variable activities in process model at different granularity level; for instance, root feature in a feature model represents the reference process model, and the child features are considered as an atomic activities or sub-processes (compound activities). Different techniques and transformations are also developed to construct feature model from business process model [14, 34].

Schnieders et al. [275] present a method, called Process Family Engineering (PFE), to design and implement a family of business process model. They adopt SPL-based approach to manage and overcome the complexity resulting from variability. The proposed methodology include three major phases: analysis, design and implementation. Analysis phase focuses on describing the requirements of the process family and variability, with feature model created to explicitly model the process variants and encapsulating the configuration knowledge. Based on the requirements, the process family architecture is designed and business process variants are developed and implemented. In the analysis phase of application engineering, business process family is configured based on feature model to derive an application-specific process, i.e., customer-tailored process-oriented business-information-system variants.

Acher et al. [8] propose an approach to model service variability in a family of process model describing service composition. In this approach, feature models are used to describe variability points in specification of services, for example, service interface. A process model family is defined as a composition of feature models; thus, a set of feature-model merge operators is defined which derive and configure composite services from a set of variable service in a process model.

In [216, 217], Mietzner et al. present a SPL-based approach using orthogonal variability model to explicitly model variability of process-based SaaS applications and support customization. The notion of variability descriptors are introduced and used to locate the variation points in software artifacts, particularly on the process layer, such as business process model, service interface, and process deployment descriptions such as BPEL. Hence, a variability descriptor consists of a set of variation points, with each variation point including a set of locators and a set of alternatives: locators link to software artifacts, and alternatives provide possible values for a variation point. A mapping from variability models to
WS-BPEL process models is introduced in detail in [216]. Variability model and descriptors are utilized to systematically derive customized SaaS applications.

In [232], Nguyen et al. propose a SPL-based approach to model variability of process models and reduce the complexity of process and service customization. They adopt a feature-oriented approach to develop Web Service Variability Description Language (WSVL), as an extension of WSDL, to describe customizable services. A customizable service variants expressed by WSVL includes (i) a customizable service descriptions, for instance, the specification and capabilities of service; (ii) the feature description which describes the variability of service variants in terms of feature; and (iii) the mapping description which provides the mapping between the service features and service capabilities. The approach also extends the BPMN meta-model to define variation points within business process models. The extension focuses on modeling variability of three aspects of process model: control-flow, data-flow, and message-flow. A variation point in a control-flow is interpreted as any location in a process model at which different execution paths can take place, and variants can be arbitrary process fragments. Variability in data-flow is considered as a different way to store data objects. Variability in message-flow is identified as alternative conversations and interactions between two parties, i.e., the process and a partner service (or a consumer).

SPL-based approaches enable to capture and represent variability relations and configuration dependencies using feature models, and separate the variability modeling from the process modeling which reduces the complexity of variability management and configuration.

### 2.2.4 Service Composition Synthesis Approaches

A large body of research works exists in the area of service composition, and a vast number of approaches have been proposed for several purposes with distinctive degrees of scale and automation. This section presents an overviews of selected approaches. Different applications to compose service may require different composition approaches [94]. These approaches can be distinguished and classified according to their underlying strategies for design and run-time composition [12, 94, 258]: 1) static vs. dynamic composition, 2) manual vs. automated composition, 3) model-driven and business rule-based composition, 4) ontology-driven web service composition and 5) declarative composition.
2.2.4.1 Static and Dynamic Composition

Static and dynamic composition concern the time when the services are composed and assembled. Static composition occurs at design time when the software system design and architecture is planned and constructed. The composition structures are modeled by business process models, and the services are either discovered and selected or implemented at design time. However, the static approach is applicable to the cases where business processes, business partners, and services are known at design time and do not change frequently [94]. Static composition is useful to provide complex interaction pattern among known components. This approach leads to commercial product such as Oracle BPEL Process Manager, IBM WebSphere Business Modeler, BEA Web logic, and Microsoft Biztalk which support static composition. Dynamic service composition is an approach with business process partners and services changing at runtime, new services likely to become available and replaced, and partner policies changed [12]. As a consequence, business processes should be flexible and adaptable. Service selection should also be implemented with respect to dynamic change in the context and response to stakeholders’ requirement changes, so dynamic composition approaches require generating composition plan with high level of automation. Dynamic composition are consequently appropriate for applications where users and services are dynamic, for example, ubiquitous and mobile computing. Different dynamic composition platforms have been presented; for instance, Stanford’s Sword and HP’s eFlow, StarWSCoP, SeSCo, WebDG, Component Service Model with Semantic (CoSMoS), Component Runtime Environment (CoRE), and Semantic Graph based Service Composition (SeGSeC) [12, 149].

2.2.4.2 Manual and Automated Composition

Composition approaches can be categorized as manual or automatic. As it is mentioned above, in manual composition approaches, designer manually design and model composition by workflow and interaction among services to generate the composition plans; as a case in point, BPEL is lower-level process modeling and execution language to define and model composition process by control flow constructs. In contrast, automatic composition basically implies that software agent performs the composition based on predefined algorithm. The vast amount of research works for automatic and semi-automatic service composition have been proposed and advocated using AI planning techniques [248]. AI planning methods are often used when there is no predefined process model, but a set of constraints and
preferences are defined. Accordingly, service composition has been investigated to determine how to synthesize complex behaviors which are given an initial state and an explicit goal representation as well as a set of possible state transitions. Hence, it is often assumed that a business process or application is associated with some explicit business goals; because of this, these goals can guide a planning-based composition tool to compose and select the right services [211]. Most AI-planning approaches often propose planning frameworks which can be characterized by the representation of planning domain and problem to enable automatic reasoning; the planning method is applied to solve the given composition problem in the domain and the part of service semantics used for this purpose [175]. The majority of such frameworks include regression-based planners [208], which are based on model checking (e.g., [310]) highly optimized hierarchical task network (HTN) planners such as SHOP2 [288], and a combination of classical and HTN planning, called XPLAN [176], and GOLOG-SPC [211]. HTNs enable the user to define template of how to perform a composition and customizing user constraints [291].

2.2.4.3 Model-driven and Business Rule-based Composition

This approach, known also as workflow-based approach [12], employs a business process modeling language to define abstract process models that include a set of tasks and their data dependencies. Each task contains a query clause that is used to search the real atomic service to fulfil the task. This requires the requester to specify several constraints, including the dependency of atomic services, and the stakeholder’s preference among others. Business rules can be used to structure and schedule service composition, describe service selection and service bindings.

Business rules can be modeled and expressed by Object Constraint Language (OCL). In [240] Orriens et al. introduced a classification schema for business rules: structure-related rules facilitate the specification of the way in which service composition is to be performed, role related rules govern the participants involving in the service composition process, message-related rules control the use of information, event-related rules govern the behavior of service composition in reaction to expected or unexpected events, and constraint-related rules represent conditions in service composition.

Model-driven composition was originally introduced by [240]. Their approach relies on static and dynamic service composition where UML is employed to model service composition providing a high-level of abstraction that can be directly mapped to other standards
(i.e., BPEL), and business rules are expressed by OCL, which governs and steers the process flow. Grønmo et al. [124] proposed a model-driven semantic Web service composition, using semantic Web service languages such as OWL-S and WSML. Furthermore, they propose a methodology to guide developers to compose services in four phases, starting with the initial modeling, and accomplishing a new composite service that can be deployed and published. They identified that models are the primary artifacts that enable the developer during the composition life-cycle to work with higher-level graphical models instead of lower-level and more verbose lexical counterparts. In the proposed methodology, high-level of automation is achieved by identifying model-driven transformations between models and lexical descriptions about the Web services.

2.2.4.4 Ontology-driven Service Composition

Several approaches are proposed in the direction of automated (ontology-driven) service composition by semantic web community. In fact, majority of existing composition planners in this stream draws its inspiration from the vast literature on logic-based AI planning [175]. This approach is based upon ontological descriptions and relationship between services. Ontologies are used to describe semantically the description of functional and non-functional properties of services, to facilitate and automated discovery, selection and dynamic composition of services [258]. Further, the possible compositions are enacted by checking the semantic similarities between interfaces of individual services (semantic matching) and considering the service quality (QoS matching). Semantic Web service (SWS) composition planners are surveyed in [175]. The existing SWS planners are classified according to the latter two criteria, which provide the following classes: Dynamic or static SWS composition planners which rely on whether the plan generation and execution are inherently interleaved in the sense that services can be executed at planning-time, or not. Furthermore, functional-level or process-level SWS composition planners which depend merely on service-profile semantic, or the additional semantic description from process model (data and control flow) [190]. Most implemented SWS composition planners support OWL-S, OWLXplan [176], SHOP2, and GOLOG-SPC. However, there is less support for the standard SAWSDL and WSML available to date [175].
2.2.4.5 Declarative Composition

In declarative composition, high-level declarative descriptions generate the composite services. These approaches use compensability rules to determine whether two services are composable or not [12, 94]. These rules often act as constraints that must be satisfied in order to compose a service and are used to generate composition plans. These approaches usually tend to reach optimality of composition against some defined objectives (e.g., cost, time, etc.) which can be modeled mathematically by utility (objective) functions. The optimality can be achieved by mapping rules to constraints and trying to solve them by operation research methods (e.g., constraint satisfaction problems (CSPs)). [63]. The declarative approach includes two phases: the first phase takes an initial situation and the desired goal as starting point, and constructs generic plans to reach the goal. The second one chooses one generic plan, discovers appropriate services, and builds a workflow out of them. For instance, Benatallah et al. [42] present a framework for dynamic and peer-to-peer provisioning of Web services. Web services are declaratively composed, and the resulting composite services are executed in a decentralized way within a dynamic environment. The framework uses and adapts the state-charts as a visual declarative language.

2.2.4.6 Comparative Analysis

As we discussed above through the literature surveys on service compositions [12, 41, 94, 149, 248, 258], a significant amount of work and efforts has been invested on service composition. However, we argue that the proposed approaches and methodologies for service composition are different with respect to the application and context. Rao et al. [258] present a survey of service composition and different approaches on service composition and provide a comparison study to identify common characteristics and features of an abstract framework. Accordingly, different composition approaches discussed formerly can be basically fallen into two major streams: AI planning-based service composition and Workflow-based service composition. In essence, AI planning-based service composition approaches are adopted when there is no predefined process model, but a set of constraints and preferences are defined. Most of distinguish works in service composition employing planning technique use the closed-world assumption with negation as failure and have limited reasoning capabilities. Even though AI planning techniques are appropriate to achieve automatic service composition, these techniques are not often applicable to development of service-based software.
The workflow-based approaches are the most suitable when a process model is defined. In these approaches, processes of any complexity can be created manually, and then, they can be configured by selecting the services on the basis of the constraints. Although the topology of service composition is generally fixed, the selection of actual competing services could be highly dynamic. Furthermore, such approaches often use semantic in expressive languages and ontologies to describe services and domain concept which enhance the process of discovery and selection [10, 25, 124]. In [10], Aggarwal et al. discuss that having well-designed composition process as a starting point is pragmatic initial step for service composition. One of the benefits of these approaches is enabling to fully control in the life-cycle of process design and to create complex processes which convey the real world requirements. As with other techniques for automation, automated composition reduces the burden to create the process; however, it lacks of the ability to deal with the construction of complex processes compounding like branches, parallelization and loops. In addition, most AI planning-based approaches do not consider the complex constraints that may be involved in selecting the services or stakeholder preferences [10, 94], since the focus is on creating compositions by preconditions and effects. While some semi-automatic approaches provide semantic interfaces to service registries to choose services, the domain knowledge and logical reasoning as well as using the experts’ knowledge are devised to pragmatically select the most appropriate services [124]. Development and deployment of software service specially in the context of software product line(software family) are generally performed in an iterative and incremental fashion.

2.2.5 Business Process Optimization

Business processes allow the specification of complex composite service-based applications to be expressed by abstract services acting as place holders of service components selected at design time or invoked at run-time. In this case, business process require to be optimized to select the best set of services selected by solving an optimization problem.

Service selection is an important phase in service-oriented development life-cycle. Services are selected based on the two kind of criteria: functional and non-functional. Service description matching provides design-time automation in order to discover services with desired functionalities. During the service discovery phase, after the functional matching
at the specific service registry, a list of services is constituted with equivalent functionalities. However, it is required to select and rank among functionally equivalent services with respect to their non-functional properties (QoS), taking into account the explicit quality requirements. The output of the service selection and ranking algorithms is the list of services fulfilling all quality requirements in business process model ordered by their prospective level of satisfaction of the given QoS criteria. A challenging issue toward this purpose is the selection of the best set of services (i.e., in terms of QoS) to participate in the composition, meeting global QoS constraints and preferences imposed by either developer or further by user. This problem is known to be NP-complete \cite{10, 16, 23, 24, 336, 338} and becomes even more relevant for configuration and product deviation in the context of SPL. Several solutions and selection algorithms have been proposed for service composition. A taxonomy of these solutions may be produced based on their objectives and the way they proceed.

Two general approaches exist for QoS-driven service selection as it is stated in the literature \cite{16, 198, 338}: local selection and global selection. In local selection, one service is selected independently from each class (e.g., for individual abstract service or sub-tasks, part of the composition). Local approaches can guarantee only local QoS constraints, i.e, candidate services are selected according to desired characteristics, the price of a single-services invocation being lower than a given threshold is an apt example. When a developer uses a given utility function, each service candidate is assigned a utility value and the service with maximum utility value is selected. Local selection is especially useful for distributed environments where central QoS management is not desirable \cite{24, 42, 194}. However, local selection approach is not suitable for service composition, with global level, i.e, constraints posing restrictions over the whole composition, that is, maximum total price or minimum response time, because such global constraints cannot be verified locally \cite{16, 23, 338}.

On the contrary, in the global selection approaches, the selection is performed by solving an optimization problem that takes into account the (global) QoS constraints of the composite service, the process structure, and QoS metric aggregation rules \cite{23, 338}. These approaches attempt to solve the problem on the composite service level. In essence, these approaches not only compute the aggregated QoS values of each service combination based on composition structure and workflow patterns \cite{155} but also seek the service combination which maximize or minimize the aggregated utility value while guaranteeing global constraints. The global selection problem generally can be modeled as Multiple choice Multiple dimension Knapsack (MMKP), which has been shown to be NP-complete \cite{204}. Therefore
it can be expected that an optimal solution may not be found in a reasonable amount of
time.

Zeng et al. [338] present quality-driven service selection for service composition. The au-
thors propose a quality model and a mechanism for service selection based on multi-attribute
utility theory is proposed. The selection process aims at maximizing user satisfaction which
is expressed as utility function over QoS attributes, while satisfying all the constraints set
by the user or by the structure of composition service. Global planning approach is used
in order to find the best service candidate for the composition. Mixed linear integer pro-
gramming (MILP) [83] is employed to achieve optimal service selection. This work has been
extended by Ardagna et al. [23] where service selection problem is formalized as MILP. In
this approach local constraints are considered, and both service selection methods are based
on linear programming and best suited for small-size problems as its complexity increases
exponentially with the increasing problem size. The result indicates that these methods
suffer from poor scalability in view of the exponential time complexity of the applied search
algorithms [336]. The need to deal with more general constraint criteria, such as service
dependencies, is strengthened in the work of Aggarwal et al. [10]

More recently, Alrifai et al. [16] present an approach in which global optimization and
local selection techniques are combined to address QoS-aware service composition. They
argue that finding the reasonable set of services that avoid obvious violation of specified
constraints at acceptable cost is more important than finding the optimal set of services
with a very high cost. The proposed approach starts with the global level and resolves the
selection problem at the local level. To ensure the fulfillment of global QoS constraints in a
service composition problem without enumerating all possible combinations of component
web service, QoS global constraints (i.e., imposed by the user on the whole composition) are
decomposed into a set of local constraints serving as conservative upper bounds, such that
the satisfaction of local constraints guarantees the satisfaction of global constraints. They
used MILP to find the best decomposition of QoS constraints. However, the main drawback
of this approach is that it represents a greedy selection approach, since it selects services at
the local level and does not ensure that the global constraints are respected.

Second generation solutions proposes heuristic approaches for service selection. In [336],
Yu et al. propose two models for QoS-based service selection and composition: the combinatorial model, by defining the problem as MMKP, and the graph model by defining the problem as multi-constraint optimal path (MCOP). In both models, a user-defined utility function of some system parameters can be specified to optimize application-specific objectives. For each model a heuristic algorithm is introduced to find a feasible and near-to-optimal solution. Two heuristic algorithms are proposed, namely WS.HEU and MCSP-K, for combinatorial and graph models, respectively. WS.HEU is particularly applied for sequential workflows (i.e., workflows structured as a sequence of activities), whereas the WS.HEU is designed for general workflow. The WS.HEU algorithm starts with a pre-processing step to seek an initial feasible solution, that is, a service combination that meets all constraints but is not necessarily the best solution. The post-processing step improves the total utility value of the solution with one upgrade followed by one or more downgrades of one of the selected services. The time complexity of the heuristic algorithm for the combinatorial model is polynomial, whereas the complexity of the heuristic algorithm for the graph model is exponential. Despite the significant improvement of these algorithms compared to exact solutions, they lack scalability through the increase in the number of services and are as a result not a viable solution in terms of the real-time requirements [16].

Others heuristics-based solutions are constructed on genetic algorithms (GAs) [56–58, 154, 339]. Canfora et al. [56, 57] proposed genetic approaches to QoS-aware service selection and composition. GA is used to determine the set of service concretizations [56] (i.e., bindings between abstract and concrete services that lead to QoS constraint satisfaction while optimizing the objective function. Hence, the problem is modeled by means of a fitness function (i.e., objective function) that prescribes the optimality of a solution and need to maximize some QoS properties. The specification of the fitness function and number of iteration are left to the workflow designer.

The research literature indicate that adopting genetic algorithms can be more flexible than integer programming-based approaches, that is, the most widely adopted approach for service selection [10, 16, 23, 24, 336, 338]. In [57], Canfora et al. present empirical data to assess the performance of GAs-based methods against integer programming solutions. Conducted experiments showed that GAs is more scalable while being slower than integer linear programming. GA-based approaches for optimization process can efficiently deal with non-linearity of aggregation function; however, they are less computationally efficient [56–58, 154, 339]; as a result, GAs do not impose constraints on the linearity of the QoS
composition operators (and thus of objective function and constraints). This allows the usage for all possible (even customized) QoS attributes, without the need for linearization. Nevertheless, an alternative to GAs could be the usage of nonlinear integer programming techniques [48].

In [198], the authors argue that the application of GA algorithms to the service selection problem presents two main drawbacks. First, the order in which service candidates are checked is randomly chosen (with equal probability [154]), while there are cases in which services should be checked in an orderly way to optimize timeliness and the optimality with respect to user preferences [198]. Second, as the GA algorithm is iterative stochastic searching technique, iterative operators can run for long periods of time, and algorithm can run endlessly, so the user should define a constant number of iterations fixed a prior. However, fixing a high number of iteration does not give any guarantee concerning the quality of result.

Harman [133] presents different application of optimization techniques in software engineering. The author reviewed widely used optimization techniques and the key ingredients required for their successful application. It is discussed that no matter what technique is employed, it is the fitness function (i.e., objective function) that captures the crucial information; it differentiates a good solution from a poor one.

2.3 A Systematic Mapping Study of Service-Oriented Software Product Lines

Combining and integrating SOSE and SPE have been a subject of considerable research interest in recent years, observed through the literature (e.g., [20, 69, 73, 109, 137, 152, 192]), and dedicated workshop series [74, 183, 184]. Hence, we need to synthesize the evidence regarding the usefulness of combination. Therefore, we conducted a systematic mapping study to analyze the existing research and relevant literature published on this topic. Systematic mapping studies (a.k.a. scoping studies) are designed and performed to provide a wide overview of a research area [174]. In software engineering, systematic mapping studies have been recommended when a research topic is new or not mature enough to comprise a set of comparable empirical studies [172, 174]. The aim is to “map out” the research undertaken instead of answering detailed research questions in contrast to systematic literature reviews (SLRs), which derive very specific research questions [54, 249]. Thus, a mapping study as
CHAPTER 2. LITERATURE REVIEW

a part of evidence-based software engineering is conducted if research evidence exists on a topic and provides an indication of the quantity of the evidence [174].

This section presents a systematic mapping study that aims at collecting evidence about how SO and SPLE are combined or integrated with the aim to identify the research trends and categorize studies at a higher granularity level. Our goal is to provide a map of existing research and synthesize current evidence on the integration of two paradigms. Moreover, the outcomes of our mapping study can help identify research challenges and to direct ongoing research.

The rest of this section is constructed as follows: Section 2.3.1 and 2.3.2 describe the methodology followed in our conducted mapping study and classification scheme. Section 2.3.3 reports results. The threats to validity are given in Section 2.3.4. Section 3.6 presents a discussion.

2.3.1 Systematic Mapping and Research Method

Efforts in software engineering research are dedicated toward developing a standard methodology for conducting mapping studies [54, 174, 249]. Peterson et al. [249] describe methods for conducting mapping studies and discuss differences between systematic mapping studies and systematic literature reviews. Moreover, they provide guidelines for a broader set of situations where either or both systematic mapping studies and systematic literature reviews are appropriate and required to be conducted. A systematic literature review is conducted as a means of identifying, evaluating, interpreting, and comparing all available researches that are relevant to a particular research question and relative merits of competing technologies. In contrast, a systematic mapping study provides a systematic and objective procedure to determine the nature and extent of the empirical-study data to answer a particular research question [54].

Mapping studies often use the same basic methodology as systematic literature reviews; however, they aim at identifying and classifying all related research into a broad software engineering topic. The software community has been working towards well-defined methods to conduct mapping studies [249]. The approach of the systematic mapping study presented in this chapter combines a well-organized set of proper practices both to undertake a mapping study and to systematically review guidelines in the context of software engineering [171, 249]. The combination of guidelines help us to leverage both systematic mapping and literature review techniques. The major steps of our systematic mapping process comprise: (1)
definition of a protocol of the study and research questions; (2) exploratory search and data collection; and (3) analysis of the collected data and reporting. Figure 2.3 summarizes the process followed in our systematic mapping study. The details of each step are described in the following subsections.

2.3.1.1 Review Protocol

A systematic mapping study starts by defining a review protocol specifying the plan of research and primarily including the research. The review protocol primarily includes the research questions, search process, study selection criteria, the classification schema, and the method to extract and analyze data. The review protocol is also important for other researchers who would like to either extend or replicate the study [174]. It is sufficient, in the case of the former, to include only new studies through the subsequent literature search. Our literature review protocol is adopted from the systematic literature review guidelines defined in [171]. We adopted the concepts such as protocol definition that derives the research in the study during the study definition, which improves reliability by describing different aspects of the review to conduct an unbiased mapping study [249].

The protocol used in our study included the following steps: 1) defining the research questions, the scope and search strategy, the inclusion and exclusion criteria to select studies; 2) performing search, collecting and selecting studies using established protocol, and 3) classifying and analyzing papers and extracting and aggregating data to produce a comprehensive overview of the published literature on the studied topic. The protocol was reviewed incrementally and updated according to the newly collected data in the course of study. The protocol was also evaluated and incrementally revised by individual authors. The following
2.3.1.2 Research Questions

The study aims to provide an overview of approaches combining or integrating SO and SPLE, and of their goals. Our main objective is to characterize and summarize evidence and identify the existing research. The research questions for present mapping study is formulated with the help of Population, Intervention, Comparison, Outcome, Context (PICOC) criteria defined by Higgins et al. [140]. Because the objective of this mapping study is not to find evidence on the comparison of approaches, methods or models; the “Comparison” attribute of PICOC was not utilized. The population includes the studies that adopted both SO and SPLE. The intervention includes concepts, principles, approaches or techniques employed from one paradigm to another. The outcome of interest represents a mapping of studies, classification, type and quality of evidence relating to the adaptation of SO and SPLE. The context is within the domains of SOA and SPL with a focus on empirical studies and their application domain. The primary research question that guided our systematic mapping study is: **What is the available evidence regarding the integration or adoption of methods and principles of both SPLE and SO paradigms?**

To answer this question a set of sub-questions was derived from the primary research question to identify and analyzing the relevant literature discussions.

**RQ1:** *In which fora is research on integrating or combining SO and SPLE principles and practices published?* There are a few workshops specifically devoted to service-oriented and software product lines; however, our experience from earlier reviews shows that research may be published in different fora. What are the types of papers typically published (e.g., conference proceedings, journals, and workshops), and the actual publication venues? Publication chronology of the papers and distribution of studies in different venues substantially indicates the existence of research activity.

**RQ2:** *What is the focus and objective (motivation) of the existing research results on integrating or combining SO and SPLE principles and practices? Based on identified synergies, what are the identified characteristics of the possible exploitation?* To identify the focus and key issues which have motivated adoption of one paradigm into the other is the aim of this research question. We have also focused on giving an overview of approaches which are proposed or applied by the integration or combination of SO and SPLE principles.

**RQ3:** *What types of research and contribution are represented?* This question identifies
the types of research (e.g., evaluation research, solution proposal, etc.) and contribution (contribution facet), such as method, model, language, etc.

The answer to RQ1 was retrieved by collecting and selecting related studies and providing distribution of publication fora which is given Section 2.3.3.1. A preliminary classification scheme was established and developed through keywording \[174, 249\] with RQ2 answered through analyzing the 81 selected studies described in Section 2.3.3.2, and the answer to RQ3 retrieved by analyzing the types of research and contribution, summarized in Section 2.3.3.9.

### 2.3.1.3 Data Sources

To collect primary studies, our exploratory search process included digital libraries as well as manual search of journals, conference and workshop proceedings of the most relevant organizations in the software engineering community. The pre-review search was performed using the IEEE Computer Society Digital Library, ACM Digital Library, Science Direct, Citeseer, SpringerLink, InderScience, and Google Scholar. These search engines cover the vast majority of published studies in software engineering. Moreover, the manually conducted search took into account topic-specific conference, workshop proceedings and technical reports regarding the research topics such as Software Product Line Conference (SPLC), Software Reuse, and Software Engineering Institutes (SEI) workshop series and technical reports on SOA and SPLE. These resources are considered the key publication resources in SPLE research area.

### 2.3.1.4 Search Strategy

The search terms in our mapping study were constructed using the guidelines described in \[90\], i.e., through the following steps: (1) deriving major terms by breaking down the research questions; (2) identifying alternative spelling and synonyms for major terms; (3) checking the keywords in any relevant papers we already retrieved; (4) incorporating alternative spellings and synonyms using the logical operator ‘OR’; and (5) using the logical operator ‘AND’ to link the major terms.

The main search terms are shown in Table A.1. The devised search queries were launched for individual selected sources to identify and collect publications. The numerousness of features of digital libraries necessitated slight modification and calibration for respective query
interfaces. Furthermore, we accumulated all the keywords given by the authors of the selected papers with the outcome being a tag cloud generated of terms including more than one frequency. We considered four major keywords deriving our initial search: “service orientation/SO”, “service-oriented architecture/SOA”, “software product line/SPL” and “software product line engineering/SPLE”.

In this chapter, we only report the results of a broad search covering the period from 2001 to 25th September 2011. Therefore, we are nearly certain that we have grasped every paper published in the period of the span of our study despite publication time lag.

Figure 2.4 shows the number of papers identified at each stage of exploratory search and study selection. In stage one, the titles, abstracts, and keywords of publications in the included digital libraries and databases were searched with the combination of given search terms. The outcome of manual search within selected publication fora is also included in this stage. This state resulted in 816 publications being identified.

![Figure 2.4: Stages of search and study selection.](image)

In stage two, the duplicated studies were removed, and it involved initial inclusion and exclusion based on the title, abstract, introduction and conclusion sections. The initial inclusion criteria were studies with both paradigms. We collected 132 studies for further consideration. Furthermore, the study collection is followed by performing “snow-balling”, such as pursuing related works and papers listed in the references of papers. We also communicated directly with some authors for secondary research to pinpoint any other studies that the database searches and manual search might have missed, all leading to nine
additional publications. In total, we finally identified and collected 141 relevant publications and cleaned out the duplicated studies of the same author published in different venues. Typically the most recent ones were considered and included. In the final stage, 81 studies are included for this mapping study after applying selection criteria that are described in the next subsection.

2.3.1.5 Inclusion and Exclusion Criteria

The study selection processes is performed based on careful reading of the collected papers in order to select the most relevant studies which address our research questions. We define following inclusion and exclusion criteria for the selection of collected publications:

- **Inclusion**: Publications with a focus on some aspects of combination, integration or applications of both SO and SPLE.

- **Exclusion**: publications with no concentration on or proposal of an approach related to combination, integration or applications of both SO and SPLE.

Quality assessments of studies are important to restrict bias in conducting the review and to guide interpretation of findings [171]. However, evaluating the quality of a study is not straightforward due to the fact there is no unanimous or universal definition of quality. Because the goal of systematic mapping studies is to structure a large area within the research domain under review, systematic mapping study guidelines do not constitute a formal quality assessment criteria [249]. We used accepted criteria to assess the quality of the studies to test the suitability and selection of studies. Articles must have a minimal description of the context, clear objectives or research goals, and a consistent description of the proposed approach.

In this work, we further consider selected studies for more discussion about their proposed approaches by principles of good practice of conducting empirical research in software engineering proposed. Owning to the fact that the majority of identified studies have been published in recent years, we did not subject them to stringent critical appraisal based on the number of citations.
2.3.1.6 Data Extraction

The data extraction forms were designed to collect required information. We collected the following information from each study: title and authors; source and type (journal, conference, symposium, workshop, book chapter or technical report); date of publication; and collected information based on our classification scheme (cf. Section 2.3.2) and answers to individual research questions. The summary and description of research objectives and focus, research method and contributions were also included. The data are extracted and stored in spreadsheet. The workbook was shared among the authors for the collaboration and synchronization using the share workbook feature in MS Office. During the data extraction, each paper was read entirely and was also scrutinized using the inclusion and exclusion criteria.

2.3.2 Classification Scheme

To address the research questions of this mapping study, a classification scheme is defined to analyze different facets of the analyzed studies in terms of research focus, research type and type of contribution as well as proposed approaches – for primary studies. We developed the classification schema by following the keyword method [174, 249]. Constituting the mapping publication and classification scheme were performed iteratively as a new study was added. The classification schema is created and developed by keywording the abstract, introduction and conclusion, and reading the full-text of studies in order to identify and cluster different facets and contributions of the papers. The set of keywords from different papers was incorporated to develop a high-level realization about the nature and contribution of the papers. This helped us to come up with a set of categories which are representative of the different facets of the research and underlying population.

To address the research question RQ2, we identified the following primary classes of research focus: Service Variability Modeling, Service Reuse, Service Identification, Service Configuration and Customization Management, Dynamic Software Product Line, and Adaptive Systems.

The contribution type (RQ3) is classified into eight categories: Metrics, Tooling support, Method, Model, Language, Open items, Analytical discussion, and Comparison analysis. Research type is classified into six categories based on adopted scheme proposed by Wieringa et al. [330]: Evaluation research, Validation research, Solution proposal Conceptual
proposal, Opinion-oriented papers, and Experience reports.

During the data extraction, some studies could be classified into more than one category according to schema. Based on the classification schema we identified different classes of research focus and type of contribution discussed below.

2.3.3 Results

This section describes the results obtained after conducting the systematic mapping study by following the method described in the previous section.

2.3.3.1 Demographic Data

We provide an overview of the reviewed papers with respect to the types of publication, sources, and research trends. The answer to RQ1 was addressed by collecting information and distribution of publication fora. Figure 2.5 depicts the distribution of the publication types and the grand total number of studies published according to the chronological order of publication years. Evidently, there is an increased number of research studies published

---

5The search process ended up on 25th September 2011.
between 2005 and 2011, when an important international workshop was devoted to the
intersect of SO and SPLE was organized (i.e., the SOAPL workshop series).

Research directions of the series of this workshop mainly focused on exploring how
these two paradigms can benefit from each other \cite{74, 183, 184}. The growing number of
publications in recent years generally corroborates this trend, and many research works
have recently become available (disregarding 25th September 2011 and 2012 when search
processes ended since many studies may not have been indexed in digital libraries or made
available by search engines). The trend testifies the interest of the research community in
expanding this research area.

The distributions and the source of research studies published are shown in Appendix A.
Figure A.1 shows the distribution of studies published in workshops series, and Figure A.2
demonstrates the distribution of publication fora categorized by source including journal,
conference, and symposium papers (cf. Tables A.2, A.4, and A.3).

The distribution of publications in terms of types and resources in this mapping study
(RQ1) indicates the important data sources for this research area and further studies.

2.3.3.2 Research Focus

The following subsections address the research questions RQ2, where each of the research foci
is discussed, and an overview of studies is given with respect to their approaches reported.
Figure 2.6 shows the distribution of research foci of identified and collected primary studies.
If the focus or contribution was within more than one area, we assigned the paper to several
research foci (cf. Table 2.1).

![Figure 2.6: Distribution of research foci of collected primary studies.](image-url)
2.3.3.3 Service Variability Modeling

It is vital that variability modeling and management be supported to enable and promote reuse and customization of software-intensive systems. Variability modeling, as a main principle of SPLE, is employed by many studies in the context of SO to develop highly variable services in a managed environment.

Variability modeling in SO differs from variability modeling in SPLE in several aspects. In SO, different levels of abstraction including business requirements, service compositions, service interface, and service implementation are all variables. Not only should we consider variability of intra-organization services and legacy systems, but also variability in third-party services should be taken into account, incorporated into methods and languages dedicated to SO (e.g., BPEL, WSDL), and considered in dynamic runtime [109, 231].

In SPLE, feature modeling [162] is a widely used variability modeling technique to describe point of variability in an SPL. A feature model leverages the notion of features – visible characteristics of software systems to describe the variation points and variants. Features are defined as visible characteristics of software systems [162]. Feature models provide several types of variability relations (e.g., optional, alternative, and OR) as well as cross-tree dependency relations called integrity constraints. Several variability modeling techniques have been utilized to represent variability at various abstraction levels including requirements, composition of services (i.e., business process), service interfaces, and implementation of services. Notably, a large body of work has concentrated on modeling functional variability in the context of SOA while few approaches model variability in the quality of services.

A large number of the papers is incorporated in our study, employed feature-driven approaches for modeling functional and data variabilities in the context of SOA. Gruler et al. [S33, S38] and Zaupa et al. [S81] employed feature models [36] to represent functional variabilities in service operations. Similarly, Acher et al. [S2] modeled functional variabilities using feature models. For each service, they developed three feature models representing variability in service interfaces, service inputs, and service outputs. Schnieders and Puhlmann [S68] employed feature models to represent functional variability in the workflow systems. In addition, they extended Business Process Model and Notation (BPMN) by variability realization mechanisms such as inheritance, parameterization, extension, and design patterns to incorporate variability in the design artifacts. Diaz et al. [S25] extended
Table 2.1: Research foci and references.

<table>
<thead>
<tr>
<th>Research foci</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Variability Modeling</td>
<td>S33, S38, S2, S68, S25, S58, S31, S28, S43, S47, S5, S73, S62, S80, S72, S75, S1, S63, S42, S67, S79, S61, S11, S9, S14, S13, S56, S12, S32, S34, S49, S69, S76</td>
</tr>
<tr>
<td>Service Identification</td>
<td>S48, S50, S71, S29, S41, S4, S23, S16, S51, S15, S81, S25, S25, S10, S59, S40, S8</td>
</tr>
<tr>
<td>Service Reuse</td>
<td>S55, S54, S74, S2, S62, S35, S42, S56, S77, S66, S60, S72</td>
</tr>
<tr>
<td>Service Configuration &amp; Customization</td>
<td>S51, S64, S65, S36, S20, S19, S18, S39, S70, S80, S46, S49</td>
</tr>
<tr>
<td>Dynamic Software Product Line</td>
<td>S27, S26, S17, S3, S78, S22, S21, S45, S52, S6, S57, S44, S53, S8, S37, S24</td>
</tr>
</tbody>
</table>

the Web Services for Remote Portlet (WSRP) specification to support developing a family of Portlets. They also used a feature models to represent variability in consumer profiles (i.e., functional requirements of consumers) and mapped features to artifacts using a feature x artifact matrix. Nanjangud and Karthikeyan [S58] proposed a variation meta-model consisting of variation points and variant features for service modeling to support modeling functional variability that may exist in composite services and service interfaces.

Some of the collected studies employed feature models for modeling variability of both functional and non-functional (QoS) aspects of services. The results presented in Asadi et al. [S9], Boskovic et al. [S13, S14], and Mohabbati et al. [S56] represent variability in the business requirements, service composition, and quality of services using feature models. They extended feature models with annotation properties to annotate features with required quality values. The DiVA approach [S31] proposed by Greenwood et al. captures variability in both functional and non-functional requirements of services and employs feature modeling to represent variability in the requirements and the composition of services. Moreover, an aspect-oriented model is utilized to encapsulate the implementation of service variants. Fantinato et al. [S28] used feature models to represent variability in Web services e-contract (WS-contract) definitions and quality of service. In their approach, mandatory, optional, and alternative elements of Web service contracts are identified and represented using feature models concepts. The feature model is mapped to the contract template, and
configurations of the feature models are used to derive a concrete Web service contract. Kattepur et al. \cite{S43} modeled variability and constraints of composite service orchestrations via feature models and proposed an effective technique for achieving feasible service configurations. La and Kim \cite{S47} categorized different types of variability in Software-as-a-Service (SaaS) applications into attribute, logic, workflow, interface, persistency, context, and quality of service. They propose a high level feature-oriented process to capture and model variability. As a result of variability modeling activity, decision tables containing a set of variable features along with their variability characteristics are generated. Medeiros et al. \cite{S5} adopted feature-driven approach to illustrate both functional and non-functional variability in a set of services. Their work discusses variability possibly occurring in service interfaces, business processes, and service components.

Furthermore, other variability modeling languages, such as Orthogonal Variability Model (OVM) \cite{250} and COVAMOF \cite{285}, have also been considered for modeling variability in the context of service-oriented software development. Sun et al. \cite{S73} employed the COVAMOF variability modeling framework as a separate variability dimension used in combination with UML diagrams to model variability in the context of service-based systems. They concentrated on variability in service compositions and service interfaces and addressed both functional and non-functional aspects. In \cite{S62}, Nunes et al. focused on modeling functional variability at the requirement level (represented by using goal models) and process models (represented by using business process model notation) by OVM.

UML was used to model and represent variability in either a separate dimension or in internal models. Yu et al. \cite{S80} extended the UML to represent variability in service compositions. The approach focuses on representing functional variability (i.e., difference in services) that provides various functionalities in a variation point. Stollberg and Muth \cite{S72} extended SOAML (a UML profile for SOA modeling) and defined a service variability metamodel to describe necessary constructs to model variability in services. Moreover, they proposed a service variability specification models to define variable aspects of service interfaces including data and operations. An approach proposed by Wada et al. \cite{S75}, a UML profile is used to represent services and their non-functional requirements. They applied feature models to describe non-functional requirements and their variability and constraints relations in a separate model. A service application is generated according to configuration and UML models defined in the UML profile. Abu-Matar and Gomaa \cite{S1} proposed a
CHAPTER 2. LITERATURE REVIEW

feature-driven multi-view variability modeling technique to model variability in service contracts, service composition, and service interfaces. Feature models are represented, using the UML, and SoaML is adopted to represent service contracts, composition and interfaces. Park et al. extended the meta-model of UML activity diagrams to integrate and show different kinds of functional variability in business processes. In their approach, variations were analyzed at different abstract levels and variability is refined by using several concepts – variation point, variation point type, variation point cardinality, and variants.

Variability notions, borrowed from SPLE, have been utilized to develop domain-specific languages to model variability in the SOA context. Karam et al. proposed a domain-specific visual language, called Visual Web Composition (Visual WebC), to demonstrate workflows and their variability. The proposed language consists of webpad elements, which are generic elements that can be replaced with Web services. Reinhartz-Berger et al. proposed an application-based, domain-modeling platform for expressing and managing variability in reference business process models for enterprise service-oriented applications. These models can be specialized and customized according to the requirements. Ye et al. proposed an XML-based language to model variability in business process models. At the business process model level, variations may exist in the activities, events, data, and gateways. Similarly, the approach proposed by Nguyen et al. aims at modeling variability in control flow, data flow, and message flow. Their approach extends BPMN to describe variability in a process-based service composition. In order to describe in a separate variability dimension, a new language called, Web Service Variability description Language (WSVL), based on the notion of features, was introduced. Bayer et al. proposed variability mechanisms to showcase families of business processes. Their approach extends process models with stereotype annotations which accommodate variability and are applied to both UML activity diagrams and BPMN. The business process model family is further configured and tailored by means of a configuration model to resolve the variation points and instantiate final service product. Boffoli et al. also modeled functional variability at the level of business process models. They used decision tables to capture variability and manage variations points and variants in process models.

In addition to modeling variability in the services, some approaches concentrate only on the realization of variability in the services or validation of the variability models and service templates. Van Gurp and Savolainen present variability realization techniques that can be used in SOA context. They described nine variability realization techniques.
and proposed a process to select the right realization technique for implementing variability. Gröner et al. [S32] proposed an approach for validation of configurable process model describing the composition of services. Their approach employed description logic reasoning to validate the mapping between feature model describing the configuration space of services and composition of the services that are expressed by BPMN models in the domain engineering.

2.3.3.4 Service Identification

A service can be seen as an abstract resource representing a capability of performing a task and providing a coherent functionality from the provider and requester entities [99, 244]. Service identification is a critical challenge in early phases of the SOA life-cycle for developing service-oriented systems [177, 245]. Service identification focuses on reusability and determination of what business functionalities should be provided by the target service in the application domain. Hence, it primarily involves identification of the right level of service abstraction and granularity, and provisioning of service capabilities from users’ requirements.

Lee et al. [S48, S50] introduced a feature-oriented approach, adopted from SPLE, to analyse and identify reusable services. Feature analysis and modeling are employed to identify and group units of functionality (specified as features) to provide services at the right level of granularity in a service-oriented system. In [S71], Souer et al. discussed feature modeling for commonality analysis which leads to the identification of business process models and services in the context of content management systems and web applications. Galster et al. [S29] presented an approach for the identification of potential core services for service-oriented product lines based on a trade-off analysis between the added value of services and the structural stability degree of a system.

Kang et al. [S41] proposed a method for service identification for software product lines by using variability analysis and feature modeling. The proposed method provides guidelines about how to group features as service candidates and refine them to achieve a proper level of granularity, which specifies the scope of variability in functionalities exposed by services. A top-down approach for the systematic identification and documentation of service-oriented core assets is described by Medeiros et al. [S4]. In their approach, variability and commonality analysis inspired by SPLE is adopted for service identification and analysis.
CHAPTER 2. LITERATURE REVIEW

66

of similarities in functionalities of services in the early stages of service development. Furthermore, variability analysis is performed to generate architectural decision, which derives service customization.

In the work of Chen et al. [S23], feature-oriented analysis is employed to identify services to develop service-oriented systems. Their approach focuses on reengineering of service-oriented systems and restructuring legacy systems, where feature analysis of a particular system is performed and its results are used in service identification.

2.3.3.5 Service Reuse

SOA, in which one of the key design principles is to provide pieces of reusable functionality to be exposed as service, utilizes services as fundamental elements to develop applications. To fully utilize the benefits of SOA, we need effective methodologies to support systematic, agile, and cost-effective reuse during the development of service-oriented applications. SPLE relies on a fundamental distinction of development for reuse and development with reuse to minimize reusability—an insight having been leveraged to improve design, development, and evolution of service-oriented applications [71, 250].

In [S16], Butler postulated that SPLE can be employed to promote service engineering by systematic reuse. Lee and Kotonya [S51] offered an argument that SOA lacks support for systematic reusability, discussing that not adopting variability management approaches causes more difficulty to reuse services for mass customization and to achieve effective construction, maintenance and evolution of similar software service products. Hence, a motivation to improve reusability and increase flexibility leads to integrating SPLE with service-oriented development.

Bubak and Gomaa [S15] discussed how concepts of SPLE (e.g., static and dynamic feature modeling) can be leveraged to support and escalate service reuse. Their work also provided an argument on how an enterprise can possibly gain competitive advantage by coupling SOA capabilities with systematic and strategic reuse by adopting SPLE approaches. In [S81], Zaupa et al. also offered a discussion and reported lessons learned about creating Web applications based on SOA by following the principles of SPLE. Their work focused on an application generation and offered a method that consisted of a set of activities to define the application domain and service model and customize applications by applying feature modeling. In [S25], Diaz et al. [S25] described a feature-oriented method to develop and construct portlets as reusable services. Balzerani et al. [S10] proposed a software product
line architecture for Web-based applications, built in a prescribed way on compositions of reusable service components with explicit variability management.

In \cite{S7}, Alferéz et al. argued about needs for systematic reuse and proposed a method to support reusability of Web Services and service compositions for mass-production environments. The proposed approach adopts SPL-based and model-driven development methods. Feature models are used to capture structural variability of business process models which are one of several ways to describe service composition. Alferéz et al. also described how the configuration of feature model, involving the selection and deselection of reusable features mapped to services, guided the construction of specific service composition. Street and Gomma \cite{298} discussed architectural issues for reuse in SOA and, in particular, how to support sound architectural design practices necessary to build systems out of reusable services from ground up. In another work \cite{S30}, the same authors described an approach to exploit Web services to increase the reusability in SPLE.

Narendra et al. \cite{S59} presented a formal approach to modeling variability at the business process and service levels to enhance and maximize reusability of SOA-based solutions. Their approach generates different variation of solutions to meet changing requirements. In \cite{S40}, Jiang et al. discussed reuse in service construction at the level of service interfaces and implementation in the process of developing families of Web Services. They aimed at improving the degree of service reusability in a product line of services by identifying and managing the points of variability, and as such, they proposed a patterned-based method based on UML to model and manage variability of services functionalities. Altintas et al. \cite{S8} presented an approach that integrates product line architecture with reflective rule-based business-process modeling for service-oriented application development.

2.3.3.6 Service Configuration and Customization

Software-as-a-Service promises to enable software application vendors to deliver software functionalities in a flexible and scalable way to serve as many users with different business objectives as possible. Configuration and customization is another crucial issue in service development on account of different functional and quality requirements of users for particular domains or contexts. Most enterprises and service providers are inevitably obliged to tailor their service to meet requirements.

In \cite{S55}, Mietzner et al. used variability models, described by OVM \cite{250}, to systematically derive customization and deployment of services for individual users. In \cite{S54}, the
same authors described how to generate customized processes and create deployable services by means of explicit variability models that define variability descriptors for the process at the logic layer. They put forth how the variability descriptors are transformed into WS-BPEL process models that guide the customization. Sun et al. \cite{S74} introduced a method to develop customizable Web services by incorporating the SPLE principles into service composition. The proposed approach adopts domain and application engineering processes from SPLE to enable users to construct and customize composite Web services. Acher et al. \cite{S2} proposed an approach to managing services as a product line architecture. Their approach uses multiple feature models to structure information concerning service variability. A proposed set of composition operations enables a composition of feature models for further configuration of composite services. In work of Nunes et al. \cite{S62}, SPLE and SOA were incorporated to develop service-oriented user agents. The proposed approach compromises the activities to construct and develop customized agents encapsulating functionalities of services that automate users' tasks.

Hadaytullah et al. \cite{S35} proposed an approach to modeling and managing variability in business process models for service customization. UML activity diagrams are utilized to model and manage business process models. A customization is enacted and performed by using pre-defined specialization rules. In \cite{S42}, Karma et al. adopted a lightweight product line model to perform calibration and customization of Web service-based Web applications. The proposed approach aimed at facilitating the composition and customization of service for specific users. Their approach supports agile methods, which were particularly relied on a domain-specific visual language.

Mohabbati et al. \cite{S56} proposed a feature-driven approach for the configuration and optimization of reference business process models and derivation of customized composite service according to users' functional and non-functional requirements. In \cite{S77}, Wang et al. used feature modeling to model variability in composite services in order to enable end-user service customization corresponding to variable requirements. Utilizing variability modeling for decision making, configuration and selection of alternative composition of services is also discussed by Petersen et al. \cite{S66}.

Nguyen and Colman \cite{S60} proposed a feature-oriented approach to customize Web service interface. The feature model is used to describe variability and variation points existing in the service interface, which guides the customization of service. In \cite{S72}, Stollberg et al. proposed a method for service customization by variability modeling of service specifications,
i.e., optional operations, message types as well as their dependencies.

2.3.3.7 Dynamic Software Product Lines

Dynamic Software Product Lines (DSPLs), as specific type of product lines [18], focus on the development model for adaptable and dynamically configurable software systems. DSPLs’ goal is to support variability management at run-time (i.e., dynamically (re)binding variation points), variable quality and performance management, (re)configuration, and dynamic product derivation. Moreover, DSPL approaches aims at identifying reusable and dynamically reconfigurable core assets at development time, which are explicitly modeled as dynamic variability [18, 131]. Hence, from service-oriented perspectives, service characteristics (e.g., dynamic discoverability and binding) can be leveraged to realize and implement product line architectures and support the development of DSPLs.

Lee and Kotonya [51] posit that a service-oriented product line is a DSPL domain application. In their approach, features, which provide functionalities of applications, can be mapped to activities of workflow models or dynamic services by means of service analysis [46, 49]. Hence, the variation of the features can be dynamically bound during run-time by means of selecting services according to their quality.

In [64], Parra et al. argued that adopting SPLE to construct context-aware system based on SOA enabled a complete service development ranging from requirements to implementation. SPLE also enabled the context management throughout the software lifecycle. Parra et al. proposed a homogeneous context-aware DSPLs to build service-oriented applications and adapt them at run-time in accordance to their usage context. They, in another work [65], proposed a feature-oriented approach to automating the derivation of product architectures from feature configuration by combining model-driven and aspect-oriented development. In their approach, service-components architectures are employed for a target platform. In [36], Hallsteinsen et al. also discussed that the integration of SOA and SPLE principles enabled DSPLs and provided a suitable development model for configurable systems.

Cetina et al. [20] utilized model-driven development and SPLE for modeling run-time variability, adaptation policies and reconfiguration of services. Some guidelines to design and implement a DSPL in the context of autonomic and smart homes is given in [18, 19].

Istoan et al. [39] investigated the feasibility of synergy between SOA and DSPLs in the context of home automation systems. They discussed significant advantages to address
DSPL by combining converging synergy points of SOA and SPL. To this end, they proposed a service-based middleware designed to solve issues such as devices interoperability, run-time variability binding, and linkage facilities. In the research reported in [S70], a feature-oriented method to support run-time variability reconfiguration is proposed by Shen et al. They used feature model to capture run-time variations and their dependencies in order to provide high-level business views for adaptation.

In [S80], Yu et al. argued that SOA provided dynamic capabilities needed in many product lines while SPLE approaches enabled useful mechanisms to model dynamic applications implemented through service compositions. Their proposed approach was to build and adapt dynamic service applications based on these two paradigms, including three main phases: domain engineering encompassing the phase to define reference architectures as well as domain specific and reusable core assets based on services; application engineering that defined application architectures according to particular requirements; and run-time phase executed autonomously and configure requirement-based services.

2.3.3.8 Adaptive Systems

One of the most promising characteristics of adaptive software systems is their ability to optimize service provisioning and automatically adapt their behavior at run-time based on the environment and guided by objectives and needs of the users. The typical goal of the adaptations is to provide the best possible services to users according to the users’ requirements within a particular context and resource constraints and to support fault tolerance in a dynamic environment [167].

Elkhodary et al. [S27] discussed the notion of features and the role of feature modeling techniques from SPLE to alleviate the challenging difficulties of building self-adaptive systems. They showed that feature modeling could provide an effective mechanism for analytical modeling, which could capture interactions and identify conflicts among the goals in a system. Feature modeling enabled to manage the configuration space of an adaptive system and for reconfiguration of the adaptive system. In another work of [S26], they utilized feature models, specified at design-time and employed as analytical and decision models by which to evaluate behavior and characteristics of a service-oriented application at run-time to make appropriate adaptation decisions. In [S17], Cetina et al. proposed a method based on the principles SPLE to develop adaptive pervasive service-oriented applications. In a like manner, feature models, as decision models for the run-time, are employed to guide the
dynamical reconfiguration of an application. Acher et al. [S3] applied feature modeling to define the adaptation rules to determine which configuration of the adaptive system to be executed in each specific context. In [S78], White et al. employed a model-driven approach to managing complexity of developing adaptive services. Their approach used high-level adaptive models, defined by means of feature models, to derive a new service composition when a service failed.

A service-oriented analysis and design method to develop adaptable services is presented in [S21, S22, 62] by Chang et al. The proposed method considers three types of variation points in service design: workflow, service composition, and business logic. At design-time, service variability is modeled and designed into service components and compositions to support service adaptation. By following SPLE concepts, Kim et al. [S45] used variability modeling to design reusable services requiring run-time adaptation and evolution.

In the work of [S52], Lee et al. employed SPLE concepts (e.g., variation points and variants) for service decision modeling of content adaptable services in the context of pervasive computing. Lee et al. argued that using SPLE enabled to define systematically decision strategies needed for adaptation. Alferez et al. [S6] proposed a method to design and implement context-aware autonomous services. In their method, variability models represented by feature models are used as adaptation policies to generate automatically execution plans and reconfigure service compositions. Mohabbati et al. [S57] and Kaviani et al. [S44] proposed a method for adaptable service composition in the context of ubiquitous computing by leveraging SPLE principals of variability modeling. In those two methods, feature models annotated by domain knowledge were used as decision models for run-time service composition and adaptation according to functional capability supported by devices in environments. Marinho et al. [S53] adopted feature-oriented approach based on SPLE principles to achieve dynamic adaptation and reconfiguration in the development of mobile and context-aware applications.

In [S8], an approach was proposed to integrate SPLE principles and rule-based business process management with architectural point of view being the main idea behind integration and proposed to enhance run-time adaptability. Their approach was set up to support dynamic composition of feature aspects. In their proposed framework, rules and rule sets are expressed in terms of dynamic aspect and delegated facts, which are implemented with adaptive object model. Allsteinsen et al. [S37] proposed an approach to building adaptive service-oriented applications based on SPLE principles through the employment of explicit
variability models and product line architectures at run-time to deal with complexity of adaptation by an adaptation platform. His proposed platform provided a conceptual model and reference architecture for the adaptive applications. In the work of [S24], Clotet et al. presented an approach that integrated goal-oriented modeling and variability modeling techniques to support run-time monitoring and adaptation and argued that adaptive systems can’t be fully pre-specified because of variabilities and changes in users’ requirements. Therefore, design and modeling techniques are required to describe evolvable system to systematically specify adaptation rules as far as possible in a declarative way.

2.3.3.9 Contribution Facets

Figure 2.7 depicts a map of studies with respect to research foci, which is distributed over research contribution and type (RQ3) according to classification scheme. The vertical axis describes the research focus, and the left and right horizontal axis shows types and contributions respectively. The scatterplot represents the interconnected frequencies in each class intersections. The size of circles is proportional to the number of studies within the pair of classes. The number of studies on each side may differ, in view of the fact that some studies provide multiple contributions.

![Figure 2.7: Map of research foci with respect to contribution and research types.](image)

Different types of contribution are made by publications in this study. We categorized following classes with respect to study descriptions: Metrics, Tooling support, Method, Model,
Language, Open items, Analytical discussion, and Comparison analysis. Metrics concentrates on what to measure to characterize certain properties of the SO and SPLE integration. Tooling support refers to any kind of tools supporting integration of SO and SPLE, with tools mainly including research prototypes. Method includes descriptions of two processes: (1) a proposes approach and/or technique; and (2) the method of performing the integration. It may sometimes also involve a third process: the adaptation of one paradigm into another. The Models and Languages categories include any modeling methods or representation of information to be used by adopting existing languages or by introducing new languages to be used in the integration of SO and SPLE. Open items typically discuss open challenges and issues for integration of SO and SPLE. Analytical discussion includes papers which provided discussions or theoretical models; or taxonomies or categorization of concepts. Comparison analysis includes papers whose research is based on either a survey or categorization and comparison of existing approaches available in the literature.

The classification of research types is based on [330]. Research types are independent from specific focus areas and are categorized into the following classes [249, 330]: (1) Evaluation research, which assesses a problem or implemented solution in practice and includes methods such as case study or field experiments; (2) Validation research, which focuses on investigating and proposing a novel technique or a solution not yet implemented in practices. Research is performed systematically using methods such as rigorous analysis, experiments, and simulations; (3) Solution proposal, which can be either a novel or significant extension of an existing approach or technique. In this category, pros and cons of research results are typically exemplified and critically discussed; (4) Conceptual proposal portrays new perspectives of existing phenomena and structuring the field in form of a taxonomy or a conceptual framework; (5) Opinion-oriented papers report authors positions on certain approaches; and (6) Experience reports offer insights in experience gained from one or more real-life projects.

2.3.4 Threats to Validity

On account of the fact that there are some potential threats to the validity of the present mapping study and its results, the validity of the results from the mapping study has to be evaluated. Owing to the fact that subjective measurements are involved in the process of searching and collection, selecting studies, defining classification scheme, data extraction, and data analysis in the course of the review, we need to address aspects of the validity and
limitations of this study. To validate the credibility of the results, three types of threats should be discussed: (1) **construct validity** refers what extent the inferences can be made and to what is investigated with respect to the research questions under study; (2) **internal validity** focuses on design and enactment of the study; and (3) **external validity** refers to the extent which the effects observed in the study are applicable to outside of the context of the study and can be generalized.

In terms of construct validity, the research questions and objectives of the present study is explicitly defined in the review protocol, which helps achieve the same interpretations for other researchers to replicate the review if needed in the future. To formulate research questions and to derive the search strings, we took into account PICOC criteria, as recommended by [171] and adopted commonly in other mapping studies in different disciplines. We selected the digital libraries that include a very large set of publications in the software engineering fields. Furthermore, we collected relevant conference proceedings and journals as well as sources in which studies concerning service computing and software product lines are normally published. We defined and refined search queries based on the obtained results to maximize the selection of relevant papers for mapping study. In addition, we considered synonyms and the lexeme of the words. We applied the procedure which may be applied by the other researchers.

Moreover, another aspect of construct validity is to ascertain that all the relevant publications on the selected topic are included. Therefore, we incorporated as many studies as possible, excluding “gray publications” such as short papers, works in progress, unpublished, or unpreviewed literature [171] with the possibility of having left out some relevant studies; thus, we don’t guarantee that all the primary studies have been covered.

One way to address both internal and external validity threats is to replicate the mapping study. In terms of the internal validity, the main limitation is increase of bias which is due to the fact that synthesizing the extracted data is quite subjective. However, to mitigate threats, we defined and revised a review protocol given in Section 2.3.1.1.

Another source of threats is the classification scheme on account of the fact that schemes are subjective with no agreement about which one is the best. Furthermore, the classification is based on the research focus and contributions of studies according to terminology used by authors. That many researchers utilize different terminology to demonstrate their approaches is indicative of lack of standard terminology.

Our classifying related studies with slightly different terminology in the same class was
a measure we took against this threat. Another measure was to assign different classes to
the studies possibly having multiple contributions. It is conceivable that others researchers
may have encountered similar threats with different schemes in general; nonetheless, the
authors verified the consistency of classifications of the first authors and both validated and
cross-checked by other researchers.

Concerning the external validity threats, the result of the systematic mapping study
were considered with respect to approaches given in both SO and SPLE topics. Thus, the
classification presented and the conclusion drawn are only valid in the given context.

2.3.5 Discussion

SO engineering and SPLE have co-existed for a long time and a fair amount of research
has been dedicated to each area by two different research communities. The research trend
indicates that the combination and integration of both concepts, principles, and approaches
constitutes a new research direction [69, 73, 74, 109, 183, 184, 192]. To the best of our
knowledge, there has been no mapping study in both SO and SPLE topics. Therefore, we
carried out a systematic mapping study to provide a classification of research objectives and
an overview of the existing studies that have investigated the integration of SO and SPLE
principles, methods and approaches. The collected data presented indicates a growth in the
number of works published in this area in the coming year, as illustrated by the trendline in Figure 2.5.

The results of systematic mapping study enabled us to identify what research topics
were pursued in the available literature. The primary research objectives and foci of the
identified studies in this field have been on service variability modeling, service reuse, ser-
vice identification, service configuration and customization, dynamic software product line,
and adaptive systems. The thematic analysis of mapping study shows that the majority of
studies focused on service variability modeling (38%) and adaptive systems (19%) by apply-
ing SPLE principles and approaches. Moreover, the majority of proposed approaches had
been pursued the advantage of feature-oriented and methodologies introduced by SPLE. We
identified that this body of research works aims at achieving resilient and effective modeling
of service-oriented systems, supporting configurability in dynamic and adaptive systems,
and reducing the system complexity. Whereas, SOAs are adopted in the SPLE context to
reconcile the flexibility, scalability and dynamism in product line deviations and creating
DSPL.
Furthermore, the classification scheme, which was applied in this study, paved the way for us to infer that the majority of studies are solution proposals (41.4%) and conceptual proposals (23.4%) with the primary focus on variability modeling and management. Some research foci present a relevant amount of entries in this mapping study; however, evaluation research and validation are weakly addressed in these studies. Our observation has pinpointed the shortcoming as caused by the recent research literature being in preliminary state or proposing some ways to integrate and combine SO and SPLE. However, we believe there is need for the evaluation and validation of approaches, for instance, in industrial settings from a practitioners point of view.

The mapping study indicates that the combination of SO and SPLE are promising to help develop architectures for adaptive systems in responding effectively to dynamic functional and non-functional requirements, where the advantages stems from SO principles. Furthermore, it enables architectures to be reused in different instances, and supports variability managements, configuration and customization, where benefit stem from SPLE principles. In addition, dynamic software-product lines have been presented as a new direction in SPLE to deal with run-time adaptation to changing requirements and evolving context at different binding times. Hence, SOAs can be utilized to implements software core assets of product lines as services. From this perspective, dynamic behavior of service can reconcile dynamic characteristic into SPL architecture to support DSPLs. It is noticeable that the research community with even specific venues, such as the Dynamic Software Product Line (DSPL) workshop at SPLC, has begun to explore variability management and adaptation at run-time. From the SPLE perspective, this is motivated by the need for more consolidated approaches that address run-time variability. Nevertheless, the results indicated that SPLE approaches for SO are still at an early stage and gaining maturity.

Our observation reveals that the identified studies mostly proposed a method (46.7%) or model (21.6%). There is lack of tooling supports with only 9% of included studies reporting some form of tooling supports. Similarly, we realize that research foci on service configuration-customization management and adaptive systems are topics with more entries than the other research foci with the distribution across the solution and conceptual proposals.
Chapter 3

Service-Oriented Software Product Line Design and Development

3.1 Introduction

Nowadays enterprises and companies deal with several challenges for developing SOA-based solutions. To stay relevant with the global competition, they need to rapidly and cost-effectively develop and deploy consumer-tailored services to meet a wide variety of demands of their particular domain or targeted market sectors within a particular domain. These challenges motivate enterprises to shift from mass software production to mass software customization. A trend towards developing software applications composed of reusable software assets for different requirement sets. To enable mass customization in the context of Service-Oriented Architectures (SOAs), we need innovative software engineering methods and models: 1) to capture the knowledge of variable requirements and reflect variability of services 2) to support the reuse of services and all other software development assets 3) to enable customizing and managing of services according to stakeholders’ functional and non-functional requirement [27, 72, 184].

Software Product Line Engineering (SPLE) is one of the most promising and well-established paradigms focusing on the development of software product lines [71, 250] based on the principles of variability modeling and mass-customization. SPLE research has proposed numerous approaches and techniques for the efficient production of similar software
systems (software families). Hence, the adaptation of SPLE approaches for mass customization is the center of attention now and has already been applied successfully to many enterprises [210]. Employing SPLE techniques results in the reduction of cost, effort, time-to-market and the improvement of quality, together decreasing the complexity of design, facilitating and expediting the customization, maintenance, and evolution of software [71, 234, 250].

SPLE offers promising prospects to provide scalable solutions to the current challenges of the development, management and customization of Web services and generally SOA-based systems [72, 74, 183, 184] to which we refer as Service-Oriented Product Lines (SOSPLs). In this chapter, we provide a comparison of SPL and SOA from different perspectives then present a method for a systematic development of a service family. The underlying idea of which is to guide the development process of an SOSPL and which extends the conventional SPLE life-cycle to support modeling, developing and managing variant-rich service-oriented applications.

This chapter is organized as follows: Section 3.2 reviews the SPL development and outlines some of the main SPLE activities. Section 3.3 presents a holistic comparison of SPL and SOA by focusing on reuse, architectural and variability aspects of the two paradigms. A case study is presented in Section 3.4 to show the concepts and approaches. Section 3.5 introduces a methodology for SOSPL development by focusing on the main engineering activities of the approach. Section 3.6 presents a discussion, and Section 3.7 provides a summary.

3.2 Software Product Line Development

SPLE addresses the issues of software reuse and mass-customization. An SPL or a software product family is defined as: “a set of software-intensive systems, sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way” [71]. The ‘particular market segment’ refers to a domain (i.e., a business area) and the business strategies of an enterprise or organization whose objectives of the business area are determined with changes in its stakeholders’ requirements in mind.

A key idea in SPLE is to capture the essential concepts of ‘commonality’ and ‘variability’ among a set of similar software products belonging to the same domain. Therefore, rather
than describing a single software system, the model of software product lines describes the set of products in the same domain. A product line includes predicted variations that are introduced by tailoring the core assets using variation mechanisms. Variability introduced in SPLE is an abstraction that enables and facilitates customization. It empowers product derivation of different applications by explicit modeling and management of variation points which define decision points determining how the product family members may differ from each other [250, 319]. Variations along with their possible choices, functions or qualities, can be defined at each level of abstraction (e.g. requirements, architecture, or components).

SPLE relies on a fundamental distinction of development for reuse and development with reuse with aims at maximizing reusability and eliminating wasteful generic development of components used only once. This insight can be leveraged to improve software development life-cycle that SPLE shifts from the development of a specific application or individual system to a domain, in turn, leads to two characterized development processes commonly referred to as domain engineering and application engineering [71, 250].

SPLE shifts from the development of a specific application or individual system to a domain. This, in turn, leads to two characterized development processes which are commonly referred to as domain engineering and application engineering. Domain engineering models variability among product family members and develops the reusable software platform by focusing on developing-for-reuse. The software platform encompasses all software development artifacts that are liable to be reused. On the other hand, application engineering adopts the developing-with-reuse approach, where products are customized and derived from product family and reference platform which is constructed and developed in the domain engineering phase. Reuse of the software platform and binding variability for different applications are then enacted in application engineering. Differentiating these two development lifecycles allows for establishing the software platform, application customization, and product derivation.

Approaches to the analysis and construction of SPLs can be classified into three strategies: i) proactive, ii) reactive, and iii) extractive [106]. A proactive strategy is similar to the waterfall approach in conventional software engineering, where all product variations on the foreseeable horizon are analyzed and designed, while architectures for the target domain are defined and implemented upfront. This approach is appropriate for enterprises to foresee and plan ahead of their product line requirements well and that have available resources and time for a long development cycle. A reactive strategy is an incremental approach
where only those product-line reusable assets needed in immediate terms are developed and built. Hence, this approach typically requires less upfront effort than the proactive one. In a reactive strategy, one or several variations of software products can be analyzed, designed and implemented in each development spiral. Such an approach is appropriate where the upfront requirements for product variations cannot be predicted well in advance or where enterprises have to meet tight schedules, which is usually limited in resources, through the transition to an SPL approach. An extractive strategy is between proactive and reactive ones and reuses existing software products as the product line initial baseline.

3.3 Comparison of SPL and SOA

SPL and conventional SOA-based approaches to software development share common goals with both promoting the concepts of reuse and foster organizations to recycle existing assets and capabilities rather than repeatedly redeveloping them for new software systems. Recent years have witnessed growth of research in the exploration of synergies of the combination of SPL and SOA [62, 72, 74, 108, 138, 184, 334]. Even though two paradigms support software reuse, there are different perspectives and outlooks [192]. In this section, commonalities and differences corresponding to the two paradigms are discussed to show how SPL can be adopted and leveraged for the development and customization of a family of SOA-based applications. To compare SPL and SOA, we consider four main aspects including development processes, reusability notions, architectural styles, and variability modeling and management.

3.3.1 Development Processes

SPL and SOA follow different engineering goals. Thus, the activities associated with their software development life-cycles are different. One of the main objectives of SPLs is to reduce the overall engineering efforts required to produce a set of similar software applications by capitalizing on the commonality and by managing the knowledge of variability and customization. Therefore, the engineering goal of SPL is the systematic development and management of core assets and software platform in order to achieve a high level of reusability [71, 106, 234, 250]. In contrast, service-oriented approaches set the goal of achieving system agility and of enabling automation to cope with integration, interoperability and dynamic execution in heterogeneous environments, and providing run-time flexibility [43, 99].
Table 3.1 shows a summary of major life-cycle phases of the two paradigms essentially including requirement and domain analysis, design and implementation and deployment.

Table 3.1: Comparison of the major engineering activities of software product line engineering and service orientation

<table>
<thead>
<tr>
<th>Engineering Paradigm</th>
<th>Requirement Analysis</th>
<th>Design and Implementation</th>
<th>Deployment and Maintenance</th>
<th>Main Engineering goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service-Oriented Software Engineering</td>
<td>• Planning and requirement analysis</td>
<td>• Business process specifications</td>
<td>• Service publishing</td>
<td>• Integration and Interoperability</td>
</tr>
<tr>
<td></td>
<td>• Business process models</td>
<td>• Service construction</td>
<td>• Service matching</td>
<td>• System agility through run-time flexibility</td>
</tr>
<tr>
<td></td>
<td>• Service identification</td>
<td></td>
<td>• Execution and monitoring</td>
<td>• Dynamic execution</td>
</tr>
<tr>
<td>Domain Engineering for reuse</td>
<td>• Product line scoping</td>
<td>• Domain design</td>
<td>• Product line maintenance evolution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Product line requirement analysis</td>
<td>• Domain realization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Variability analysis</td>
<td>• Domain testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Product Line Engineering</td>
<td>• Application requirement analysis</td>
<td>• Application design</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Application realization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Application Testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application Engineering</td>
<td></td>
<td>• Application deployment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development for reuse</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Requirement and Domain Analysis**: Service-oriented design and development are basically based on an iterative and incremental process. The process is initiated with planning proportional to the requirements which, for a new application, are investigated in analysis phase. This process comprises of reviewing business goals and objectives that derive the modeling and development of business processes. Business processes and services are identified in a stepwise manner in the analysis phase with the main objective of which is to facilitate the reuse (or reproposing) of the business process functionality through the identification and orchestration of services when constructing new applications. The requirement-analysis phase in SPLE also consists of determining the requirement and using domain information. Nevertheless, SPLE focuses on the analysis and specification of requisites for the entire product family (i.e., product line). To this end, domain engineering of SPLE mainly concentrates on a systematic analysis and the settlement of variability of both functional and non-functional (quality) requirements performed by scoping the product line, analysing product line requirements, and identifying commonalities and variabilities among product line members. Requirement analysis in the application
engineering life-cycle further focuses on the analysis and determination of prerequisites of individual stakeholders. In the application engineering life-cycle, requirement analysis is established for configuring reusable software assets developed and produced in the domain engineering life-cycle.

- **Design and Implementation:** Service-oriented design and implementation is followed by the design and specification of business processes and service components corresponding to the requirements. Service implementation and testing involves the discovery of available services in local or remote repositories and the development of services, using the specifications in the design phase. In SPL, domain design and implementation involve the detailed design and realizing the reusable software components for the entire product family. It starts with the domain design sub-process which consists of 1) defining and modeling the commonality and variability based on the domain-specific requirements identified in the requirement-engineering phase; 2) specifying the product-family reference architecture which provides a common, high-level structure for all the product-line applications. Furthermore, the domain design incorporates configuration mechanisms into the reference architecture for supporting variability management in order to enable further product customization and derivation. The domain realization sub-process focuses on the implementation and the testing of each component planned and designed for the reuse in different contexts (i.e., the applications of the product line). The application design sub-process in the application design sub-process in the life-cycle of application engineering incorporated application specific adaptation and employs the reference architecture to refine and instantiate the application architecture. Afterward, the application realization sub-process focuses on the selection and configuration of reusable software components and testing for specific application- already contained product line architecture developed in domain engineering phase.

- **Deployment and Maintenance:** Service-oriented development deals with packaging, provisioning, publishing services, service-matching based on requirements of stakeholders, executing stakeholders-acceptance testing, and monitoring performance in the production environment. The development phase including the
configuration and deployment of a final product is associated with application engineering with activities for custom-building systems according to the result of domain engineering.

3.3.2 Reuse in SPL versus SOA

Software reuse, as one of the important goals in software engineering, can improve the quality and productivity of software development. For this purpose, several software reuse approaches have been devised. Component-based software engineering (CBSE) facilitates software reuse and promotes quality and productivity.

CBSE focuses on the interoperability, reusability, and extensibility to facilitate fast delivery of scalable, evolving software systems with research on SOA being a modern instance of this vision, leveraging a logical framework by decoupling several logical units of functionality (services) [179]. This logical framework yields facility reusability through obviating the recreation of common services, thus the achievement of business goals by way of loosely connected services with their variability guided by SOA policies [43, 246].

Reuse in SPL vs. services in SOA have different characteristics (cf. Table 3.2). As mentioned, reusable assets in SPL encompass all the reusable software artifacts. A core asset is the most essential element of SPL since it is a common asset which is reused within multiple products and the reusability of which will largely determine the success of the whole product line [250]. For instance, the most distinguishable reusable assets in SPL context are as follows [71]:

- **Analysis and design models**: including the requirements and variability models, which describe the common and variable features for all family members
- **Domain models**: describing and representing all the entities and concepts that can be utilized in the context of software product families
- **Architectures**: specifying and determining which of the reusable components are needed for configuring executable applications and how to configure software families that best satisfy non-functional requirements
- **Design decision models**: specifying the family configuration model and determining how to derive software products based on specific requirements
- **Software components**: supporting variation points and implementing the required functionalities of software families
• **Interfaces**: enabling different implementation of the same functionality

• **Test artifacts**: reusing test plans, test cases and scenarios, and test data

### Table 3.2: Reuse in SPL and SOA

<table>
<thead>
<tr>
<th>Reuse Characteristic</th>
<th>Reuse in SPL</th>
<th>Reuse in SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuse Units</td>
<td>Analysis and Design Models (Requirement Models), Domain Models, Architectures, Decision Models, Software Components, Composition Models, Interfaces, Test Cases, Documentations</td>
<td>Service, Business Processes or Collaboration Templates, Application templates, Data Schema and Data Provenance, Policies and Business Rules, Test Scripts, Interfaces</td>
</tr>
<tr>
<td>Reuse Context</td>
<td>Software Family Members</td>
<td>Various Contexts</td>
</tr>
<tr>
<td>Coupling With Reuse</td>
<td>Tightly Coupled</td>
<td>Loosely Coupled</td>
</tr>
<tr>
<td>Reuse Method</td>
<td>Instantiation</td>
<td>Service Invocation Composition Invocation</td>
</tr>
</tbody>
</table>

On the other hand, in SOA, services are intended to be reusable building blocks and units of sharable software assets for different applications which implement different business processes. As a consequence, services can be orchestrated to construct composite services through business processes.

Core assets in SPL including a generic architecture and components develop applications whereas services are basic building blocks to support compositional software development in SOA in which business processes or application templates specify entire application through the definition of execution sequences of valid workflows. Services can be reusable artifacts which enable rapid SOA application development [313].

Assets and applications are generally tightly coupled in SPL, while services are loosely coupled which is one of the most pronounced properties of services in SOA research [244]. Services are highly independent of context and the state of other services.

Software components often operate within a context defined by a generic architecture for product family members in SPLs. SOA is grounded on the idea of open integration of business processes by means of shared services where services are described through standard-interface and are intended for reuse in different contexts. Nevertheless, services can also be developed and reused for internal processes within organizations. In essence, SOA basically envisages and focuses on large scale reuse [145] because SOA promotes services to be seamlessly consumed by diverse applications where they can be published, discovered
and invoked through standardized specifications [43]. Unlike core assets, services can be reconfigured at run-time [62].

3.3.3 Architectural aspects of SPL versus SOA

Both SPL and SOA require defining the architectural context and composition rules with SPL architecture often to be centralized, static, and specialized into concrete products but SOA as decentralized. Composition rules are predefined in SPL, which describe common and variable behavioral characteristics of architecture, while in SOA composition or business rules are generally defined to govern the way in which a composition is constructed. SPL basically aims at providing a common architecture for reuse, whereas SOA lacks enough support for large grained software reuse at the architectural level. Gomma et al. [118, 299] discuss software architectural issues in SOA and describe various practices to develop reusable services in order to craft and compose systems from services efficiently. They draw attention to the fact that the architectural solution space offered by SOA promises to provide potentially significant benefits for reutilization. However, achieving SOA’s benefits may not be guaranteed just by implementing based on the SOA solution. Accordingly, the important software architecture and reuse issues should be addressed prior to creating a SOA [299]. Tsai et al. propose a classification schema of architectures for SOA-based applications in order to evaluate variety of architectures [313]. The slackly coupling characteristic and platform-independent view inherited in SOA may address many architectural issues that are open-design and integration problems. Furthermore, architecture style offered by SOA is potential to maximize reuse beside interoperability and flexibility; however, SOA lacks support to manage variability at the architectural level [72, 184] whereas SPL enables managing variability to improve reutilization at such level.

3.3.4 Variability in SPL and SOA

The concept of variability refers to the ability of software systems or artifacts to be efficiently extended, modified, specialized, or configured (customized) for (re)use in the specific context for a particular application [319]. This characteristic enables for applying changes at different levels from software architecture to implementation. Two important concepts related to variability discussed in the literature are variation points and variants with the former being placed in the design or implementation at which variants occur and with the
latter being the alternatives to be selected at those variation points. Therefore, variability can specify a part of an architecture which remains variable, as variation points, or what is not completed at design time. Variability can be implemented at design time or run-time. It is noteworthy that variability and flexibility are closely interconnected. Flexibility offers adaptation and changes of architecture, while variability deals with various version of architecture.

Variability in SPL encompasses all software artifacts from requirements to code. Therefore, there are numerous modeling methods proposed with the objective of modeling variability within software artifacts and at different levels of abstraction. Van Gur et al. discuss about the notion of variability in SPL, where variability is exposed at different levels: platform technologies and user expectations, requirements specifications, designs, component source code, compiled code, linked code, and running code in the context of which variability refers to the ability to select among these artifacts at various stages.

Effective management of variability, which determines how flexibly new members of a given SPL can be obtained and defines SPL boundaries is essential for the success of SPLs. The distinction between variability modeling and other techniques is based on the diversity between variability modeling and variability mechanism. Variability modeling techniques model the variability provided by the product line artifacts while variability mechanisms, several of which have been proposed in the literature such as conditional compilation, patterns, generative programming, macro programming, and aspect-oriented programming are commonly considered ways to introduce or implement variability in those artifacts.

Accordingly, variability in SPL is an essential concern in all phases of development lifecycle. Variability identification, modeling and management is rather a large field of research in SPL. Most current works address identification and management of variability by modeling the concepts as features which considered as the first-class representation of variability and in terms of which the major advantages of discussing a software system is that the concept of feature bridges the gap between the requirements and technical design decisions in view of the fact that software components rarely address a single requirement but rather an entire set of essentials (details are given in Section 3.5). As we discussed in Chapter 2, there are number of well-studied feature-oriented approaches for domain analysis and modeling common and variable requirements in SPLE such as FODA (Feature-Oriented Domain Analysis) and its extension FORM (Feature-Oriented Reuse Method).
RSEB (Reuse-Driven Software Engineering Business) \cite{123}, GPM (Generative Programming Methods) \cite{77} and PLUSS (Product Line Use case modeling for Systems and Software engineering) \cite{98}. Every method generally shares feature as the common concept used in the analyses of commonality and variability. Some approaches are architecture-centric such as Hoek \cite{141}, Koalish \cite{29}, and Thiel \cite{304}. Some of which are configuration-based (e.g., COVAMOF \cite{285} and Koalish \cite{29}). Some of which extend UML to model variability like VPM \cite{325}. Some proposed approaches focus on separating variability representation from the representation of various SPL artifacts such as Bachmann \cite{33}.

The development of SOA-based applications is accomplished through different abstraction layers: business process or orchestration layer, service interface layer, and service implementation or component layer with the business process layer or orchestration consisting of composite services implementing coarse-grained business activities, or even an entire business process \cite{245}. The service layer is composed of self-contained and business-aligned services which provide the implementation for fine-grained business activities. The service interface layer comprises the interface of services published by a service provider. Finally, the component layer (i.e., implementation layer) consists of a set of components that realize service interfaces and provide the implementation for services. Variability in SOA affects these different layers thoroughly. Chang et al. \cite{62} discuss four types of variation points which occur in a general four-layered SOA architecture: workflow variability, composition variability, interface variability, and business logic variability with workflow variability identified as variation of the control flow of a business process, i.e., tasks to be alternatively and optionally completed in a workflow depending on the individual service user, composition variability identified as variability when there is more than one possible service interfaces for activity construct in the business process which implement the service with either different logic or quality attributes and with Interface variability occurring when the candidate services interfaces are different. Finally, components which realize and implement service interfaces by different logic impose logic variability.

**Granularity Levels:** Granularity in SPL refers to the degree of detail and precision of variability as produced by design or implementation artifacts. SPL variability may exist at different levels of granularity ranging from entire components to single lines of code \cite{77,166}. SPLE takes a top-down approach and decomposes artifacts into fine grained artifacts whereas a bottom-up compositional approach is often adopted in SOA to combine artifacts into larger entities- inserting service into composite services (i.e., business processes) that
finally form the application. Decomposition or top-down modeling means that an SPL architecture specifies the decomposition of a family into architectural components. However, there are also hybrid approaches, such as product populations modeled using Koala [239], where the mixture of bottom-up and top-down approaches are leveraged. In SOA, generally there is no particular architecture specifying the decomposition.

In SOA, granularity specifies the scope of variability in functionality exposed by a service. A component which provides an implementation for a service interface can be of various granularity levels that software developers can always encapsulate the entire functionality of a solution into a single service is possible due to the well-known ‘fractal’ nature of services, where a higher-level service can encapsulate lower-level services to any level of granularity [55]. However, a fine-grained service is more easily reused; in distinction, coarse-grained service is more difficult to be reused [245, 299]. Nevertheless, services with high-level interfaces increase the reusability because providing interfaces with a coarse-grained granularity masks specialized or implementation-specific methods, thereby, making a service adoptable and reusable by multiple applications. Moreover, creating and designing high-level, coarse grained interfaces that implement a complete business process is desirable from the perspective of service-oriented design and development [99, 245]. However, there is a trade-off between fine-grained and coarse-grained. Services at different levels of granularity can be generally classified into different categories [180]: basic services, intermediary services, process-centric services and public enterprise services.

Basic services that represent the elements of a vertical domain are simple logic-centric or data-centric services with data-centric services handling persistent data and logic-centric services encapsulate algorithms for complex calculations or business rules and intermediary services designed to bridge a technical gap in architecture. They provide service links with other services or application front-ends and services in gateways, adapters (mapping message formats to enable interoperability), facades (providing a different view on one or more services), and other functionality-adding services (extending functionality of existing services without altering them internally). In SOA, process-centric services control and maintain the state of the enterprises business processes which uses basic or intermediary services to perform task and deal with business data. These services separate process logic from representation layer and encapsulate the process complexity for a single point of administration. A common example is an online shopping process which includes filling the shopping cart, ordering products, and executing billing. Public enterprise services offered...
to partner companies as an in-house-system interface which, in turn, have the granularity of business documents and are coarse-grained integrate enterprises (B2B).

### 3.4 Case Study

To illustrate the concepts and the approach presented in the following sections, we select a part of case study of a family of online marketplace portals providing applications for online trading like eBay. A *portal* provides access to diverse services and content and can be customized and deployed based on different business requirements of targeted stakeholders. In this context, each service is also known as *portlet* which is a “presentation-oriented Web service”. For more details, interested readers are referred to [312] in which a case study on a product lines of portlets is discussed to illustrate feature-oriented model driven development.

Figure 3.1(a) illustrates a service scenario of customer and payment processes – part of a large product family that defines a common framework for online payment provided in online marketplace. For the simplicity, a high-level view of the process is represented, and the details are omitted.

For instance, different methods of online payment can be considered for different instances of service products from a family. Therefore, the number of possible payment method variations of a reference payment process, as a catalog and template, can be derived and configured according to the stakeholders’ requirements and business objectives. Some services are indispensable and prerequisite of the payment process (e.g., Credit Card payment feature as the dominant online payment), which should be included for all the stakeholders’ service product instance whereas some functional services (for example Smart Card e-Check and Debit Card) or extra-functional services (for instance Logging and Monitoring) can be determined as optional that can be included or excluded based on stakeholders’ needs (cf. Figure 3.1(b)). As a case in point, *Stakeholder A* may require additional features for having highly-secured payment transactions, incorporating a fraud protection service, even though this service is not required in the payment process of the final customized portal for *Stakeholder B*. In another scenario, *Stakeholder C* could ask the payment process to

---

2. [https://www.oasis-open.org/committees/wsrp/](https://www.oasis-open.org/committees/wsrp/)
Figure 3.1: a) A holistic view of customer and payment service family. b) Examples of process variants configured for different stakeholders.
be supported by a Mobile-based Notification service in addition to the common payment notification services such as the Mail-based Notification service.

Therefore, in the context of a product family, a business process should be imposed inevitably by variants (optional and mandatory services) which are required to be managed, specialized and customized in order to meet different stakeholders’ functional or quality requirements.

3.5 Applying SPLE to Develop Service-Oriented Software Product Lines

We already mentioned that even though SOA has been widely adopted, there are still no systematic methods to support modeling and managing variability during the development of SOA-based applications and further service management, which calls for a well-defined development process and understanding variability in functional and non-functional requirements in the course of development.

This section outlines the activities of a proactive methodology. The proposed method is an extension of a traditional software product-line life-cycle in order to support development and customization of a family of SOA-based applications. The proposed top-down method follows a two-life-cycle approach that separates two core activities related to Service-Domain Engineering and Service-Application Engineering (cf. Figures 3.2 and 3.5). Service-domain engineering constructs and evolves the reuse infrastructure by analyzing the requirements and scoping the product line as a whole and producing any common, reusable business processes and services. On the other hand, service-application engineering derives individual services (i.e., customized services) from the reference architecture. Domain and application engineering life-cycles can rely on fundamentally different processes, namely, plan-driven and agile methods. In the following, we describe the major activities and their artifacts for three major development phases: (1) analysis (requirement engineering), (2) design, and (3) implementation and testing.

3.5.1 Service-Domain Engineering

The overall service-domain engineering processes of an SOSPL is depicted in Figure 3.2. These activities (D1-D6 in Figure 3.2) are performed iteratively.
Domain analysis in SOSPL mainly encompasses product-line requirements engineering stage (D1) along with the analysis of variability by using feature modeling (D2). A feature model, as software artifact outcome of the feature modeling process, include the knowledge of variability associated with the functional and non-functional requirements and describes the permissible configuration space further guiding the customization process and determining how the reference business process model should be tailored according to the stakeholders’ requirements in the application engineering life-cycle. During the domain design phase (D3), a reference business process model (also known as business process family) is designed and constructed for the product line architecture based upon the outcomes of the requirement engineering phase (D1).

Figure 3.2: Service-Domain Engineering life-cycle.

The model mapping (D4) establishes the mapping relationships between the features
within the feature model and the corresponding activities specified within the reference business process model. The activities of the reference business process are delegated to the service(s) in SOSPLs. Inasmuch as non-functional (quality) requirements may also vary for different stakeholders, variability in the quality properties of services should also be captured and specified during the construction of an SOSPL (D5). To this end, features in the feature model are annotated by quality ranges which are supported by the entire product line architecture progressively helping service engineer and developers to evaluate the impact of variant features selected according to the quality characteristics that services provide [223].

In the final phase, the reference business process model is realized and implemented either by binding to the existing services or by developing new services. In the following, we detail these activities.

3.5.1.1 Product Line Requirements Analysis

Similar to traditional requirements engineering, domain requirements engineering should at least include the following activities [292]: 1) elicitation, in which the product line business goals and stakeholders’ requirements are discovered and scoped; 2) specifications, in which the requirements are analyzed in detail; 3) validation, in which the requirements are validated and consistency and completeness are checked, and 4) management, in which the requirements can be managed in terms of changes or refinements. In addition to these activities, domain requirements engineering captures commonality and variability between the requirements of several stakeholders. Moreover, an important activity of the requirements analysis of an SOSPL is to define the product line scope [72, 234, 250] and decide on the boundary of the product line.

A successful scoping which is determined by factors such as the knowledge of similar domain services and the future demands of stakeholders is required to be performed carefully because a scope – either too large or too small – will impair the capability of a SOSPL in achieving the goals of stakeholders [71]. Goal-oriented domain analysis can be employed at an early stage of the requirement analysis in order to capture the product line goals for requirement elicitation and to further align the final service products with the business goals and intentions of both the stakeholders and service providers. This is accomplished at different levels of abstraction by goal modeling about which interested readers can further read in [28]. The outcome of this phase is the requirements models which can be described by goal models, use-cases, documentations and details, which are used subsequently for the
variability analysis of the product line under development.

3.5.1.2 Variability Analysis and Modeling

The product line requirement engineering activity follows the variability analysis and modeling of the entire family in order to identify common and variable features. A feature is commonly defined as a visible incremental functionality and quality in software system(s) [161]. Nevertheless, depending on the stage of development it may also refer to a requirement or a coarse-grained or fine-grained component in the system(s) which provide the required functionality from different technical views. The emphasis in the variability (i.e., feature) analysis is on optional features because optional features substantially differentiating one member of the family from the others.

In SOA, services constituting the orthogonal concept to the components notion, encapsulating functionality, and providing individual non-functional properties (i.e., QoS) through a well-described and published interfaces are characterized as building blocks of software that are loosely-coupled. From this view in the context of SOSPL, we define a feature as an increment in service functionality [20], which reflects stakeholders’ both functional and non-functional requirements. Therefore, a feature, based on the granularity levels, can be realized and associated to a composite service at the high-level business processes, or associated and realized by an atomic service at the lower-level.

Feature-oriented development [77, 161, 163] is widely employed as a means for analysis, management, and visualization of commonality and variability in SPL in terms of features at different abstraction levels. Feature modeling essentially organizes features of a software product family into a model called feature model residing between the requirement model and the design specification model (i.e., the reference business process model described in Section 3.5.1.3).

Figure 3.3 shows a part of a feature model representing the variants (namely optional and alternative features) that characterize a requirements model. These features are selected to derive service products during the application engineering. Moreover, this model serves as a catalog of the variability space offered by a product family to accommodate the idiosyncrasies of the stakeholder enterprise or company.

**Feature Model:** A feature model consists of both formal semantics and graphical representation (e.g., feature diagram) and encompasses the knowledge of configuration for a
product line. A feature model is a hierarchical decomposition of features in terms of parent-child relations on different levels of abstraction. A feature diagram as a tree-like structure represents SPL variants, where fine-grained configuration option for the product-line variant corresponds to each successively deeper level in the tree as shown in Figure 3.3.

![Feature model of e-Payment (conforming stakeholders’ requirements model).](image)

Some of the features are not assumed to be present in every product during application engineering, this differentiation is expressed by the indication of feature types and their relationships. Contrary to a mandatory feature always being selected if its parent is selected, an optional feature may or may not be selected. For instance, in Figure 3.3, all the products should include the Credit Card feature as a mandatory feature. Other payment methods are specified as optional features.

A feature cardinality and group cardinality can also be determined in cardinality-based feature modeling. A cardinality defined as an interval, from zero to a given value and associated to a feature determines the lower and upper bound of the number of features required in any product in a product family. In the SOSPL context, this attribute specifies the number of service instances that should be linked to a given service at run time.

Or feature groups with defined cardinality indicate that at least k and at most k’ features that can be included out of the n features (k ≤ k’ ≤ n) in a group if the parent is selected. Moreover, Alternative feature groups with specified cardinality indicate that only k out of n features in the group must be included if the parent is selected. Back to the simplified feature model example from Figure 3.3 all the products should include the Payment

CHAPTER 3. SOSPL DESIGN AND DEVELOPMENT

Notification features. Also, all the final derived service products should include at least two methods of notification according to the feature model.

Furthermore, because features are not always independent, integrity constraints (i.e., the includes and excludes) can be defined over features of a feature model to model dependencies and relations which can exist among them. They are the means to describe that the presence of a certain feature in the product imposes the presence or exclusion of another feature (cf. Figure 3.3).

Feature models are an efficient abstraction of variability derived from the domain and stakeholders’ requirements. They also help derive the design and the development of variability through all the stages of the development including service identification and design, and further customization [72, 224].

In the feature-oriented analysis phase, which subsequently guides the identification of candidate services with right granularity, we organize feature based on the following criteria:

- Features supporting a particular business process can be grouped and abstracted as a higher-level feature on a coarse-grained level (e.g., Payment);
- Features supporting specific functional or non-functional services can be grouped and abstracted as a higher-level feature (e.g., Payment Notification and Logging services);
- A feature which incrementally realizes a feature at the upper-level, subsequently becomes as a sub-feature at the lower-level;
- Features at the leaf-level are realized by fine-grained services.

3.5.1.3 Reference Business Process Model

The previous activities, domain decomposition, and top-down variability analysis and modeling provide an insight into product features of a target domain. A feature model is generated as an output of the domain analysis. This model, which is generated as an output of the domain analysis, is then used to derive reference architecture and develop reusable components (business processes and services) in the course of the domain design. The design phase produces an architecture-interdependent model defining reference architecture as the behavioral model of features for the entire family and specifies how features are composed at run-time.
A template-based approach has been widely adopted in SPLE to create the reference model. In the context of SOSPLs, such a reference model is designed as a template for the entire service products family in a superimposed way \[78\]. A reference business process model, as a model template, describes and specifies the execution sequence of services for all instances of the product line. That is, a reference business process model is a union of all the business processes of the product line and provides the common business logic for orchestration and choreography of services, which implement features. The reference model comprises functional interfaces specifying services capabilities, pre and post conditions of the services, and configuration properties representing the data needed to configure a service before its use, and service bindings.

In practice, several languages have emerged to model and execute business processes such as Business Process Modeling Notation (BPMN) \[238\], and Business Process Execution Language (BPEL)/WS-BPEL, to name a few. Thus, the reference business process model can be modeled by using such process-oriented modeling languages, and incrementally refined and optimized. For example, Figure 3.1(a) illustrates a part of reference business process model, where variability and configuration knowledge have been modeled and encapsulated in given feature model depicted in Figure 3.3.

The reference business process model configured through the selection/elimination of features from the feature model during the application engineering and executive instances are derived (cf. Figure 3.1(b)). In other words, according to the fact that architectural variations in the reference model are encoded as features, various parts of the reference business process model are organized in variation points. These variation points are managed and configured by means of feature models. To point out the differences between design and run-time variability, it should be mentioned that feature models capture and encapsulate only architectural variability at design time. In contrast, business process models describe behavioral variability, i.e., how features are composed, which drives run-time variability through composition patterns (discussed in the next section).

Furthermore, feature model configuration (i.e., specialization and customization) is performed during the build time. The configuration can be done through the process of staged-configuration \[80\] where features further are prioritized and selected according to the (non-) functional requirements of the stakeholders \[224\]. All configured service products, which are instances of the family, have to conform to the reference architecture.
3.5.1.4 Feature Resolution and Mapping Model

During the design phase, feature resolution is the analysis and connection of the feature model and the reference business process model in order to specify explicit mapping links between the two models: feature and reference business process model, the outcome of which constitutes a mapping model including links between features in the feature model activities in the reference business model. This mapping model enables configuration of the reference business process model through feature section during application engineering. From a certain point of view, this mapping model also provides the traceability links between the requirements and implementation [78, 290].

Our mapping model is based on the annotation-based approach proposed by Czarnecki et al. [78]. In this approach, model elements (i.e., process activities) can be annotated using presence conditions (PCs) and meta-expressions (MEs). A PC indicates if the model element it refers to should be present or be removed, and MEs are used to compute attributes of model elements relevant to the language expressing the model, for instance, the name of an action in the UML ADs. Both PCs and MEs are expressed by boolean expressions over the features and feature attributes of a feature model, and are evaluated against a feature selection (configuration). These expressions can be represented in disjunctive normal form or as XPath expressions.

Our mapping model consists of boolean expression specifying presence or removal of a modeling element (e.g., activity (abstract service)) in a reference business process model based on the selection of features in the feature model [78]. In our approach, we consider a boolean variable $\psi_i$ corresponding to each feature $f_i$. Our model uses PC as annotation properties for each activity within the reference business process model. The PC of an activity is formulated as a boolean expression of $\psi_i$ variables corresponding to the features mapped to the activity (cf. Figure 3.3 and 3.4). Both the feature and activity constructs refer to model elements of feature models and reference business process models. Thereby, when domain engineers map features to activities, the PCs of activities are defined. In application engineering, reference process configuration is achieved by evaluating PCs and MEs against a feature selection. For example, when a feature $f_i$ is removed from the configuration, its corresponding $\psi$ variables are set to false. Consequently, those business process fragments whose PCs evaluate to false are removed from the reference process model, while model attributes that are affected by MEs change accordingly (e.g., the name of
activity is changed).

Feature resolution also helps identify cross-cutting concerns related to general non-functional requirements. For example, feature Monitoring with given mapping annotation $\psi_i$ in Figure 3.3 is mapped to activity Monitoring as an extra-functional abstract service in the reference business process model (cf. Figure 3.4). Based on the selection of features from the feature model in application engineering, the reference business process model is configured (Figure 3.1(b)).

![Diagram](image_url)

Figure 3.4: A part of reference business process model and associated mappings.

### 3.5.1.5 Non-functional Specifications

The domain design phase is followed by the specification of non-functional properties based on the non-functional requirements (NFRs) analysis because NFRs are interlaced and related to functional requirements. Variability in NFRs influences the SOSPL design and implementation. Non-functional variations often exhibit different types and levels of quality properties (e.g., normal and strong authentication or security). For instance, NFRs for feature Credit Card can include cost, security, availability and reliability or they can also entail defined domain-specific non-functional aspects such as usability and convenience of use. Furthermore, in application engineering, non-functional variations directly impact the selection of appropriate services from candidate services, all of which provide equivalent functionalities even though with different degrees of non-functional properties related to the service quality specification. To this end, there are a number of proposals [82], in which feature models are extended to support feature attributes to comprise non-functional properties which can be measured (or to be measured).
In next chapter, we introduce a generic evaluation model and method for aggregation and computation of ranges of quantified values of quality properties defined for a product line architecture.

### 3.5.1.6 Reference Business Process Model Implementation

The domain design phase produces a reference business process model and architecture for a family of service products (i.e, SOSPL). In the domain implementation and realization, the reference business process model is realized and implemented, which involves implementing and testing the detailed architecture of the family modeled by reference model. Abstract services specified by the reference business process model are implemented using component models such as Java class, Enterprise Java Beans, or .Net components. However, some of the services needed for the implementation might already be available; for instance, it can either be found in service catalog or retrieved through a service discovery process, and some of the services could potentially be built by partly reusing or modifying existing solutions.

### 3.5.2 Service-Application Engineering

This section describes a holistic view to the application engineering life-cycle which includes the major phases of service customization and derivation from the business process family. Regardless of the chosen variability modeling approach, the ultimate of in-service-application engineering is to employ the variability defined in domain engineering by selecting shared assets similarly developed in domain engineering. Figure 3.5 depicts a high-level application engineering process which starts with the elicitation and capturing of both the functional and non-functional requirements of an individual stakeholder through the phase of application-requirements analysis (A1). In the application design phase (A2), features are prioritized based on the stakeholder’s captured preferences and business objectives concerning the optional features and quality needs. Thereafter, the feature model is specialized through the decision-making process of selecting optional features. Subsequently, the reference business process model is configured and corresponding services are selected and bound in the deployment and integration phase (A3). The details of these application engineering phases can be found in [224].
CHAPTER 3. SOSPL DESIGN AND DEVELOPMENT

3.5.2.1 Application Analysis

This phase focuses on the elicitation of requirements of a particular stakeholder to derive customized process variants, which can be deployed as the final product. The preferences of the stakeholder are captured and later utilized for feature prioritization and selection. Similar to the requirements engineering phase in service engineering methodologies like SOMA [27], activities in the application analysis phase capture requirements for a single service (application); however, the application analysis activities reuse the family requirements models to develop requirements models of a target service. For example, assuming a family requirement model is represented in a goal model, the service goal model is developed through
reasoning on the family goal model based on the inputs of current stakeholders [28]. Validation and verification of the application requirements model with respect to the stakeholder’s needs and product line constraints are performed. In the context of marketplace portals, stakeholders of a target service application may request payment, shipment, order management, and manage customer functionalities as well as high security and low cost. The detail of stakeholders requirements can be achieved by applying label propagation algorithms on the marketplace family goal model.

3.5.2.2 Application Design and Implementation

During this phase, the feature model is utilized to manage and select variants constituting service product instances. This is accomplished through the feature prioritization and selection of sub-processes. This activity includes the selection of the best and permissible combination of optional features along with the selection of the corresponding services that would optimally satisfy the stakeholder’s functional and non-functional requirements. Several automatic or semi-automatic and manual feature model configuration techniques have been proposed to guide the final product configuration (i.e., customization) according to the requirements and preferences of stakeholders. Automatic configuration approaches employ AI optimization techniques such as Genetic Algorithms (GAs), Bayesian Belief Networks, and Constraint Optimization Problem (COP) to create the final customized product [82]. On the other hand, manual configuration techniques through staged-configuration [80] provide specialization steps for service engineers and help them resolve variability in the process of configuration and customization (cf. Section 3.6). After configuring the feature model, due to the established mappings between the feature model and the reference business process model, a concrete business process for a target service-oriented application and its realizations are derived from the family design and implementation models. However, since there may be some requirements which could not be satisfied by existing assets (i.e., services) contained in the developed SOSPL architecture, further refinement of instantiated service products from the reference model can be performed, and new required services can be implemented. In our running example, according to the requirements of the current application derived in the previous stage, application engineers can configure marketplace feature model and derive a business process model for the service-oriented application under development. Also, proper services based on the requested quality of services (e.g., high security and low cost) are selected.
3.5.2.3 Application Deployment

This phase focuses on creating an executable business process, test, and deployment of the configured service in the production environment after validating the customized services against the application requirement specification. After the deployment of the final service product onto the stakeholders’ environment, the execution of the customized services is monitored to ensure the compliance of the service execution to stakeholders’ requirements and any service level agreements.

3.6 Discussion

The development, management and evolution of many modern software systems rely on the notion of variability and suitable design techniques. SPL research has devoted a considerable amount of resources to the development of various approaches to deal with variability analysis, modeling, management, customization and related challenges over the last decade. These approaches can be employed in the design and development of variant-rich service-oriented applications (referred to as SOSPLs in this chapter).

Feature-oriented analysis is a means to create variability in services at different levels of abstraction and to subsequently make the managing and customizing it possible. Variability can be considered in terms of four different general levels of abstractions in service-oriented development [245]: requirements, business process models, service interface model and service component. In that sense, variability at a lower-level of abstraction realizes variability at a higher level. As described earlier, we leveraged feature modeling for managing variability by focusing on the requirements and business processes at the higher levels of abstraction. However, feature models can also be employed to model, represent and manage variability at the levels of service components and interface to support efficient service management. For instance, in [102], Fantinato et al. employed feature modeling to manage and enable customization in service contracts.

Nguyen et al. [229, 230] propose a feature-based service customization framework to model and manage variability of complex Web service specifications. Feature models are used to capture and represent the service variability at requirements levels. The proposed approach employed feature models as an extension to service description artifacts to facilitate the customization of service interfaces and composite services [232]. Stakeholders can configure the service by selecting required features and eliminating irrelevant features, which
result in feature configuration used as the customization requests from which the provider produces appropriate service variants.

Koning et al. [178] investigate how variability can be incorporated into service-based systems in order to enable variability modeling and management. They describe how variability management helps support run-time reconfiguration of systems by service replacement corresponding to the non-functional requirements of stakeholders. VxBPEL is proposed as an extension of BPEL to the process description and definition. Variability information is defined in-line with the process definition. VxBPEL builds upon COVAMOF [285], a framework to model variability. The authors note that the architectural modeling and management of variability in Web service-based systems provides the following advantages: enhancing the extensibility of systems through service replacement, improving run-time flexibility to configure and rebind services (e.g., being able to optimize quality through reconfiguration).

As already mentioned, improving reusability in service-oriented development is an often-stated goal in the literature. There are a number of important concerns that can influence highly-important analysis and design decision for the quality of service design. The major concerns include analysis and design for service reuse, service granularity management, and design of composable service [245]. Hence, it is a challenge to develop service-oriented systems by identifying reusable services at the right level of granularity to facilitate service composition and in turn, attesting that the identification of service candidate is a challenging task in service engineering [27, 245]. SPL approaches can be adopted to consolidate design principles and service identification during the course of service engineering.

Lee et al. [191] present a feature-oriented approach to the analysis, identification and development in order to improve reusability of service-based systems. The proposed approach provides guidelines about how to address the key issue of granularity and orchestration of services by using feature models. They show how reusable service can be identified and specified based on software features. The proposed method is based on analysis of features that may vary from a user’s point of view and will be subject to reconfigurations at runtime. Another approach to using feature-oriented analysis for service identification during the analysis and design phases is proposed by Chen et al. [65]. They focus on re-engineering towards service-oriented systems and the remark of whom that feature-oriented analysis bridges the gap between the abstract architectural and source code level, whereas business processes are excluded.

Service-Oriented Modeling and Architecture (SOMA) proposed by IBM [27] has been
developed as a generic development method for SOA-based applications. SOMA provides the guidelines for identification and specialization of services that realize and implement business processes through service composition. The authors of SOMA remark that variability analysis in the practical SOA solution design is crucial for the initial finding-binding relationships between a service consumer (i.e., stakeholder) and a service provider. Moreover, they noted that the publishing and discovery of relationships are often affected by variations, which are identified later in the design process. Hence, such variations may cause expensive fundamental re-design of SOA-based solutions [27]. To address this problem, the authors remark that a development life-cycle for SOA-based solutions should be extended by a variation-oriented analysis as an extra dimension that should be performed.

3.7 Summary

We observe that the convergence of service-oriented and software product line engineering is gaining a considerable amount of attention and rapidly emerging as a viable and important software development paradigm. As we discussed in this chapter, they both share common goals and promise to collaborate in the development of flexible, cost-effective software systems and to support a high level of reuse. Yet, their main goals are somewhat different. In this chapter, we discussed how service-oriented development can benefit from SPLE approaches for variability modeling and management in the process of identification and design of variant-rich service-oriented applications.

By combining ideas of service-oriented development and SPLE, we expect to derive new software engineering approaches using the best from both paradigms: (a) development of generic software architectures for highly adaptive Web services that can respond effectively to fluctuations in stakeholders’ (non-) functional requirements, and (b) development of shared architectures that could be reused in different instances (benefits from the SPLE principles).
Chapter 4

QoS and Aggregation Models for Service-Oriented Software Product Line Architecture

The Service-Oriented Architecture (SOA) enables the realization of Software-as-a-Service (SaaS). Although various stakeholders (e.g., companies and/or individual software developers) can employ services as the building blocks of their target application, in reality, they often have dissimilar requirements from each other, resulting in the need for consideration of different features in the final product. This, along with the importance of sustaining market changes, necessitates mechanisms for the rapid development of software systems that best meet the stakeholders’ feature needs. Some service providers have already moved towards the adoption of customizable product-development models to efficiently tailor solutions for their stakeholders. Within this process, they need to consider, manage and withstand both variable functional and non-functional (quality) requirements to produce new applications systematically \[72\]. As we discussed, the *Software Product Line Engineering* (SPLE) provides a platform to capture both functional and non-functional aspects and allows for the rapid customization of new products. Several researchers have proposed to integrate SOA and SPLE paradigms into *Service-Oriented Software Product Lines* (SOSPL) as a way to formalize *customizable product-development* and take the benefits and synergies of both paradigms \[72\] \[192\].

Researchers have explored various strategies for the realization of software applications,
e.g., how the most appropriate services can be selected for a given product line and how they can be efficiently composed. However, previous works often fail to consider Quality-of-Service (QoS) in the context of SOSPL and configurable business process models.

From a stakeholder’s point of view, the quality characteristics (attributes) can be mandatory or optional with different levels of importance. Moreover, quality characteristics can be affected by variation in functional features. The impact of functional variable features on quality characteristics is complex and difficult to identify and measure. On account of the fact that one variable feature of a service product family may be influenced by several quality properties required by stakeholder or one quality dimension may be affected by several variable features. It is necessary to identify and measure the diverse impact of functional variable features on a quality attribute, because it plays a key role in assessing the quality characteristics for an SOSPL, which imposes heavy human effort for the assessment.

Furthermore, to evaluate the potential quality of the developed service-oriented applications, it is essential to consider constraints and quality restrictions of individual services and overall requirements. Because some QoS dimensions values may vary during the service life-cycle. The QoS aggregation is a prerequisite for composite service to assure and verify if the QoS expectations of stakeholder are satisfied.

Quality evaluation is a challenging task in monolithic software systems, and is even more complex when it comes to SOSPL, as it requires to analyze the attributes of a family of SOA systems. In SOSPL, architectural quality evaluation is crucial as it allows the examination of whether the final service product satisfies and guarantees the ranges of quality requirements within the envisioned scope of the product line. This chapter contributes a solution to the following open research problem **How can the quality dimensions of a service family be aggregated with respect to architectural variability?** The novelty of our approach is in accounting for variability during architecture quality aggregation, which has not been considered in any related work, to the best of our knowledge. Our work focuses on the development of a framework for computing the quality ranges of features in an SOSPL by aggregating quality properties at the architectural level.

In particular, this chapter addresses the third research question of this thesis (RQ3) and makes the following contributions:

1. The introduction to an extensible multidimensional QoS model that captures non-functional properties inherent to SOSPLs;
2. The introduction and classification of a set of possible variability patterns that occur at the architectural level of a SOSPL. This can be seen as a catalog of patterns for variability that may occur in the structure of business process models expressing composite services;

3. The development of a quality model framework for the aggregation of QoS based on different architectural patterns (structural variability and composition patterns);

4. The formalization of a computational model for architectural quality evaluation, which takes into account both variability and composition patterns and allows for trade-off analysis and architectural decision making between options that provide similar functional properties but different levels of quality.

The rest of this chapter is organized as follows: Section 4.1 offers a running example reviewing and illustrating the problem under study. Section 4.2 describes related conceptual models and formalizes the main concepts used in the proposed approach. The proposed QoS model for SOSPLs is described in Section 4.3. Section 4.4 describes and exemplifies the QoS aggregation and computation method. Section 4.5 presents a discussion and related work. Section 4.6 summarizes the chapter.

4.1 Illustrative Example

As discussed in previous chapters, SPLE has been recognized as a successful approach to variability management and reuse engineering, which enables mass customization, enhances software quality and reduces the time-to-market of new software products. This is achieved since SPLE derives products from a family of software applications using reusable assets. Different software products derived from a software product line are distinguishable based on their included features. A feature reflects the stakeholders’ requirements, and it is an increment in the product functionality.

As described in Chapter 3, SPLE consists of two main life-cycles: Domain Engineering and Application Engineering. Domain Engineering is concerned with the analysis and identification of the scope of the product line and the capturing

\footnote{In particular, we consider that structural variability is captured by feature models and composition patterns are captured by business process models where there is a mapping between elements of feature and business process models.}
of the entire domain of interest through modeling of common and variation points. The application engineering cycle builds the understanding of specific requirements of different stakeholders, for whom the customization and configuration of the product line is carried out. Let us review the case study presented earlier. We focus on the simplified business process model for payment service of our case study in a global online retailer scenario to illustrate the concepts and further discussions in this chapter.

Figure 4.1 (a) illustrates a feature model representing commonalities and variability in the configurable process model of a payment service (cf. Figure 4.1(b)). The feature model focuses on the structural relationships and the configuration dependencies between features, and it is used to distinguish products of a SOSPL and guides the configuration of a reference process model.

*Figure 4.1: An online store product line (a) feature model representing the structural variability of the product line and (b) a business process model representing the design of the Payment feature from the feature model*
As shown in Figure 4.1, different features from the feature model (a) can be used within the business process model (b) to provide different functionalities of the payment service defined as a composite service. These features can be realized using different services that offer the same functionality, but differ in QoS characteristics. The activities represented in the gray color in the process model indicate optional features, i.e., those features can be optionally included or excluded from the target product based on stakeholders’ requirements. For instance, the Notification feature can be included in a final service-oriented software product derived from the SOSPL under study by selecting either the Mobile-based notification, Phone-Fax notification, or Email/Voicemail notification features, which may have different ranges of quality values since they are implemented by different services. Hence, the QoS characteristics of a developed product are closely dependent upon the features included in the final product.

A feature model such as the one in Figure 4.1 (a) is a model of the payment process product line representing, in turn, a configurable SaaS while each variant (configured service) is a member of that family. As already discussed, variation points are considered those places in the design of a configurable process model. However, the options to be selected for a particular service-oriented product to be derived from the SOSPL based on the stakeholders’ requirements are left open for the configuration. Hence, variation points provide the possibility to derive different products, i.e., different final composite services. In SOSPL, particularly during the domain engineering cycle, determining the implied QoS ranges for individual features based on the underlying architecture and implementation helps domain engineers ensure that the product line architecture will fulfill and deliver the lower and upper bounds of the values of quality requirements of the stakeholders; in other words, the aggregation of the QoS characteristics of a feature model according to the quality of its features provides the means to estimate the likely lower and upper bounds of QoS properties for potential service-oriented software products that will be derived from that SOSPL. In addition, in the context of SOSPL, quality-range computation enables for keeping track of the product line quality ranges even after the QoS properties have changed. In Section 4.4 we discuss how these QoS ranges are computed in the presence of variability inherent to SOSPL.

A feature model such as the one in Figure 4.1 (a) is a model of a family of payment process, which in turn represents a configurable SaaS, while each variant (configured service) is a member of that family. As we discussed, variation points are considered those places
in the design of a configurable process model. However, the options to be selected for a particular application with respect to stakeholders’ requirements are left open for the configuration. Hence, variation points provide the possibility to derive different products, i.e., different final composite services. In SOSPL, particularly during the domain engineering cycle, determining the implied QoS ranges for individual features, based on the underlying architecture and implementation, helps domain engineers ensure that the product line architecture will fulfill and deliver the upper and lower bounds or values of quality requirements requested by stakeholders. In other words, the aggregation of the QoS characteristics of a feature model based on the quality of its features, as derived from underlying processes and services implementing those features, provides the means to estimate the likely lower and upper bounds of QoS properties for potential service products that will be derived from that product family. In addition, in the context of SOSPL, quality range computation through the construction of a generic evaluation model enables us to keep track of the product line quality ranges even during or after specifications of the service quality have changed. In next sections, we describe our proposed approach how these QoS ranges are computed in the presence of variability.

4.2 Preliminaries

In this section, we describe some underlying concepts and definitions that are used for the quality analysis and expressing configurable process model in the remainder of this thesis.

4.2.1 Process Model and Specification

As already mentioned, complex software applications can be built out of composite services. A service can be modeled as a software component with a well-defined interface that implements a set of operations offering a piece of functionality. In other words, a process model defines the workflow of a set of ordered activities intended to realize and implement one or a set of goals associated with a feature w.r.t. stakeholders’ requirements. A (business) process model describes composition logic defining a workflow (i.e., control flow) of a set of ordered activities (also called service activities or abstract services) and their transitions which implement features with respect to stakeholders’ requirements. The process models enables services to be composed at the different levels of granularity and dictates how services can be combined, synchronized, and co-ordinated.
We refer to a reference process model as an abstract representation of all valid service compositions in an SOSPL. We assume that a reference process model, as a higher level model of service compositions, is generated by template-based and parametric-design-based approaches or by any languages used to describe the configurable process model \[3, 78, 187\]. A reference process model $PM$ consists of a set of interrelated abstract activities $A = \{a_1, \ldots, a_n\}$ and their data dependencies (inputs, outputs, and pre- and post-conditions). An activity $a_j$ represents a well-defined business function, as a functional abstraction of a service providing an implementation for feature $f_j$. Hence, an activity can be atomic (e.g., atomic service) or non-atomic (composite service). Each activity is delegated to one or more concrete services providing the required functionality with different value range of QoS properties (e.g., cost and performance). A service can be defined formally as follows:

**Definition 1 (Service).** A service is defined as a tuple $s = (I, O, P, E, G, NF)$, with $I/O$ denoting the sets of input and output of service operations, $P$ and $E$ sets of pre- and post-conditions that must hold before and after service execution, respectively, $G$ a set of goals the service realizes, and $NF$ a set of non-functional properties.

Service modeling languages (e.g., SoaML\[^2\], WSML\[^3\]), semantic annotations and ontologies (e.g., SA-REST\[^4\], SAWSDL\[^5\], WSMO\[^6\], and OWL-S\[^7\]) can be employed to model and describe service specifications and supported goals and define non-functional properties which are used in service descriptions and a service requests. Non-functional properties includes context and QoS information. Activities are defined in the same way as services, but no binding to specific service is attached. We formally define a process model as follows:

**Definition 2 (Process Model).** A process model $PM$ is modeled as a directed acyclic graph $G_{PM} = (V, E)$, where $V = \{V_v, V_o, V_A, V_G\}$ denotes a set of disjoint nodes, with $V_v$ and $V_o$ being sets of initial and final nodes, respectively, $V_A$ being a set of activities, and $E \subseteq V \times V$ representing the workflow (transitions) between the nodes, and with $V_G$ being a set of nodes as gateways which have a type $T(V_G)$ such that $T(V_G) \in \{\text{AND, OR, XOR, DISC}\}$.

\[^3\]Web Service Modeling Language: [http://www.w3.org/Submission/WSML](http://www.w3.org/Submission/WSML)
\[^4\]Semantic Annotations for REST: [http://www.w3.org/Submission/SA-REST](http://www.w3.org/Submission/SA-REST)
\[^5\]Semantic Annotations for WSDL: [http://www.w3.org/TR/sawSDL](http://www.w3.org/TR/sawSDL)
\[^6\]Web Service Modeling Ontology: [http://www.w3.org/Submission/WSMO](http://www.w3.org/Submission/WSMO)
\[^7\]Semantic Markup for Web Services: [http://www.w3.org/Submission/OWL-S](http://www.w3.org/Submission/OWL-S)
Workflow patterns are the types of structures and behavior observed to commonly recur in real-world workflows [317]. Workflow patterns are defined in terms of how the process flow proceeds in sequences and splits into branches and how those branches merge while executing process activities.

To model peculiar characteristics of process models and stakeholders’ requirements, the specifications of process model are enriched by annotations in our work. We assume the execution probability of every conditional branching i.e., switch (XOR-split), is specified by annotations; therefore, for each gateway with conditional branching with \( k \) disjoint branches, the sum of probabilities is \( \sum_{i=1}^{k} \rho^b_i = 1 \), where \( \rho^b_i \) indicates the probability of execution of \( i \)th branch. Furthermore, loop constraints in a process model can be defined to express the number of iterations for a particular (group of) activity/ies; hence, the probability distribution of every loop with the maximum number of iterations \( c \) is specified such that \( \sum_{i=0}^{c} \rho^l_i = 1 \), where \( \rho^l_i \) indicates the probability that loop \( l \) is executed \( i \) times. We assume that an upper bound \( c \) for the loop \( l \) is determined. Otherwise, the process further cannot be optimized because infinite resources might be required for execution. The distribution of probabilities of execution of conditional branches and loops is specified during design time or is evaluated based on the past executions captured by the system logs or service brokers [338].

Functional and non-functional constraints over QoS, in the form of global and local constraints, need to be defined and fulfilled. Global constraints stipulate (end-to-end) requirements for the entire process model, and local constraints specify the desired QoS value to be provided for a particular activity (i.e., feature); therefore, each process can be subjected to global constraints \( C_{gc} = \{I_{gc}, O_{gc}, P_{gc}, E_{gc}, NF_{gc}\} \). Each activity associated with a feature can also have a set of constraints \( C_{fc} = \{I_{fc}, O_{fc}, P_{fc}, E_{fc}, NF_{fc}\} \). These constraints represent a multi-set containing input constraints \( I \), output constraints \( O \), pre-condition constraints \( P \), post-condition constraints \( E \), and non-functional and QoS constraints \( NF \). Constraints \( I, O, P, \) and \( E \) particularly set apart restrictions on input/output data of service or the composition itself and are not further considered in this work because they do not play a major role during the QoS-aware optimization process of SOSPL configuration. Constraints can also be expressed and specified by using semantic annotations or any languages,
like as OCL\(^8\), RuleML\(^9\), Slang\(^{10}\) and WSLA\(^{11}\).

To describe a set of functionally equivalent services with different QoS properties (values), we define a service candidate set denoted as \(S_{a_i}\) for each activity \(a_i\) providing an implementation for feature \(f_i\). Given the above consideration, a set of \(\bigcup_{i=1}^{n} S_{a_i}\) includes binding links to services providing multiple implementations for \(n\) activities in the reference process model. The best services for each activity is further selected by maximizing the coverage of the stakeholders’ QoS requirements. Selected services are invoked at run-time through a dynamic/late binding mechanism. The information about services is managed by a service broker.

### 4.2.2 Process Structure Tree

In this thesis, we consider structured process models which have a number of desirable properties \([170, 251]\). A process model is referred to as structured if it has the following properties: (1) the soundness property of a process model can be checked in polynomial time. A process model with no structural errors such as deadlocks or lack of synchronization \([271]\) is said to be sound \([6]\), and (2) it has the properties of modularity, readability and maintainability. The soundness of a structured process model can be analyzed by Petri net reachability graphs.

To allow further analysis of activities and their relationships according to workflow patterns, we rely on the concept of Process Structure Tree (PST). This concept is based on partitioning a process model into smaller fragments (regions or components) and structuring them in a hierarchical way according to the nesting relation. PSTs can be created using different algorithms. Our approach is based on a method described in \([320]\), which decomposes a process model into canonical Single-Entry-Single-Exit (SESE) regions. The SESE regions are known from compiler theory and are utilized for control flow decompilation.

We consider the (well-) structuredness property of a process model \([170, 251]\). A process model is well-structured if and only if for every node with multiple outgoing arcs (i.e., a split), there is a corresponding node with multiple incoming arcs (a join), such that set of nodes between the split and the join forms SESE region. An structured process model can

---

\(^8\)Object Constraint Language: [http://www.omg.org/spec/OCL](http://www.omg.org/spec/OCL)

\(^9\)The Rule Markup Initiative: [http://ruleml.org](http://ruleml.org)

\(^{10}\)The SLAng SLA Language: [http://uclslang.sourceforge.net](http://uclslang.sourceforge.net)

be decomposed into SESE regions\cite{321}, which can be computed in linear time (cf. \cite{158}).
A SESE region is defined formally as as follow:

**Definition 3** (Single-Entry-Single-Exit Region). Let $G_{PM}$ be a process model graph with the distinguished initial and final nodes, such that every node is reachable from the initial node and the final node is reachable from every node in $G_{BP}$. Two distinct nodes $V_i$ and $V_j$ in $G_{BP}$ enclose a Single-Entry-Single-Exit (SESE) regions if

1. $V_i$ dominates $V_j$, i.e., every path from initial node to $V_j$ includes $V_i$
2. $V_j$ postdominates, i.e., every path from $V_j$ to final includes $V_i$
3. every cycle containing $V_i$ also contains $V_j$

We define *Process Structure Tree* (PST) as a hierarchical representation of a structured process model according to Definition 2 and 3.

**Definition 4** (Process Structure Tree). Given a process model PM, $PST_{PM} = (V, E)$ is a tree of canonical fragments of $G_{PM}$, where $V = \{V_r, V_R\}$ is a finite set of nodes. $V_r$ is the root of tree with $V_R$ corresponding to canonical SESE regions, $E \subseteq (V_R \times (V_R \setminus V_r))$ being the set of edges, and nodes being either activities $V_A$ or gateways $V_G$.

### 4.2.3 Feature Model Specification

In the previous chapters, we discussed that feature models can be extended to support a configuration framework to allow dynamic deployment of variant business processes. Feature models are used to facilitate configuration of software product lines. In SOSPLs, configuration also implies selection of a particular process variant from the original *reference business process model*. A feature model is a means for describing a permissible configuration space of all the products of an SOSPL in terms of its features and their relationships. Figure 4.1 (a) depicts a part of the feature model in our example of the online shop product line, which breaks down the variability and commonality of the product line into a hierarchy of features and represents structural variability in the reference process model. Parent-child relationships in the feature diagram indicate the refinement of the product line functionality. As not all features are assumed to be present in every product, this differentiation is expressed by a classification of feature types and relationships which drive *structural variability patterns* as follows:
• **Mandatory/Optional:** A mandatory feature must be included in every member of a product line of services if its parent feature is selected. An optional feature may or may not be included if its parent is included;

• **Or-group:** An Or-feature group is a non-empty subset of features that can be included if a parent feature is included;

• **Alternative-group:** Alternative-feature group indicates that from a set of alternative features exactly one feature must be included if the parent of that set is included.

Variation points are those features that have at least one direct-variable sub-feature (cf. Figure 4.1). It can be observed that, except for the mandatory feature type, all the other types of features imply structural variability.

A feature model can be expressed as a rooted directed acyclic graph (DAG). Feature diagrams, which are a tree-like graphical notation, is more widely used due to its visual appeal and easier understanding. In the tree-like structure, each node (feature) represents a variation or increment in application functionality. We formally define a feature model as follows:

**Definition 5** (Feature Model). A feature model $FM$ is a directed acyclic graph $G_{FM} = (V_F, r, DE, CE, \lambda)$ where:

- $V_F = \{f_1, \ldots, f_n\}$ is a finite set of nodes representing features and theirs attributes;
- $r \in V_F$ is the root feature of a feature model;
- $DE \subseteq N \times N$ is a set of decomposition edges representing the parent-child relations;
- $\lambda : P(f) \rightarrow NT \subseteq N \times N$ is a function assigning feature types and cardinalities for child features with parent feature $f$, where $NT \in \{\cdot, \circ, \bigtriangleup, \bigtriangleup\}$. The $\cdot$ and $\circ$ denote mandatory and optional parent-child feature relations, respectively; $\bigtriangleup$ and $\bigtriangleup$ denote Or and Alternative group relations between parent-child features with common parents, respectively;
- $CE \subseteq N \times N$ is a set of constraints edges expressing the integrity constraints among features.
Figure 4.2 shows the meta model of cardinality-based feature model and the major constructs. A feature model with a root feature is composed by an optional set of relations. The cardinality-based relations can be two distinctive types: “binary” relations expressing optional and mandatory features, and “set” relations constructing grouped features.

In general, cardinality specifies reasonable interval for number of child-feature in a group of features \( F \subseteq N \times N \), which is denoted by \( \langle k-k' \rangle \), where \( 1 \leq k \leq k' \leq m \), and \( m = |F| \); hence, \( \langle k-k' \rangle \) cardinality defined over Or feature groups indicates at least \( k \) and at most \( k' \) features that can be included out of the \( m \) features in a group if the parent is selected. Moreover, the \( \langle k-k \rangle \) cardinality specified for Alternative feature groups implies that only \( k \) out of \( m \) features in the group must be included if the parent is selected. The concept of cardinality also emulates four types of decomposition for features in a group as follows: \( \langle m-m \rangle \), \( \langle 0-0 \rangle \), \( \langle 1-m \rangle \), and \( \langle 1-1 \rangle \) specifies the mandatory, optional, Or, and Alternative features/groups, respectively.

A feature and corresponding service may depend on other features for its correct functioning or operation; therefore, operational dependencies among features can be defined by
constraints which are known as integrity constraints (or cross-tree constraints) \[193\]. For instance, a feature can refer to another feature that requires (includes) or excludes. Therefore, the presence of a certain feature in the product may impose the presence or exclusion of another feature. Integrity constraints can be expressed and specified by the constraints edges in a feature model; thereby, service designer can express implicitly or explicitly dependency among features.

We extended conventional feature models to incorporate non-functional properties and specify quality characteristics (cf. Figure \[1.2\]); as a result, each feature can comprise properties to include quality information. In SOA, non-functional properties can be viewed as QoS characteristics. In the next section, we describe our proposed QoS model which describes quality characteristics and extends feature model.

4.3 Quality Requirements and QoS Model

This section introduces a QoS model and also provides the foundation for QoS aggregation and computation, and further quality-aware SOSPL configuration presented in Chapter \[5\] of this thesis.

4.3.1 Quality Requirements and Criteria

In contrast to functional properties defining a particular functionality or behaviour of a software system, non-functional properties (also known as extra-functional or quality) describe how well the functionality of a system is fulfilled. We discriminate between the two general types of quality characteristics for an SOSPL: (1) the quality attributes which are specific to the product line. Such quality variability characteristics are related to functional variability causing variation in the quality of product-line members, and (2) the general quality or domain-specific quality characteristics, i.e., cost, performance, cost, safety, etc. To illustrate our approach, we only consider general quality type for further quality aggregation and computation in this thesis.

Existing standards for software product quality (e.g., ISO/IEC 9126-1 and the updated version ISO/IEC 25010 2010) provide insights into the general and common characteristics for almost every type of software. However, different types of software products have specific characteristics; hence, the actual application of software quality models usually requires reusing an existing quality model and extending it for a specific software product or domain.
In the SOC literature, the “Quality of Service”, or QoS for short, is referred to as non-functional properties associated with a service which are measured to evaluate the degree a service meets the quality requested by its stakeholders \[213\]. These qualities are described in a Service Level Agreement (SLA) serving as the foundation for expressing non-functional requirements and the expected level of service between the consumer and provider.

The importance of QoS for SOA is due to the fact that service-oriented applications often use highly-distributed and loosely-coupled services over the network. These services may be invoked and executed by a variety of stakeholders in heterogeneous environments. Accordingly, the modeling, describing and managing service quality are of utmost importance in all the SaaS life-cycle phases, requirements specification to design, deployment, execution, and monitoring.

### 4.3.2 QoS Languages

Several QoS languages have emerged to deal with QoS definitions and specifications and describe SLAs for service-based applications, which can be grouped based on their focus during standardizing efforts. These include IBM’s Web Service Level Agreement (WSLA) Language \[196\] and HP’s Web Service Management Language (WSML) \[272\], the WS-Agreement \[19\] from the Open Grid Forum (OGF), which are XML-based languages. The use of XML is clearly intended to facilitate integration with other Web-service technologies that are also dependent on XML. WSLA and WSML are developed for managing enterprise-oriented application scenarios, which are accompanied with QoS management infrastructure. Another work is WS-Policy \[322\] developed by the World Wide Web Consortium (W3C), which allows services/components to specify their policies and enables consumers to specify their policy requirements. These policies primarily defines non-functional properties.

Moreover, there are other notable ongoing research projects. Rule-based Service Level Agreement (RBSLA) \[247\] employs knowledge representation concepts for describing requirements and the specification of SLA. Web Service Offering Language (WSOL) \[308\] enables to specify functional and QoS constraints as service policies. Service Level Agreement definition Language (SLAng) \[188, 289\], Web Service Modeling Language (WSML) \[52\], and UML Profile for QoS and Quality of Service Modeling Language (QML) \[107\], all these languages enable to express non-functional aspects at different levels of description. However, no language has yet achieved broad adoption.
Several quality ontologies have been proposed which aim at conceptualizing and integrating concepts and constructs intended for the representation of similar aspects of quality related to SOA-based applications \[181, 205, 242, 257, 309\]. Current ontologies and approaches lack in providing supports to express QoS relations, dependencies, and relative importance and conditions specified by decision-makers involved (e.g., either stakeholder or system). The managing and reasoning QoS requirements are complex tasks because of the nature of involved properties, and several aspects must be also considered \[110, 243, 289\]. Moreover, because of diversity of application domains, QoS models may use different vocabulary, concepts, metrics, units, quantitative and qualitative values to define and represent QoS characteristics and dimensions which may be required from system and stakeholder’s perspectives \[110, 241, 307, 309\].

These problems can be addressed by developing QoS model to define relations, and dependencies of non-functional aspects relevant to QoS specifications, and stakeholder’s preferences \[189, 307\].

4.3.3 QoS Model

The proposed QoS model is utilized to model and define quality properties for SOSPLs. The QoS model is used by service requesters (i.e., stakeholders) to express QoS requirements and preferences and by service providers to describe and advertise services QoS, determine the priority and relationships among QoS aspects that are used for further reasoning, decision-making, and service configuration and optimization.

To this end, we first introduce primary general requirements and design criteria guiding to develop a QoS model. An extended review of a number of existing QoS languages \[19, 196, 247, 308, 322\], ontologies \[181, 242, 257, 309\], and models \[139, 159, 241, 307, 324\] specified these general design requirements which are also summarized by Tran et al. \[309\] as follows: A QoS model should enable to conceptualize and express explicit specifications of QoS properties that can be applicable for various application domains. In addition, it should be extensible to define and include custom QoS properties of importance for specific domains of interest. The QoS model should support different participants (e.g., providers, stakeholders, third-parties, and brokers) in specifying quality information in details and requirements at different levels of expectations. Quality information can be applied to different elements of an atomic or composite service (e.g., service operation, input/output, and interface). For product-line engineering, different types of quality variability can be
identified: (a) variability among different quality properties, (b) different priority levels or preference on quality properties, and (c) indirect variation, which all result in functional variability. Therefore, the QoS model should support to express stakeholders’ preferences, priority and conditions concerning QoS properties (e.g., mandatory and optional properties). Also, it should allow to describe the classification, relation and prioritization of quality aspects.

Our QoS model is based on the OMG UML QoS profile [316] and extended version proposed by [159], which conceptualizes various aspect of quality and enables the definition of standard QoS modeling elements. We also considered existing QoS ontologies [181, 205, 242, 257, 309]. The OMG UML profile and existing ontologies do not take preference into account to express the relative importance of QoS for systems and stakeholders, which is a prerequisite and used for decision-making and reasoning over QoS properties and requirements.

The important characteristics of the QoS model is that it enables to define general and domain-specific quality aspects with their relations and allows to manage complex quality characteristics. In the following, we first overview the key concepts of the UML QoS profile (the details of the meta-model are available in [316]), and then we describe the extension to the QoS meta-model integrating quality information in feature model.

In the following, we first overview the key concepts of the UML QoS profile (the details of the meta-model are available in [316]), and then we describe the extensions to the QoS meta-model by accommodating the preference model with the aim of supporting decision-making and optimization in the course of process configuration, customization, and service selection.

To describe quality concepts, the UML QoS meta-model introduced by OMG comprises the three main conceptual submodels: Characteristics, Constraints, and Levels. Figure 4.3 shows the key elements of this meta-model. The QoSCharacteristic package includes the modeling elements to describe quality characteristics and dimensions. The QoSConstraints describes quality constraints and contracts, and QoSLevel specifies quality model and transitions. In the following, we review some of important elements of these submodels.

1. QoSCharacteristic includes the model elements for the description of QoS properties representing a measurable non-functional aspect of a service within a given domain, for instance, price, availability, throughput, scalability, accuracy, and reliability. The
Figure 4.3: QoS meta-model with proposed extension.
meta-class QoSParameter provides methods and parameters to quantify a quality characteristics.

- **QoSDimension** defines dimensions to quantify a quality characteristic. A QoS characteristics can be quantified and composed by multiple quality dimensions, and a quality dimension of QoS characteristics can be defined based on another QoS characteristics. For instance, reliability characteristic can be an aggregation of quality dimensions such as time to failure, time to repair, and number of failures supported. The statistical qualifier, unit, and direction, as the core attributes, define the type of statistical qualifier, and the unit for the value dimension and direction (or tendency). The types of statistical qualifiers can be defined through elements such as quality range, maximum value, minimum value, mean, variance, standard deviation, percentile, and distribution.

- **QoSPParameter** provides parameterization methods to quantify QoS characteristics in terms of types, units, and metrics for the description of value definitions, that is, time, currency, etc.

- **QoSDependency**, as an extension proposed by [159], specifies the interdependencies between values of quality dimensions; for example, the reliability depends on Mean Time To Failure (MTTF), Mean Time To Repair (MTTR), etc.

- **QoSCategory** enables grouping QoS characteristics with their priorities to manage a complex class of QoS properties sharing similar characteristics to facilitate the process of service ranking and selection. For instance, grouping enables service developers to define more abstract quality aspects (e.g., “Performance”, which is qualified from the stakeholders’ perspectives) can comprise throughput (the rate of successful service-request completion) and response time (the time elapsed between a consumer sending message and receiving response), or “Security” can capture the level and the kind of security a service provides and generally includes authentication, confidentiality, encryption, and traceability and auditability.

- **QoSContext** provides supports for the description of quality characteristics. A context comprises several QoS characteristics. It also can be associated with QoS Contract (e.g., hand-off latency and packet loss).
• **QoSValue** instantiates QoS characteristics and determines value through a dimension slot for quality dimension, for instance, service reliability is 98%.

• **QoSAggregation**, as one of our extensions to the UML QoS meta-model, provides methods for quality aggregation of a composite service based on the variability and composition patterns (See Section 4.4). The **aggRule** expresses the rules and operators for quality aggregation and computations. Some quality dimensions may follow the same aggregation rule.

2. **QoSConstraint** restricts the accepted values of a QoS characteristic, which can be imposed by stakeholders or service providers based on application requirements or architectural decisions. **QoScontext** expresses the characteristics of quality and functional elements involved in a constraint.

• **QoSContract** composes service and consumer constraints and specifies overall multiple acceptable quality level of service and the requirements to be achieved.

• **QoSOffer** specifies the limits of quality values that service operations can support corresponding to the architectures and implementations designed to supply particular quality.

• **QoSRequire** enables service provider or consumer to define quality requirements which are compulsory and which systems must fulfill whilst others may be discretionary. The required constraints can also be specified either by the service provider, user, or subsystem that needs the consumer to achieve some perquisite level of quality to obtain the quality offered by provider.

3. **QoSLevel** determines the working and transition modes under which the service is executed. Quality level defines the quality behavior which relies on the algorithms, configuration, and infrastructure of the service. Different execution modes may provide distinctive range of quality level for the same service. The details of constructs are available in [316].

Figure 4.4 depicts the submodels of extended UML QoS meta-model by preference model, allowing to introduce and specify relative importance and conditional constraints over quality characteristics, which are required for decision-making.
• **DecisionMaker** defines and explicates involved decision-makers (DM) that can be regarded as algorithms, system, service provider, or stakeholder. Decision-maker can stipulate and determine the relative importance of relations of the quality in terms of preference or objective function which should be either maximized or minimized for multi-objective optimization and quality-aware service composition and configuration.

• **Preference** enables decision-makers, i.e., system, service provider, or stakeholder, to express the priorities or relative importance over QoS characteristic, dimension, and category. The preference is often to minimize or maximize an objective function representing preference.
• **Precedence** allows defining the relative importance relations for QoS model elements to specify the priority between quality pairs. The *rules* determine the order at which characteristic, dimension, and category are further prioritized.

• **Weight** expresses the strength or degree of importance associated with relative importance relations.

• **QoSPrecedenceCondition** indicates the constraints when the priorities hold. For instance, the precedence condition specifies priority that should be applied if value over a quality dimension is achieved.

• **ObjectiveFunction** enables to construct single or multiple objective functions formulating and determining DM’s preference. Quality-aware optimization may involve more than one objective function which can be constructed by aggregation.

• **PreferenceCondition** specifies conditional preference on values of quality dimensions to hold.

### 4.4 Quality of Service Aggregation and Computation for Product Line Architecture

In this section, we describe our proposed quality aggregation model for product-line architectures. We cover the following issues to provide a model for the aggregation and computation of QoS-range values in the presence of variability: (a) quality criteria and quality-range for SOSPL; (b) combinations of the variability and composition patterns; and (c) the algorithms to aggregate and compute quality range values.

We consider some quantitative QoS characteristics of Web-services, which have been taken into consideration as selection criteria in the research literature [60, 155, 336, 338]. In this work, we include the following indexed QoS dimensions: 1) cost, 2) response time, 3) throughput, 4) availability, which are denoted as \( q_{pr} \), \( q_{rt} \), \( q_{tr} \), and \( q_{av} \), respectively. Of course, our approach is not limited to these QoS properties, but we limit our discussions to these to illustrate our approach for quality aggregation and computation.

We also distinguish between two types of quality dimensions: *deterministic* and *non-deterministic*. The former indicates that the values of QoS are known or certain when
a service is invoked (e.g., execution price and supported security protocols), whereas the later are unknown or uncertain at invocation time (e.g., response time), therefore, the QoS information should be collected through run-time monitoring. We consider that the QoS information is provided by a service broker or middleware.

Based on the underlying implementation of a set of functionally equivalent services, which may be available for each feature, ranges of values of quality dimensions can be further specified and aggregated for each feature. Particularly during the domain engineering life-cycle, determining the implied QoS-ranges for individual feature helps service engineers ensure that the product line architecture will fulfill the upper and lower bounds of the values of the quality requirements requested by the stakeholders. Moreover, quality-range computation enables for keeping track of the product line quality-ranges even after the specification of the service quality has changed. Mapping models interconnecting feature and business process models enable the propagation of quantified quality values of concrete service sets, bounded to activities (abstract services) within in the process model; for example, in Figure 4.5 sets of candidate services provide different range of quality, denoted as $q^R$, for each features. The range of the $k^{th}$ quality dimension for feature $f$ can be hierarchically computed. Let us now proceed with some formal definitions as a basis for our work.

**Definition 6** (quality-range). The quality range values of the $i^{th}$ quality dimension (dimension) are defined as $q_i^R = [q_i^{LB}, q_i^{UB}]$, where $q_i^{LB}$ and $q_i^{UB}$ are lower and upper bound values of the quality dimension, respectively.

The above definition implies that each property such as response time or cost can be described by a range of numerical values. This range specifies both lower and upper bounds for that quality dimension. To be able to compute such a quality-range, appropriate aggregation operators are needed. We consider the following three types of quality aggregation operators for computing the quality-ranges of a software product line:

- **Summation**: The quality-range values of the product line is determined by a sum of the QoS-range values of the quality attributes of services. An example would be cost.

- **Multiplication**: The range values of quality attributes are determined by production of the QoS values of the services, for instance, reliability and availability.

- **Min-Max**: The quality-range values of the product line are computed with respect
to critical paths in the business process structure, for instance, response time (i.e., execution duration).

Figure 4.5: Non-functional specification and aggregation for evaluating quality-range supports by product line architecture

In order to employ the above operators, we consider the following: We assume that for each activity \( a_n \) in a business process model \( BP \), there is a bounded set of candidate services, \( S_{a_n} = \{s_{n1}, \ldots, s_{nm}\} \), in which all of the candidates provide the same functionality, but have different degrees of quality. The quality of a service \( s \) is a vector \( Q_s = \langle q_1(s), \ldots, q_k(s) \rangle \in \mathbb{R} \), where the function \( q_i(s) \) determines the values of the \( i \)th quality dimension.

The quality of each activity \( a_n \) is defined as a matrix \( [Q_{a_n}]_{i \times j}; 1 \leq i \leq k, 1 \leq j \leq m \), where each row corresponds to a quality dimension \( q_i \), while each column corresponds to a service candidate. Thereby, the range of the \( i \)th quality dimension for feature \( f_n \) corresponding to activity \( a_n \) is obtained by the quality-range function \( q_i^R(f_n) = [q_i^{LB}, q_i^{UB}] \), where \( q_i^{LB} = Q_{an}^{min} \) and \( q_i^{UB} = Q_{an}^{max} \).

For example, let us assume that there are five service candidates in \( S_{a_k} \) binding to activity \( a_k \), mapped to feature \( f_k \), and that their service cost values \( (q_{pr}) \) are given by vector \( Q_{ak}(i, j) = (100, 250, 65, 130, 95) \). The cost range values of feature \( f_k \) could be set as follows: \( q_{pr}^R(f_k) = [65, 250] \), because we are interested in the lower and upper bound values for the quality-range.
4.4.1 Combining Structural Variability and Composition Patterns

In essence, the aggregation model for quality computation in the context of SOSPL depends on: a) structural variability captured by a feature model, and b) behavioral variability captured by a business process model, which describes the composition structure.

Composition patterns\(^\text{12}\), which have their roots in workflow management systems [317], aim at building composition structures that are derived from the requirements in the process modeling phase. In other words, these patterns describe the behavior of features during execution time. Composition patterns are independent of particular composition languages or techniques. They describe the control flow of execution of features that are mapped to the activities of a reference business process model in an SOSPL. The complete collection of the workflow patterns for the different aspects of process-oriented application development has been presented in [317]. Similar to [155], we also consider the composition patterns as a mean to represent behavioral of service compositions and use them as a basis for quality aggregation of SOSPLs. These patterns can be grouped into two main groups: a) sequential patterns and b) parallel patterns. These patterns describe the structure of a process and the execution model for activities (i.e., service).

In a process model, two basic sequential patterns are defined as sequence and arbitrary cycle.

1) **Sequence**: This pattern describes the structure where an activity is executed after the completion of another activity in the same process.

2) **Arbitrary cycle**: This pattern, as a special case of sequence pattern, expresses that the execution of one or more activities is repeated for a given number of times. The number of cycle is determined during the design time or specified with respect to conditions at run-time.

\(^{12}\)We use the terms workflow and composition patterns, interchangeably.
Parallel patterns describe how the process flow splits into branches for executing the activities and how they merge or converge.

3) **Parallel split**: A parallel split pattern represented by the AND-split operation describes the behavioral structure where a single thread of control flow splits into multiple threads which can be executed concurrently in any order.

4) **Multi-choice**: This pattern denoted by the OR-split operation describes the structure in which a number of \( m \) branches among \( n \) (\( 1 \leq m < n \)) can be selected and executed in parallel based on the process control data and business rules or specified probability of the process branches. The \( \rho_i^b \) represents the probability of execution of an \( i \)th branch.

5) **Exclusive/Deferred choice**: An exclusive pattern represented by the XOR-split operation describes conditional branching where only one of several branches is selected.
for the execution based on a user’s decision or process control data. Deferred choice resembles the exclusive choice pattern, but in the deferred choice, the split relies on external input, whereas the exclusive choice pattern is based on workflow information (i.e., orchestration data). At design time, the probability of each branch can also be determined specifies the probability that a certain branch is chosen to be executed at run time.

The following general parallel patterns represent how branches model the logic of the convergence of activities that are invoked or executed in a process flow.

6) **Synchronization:** This pattern represented by the AND-join operation describes the structure of a process where multiple activities in parallel branches converge into one single synchronized thread.

7) **Simple-merge:** The simple merge pattern represented by the XOR-join operation describe the structure where two or more branches converge without the synchronization.

8) **Multi-merge:** This pattern describes the structure where branches converge without synchronization, and the activity succeeding the convergence is activated by the completion of every incoming branch.

9) **Discriminator/m-out-of-n-join:** This pattern describe the convergence of multiple branches in a process into a single subsequent branch such that the completion of one branch results in the subsequent branch being activated, but the completions of other incoming branches thereafter are disregarded and have no effect on the subsequent branch. This pattern has also been known as a partial join \[155, 317\]. The discriminator patterns can be generalized where the subsequent activity will be activated only after \(m\)-out-of-\(n\) activities. The number of \(m\) is specified over a business process model based on the process control data and business rules.

It is noteworthy that various combinations of the parallel splits and joins cannot provide valid patterns. For example, an XOR-split in combination with an AND-joint cannot occurs in process models; hence, this combination does not result in a valid pattern.

In the next section, based on the combination of structural variability patterns (variability patterns in short) and composition patterns, we first demonstrate how the effects of QoS
aggregation for the computation of quality-ranges are derived from such patterns, and then we describe algorithms to perform the derivation for other complex patterns in the same way. In addition to descriptive definitions of selected composition patterns, we show the semantics using BPMN. Figure 4.6 illustrates three main variability patterns (left side), as described in Section 4.2.3, in combination with 11 composition patterns (CP1-CP11) (right side).

4.4.2 Aggregation Rules Based on Structural Variability and Composition Patterns

Based on the patterns described above, we define aggregation rules for each QoS property by primarily taking into account the variability patterns which may occur within each composition pattern. In the following, we present the aggregation rules for four numerical QoS properties: cost, response time, availability and throughput. The cost of a feature is the cost which can be associated with the deployment, execution, management, maintenance and monitoring of a service.

The summary of the cost and response time aggregation rules grouped based on mandatory/optional patterns and Or/Alternative variability patterns corresponding to composition patterns are given in Tables 4.1 and 4.2 respectively. The aggregation rules for availability and throughput are presented in Tables 4.3 and 4.4.

According to Definition 6, the definition of a quality lower bound, \( q_{i}^{LB} \), for different quality dimensions must indispensably consider the mandatory features for sequential and parallel split patterns (CP1-CP6). In addition to mandatory features, the optional features generally contribute to the upper-bound range value, \( q_{i}^{UB} \). For instance, in the sequential patterns, the cost of a particular feature should be determined by the sum of the cost values of each mandatory feature for the lower bound; while the upper bound is resolved by the accumulated cost for both the mandatory and optional features.

By adopting a hierarchical approach, described in the next section, the range values (upper and lower bounds) for QoS properties are computed for a combination of variability and composition patterns, based on our formulated aggregation rules. To determine the upper and lower bounds for QoS-range values, \( q_{f}^{R}(f) \), for a parent feature \( f \) (cf. Figure 4.6), the aggregation rules are applied on the basis of each composition pattern according to the variability patterns.

To compute the quality-range values, the aggregation operators are applied to each
Table 4.1: Aggregation rules based on *Mandatory/Optional* patterns for cost and response time.

<table>
<thead>
<tr>
<th>QoS property</th>
<th>Cost (q&lt;sub&gt;c&lt;/sub&gt;)</th>
<th>Response Time (q&lt;sub&gt;r&lt;/sub&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Seq. Patterns</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1 CP1 Sequence</td>
<td>$\sum_{i=1}^{n} q_{pr}^L(f_i) : \forall f_i \in f \quad , \quad \sum_{i=1}^{n} q_{pr}^U(f_i) : \forall f_i \in f \lor f$</td>
<td>$\sum_{i=1}^{n} q_{pr}^L(f_i) : \forall f_i \in f \quad , \quad \sum_{i=1}^{n} q_{pr}^U(f_i) : \forall f_i \in f \lor f$</td>
</tr>
<tr>
<td>R2 CP2 Arbitrary Cycle</td>
<td>$c_{q_{pr}}^L(f_i) : f_i \in f \lor f \quad , \quad c_{q_{pr}}^U(f_i) : f_i \in f \lor f$</td>
<td>$c_{q_{pr}}^L(f_i) : f_i \in f \lor f \quad , \quad c_{q_{pr}}^U(f_i) : f_i \in f \lor f$</td>
</tr>
<tr>
<td>R3 CP3 AND-AND</td>
<td>$\min \left( q_{pr}^L(f_i) : \forall f_i \in f \right) \quad , \quad \max \left( q_{pr}^L(f_i) : \forall f_i \in f \right)$</td>
<td>$\min \left( q_{pr}^L(f_i) : \forall f_i \in f \right) \quad , \quad \max \left( q_{pr}^L(f_i) : \forall f_i \in f \right)$</td>
</tr>
<tr>
<td>R4 CP5 AND-DISC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5 CP6 AND-XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6 CP7 XOR-XOR</td>
<td>$\sum_{i \in F_{sub}}^{n} q_{pr}^L(f_i) : \forall F_{sub} \in F_{C_M}^n$</td>
<td>$\sum_{i \in F_{sub}}^{n} q_{pr}^L(f_i) : \forall F_{sub} \in F_{C_M}^n$</td>
</tr>
<tr>
<td>R7 CP8 OR-XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R8 CP9 OR-OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R9 CP10 OR-DISC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Mandatory/Optional Patterns**

Table 4.1 continued: Aggregation rules based on *Mandatory/Optional* patterns for cost and response time.

<table>
<thead>
<tr>
<th>QoS property</th>
<th>Cost (q&lt;sub&gt;c&lt;/sub&gt;)</th>
<th>Response Time (q&lt;sub&gt;r&lt;/sub&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Seq. Patterns</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1 CP1 Sequence</td>
<td>$\sum_{i=1}^{n} q_{pr}^L(f_i) : \forall f_i \in f \quad , \quad \sum_{i=1}^{n} q_{pr}^U(f_i) : \forall f_i \in f \lor f$</td>
<td>$\sum_{i=1}^{n} q_{pr}^L(f_i) : \forall f_i \in f \quad , \quad \sum_{i=1}^{n} q_{pr}^U(f_i) : \forall f_i \in f \lor f$</td>
</tr>
<tr>
<td>R2 CP2 Arbitrary Cycle</td>
<td>$c_{q_{pr}}^L(f_i) : f_i \in f \lor f \quad , \quad c_{q_{pr}}^U(f_i) : f_i \in f \lor f$</td>
<td>$c_{q_{pr}}^L(f_i) : f_i \in f \lor f \quad , \quad c_{q_{pr}}^U(f_i) : f_i \in f \lor f$</td>
</tr>
<tr>
<td>R3 CP3 AND-AND</td>
<td>$\min \left( q_{pr}^L(f_i) : \forall f_i \in f \right) \quad , \quad \max \left( q_{pr}^L(f_i) : \forall f_i \in f \right)$</td>
<td>$\min \left( q_{pr}^L(f_i) : \forall f_i \in f \right) \quad , \quad \max \left( q_{pr}^L(f_i) : \forall f_i \in f \right)$</td>
</tr>
<tr>
<td>R4 CP5 AND-DISC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5 CP6 AND-XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6 CP7 XOR-XOR</td>
<td>$\sum_{i \in F_{sub}}^{n} q_{pr}^L(f_i) : \forall F_{sub} \in F_{C_M}^n$</td>
<td>$\sum_{i \in F_{sub}}^{n} q_{pr}^L(f_i) : \forall F_{sub} \in F_{C_M}^n$</td>
</tr>
<tr>
<td>R7 CP8 OR-XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R8 CP9 OR-OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R9 CP11 OR-DISC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4.2: Aggregation rules based on Or/Alternative-feature groups variability patterns for cost and response time.

<table>
<thead>
<tr>
<th>QoS Properties</th>
<th>Cost ((q_c))</th>
<th>Response Time ((q_{rt}))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallel Patterns</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1 CP1 Sequence</td>
<td>(\min \left( \sum \frac{\text{LB}}{\text{UB}} q_{i,j} F_{\text{sub}} \subset F_{C_k^n} \right))</td>
<td>(\max \left( \sum \frac{\text{LB}}{\text{UB}} q_{i,j} F_{\text{sub}} \subset F_{C_k^n} \right))</td>
</tr>
<tr>
<td>R2 CP2 AND-AND</td>
<td>(\max \left( \sum \frac{\text{LB}}{\text{UB}} q_{i,j} \text{max} \sum F_{\text{sub}} \subset F_{C_k^n} \right))</td>
<td>(\max \left( \sum \frac{\text{LB}}{\text{UB}} q_{i,j} F_{\text{sub}} \subset F_{C_k^n} \right))</td>
</tr>
<tr>
<td>R3 CP3 AND-DISC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4 CP4 AND-OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5 CP5 AND-XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6 CP6 OR-AND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R7 CP7 OR-OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R8 CP8 OR-DISC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R9 CP9 OR-XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R10 CP10 XOR-XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R11 CP11 XOR-OR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^{(*)}\) and \((**)\) represent the feature set combinatorial operators which are applied for Or and Alternative variability patterns, respectively.

Member set of \(F_{\text{sub}} \subset F_{C_k^n}\). For instance, for the lower and upper range values of cost \((q_{pr})\), the summation operator is first applied to each element of \(F_{\text{sub}}\), which results in a new set. For this new set, the min-max operator is then applied.

The mandatory and optional features in the Multi-choice parallel patterns (cf. CP8, CP9, CP10, and CP11) follow different aggregation rules. To perform quality aggregation, the aggregation model also needs to capture which paths in the business process flow will be chosen at run-time particularly for OR-Splits; however, we consider the worst-case scenarios to determine the lower and upper bounds of quality values and assume that the execution of all possible choices for an OR-Split gateway is equally probable. The business rules defined over business process models (i.e., OR-Split gateway) specify how many paths \((m)\) can be executed at run-time. This results in a feature set \(F_{\text{sub}} \subset F_{C_k^n}\) where \(k \leq m \leq k' \leq n\).

For instance, in our example from Section 4.1, we assume that two notification features Email-voicemails and Mobile-based notification are included in an instance of the reference business process. Hence, the decision concerning which notification service should be invoked w.r.t. OR-split semantics is left to run-time.

To address Or/Alternative-feature groups variability in combination with Multi and Exclusive choice composition patterns (CP7-C11), the aggregation model must consider all possible combinations of features corresponding to the given cardinality \(<k-k'>\) over the \(n\) features of the feature group specified at design time. Therefore, the resulting feature set (i.e., \(F_{\text{sub}}\)) is a subset of \(F_{C_k^n}\) and \(F_{C_{k'}^n}\) for lower and upper bound quality-range values, respectively.
Chapter 4. QoS and Aggregation Models

Table 4.3: Aggregation rules based on Mandatory/Optional patterns for availability and throughput.

<table>
<thead>
<tr>
<th>QoS property</th>
<th>Availability (qav)</th>
<th>Throughput (qtp)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Seq. Patterns</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1 CP1 Sequence</td>
<td>$\prod_{i=1}^{n} q_{av}^L (f_i) : \forall f_i \in f \lor f'$, $\prod_{i=1}^{n} q_{av}^U (f_i) : \forall f_i \in f'$</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
</tr>
<tr>
<td>R2 CP2 Arbitrary Cycle</td>
<td>$q_{av}^L (f_i) : \forall f_i \in f \lor f'$, $q_{av}^U (f_i) : \forall f_i \in f \lor f'$</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\min\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
</tr>
<tr>
<td>R3 CP3 AND-AND</td>
<td>$\prod_{i=1}^{n} q_{av}^L (f_i) : \forall f_i \in f \lor f'$, $\prod_{i=1}^{n} q_{av}^U (f_i) : \forall f_i \in f \lor f'$</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\min\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
</tr>
<tr>
<td>R4 CP5 AND-DISC</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
</tr>
<tr>
<td>R5 CP6 AND-XOR</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
</tr>
<tr>
<td>R6 CP7 XOR-XOR</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
</tr>
<tr>
<td>R7 CP8 OR-XOR</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
</tr>
<tr>
<td>R8 CP9 OR-OR</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
</tr>
<tr>
<td>R9 CP11 OR-DISC</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
<td>$\min\left(\prod_{f_c}^{L} q_{vp}^L (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$, $\max\left(\prod_{f_c}^{U} q_{vp}^U (f_c) : \forall f_c \in F_{ac} \lor F_C \right)$</td>
</tr>
</tbody>
</table>
Table 4.4: Aggregation rules based on Or/Alternative-feature groups variability patterns for availability and throughput.

<table>
<thead>
<tr>
<th>QoS Properties</th>
<th>Availability (q(v_a))</th>
<th>Throughput (q(v_t))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Availability</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| R1 [CP1] Sequence | \[
\begin{align*}
&\min \left( \prod_{i \in \mathcal{A}_F} q_{v_a}(f_i) \cdot F_{v_a} \in F_i \right)
\end{align*}
\] & \[
\begin{align*}
&\min \left( \min_{\mathcal{A}_F} q_{v_a}(f_i) \cdot F_{v_a} \in F_i \right)
\end{align*}
\]
| R2 [CP3] AND-AND | \[
\begin{align*}
&\max \left( \prod_{i \in \mathcal{A}_F} q_{v_a}(f_i) \cdot F_{v_a} \in F_i \right)
\end{align*}
\] & \[
\begin{align*}
&\max \left( \max_{\mathcal{A}_F} q_{v_a}(f_i) \cdot F_{v_a} \in F_i \right)
\end{align*}
\]
| R3 [CP4] AND-DISC |                         |                       |
| R4 [CP5] AND-XOR |                         |                       |
| R5 [CP6] AND-OR  |                         |                       |
| R6 [CP7] OR-XOR  |                         |                       |
| R7 [CP8] OR-OR   |                         |                       |
| R8 [CP9] OR-DISC |                         |                       |
| R9 [CP10] OR-XOR |                         |                       |
| R10 [CP11] XOR-XOR |                         |                       |

4.4.3 A Method for QoS-range Aggregation and Computation

Our method is based on analyzing variability and composition patterns expressing the structural and behavioural variability of a reference business process.

The QoS-range values for features in a feature model are computed by hierarchically aggregating the QoS for variability patterns at the level of each composition pattern. Aggregation is performed by gradually collapsing features into a single feature in the feature model, by employing the notion of a virtual feature. This approach enables us to perform the aggregation from both microscopic and macroscopic perspectives. In other words, the quality-range values can be computed for each feature from a local view and also for the entire feature model from a global view.

Algorithms 1 and 2 detail the procedure for computing QoS-ranges for a feature model. The aggregation and computation of quality-range values of particular quality dimension for specific feature starts with Algorithm 1 where the feature model is traversed from a given feature node by post-order depth-first traversal, i.e., computing the aggregated QoS-ranges from leaves and the right-most nodes up to the root node.

In order to analyze and identify how features at the same level in the feature model are composed in the process model, we decompose the process graph into process components. We employed the Refined Process Structure Tree technique [251, 321] for decomposition and transformation of process model graphs and further control flow analysis.

For each feature, the Process Structure Tree (PST) corresponding to a given feature is subsequently created and parsed. Line 7 in algorithm [1] creates and decomposes the process model graph into a tree of Single-Entry Single-Exit (SESE) components. Each process
component is a connected subgraph with a SESE region, which includes all components at the lower level. Each component pertains to one out of four structural classes: i) a trivial components of a single flow arc; ii) a polygon describing a sequence of components; iii) a bond presenting a set of components that share two common node, and iv) other components are considered rigid. The SESE components at the same level are disjoint. It is noteworthy that for any process model there is a PST defining unique decomposition.

**Algorithm 1:** Aggregate QoS-range for feature: \( \text{AggregateQualityRange}(f) \)

**Input:** \( f \): given feature of feature model \( FM \)

**Output:** QoS range- \( q^R \) of feature \( f \)

\[
\begin{align*}
1 & \text{begin} \\
2 & \quad \text{// All direct child features of } f; \\
3 & \quad S_f[ ] \leftarrow \forall \text{ChildFeatureOf}(f) ; \\
4 & \quad \text{if } S_f = \emptyset \text{ then return } q^R(f) ; \\
5 & \quad \text{else} \\
6 & \quad \quad \text{for } i = 1 \text{ to } |S_f| \text{ do} \\
7 & \quad \quad \quad \text{AggregateQualityRange}(S_f[i]); \\
8 & \quad \quad \quad \text{PST} \leftarrow \text{ProcessStructure}(S_f[i]); \\
9 & \quad \quad \quad \text{if } \text{PST} = \emptyset \text{ then return } q^R(f); \\
10 & \quad \quad \text{else} \\
11 & \quad \quad \quad \text{ComputeQualityRange}(PST) \\
\end{align*}
\]

Figure 4.7(a) depicts the result of decomposition of the graph of the exemplified payment process model graph given in Figure 4.1 where each dotted box represents a SESE.
component. The resulting process structure tree which defines a hierarchy of components is shown in Figure 4.7(b). For every trivial SESE component in a PST, the control flow analysis, whose details are omitted from algorithm 2 for the sake of brevity, is performed; readers interested in the details of the control flow analysis are referred to [320].

Algorithm 2 operates over a PST, and the aggregation is performed at the level of each processes component. Lines 3 to 23 iteratively aggregate the quality for each child component. For individual process components, features are grouped into virtual features corresponding to the variability patterns. The control flow information is used for identifying the pattern in each of the SESE components. The virtual features, which are denoted as \( f_{\text{Mo}} \), \( f_{\text{Or}} \), and \( f_{\text{Xor}} \), represent Mandatory/Optional, and Or/Alternative-virtual feature group, respectively.

The range values of quality dimension of virtual feature are computed by aggregation function according to the aggregation rules introduced earlier and shown in Tables 4.1, 4.2, 4.3, and 4.4. It should be noted that aggregation rules are defined for individual quality dimension with respect to their nature. The class QoS\text{Aggregation} introduced in the QoS model (cf. Figure 4.3) defines aggregation rules and provides required methods.

In order to comply further with the aggregation rules, the type of virtual features should also be determined. Hence, the type of the corresponding virtual feature is labeled as optional if all the collapsed features are optional; otherwise, it is labeled as mandatory (i.e., line 11).

It should be noted that according to the given semantic descriptions of variability patterns for Or/Alternative-feature groups, corresponding virtual features \( f_{\text{Or}} \) and \( f_{\text{Xor}} \) are labeled as mandatory. The virtual feature \( f_{V} \) includes all of the collapsed virtual features, which are grouped in each basic composition patterns, and its type is determined such that the type of the collapsed virtual features is considered mandatory as well (line 20-23) if there is at least one mandatory feature in a grouped feature.

Example 1. To demonstrate the aggregation algorithms described above, we use a simple example of the aggregation and computation of QoS range values of cost \( (q_{pr}) \) for the payment feature in the feature model shown in Figure 4.1.
Algorithm 2: Compute QoS-range values of a process associated to feature $f$.

$\text{ComputeQualityRange}(C)$

Input: $C$: Process component - node of process structure tree $PST$

Output: $q^R$: aggregated QoS-range of process component

begin

\begin{enumerate}
\item\hspace{1em} \textbf{foreach} $C_i \in \text{ChildOf}(C)$ \textbf{do} \text{ComputeQualityRange($C_i$)};
\item\hspace{1em} \textbf{forall} the $f_k \in C_i$ \textbf{do}
\item\hspace{2em} \text{[/\拢\hspace{1em}]Group features w.r.t. variability patterns and step-wise collapsing features by means of virtual features;}
\item\hspace{2em} \textbf{switch} feature $f_k$.Type \textbf{do}
\item\hspace{3em} \textbf{case} $f_k \in f \cup f$
\item\hspace{4em} $f_{\text{mo}}[\cdot] \leftarrow f_k$;
\item\hspace{4em} // Applying aggregation rules for Mandatory/Optional patterns;
\item\hspace{4em} $q^R(f_{\text{mo}}) = \text{QoSAggregation}(f_{\text{mo}})$;
\item\hspace{4em} if $\forall f_k \in f_{\text{mo}}: f_k \in f$ then
\item\hspace{5em} $f_{\text{mo}}.\text{Type}=f$
\item\hspace{4em} else
\item\hspace{5em} $f_{\text{mo}}.\text{Type}=\circ f$
\item\hspace{3em} \textbf{case} $f_k \in f$
\item\hspace{4em} $f_{\text{or}}[\cdot] \leftarrow f_k$;
\item\hspace{4em} // Applying aggregation rules for Or pattern;
\item\hspace{4em} $q^R(f_{\text{or}}) = \text{QoSAggregation}(f_{\text{or}})$;
\item\hspace{3em} \textbf{case} $f_k \in f$
\item\hspace{4em} $f_{\text{xor}}[\cdot] \leftarrow f_k$;
\item\hspace{4em} // Applying aggregation rules for Alternative pattern;
\item\hspace{4em} $q^R(f_{\text{xor}}) = \text{QoSAggregation}(f_{\text{xor}})$;
\item\hspace{2em} $f_{\text{v.}}[\cdot] \leftarrow f_{\text{mo}}, f_{\text{or}}, f_{\text{xor}}$;
\item\hspace{2em} // Applying aggregation rules w.r.t. variability patterns;
\item\hspace{2em} $q^R(f_{\text{v.}}) = \text{QoSAggregation}(f_{\text{v.}})$;
\item\hspace{2em} if $\exists f_k \in f_{\text{v.}}: f_k \in f$ then
\item\hspace{3em} $f_{\text{v.}}.\text{Type}=f$
\item\hspace{2em} else
\item\hspace{3em} $f_{\text{v.}}.\text{Type}=\circ f$
\item\hspace{2em} \textbf{return} $q^R(f_{\text{v.}})$
\end{enumerate}
It is noteworthy that different rules are applied for aggregation of range values for other QoS properties (e.g., response time). The following is a step-wise description of the above two algorithms for computing the QoS-range values applied to the Payment feature from Figure 4.1. We refer to the respective lines of the two algorithms to show how each step of the proposed algorithms is actually performed and how the quality-range values are computed.

\[
q_{pr}^{R}(f) = \left\{ q_{pr}^{LB}, q_{pr}^{UB} \right\} := q_{pr}(f_i) | f_i = \text{Agg.} \bigcup_{i=1}^{n} f_i \text{ (Alg.1 lines 2-11)}
\]

\[
q_{pr}^{R}(\hat{f}_1) = [8, 20]; q_{pr}^{R}(\hat{f}_2) = [17, 48]; q_{pr}^{R}(\hat{f}_4) = [10, 54]; \text{ (Alg.1 lines 3)}
\]

\[
q_{pr}^{R}(\hat{f}_1) = \left\{ q_{pr}^{R}(\hat{f}_1), q_{pr}^{R}(\hat{f}_2) \right\} = \left[ q_{pr}^{LB}(\hat{f}_1), q_{pr}^{UB}(\hat{f}_1) \right], \text{ (Alg.2 lines 5-8)}
\]

\[
q_{pr}^{R}(\hat{f}_3) = \left[ 12, 51; \right. \text{ (Alg.2 lines 5-8)}
\]

\[
q_{pr}^{R}(\hat{f}_4) = \left[ q_{pr}^{R}(\hat{f}_3), q_{pr}^{R}(\hat{f}_4) \right] = \left[ q_{pr}^{LB}(\hat{f}_3), q_{pr}^{UB}(\hat{f}_3), q_{pr}^{LB}(\hat{f}_4), q_{pr}^{UB}(\hat{f}_4) \right] = [12, 105]; \text{ (Alg.2 lines 5-8)}
\]

\[
q_{pr}^{R}(\hat{f}_5) = \left[ \text{min} \left( \sum_{f \in F_{Sub}} \left( q_{pr}^{LB}(f_i) \cdot \forall F_{Sub} \in F_{C} \right) \right), \right. \text{ (Alg.2 lines 9-11)}
\]

\[
q_{pr}^{R}(\hat{f}_6) = \left\{ q_{pr}^{R}(\hat{f}_2), q_{pr}^{R}(\hat{f}_3), q_{pr}^{R}(\hat{f}_6) \right\} = \left[ q_{pr}^{LB}(\hat{f}_2), q_{pr}^{LB}(\hat{f}_3), q_{pr}^{UB}(\hat{f}_6), q_{pr}^{UB}(\hat{f}_6) \right] = [40, 250]; \text{ (Alg.2 lines 5-8)}
\]

\[
q_{pr}^{R}(\hat{f}) = \left\{ q_{pr}^{R}(\hat{f}_1), q_{pr}^{R}(\hat{f}_6) \right\} = \left[ q_{pr}^{LB}(\hat{f}_1), q_{pr}^{UB}(\hat{f}_6), q_{pr}^{UB}(\hat{f}_6), q_{pr}^{UB}(\hat{f}_6) \right] = [40, 270] \text{ (Alg.2 lines 5-8)}
\]
Figure 4.8: Steps used for the aggregation and computation of quality-range for the cost \( q_c \). a) Step-wise feature-model transformation. b) Process model and decomposition to canonical SESE components \( (C_1, \ldots, C_{14}) \). c) Process structure tree – each node represents corresponding SESE components.
Figure 4.8 illustrates the step-wise transformation of the feature model and hierarchical aggregations. In each step, the feature model is traversed from a given feature, and a variability pattern is determined. The PST corresponding to the process model is parsed which is followed by control-flow analysis to detect the workflow patterns.

4.5 Discussion

Our work is related to a number of different areas. In the following sections, we first overview the approaches focusing on modeling non-functional requirements and quality properties in SPLs. Then, we describe approaches addressing quality evaluation for SPL architectures. In the last section, we summarize approaches that aim to address QoS aggregation and computation for SOA.

Modeling Quality Characteristics in SPLs: A literature survey on approaches addressing non-functional requirements is presented in [68]. A number of research efforts have developed different modeling approaches and evaluation methods for quality characteristics in the area of product line engineering.

There are several proposals about the inclusion of quality information in feature models to support quality-driven feature selection and product line configuration [82]. Benavides et al. [44] proposed an extension to feature model to accommodate non-functional properties (quality attributes) in features and facilitate further reasoning and automatic selection of features. The relations among features and feature attributes are transformed into a constraint satisfaction problem for quality-aware product configuration. Thurimella et al. [306] proposed an extended feature model by augmenting selection criteria, as product quality attributes for each variant feature. Their proposed method assesses each variable feature based on its related criteria in qualitative values. They leveraged orthogonal variability modeling (OVM) [250] to separate concern between variability and system modeling and extended rationale model–questions, options and criteria (QOC) [201] to determine rationale behind decisions for requirement engineering.

Goal-oriented approaches are also utilized to model quality requirements. Jarzabek et al. [156] proposed an approach based on soft-goal modeling technique and a framework to
model quality attributes by a goal model linked to a feature model. Their framework integrates goal and feature models by introducing the notion of feature-softgoal interdependency graph (F-SIG) which enables to express the dependency among quality attributes and features. Nevertheless, their approach is limited to the feature model, and they did not take into account the variability in system behaviour. In similar work, Yu et al. [337] employed goal models to express and capture the stakeholder’s goals or intentions expressed in terms of quality attributes associated with features.

For variability modeling and quality evaluation, other methods also have been proposed that are not based on feature models. Deelstra et al. [87] developed a method and framework, named COVAMOF software variability assessment method (COSVAM), to support modeling variation points and dependencies in a product line. They introduced the notion of dependency to represent system properties, such as quality attributes, and specify how the selection of individual variant features influences the system properties. In another work, Niemelä et al. [233] introduced a method, called Quality Requirements of a Software Family (QRF), to support modeling and defining quality requirements, and mapping requirement. However, these approaches did not address how to explicitly model quality and evaluate quality variability.

**Evaluating Quality Characteristics in SPLs:** Bartholdt et al. [35] studied the relevance of quality characteristics to the choice of specific features for non-trivial software product lines. They proposed the integration of quantitative and qualitative quality attributes with features by incorporating quality modeling based on UML profile and feature modeling. Their method only includes aggregated quality evaluation capabilities at modeling level, where the overall quality of each product is computed using predefined aggregation functions. Authors discussed that the quality evaluation is required to devise quality aggregation methods and computational algorithms which consider both structural variability and behavioural models. They also argued that holistic modeling and evaluation of quality of SPLs comprising large set of variations face significant challenges raised by these requirements.

Etxeberria et al. [101] discussed the importance of quality analysis of product lines and argued how it can be more scalable through architecture evaluation. To reduce the evaluation efforts, their approach used an extended feature model to identify the variability influencing on quality. In their approach, feature model is extended with additional quality...
feature tree that characterizes quality attributes and utility function for each feature. The impact of individual features on a particular quality attribute is quantified and measured by evaluating product-line architecture through the execution of final configured product. In particular, for performance analysis of architecture, they used execution graph, which is derived from codes, and defined mathematical formula to quantify performance value. Quality formulas include variable quality properties reflecting variable part of the product line architecture; however, it does not address how to compute such those quality values.

Zhang et al. [340] proposed an approach to predict and evaluate quality variants of an SPL architecture using Bayesian Belief Networks (BBN). In their approach, feature model is used to model functional variants, and BBN is employed to represent the design decisions and quality attributes, and predict the impacts of different configurations on product quality. The BBN model includes probability numbers to reflect the expert’s belief of the domain in how much a given configuration influences a quality attribute.

There is limited work in software quality engineering that aims at investigating how to evaluate specific quality characteristics of SPL architectures; for instance, Immonen et al. [150] proposed method for availability and reliability predictions. Olumofin et al. [237] presented a method to examine the selection of variant features on availability, reliability, performance, and modifiability of a product line architecture. They proposed a scenario-based method, as Holistic Product Line Architecture Assessment (HoPLAA), which is an adaptation of the Architecture Trade-off Analysis Method (ATAM) [168] for the area of software product lines. For the quality evaluation of SPL architectures, the ATAM is extended with the qualitative analytical treatment of variation points and the context-dependent generation, and prioritization of quality attributes scenarios.

Sincero et al. [284] proposed a feedback-based approach to predict product’s non-functional properties based on the knowledge base including the quality information and the measurements of product already configured. They aimed at providing qualitative information about how each feature and combination of features impact a certain non-functional property by collecting quality information through the generation and testing products and processing the results. In recent work, Siegmund et al. [281] [282] presented a method for the prediction of non-functional properties in software product line based on product’s feature selection. In their approach, a set of configuration paths that are different in the single feature are generated; then, the variation of quality properties are measured to identify and approximate the influence of particular feature.
The majority of conducted studies outline and highlight the need for approaches to address the complexity of quality evaluation of SPL because of the involvement of large number of quality attributes scenarios. Most existing works only focus on the evaluation of SPL architecture by assessing and predicting the relative impacts of feature selection on quality in a product-line. However, these attempts did not provide solutions, nor does it address how quality variability and quality ranges can be obtained.

**QoS Aggregation and Computation:** There are several contributions and previous studies addressing QoS aggregation in terms of different process model structures for composite services. The works of Cardoso et al. [60] and Jaeger et al. [155] are the seminal works in the literature and basis of our work, which address the aggregation and estimation of QoS values for Web services composition in well-structured process models. Their approaches are based on (some) workflow patterns in the work by van der Aalst et al. [317]. Cardoso et al. [60] proposed Stochastic Workflow Reduction (SWR) to compute and estimate the entire workflow QoS values. The SWR algorithm iteratively applies a set of reduction rules for some sequential and parallel patterns over a given structured process graph. Their proposed algorithm for aggregation makes it possible to predict the QoS performance of the entire process by repeatedly performing substitution until the whole process is transformed into one composite service node. Jaeger et al. [155] proposed a QoS aggregation method which is the most similar to ours. In their approach, quality aggregation of composite Web services is performed by considering workflow patterns and computing upper and lower bounds for QoS values. Authors represent a process model as a graph which is collapsed step-by-step by applying composition patterns. Hwang et al. [147] proposed a probability-based method where a composite service is represented by a process structure which is recursively parsed and analyzed to aggregate quality attributes. Mukherjee et al. [228] focused on addressing the QoS computation in BPEL processes by performing aggregation on an activity graph. In a more recent work, Yong et al. [333] presented an aggregation method that also employs Refined Process Structure Tree technique [251, 321] and supports unstructured process models.

Most of the above studies are related to our proposal. However, existing solutions do not consider the constituent *structural variability* of process models and do not address modeling and managing variation points, which may occur within such an architecture and can significantly impact the proper QoS aggregation. To the best of our knowledge, this is
the first work that takes both structural and behavioral variability into account to evaluate QoS dimensions in the context of SOSPL.

4.6 Summary

In this chapter, we proposed a QoS model and provided a systematic approach for QoS aggregation to address the evaluation of quality ranges supported by SOSPL(s), which further help for quality-aware service-product derivation as well as decision-making and optimization. As the main contribution, we identified a set of variability patterns that may occur within composition patterns and proposed new aggregation rules and a method for QoS range computation for holistic evaluation and prediction of quality ranges of a product-line of services based on configurable process model.

By integrating quality modeling into SOSPL, we incorporate the concepts and constructs to define quality properties from the different decision-makers’ perspectives. The proposed QoS model provides expressive and extensible submodels to specify relations, dependencies, and relative importance of quality characteristics which are indispensable for quality-driven service configuration and optimization.

We presented a feature-oriented approach to analyze variability and composition patterns to aggregate QoS ranges for an SOSPL. We proposed extended feature model to explicate the range of functional variants accounted for in the product line of services and consider the impacts of variable features on quality aspects.
Chapter 5

QoS-aware Process Configuration and Optimization

5.1 Introduction

Configuration has been a fruitful topic of research in artificial intelligence and an outgrowth of research on rule-based expert systems. John McDermott [209] used the term configuration for a particular form of a design task, where a systems is assembled out of a collection of predefined components. Whereas more innovative design tasks often need an appropriate modeling of the behavior of components, different approach and general definition of a configuration task are given by Frayman and Mittal [222]. A configuration problem is characterized by two constituents:

1) A configuration model which describes the components that can be included in the configuration and the rules on how they can be combined to form a working product, and requirements that determine specific properties that the individual product should have.

2) Stakeholder’s preferences about the functional and quality characteristics of the desired configuration.

The configuration tasks consist of representing and reasoning about attributes, structure and connections of components to find one or more combinations that meet all requirements and optimize the preferences.
Configurable applications allow software to be reused for different requirement and constraint sets. As we discussed, SaaS applications in the large scale need to be developed with highly standardized software functionalities to serve as many stakeholder (i.e., service consumers) as possible. In the service family application engineering process to meet particular requirements of stakeholders, the final service is created by tailoring the reference process model that provides common business logic for the orchestration and choreography of services for entire family.

Configuration and Customization are two approaches to tailor a SaaS application to satisfy stakeholder’s requirements \[302\]. Configuration provides the required functionalities by offering alternative options and supporting variance through predefined parameters or configurable scope. Hence, in this context, each configuration can be seen as a variant to satisfy a desired set of end-state requirements. The application can be further tailored based on a set of predefined configuration rules. While customization can involve with operational changes in design or code implementation to create or expand the functionality in order to meet stakeholder-specific requirements. These can not be achieved by configuration. However, customization can be costly and more complicated due to change operations.

Similar to any problem-solving activity, solving a configuration problem requires a representation of problem and an algorithm which produce solution based on the problem representation. The core part of any service-product configuration systems is a service-product model representing and describing features and their quality offered by a product line. Quality-aware configuration enables to offer distinct quality levels for individual configured service product.

In this chapter, we focus on the configuration problem for a service family (SOSPL). We firstly provide an overview of the problem and configuration framework. We then describe how this problem can be modeled, formalized and solved to support decision-making during the course of configuration task.

5.2 Process Configuration and Challenges Overview

Aldanondo et al. \[13\] define software configuration as “1) given a generic model of a configurable product family including all possible variants, in which a generic model is a set of components plus a set of various constraints; and 2) given a set of stakeholder’s requirements, in which each requirement can be expressed by a constraints restricting the possible
CHAPTER 5. QOS-AWARE PROCESS CONFIGURATION & OPTIMIZATION

combination of components and/or component quality values. Configuration is finding at least one component set that satisfies all constraints”. A configurable process model renders a consolidated view of a service family. It provides multiple variants of a process describing the control-flow and data-flow among the services and composition of services. One of the fundamental challenges is to cope with process variability and the large number of variants that may exist and are captured in terms of features. Feature reflects the incremental functionality in a family of services and offers a configuration options that derive different variants of a service products. The number of features can be very high in a service family, easily overwhelming the service vendors by the huge host of decisions to derive specific service product. Hence, for different stakeholders, they have to be concerned about the selection of features which can get increasingly complex by dependencies between features and their variants relating decisions.

The key challenge is that service providers should support decision-making mechanisms during the course of configuration to achieve the optimal solutions within the constraints boundaries specified by stakeholders and configurable limits.

As discussed formerly, feature model leverages the feature notion as an abstraction to describe configuration/decision space for a service family. In the context of service family, it enables to create configurable process model to derive different service products with different functionality and quality specification. Because a feature model is a generic representation of all possible configuration, the selection of a set of desirable features from the feature model yields different variants of service products (i.e., SaaS applications). However, the number of possible configurations of a feature model increases exponentially with the size of the feature model. In the context of SPL, this problem causes one of the main challenges with regards to feature model configuration including feature selection. Indeed, the number of possible configuration by considering the combination of features increases exponentially with the size of features in a feature model even for a small models. Theoretically, a product line of services with \( n \) features can impose complexity of \( O(2^n) \) for feature selection in the course of configuration. Therefore, it is impractical to manually configure and derive the solution for targeted individual stakeholders. In view of the fact that each feature can be realized and implemented by service components with the same functionality albeit different quality, variable requirements and imposed constraints complicate the selection of appropriate existing service components.

One of critical challenge is how to solve the configuration problem when encountering
resource constraints or non-functional requirements. For instance, the QoS in shape of performance or total budget for the final service product which the stakeholder demands can lead to the complication of configuring a family of services. In the context of SPL, the feature selection and optimization with resource resembles the configuration optimization problems. Accordingly, the configuration problem that involves finding an optimal selection of features and service components that conform to both feature model and resource constraints is an NP-complete problem. Diminishing the complexity of selection of features and corresponding services meets our goal which is how to keep the feature selection proportional to the limitations of the resource or constraints and how to enable service vendors with a configurable process model and automated decision support. In the following section, we describe a formal foundation and theoretical background to define and provide solutions for configurable process model.

5.3 Problem Description

As we discussed earlier, a configuration may involve setting a collection of parameters, selecting a set of features, or more generally, selecting variation points in systems. In the context of SOSPL, we refer specifically to the high-level configuration of business process models. Therefore, we define the configuration problem as the selection of the most desirable features which capture aspects of functional value for stakeholders (i.e., service consumer), and further the delegation of relevant service implementations that would collectively maximize the stakeholders functional and non-functional (QoS) requirements.

Configuration problem often needs starting at an arbitrary or particular state and deriving a new configuration that satisfies and conforms the requirements. In configuration task, stakeholder firstly specifies requirements and her/his preferences. The concrete configuration is often carried out through the selection of features in multiple stages to form a complete configuration iteratively to meet requirements owing to the fact that in practical scenarios, for instance, stakeholders may want to select preferred customized product from a recommended set. Hence, configuration task can be seen as a step-wise refinement of the constraint defining the configuration space. This process involves with the transformation of configuration starting point through a series of intermediate configurations to a configuration that satisfies a desired set of functional and non-functional requirements. We hereafter call this sequence of activities as Multi-Staged Process Configuration (MSPC)
problem.

In the configuration task, i.e., the process of service derivation from a reference process model, we distinguish between stakeholder’s functional and non-functional requirements determined through requirements analysis phase (cf. Figure 3.5-A1). To effectively configure and customize a reference process model for the target application product with respect to (non-) functional requirements, the following issues should be addressed and resolved:

I) What features should be included in the final service product in order to optimally satisfy the need of stakeholder’s functional and non-functional requirements

II) What services should be chosen best from a given set of functionally-equivalent service candidates, which provide operational implementations for features

In this context, in correspondence of a service specification (herein referred to as abstract service), very often several services (herein referred to as concrete services) may exist with the same functionality but different QoS that match the specification (cf. Figure 5.1). We define the configuration problem as “How to find an optimal decision that selects the right set of features and service implementations based on constraints defined in system and stakeholder’s preference and objective”.

Figure 5.1: A configurable process model from an abstract view.

We consider four major requirements as perquisite for the configuration of a configurable process model:

1. functional properties expressed by features mapped to process-model activities delegated further to a set of concrete services;
2. non-functional properties related to service specification (i.e., QoS);

3. stakeholder preferences concerning functional and non-functional properties to be considered for alternative configuration options, and

4. optimization supported by the configuration approach, enabling decision supports and maximizing the profits with respect to stakeholder’s objective and preferences (profit optimization).

In addition, we require correctness, verification and consistency checking to support further customization involving changes in configurable process models, which are out of the scope of present thesis.

Modeling and reasoning (i.e., ranking) of stakeholder’s preferences is a prerequisite for any kind of further decision-making. Very often, stakeholders specify their preferences and constraints qualitatively, which are then mapped on to a quantitative utility model. Hence, it also become a non-trivial issue when stakeholder conditionally specifies preferences (i.e., relative importance). For instance, stakeholder may specify that security is more important than the cost, and etc. To address the incorporation of stakeholder’s preferences about quality requirements into the process of decision-making in the course of configuration, our proposed configuration framework utilizes the widely used decision analysis theory of analytic hierarchy process (AHP) \cite{269}. It provides a method to represent and prioritize (rank) stakeholder’s requirements related to the services with respect to the high-level concerns of stakeholder and his/her conditionally-defined preferences over these concerns.

The following section firstly gives an overview of the process configuration approach, which is followed by details descriptions of proposed configuration framework and an approach devised to address the MSPC problem.

### 5.4 Approach Overview

In our approach, the process configuration is guided by the feature model that encapsulates the knowledge of configuration and describes valid and complete configuration space. This leads to two abstraction layers of configuration as depicted in Figure \ref{fig:5.2}: 1) feature model layer and 2) business process layer, which are configured in the course of service-application engineering.
The *process layer* includes reference process models which reflect behavioural variability of a service product family. The reference process model describes the orchestration and choreography of activities (i.e., abstract service) which are further assigned and delegated to concrete services. The *feature model layer* includes feature model which expresses structural variability (i.e., variation points) exist in the reference process model and derives the configuration.

![Configuration layers diagram](image)

**Figure 5.2: Configuration layers.**

In the course of service-application engineering, at feature model layer, our proposed
approach follows the staged configuration [80]. The process model configuration is accomplished through a series of successive stages of feature model specialization. In each stage, a subset of preferred features is selected and unnecessary features are discarded. It can be fulfilled by manual or automatic selection of features corresponding to stakeholders’ business objectives and preferences over functional and non-functional requirements. This yields a specialized feature model whose set of possible configurations is a subset of the feature model.

The specialization of the feature model tailors the reference process model. At process layer, there is still one hurdle to overcome to accomplish the final configuration, and that is still-unresolved variability originating from implementations of the previously specialized feature model. As already explained, each activity in a reference process model may have different implementations provided by a set of service candidates with the same functionality and varying QoS properties. The configuration is followed by selecting and binding appropriate service for each activity mapped to feature by taking into account constraints formerly specified by stakeholder.

The goal of QoS-aware optimization is to formulate and solve the MSPC problem (Section [5.7]). We propose a Multiple Criteria Decision-Making (MCDM) approach to achieve the optimal selection of appropriate features and best service candidates where functional and non-functional requirements are satisfied within the constraints boundaries specified by stakeholders. The optimization is performed by considering the stakeholder’s preferences about QoS. We model the problem of MSPC as a Constraint Optimization Problem (COP), formulated as Mixed Integer Linear Programming (MILP). This model can be further solved by well-developed mathematical programming techniques. The conceptual configuration framework and its core components are illustrated in Figure 5.3. The inputs are abstract specification of the reference process model, feature model and decision-maker’s preference. The feature model analyzer module extracts the feature relations and transfers them to MILP model formulation. The process model analyzer generates service dependency and assignment constraints which are specified in the process model. The preference prioritization module provides a ranking procedure to compute the relative importance of quality criteria. The MILP model formulation is computed, and the optimization problem for MSPC is solved by optimization module. The output is a configured feature model which guides the configuration of the (reference) process model.
In the following sections, we firstly describe about stakeholders’ preferences and prioritization of quality requirements. We then establish background about constraint optimization problem and notation for the reminder of the thesis. It is followed by detail descriptions of our proposed approach that formally models MSPC problem to obtain effective process configuration to meet a series of requirements.

5.5 Preference and Requirements Prioritization

The critical requirement for successful implementation of intelligent decision-support systems is dealing with individual stakeholder’s preferences in terms of the relative importance of criteria or objectives. The relative importance of criteria is commonly characterized by quantized weights and represented by a set of positive real-valued importance parameters.

Multiple stakeholders have distinct preferences toward optional features and QoS. As a consequence, the choice of the “best suited” alternative features from a pool of feature satisfying all the required criteria is cumbersome and tedious task. The difficulty stems
from the fact that various optional features may also have different values of elements, and not a single feature would possess all values satisfying stakeholder’s requirements. Hence, preference elicitation requires to be undertaken to capture specific stakeholder’s preferences to a sufficient degree to further enable an optimal or close to optimal decision to be taken. Accordingly, modeling, prioritizing and reasoning preferences can help to compute most preferred course of action based on the available alternative features and stakeholder’s requirements. Effective automated preference elicitation methods have been one of recent important research in AI [91].

Modeling and prioritization of preferences are prerequisite for any kind of further decision-making and advanced issues for service configuration. Owing to the fact that preferences or relative importance cannot be expressed in a binary way and easily enumerated in terms of an explicit list. Furthermore, naive stakeholders do not have the knowledge of configuration model, because stakeholder often encounter under- or over-constrained situations in the course of specifying the requirements. Stakeholders may have neither the ability nor the patience to precisely and fully express what utility information they need. Also, decision-maker (i.e., configurator) need full objective to make good and sound decisions configuration-wise. In addition, multiple stakeholders usually specify their preferences qualitatively in particular order at various levels of relative importance or may impose constraints in terms of conditions. For instance, particular stakeholder may state that “the security is more important than response time and availability”. Stakeholder also may express her preferences concerning QoS requirements conditionally, i.e., her willingness to pay more depends on the high security and low response time. Therefore, for some stakeholders a subset of QoS dimensions could be more important than the other dimensions by taking into account business objective and preferences individually.

Accordingly, configuration process inevitably should cope with the problem of effectively reasoning the relative importance or preference of stakeholder and deal with the presence of imprecise or partial preference information. With respect to further optimization in the presence of imprecise preference and objective information, we assume that a set of bound (or more general linear constraints) on objective function parameters are provided.

In this section, we focus on a preference model and a framework to quantitatively aid the prioritization and weighting of optional features and QoS criteria corresponding to stakeholder’s business goals, high-level objective requirements expressed in form of preference.
The outcome will be further incorporated into the optimization and configuration of reference process model to derive and evolve an effective service product configuration according to stakeholder’s objectives.

5.5.1 Preference Structure

Exploiting and incorporating preference into decision-making process imperatively requires a model which express both qualitative and quantitative preferential aspects. The stakeholder’s requirements can be gathered and elicited by a variety of methods such as interviews, questionnaires, task analysis which aim at asking customers to express their needs of a service product. Requirements are usually expressed in stakeholders words. Even though they are likely too general to be directly used as requirements; however, through sorting, classifying and structuring the requirements are finally obtained. Accordingly, decision-making requires to understand the preference structure of stakeholder through a number of preferential statements. In general, decision-making is the process of selecting among alternatives courses of action to attain an objective. Each alternatives is defined by a set of criteria which are characterized by attributes. Preference is the relative importance of attributes.

We further specify high-level objective and preferences of an stakeholder related to functional and non-functional (QoS) requirements as criteria (or attributes) which are associated to stakeholder’s concerns from stakeholder point of view. In the following we use the term criteria and concern interchangeably. Hence, we represent non-functional requirements and their priorities as the preferences described in terms of expected quality properties and their mutual relations; and (hard or soft constraints) over the values of quality properties. The representation builds on the Preview framework, a well-known in requirements engineering. Examples of criteria or concerns are implementation cost, security, response time, reliability for domain-independent QoS attributes, and usability, penalty rate, computation time for domain-specific ones. Thus, the concerns of stakeholder are primarily identified, and the options that require to be prioritized are further associated with those concerns.

Generally speaking, decision theory is based on representation theorems of measurement which constitute a crucial aspects in dealing with preferences. The possible optional values

\[\text{\textsuperscript{1}The concept of concern is adopted from viewpoint-oriented approaches to requirements engineering as the high-level strategic objectives of the application domain and product family stakeholders.}\]
for a preferential criterion (concern) can be specified by single numerical value or intervals, for instance, “x is preferred to be between 5 and 50” and “y between 3 and 35”. However, decision-making is a typical human mental process in nature, and decision makers (i.e., stakeholders) usually tend to represent and stipulate their preferences by linguistic quantifier or (fuzzy) lexical information rather than in numerical values. In our proposed model, we refer to linguistic labels as qualifier and which are more appropriate to express and represent the imprecise preference relations and approximate preference values – too complex to be represented using precise numerical values. Case in point is the set of qualifier tags like as strongly, very strongly, and extremely important expresses the degrees of preferences, and the set of quantifier tags such as cheap, moderate and expensive implies fuzzy linguistic-interval values associated to a criteria (e.g., cost).

The preference expresses the order of criteria, and the relative importance information retains the relationship between criteria and the property which stakeholders value in accordance with their perceived importance. Thus we model preference relations as partial order of criteria in terms of importance.

Table 5.1: The numerical and linguistic scales for importance [269].

<table>
<thead>
<tr>
<th>Numerical scale</th>
<th>Linguistic scale</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Equal importance</td>
<td>Two criteria contribute equally to the objective</td>
</tr>
<tr>
<td>2</td>
<td>Weak importance</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Moderate importance</td>
<td>Experience and judgment slightly favor one element over another</td>
</tr>
<tr>
<td>4</td>
<td>Moderate plus</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Essential or strong</td>
<td>Experience and judgment strongly favor one criterion over another</td>
</tr>
<tr>
<td></td>
<td>importance</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Strong plus</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Very strong importance</td>
<td>Favored very strongly over another; its dominance demonstrated in practice</td>
</tr>
<tr>
<td>8</td>
<td>Very, very strong</td>
<td></td>
</tr>
<tr>
<td></td>
<td>importance</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Extreme importance</td>
<td>The evidence favoring one element over another is of the highest possible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>order of affirmation</td>
</tr>
</tbody>
</table>

The majority of preference models are based on linear and weak orders which are special cases of partial orders with the linear order being an arrangement of criteria from the best to the worst one without any ex acquo, and with the weak order describing the indifference relation as an equivalence relation (reflexive, symmetric, and transitive). A well-known
problem with both linear and weak orders is that the associated indifference relation has to be transitive. However, some of these models don’t regard indifference as being transitive, and such a property may be violated in the presence of thresholds.

We denote the relative importance relationship (preference relations) by priority operator $\alpha >$ specifying the precedence or the quality of being more important, where $\alpha$ is the degree of importance associated with qualifier tags, also referred to as relative importance weight which, in the quantitative approach, is often specified indirectly using scoring function. We use fundamental 9-point scale which is typically used to represent the intensities of judgment and to provide estimates of the ratios in the pairwise comparison of criteria. Every criterion is based on a variable considered as a reason for preference. Table 5.1 shows the numerical and linguistic scales defining the intensity of importance.

*Example 2.* Lets consider two preferentially independent criteria related to QoS ‘x’ and ‘y’. The assertions of $x \alpha > y$ implies that for the individual ‘x’ is considered more important than ‘y’ with relative importance weight $\alpha$. Lets assume three qualifier tags as low, medium, and high representing available interval values for criterion set $C = \{\text{cost, security, availability, reliability}\}$.

The assertions security.high $\alpha >$ cost.low and, security $\alpha >$ reliability $\alpha >$ availability implies that for particular stakeholder granting high security is slightly more crucial than low cost; the high security and reliability are equal in terms of significance; and the reliability is much more notable than availability.

*Definition 7 (Criterion).* A criterion $c$ is a triple $(X_c, \alpha >, D_c)$ specifying a preference ordering on alternative values $g_k(X_c)$, where $X_c$ is a variable for criterion with domain $D_c$ from which its value is chosen, and $\alpha >$, relative importance on value of $X_c$ with degree of $\alpha$, is preorder on domain $D$ iff $g_i(X_c) \alpha > g_j(X_c)$.

As an example, criteria ‘economy’ and ‘responsiveness’ can be defined as $(Price, \alpha >, Z^+)$ and $(Execution time or Latency, \alpha >, D)$, where the lower cost values of service is much more preferred than higher cost, and the higher values are significantly preferred for either execution time or latency. However, the relative importance of values can be diverse for different individuals because of subjective nature of preference, e.g., particular stakeholder may have either equal or no preference ($\alpha >$) regarding low or extreme cost value of service.

On the account of the fact that individual’s decision may be based upon multiple criteria,
single criterion can be combined into compound criteria to determine the overall preference. We define a preference structure as following:

**Definition 8 (Preference Structure).** A preference structure is a quadruple $PS = (C, A, D, K)$. $C$ is a set of criteria, $A$ is a finite set of variables associated to criteria. Every variable $X_c \in A$ has domain $D_c \in D$ of possible alternative values, and $K$ is a knowledge base includes sets of constraints on the assignment of values to variables.

Accordingly, overall preference of stakeholder is defined as determining the relative importance of values assigned to criteria variables such that no constraints declared are violated. Consequentially, the overall relative importance over a set of criteria or concerns is defined as follow:

**Definition 9 (Overall Relative Importance).** Given a set of criteria and set of alternatives $A$, the overall relative importance relations $\succ$ is a subset $R \subseteq A \times A$.

### 5.5.2 Conditional Stratified Analytical Hierarchy Process

Our goal is preference-based configuration and supporting stakeholders to find the most suitable set of options for their service application by firstly prioritization (ranking) all alternatives based on their preferences and objectives. Thus, the prioritization of QoS criteria and optional features can be viewed as the process of determining the relative importance of QoS dimensions and features.

Stakeholder specifies requirements, and criteria are ranked depending upon the best possible match. This ranking relies heavily on the weights on the basis of stakeholder’s judgment. The judgments may reflect properly when the choice of selection is few. Nevertheless, the subjective judgment process is susceptible to lead to improper service configuration when the configuration space is too large. It might be quite likely the judgment of stakeholder may be biased merely towards the key elements when specifying relative importance or assigning weights. This results in improper priority and weight being assigned to features and QoS dimensions. As a result, it is prerequisite to have quantifiable values in the place of subjective opinions in order to make informed decision. *Analytic Hierarchy Process* (AHP) [269] is one of the widely-used decision-analysis methods, which can be utilized to objectify subjective evaluation of multi-criteria decisions. It is quantitative approach based on relative judgment with the assurance of consistent outcome. This method has advantages
over ad-hoc methods, and its procedure basically consists of three phases:\cite{269}: 1) **problem decomposition**: the high-level objective is logically decomposed into a hierarchy structure and homogeneous elements including the overall goal, criteria, sub-criteria, and the final alternative solutions; 2) **comparative judgment**: pairwise comparisons of criteria based on a scale scheme is performed to form the comparison matrix determining their relative priorities and final rank (i.e., weights) for each criteria. The elements at every level of hierarchy are compared one on one with every other element to decide the importance of one element over the other. The consistency checking is also enacted for each comparison, and if the results are inconsistent, the comparison is repeated by the decision-maker to achieve consistency. The comparison matrices are normalized, and eigenvectors indicating priorities are computed. Pairwise comparisons lessen conceptual complexity thus reducing cognitive loads; and 3) **priority synthesis**: the relative ranks of all (sub-) criteria are combined to constitute the global ranking value of the alternatives.

AHP has a formal way to deal with inconsistency problem which improves the consistency of judgment and prevent arbitrary responses\cite{269}. The conventional AHP suffers from the quadratic complexity, and it is also inadequate to explicitly capture the importance assessment for stakeholder’s requirements.

The conventional AHP does not support conditional preferences. Hence, we utilize the Conditional Stratified Analytic Hierarchy Process (CS-AHP)\cite{93,236} as an extension to AHP to address different kinds of conditional preferences which may include partial information. For instance, stakeholders are often aware that having low-cost customized service may not always be possible, so they may set some conditions, e.g., they are willing to pay for a high-cost service if the customized service offers high security and fast response. CS-AHP enables to conditionally express the preference and mitigate the number of required comparisons among the available options. In the following we review the CS-AHP stages for the prioritization of quality criteria. This approach can be utilized also to rank the optional features; however, we focus on prioritization of quality requirements (quality concerns).

In the first stage, the stakeholder defines the preferences, which include the definitions of concerns and qualifier tags for each concern. The stakeholder is also allowed to express conditional preferences about the relative importance of the same pair of concerns or qualifier tags. In the next stage, CS-AHP performs a tuned pair-wise comparison. The outcomes of the procedure are: (i) the local ranks of tags, i.e., ranking of qualifier tags within one concern, and (ii) the global ranks of concerns. In addition, the global ranking of qualifier tags are
computed by multiplying the global ranks of concerns with the local ranks of qualifier tags. These rankings are used as the main instrument to measure the level of satisfaction of the stakeholders’ requirements which are utilized in the course of configuration and optimization to derive the final service product.

Let \( v_i \) be the aggregated value of the \( i^{th} \) concern, such that \( qt(v_i) \) is the qualifier tag of the concern; \( w_{qt(v_i)} \) being the local rank for the qualifier tag \( qt(v_i) \); \( L_{qt(v_i)}, U_{qt(v_i)} \), and \( m_{qt(v_i)} \) being the lower bound, upper bound and middle values of the interval of values which correspond to \( qt(v_i) \); \( next_{qt(v_i)} \) and \( prev_{qt(v_i)} \) being the first neighbor qualifier tags on the right and left side of \( qt(v_i) \), respectively. If \( qt(v_i) \) is the final left (or right) qualifier tag, the values of \( \text{prev}_{qt(v_i)} \) and \( \text{next}_{qt(v_i)} \) are considered to be 0 and 1, respectively, in the case of an increasing or decreasing characteristic. In order to deliver a rank for \( v_i \) we make a uniform distribution of the ranks of the observed qualifier tags to the ranks of the first neighbor qualifier tags \( (next_{qt(v_i)} \) and \( \text{prev}_{qt(v_i)}) \). The global rank of the aggregated value \( v_i \) of concern \( c_i \) is calculated according to the basic characteristic of CS-AHP: as a multiplication of \( w(v_i) \cdot w_{c_i} \), where \( w_{c_i} \) is the rank of concern \( c_i \).

\[
\begin{align*}
  w(v_i) &= \begin{cases} 
    w_{qt(v_i)} + \frac{v_i - m_{qt(v_i)}}{U_{qt(v_i)} - m_{qt(v_i)}} (w_{next_{qt(v_i)}} - w_{qt(v_i)}), & 
    v_i \in [m_{qt(v_i)}, U_{qt(v_i)}] \\
    w_{qt(v_i)} - \frac{m_{qt(v_i)} - v_i}{m_{qt(v_i)} - L_{qt(v_i)}} (w_{qt(v_i)} - w_{prev_{qt(v_i)}}), & 
    v_i \in [L_{qt(v_i)}, m_{qt(v_i)}] 
  \end{cases}
\end{align*}
\]

(5.1)

*Example 3.* For instance, let us assume the following service quality-range values (regardless of units) are available for the *response time*, *cost* and *availability* as well as their associated qualifier tags: *response time*-range \([15, 55]\), where Low \(< 20\), Medium \([20, 40]\), and High \(> 40\); *cost*-range \([100, 350]\), where Low \(< 200\), Medium \([200, 300]\), and High \(> 300\); and *availability*-range \([5, 20]\), where Low \(12\) and High \(> 12\). To illustrate, let the preferred values for the three QoS concerns--response time, cost, and availability--be equal to 19, 275 and 15, respectively. Hence, those given values belong to the ranges specified as *response time*.low, *cost*.medium, and *response time*.high. We assume that CS-AHP gives the rank for concern throughput:0.54 and its qualifier tags: *response time*.low, *response time*.medium
and \textit{response time.high} as 0.16, 0.65, and 0.19, respectively. Given that the rank of 0.65 is defined for the whole interval [15, 20], we assign it to the middle value of the specified interval (i.e. to 17.5). The uniform distribution of ranks from 0.65 to 0.19 (the rank of medium response time) over values 17.5 to 20 yields the rank to the value of 19 as follows: 
\[
0.65 - \frac{19 - 17.5}{20 - 17.5} \cdot (0.65 - 0.19) = 0.37.
\]
Finally, the global rank for the response time of 19 is calculated as follows: 
\[
0.54 \cdot 0.37 = 0.20.
\]

Having defined preferences about the criteria in terms of concern and qualifier tags, we construct the preference structure, and the relative importance of criteria are computed by CS-AHP process which are used in the course of QoS-aware optimization.

5.6 Constraint Satisfaction and Optimization

A wide variety of real-life computationally-intractable problems can be formulated and solved as \textit{Constraint Satisfaction Problems} (CSP) which have been an important research topic in Artificial Intelligence (AI) and Operational Research (OR). A problem consists of requirements and constraints on variables with permissible and acceptable variable assignments corresponding to solutions to the problem. Constraints are relations, and a CSP determines which relations should hold among the given variables.

\textit{Constraint programming} (CP) is a paradigm by which to solve CSP. It involves a combination of computing and logic to reason about variable assignment problems. It is widely used with great success to tackle with various large-scale combinatorial (optimization) problems in many domains, such as planning, configuration, scheduling and resource allocations \cite{267, 314, 332}. Furthermore, recent and ongoing research, notably from operational research (OR), has broaden the scope of the field promisingly, and industrial interests in CP techniques also endures to provide a significant impetus. In theory, CP generally consist of two phases: 1) modeling or formulating a problem, and 2) solving the problem by means of general or domain-specific methods that basically use logical inference to save the search space, such techniques as domain reduction and constraints propagation \cite{314}.

It has been shown that the class of all constraint satisfaction problem instances is NP-complete \cite{200}. Therefore, even though it is unlikely that general-purpose algorithms exist to solve all forms of CSPs, the instances with particular forms in many practical applications can be solved more efficiently \cite{267}. Moreover, there are problems that do not directly correspond to CSPs. However, such problems may include CSPs as subproblems, for
instance optimization problems, decomposable into constraints describing what a solution is and an objective function (i.e., cost or profit function) to determine the quality of solutions, leading to the concept of constraints to the concept of Constraints Optimization Problem (COP).

### 5.6.1 Basic Definitions

A CSP (or a constraint network [200]) problem can be defined as a set of finite set of decision variables, each of which must be assigned a value from a given domain, and a set of constraints, each of which limits the set of allowed combination of variable assignments. Solving a CSP consists of finding a value for each variable from its domain so that all constraints are satisfied. A CSP is defined formally as follows:

**Definition 10 (CSP).** A constraint satisfaction problem (CSP) is a triple \( P = (X, D, C) \) representing a combinatorial decision problem \( P \), where \( X = \{x_1, \ldots, x_n\} \) is a set of variables, with respective domains \( D = \{D_1, \ldots, D_n\} \). Each domain \( D_i \in D \) is a finite set of possible values for variable \( x_i \in X \); \( C = \{c_1, \ldots, c_m\} \) is a finite set of constraints. Each constraint \( c_i \) is a pair \( c_i = \langle S_i, R_i \rangle \), where \( S_i = (x_{i_1}, \ldots, x_{i_k}) \), the constraint scope, is a \( k \)-tuple of variables, and \( R_i \subset D_{i_1} \times \ldots \times D_{i_k} \), the constraint relation, is an \( m \)-ary relation over \( D \).

It should be noted that both domains and constraints can be of arbitrary type. The search space in a CSP is \( O(|D|^{|X|}) \), where \( |D| \) is the domain size and \( |X| \) is the number of variables in the problem. The size of an instance \( P \) can be defined as \( |X| + |D| + \sum_{c \in C} \|c\| \), where \( \|c\| \) is the size of a constraint \( c = \langle S, R \rangle \).

**Definition 11 (Assignment).** An assignment is a mapping \( \alpha : X \rightarrow \bigcup_{i \in [1,n]} D_i \) such that \( \alpha(x_i) \in D_i, \forall i \in [1,n] \). An assignment satisfies a constraint \( (\alpha \models c) \) if it gives a combination of values to variables that is permitted by the constraints.

A partial assignment is a collection of assignments to a subset of the variables, and a complete assignment is an assignment for every variables in the problem space. An instance of satisfiability problem is specified by giving formula in propositional logic that can be expressed as a conjunction of clauses. It determines if there are values to be assigned to the
variables which makes the formula true.

**Definition 12 (Satisfiability).** Let \( \phi = c_1 \land \ldots \land c_m \) be a logical formula in conjunctive normal form, where each clause \( c_i = l_{i1} \lor \ldots \lor l_{ik_i} \) is a disjunction of literals. A literal \( l \in L = \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\} \) is a variable \( x_j \) or its negated form \( \overline{x}_j \). The satisfiability question, as the instance \((X, \{1, 0\}, C)\) of CSP, consists of testing if clauses in \( \phi \) can all be satisfied by some consistent assignment of truth values \( \alpha(x) \in \{0, 1\}^n \), such that the formula \( \phi \) is satisfied, i.e., each clause \( c_i \) is evaluated to 1; otherwise, \( \phi \) is unsatisfiable for all \( x \in \{0, 1\}^n \) at least one \( c_i \) evaluates to 0.

A solution to a CSP is a complete set of assignments of values to variables, such that all constraints are satisfied:

**Definition 13 (Solution).** A valid solution of an instance of \( P \) is a complete assignment \( f(X) \), such that \( \forall (S_i, R_i) \) with \( S_i = (x_{i1}, \ldots, x_{ik_i}), (\alpha(x_{i1}), \ldots, \alpha(x_{ik_i}) \in R_i) \).

If all the constraints must be satisfied mandatorily in a solution, these constraints are known as **hard constraints**. On the other hand, **soft constraints** are those that we prefer a solution to satisfy, but are not mandatory. A constraint is **relaxed** if further elements are added or modified in its relation. A constraint satisfaction instance is **satisfiable** or **consistent** if there is at least one solution and it is **under-constrained** with more than one solutions.

### 5.6.2 Constraint Optimization Problem

The CSP paradigm can be both specialized and generalized in different ways. For instance, the boolean satisfiability problem (SAT), is one of specialization of CSP, where domain is limited to \( D \in \{T, F\} \), and the overall constraint is conjunction of a set of clauses \[199\]. Also, various generalizations of CSP have been developed; however, one of the most significant is the **Constraint Optimization Problem** (COP). In practice, we may search for one solution, all solutions, or often for some of the best solutions regarding a given criteria, subject to hard and soft constraints. Seeking the best solution among all feasible solutions leads to an optimization problem. A COP is a CSP with either one or multiple functions, formulated in terms of **decision variables**, required to be optimized (i.e., minimized or maximized) with the possibility of additional restrictions, or constraints, on the values that the decision variables
may attain. A function can be given a variety of names: \textit{objective function}, \textit{utility function} (maximization), \textit{cost function} (minimization) or \textit{scoring function}, that encodes the desired objectives and measures relative satisfaction concerning the decision problem. The stakeholder’s \textit{preferences} are reflected by an objective function. In another words, preferences can be regarded as \textit{soft constraints} (i.e., cost and performance) which are required to be either minimized or maximized by objective functions.

In some problems, we may like to optimize a number of different objectives concurrently, e.g., minimizing the cost and maximizing the response time and reliability of systems. There for the primary goal of multi-objective optimization is to model a decision-maker’s preferences (relative importance of objectives and goals). The different objectives often are not compatible because the variables that optimize one objective may be far from optimal for the other. In practice, a problem with multiple objectives can be reformulated as a single-objective problem. This can be achieved by either forming a weighted combination of the different objectives or treating some of the objectives as constraints.

\textbf{Definition 14} (Objective function). An objective function $F: \mathbb{R}^n \rightarrow \mathbb{R}$ is a function defined from variables $X$ to a totally ordered set $\langle T, \leq \rangle$, that maps each assignment to a value.

An assignment $\alpha^i$ is preferred to the assignment $\alpha^j$, if $f(\alpha^i) < f(\alpha^j)$ - the value of the objective function under $\alpha^i$ is less than the value under $\alpha^j$. The standard form of an optimization problem is stated conventionally in terms of minimization. Maximization problem can be transferred into an equivalent minimization problem by simply negating the value of objective function. A constraint optimization problem (COP) is defined formally as follows:

\textbf{Definition 15} (COP). A COP is a quadruple $P = (X, D, C, F)$, where $P$ is a CSP, $X$ is a set of variables, with respective finite domain set $D$, $C = C_H \cup C_S$ is set of hard and soft constraints, and a set of objective functions $F = \{f_1, \ldots, f_m\}$ where each $f_i(d_{i,1}, \ldots, d_{i,j})$ is function $f_i: D_{i,1} \times \ldots \times D_{i,j} \rightarrow \mathbb{R}$. The problem is to find an assignment $A^\alpha = \{d_1, \ldots, d_n | d_i \in D_i\}$ such that minimizes $F$ subject to constraints set $C$.

An \textit{optimal solution} is a complete assignment that either minimizes (maximizes) the objective function. A problem with no feasible solution is called \textit{infeasible}. On the other hand, the problem called \textit{unbounded} if feasible solutions have no optimal solutions because
of possible infinitely good objective function values with feasible solutions.

5.6.3 Integer Programming

Integer Programming (IP) is one of the most commonly applied class of COP, which expresses the optimization of a mathematical function subject to a set of linear constraints where some or all of the variables should exclusively be integers. The term also refers to Linear Programming (LP) in many settings and is called a pure integer-programming problem when all decision variables must be integers. Despite these restrictions in modeling, LP is shown to be very successfully applied to many practical real-world and operation research problems [64, 332]. In this contexts, the term “programming” traditionally refers to planning activities that consume resources and/or meet requirements, i.e., the process of determining a mathematical model or plan of actions to allocate the resources among all feasible alternatives to meet requirements in an optimal way.

The theory of LP is concerned with a method to optimize a linear objective function, subject to linear equality and linear inequality constraints that express requirements in a given model [278, 332]. From the complexity point of view, even though COPs are generally NP-hard [314, 332], LP problems are shown solvable in polynomial time [165]. Owing to the fact that, for LP problems there are necessary and sufficient optimality conditions to check efficiently if a given feasible solution is an optimal solution or not. These optimality conditions have been used to develop algebraic methods such as the simplex method and other methods for solving LPs [64, 332].

5.6.4 Mixed-Integer Linear Programming

An LP with integer restrictions on some or all of the variable values is known as Mixed-Integer Linear Programming (MILP). Thus, variables can be combinations of continuous and discrete decisions. A general MILP problem is expressed conventionally in the following form [332]:

\[ z = \min c^T x \]
\[ \text{s.t.} \quad Ax \leq b \]
\[ l \leq x \leq u \]
\[ x \in \mathbb{R}^n \]
\[ x_j \in \mathbb{Z} \ \forall j \in I \]

where \( A \in \mathbb{Q}^{m \times n} \) is an \( m \) by \( n \) matrix, \( c \in \mathbb{Q}^n \) is an \( n \)-dimensional row vector, \( b \in \mathbb{Q}^m \) is an \( m \)-dimensional column vector, \( c^T x \) is the \textit{objective function}, \( Ax \leq b \) the \textit{linear constraints}, \( l \in (\mathbb{Q} \cup \{-\infty\})^m \) and \( u \in (\mathbb{Q} \cup \{+\infty\})^m \) are \textit{lower} and \textit{upper} bounds on the problem variables \( x \), \( I \subseteq \mathbb{N} \) is the subset of indices denoting the variables required to be integer. Integer variables with bounds \( x_j \in \{0, 1\} \) are called \textit{binary variables} and play a special role in MILP modeling, which are a very important tool to model “yes/no” decisions. The feasible solutions of MILP includes vectors in the variable set \( X = \{ x \in \mathbb{R}^n | Ax \leq b, x_j \in \mathbb{Z} \ \forall j \in I \} \). A feasible solution \( x^* \in X \) of MILP is \textit{optimal} if its objective value satisfies \( c^T x^* = z \).

The algorithmic approach to solve MILP problems heavily relies on iterative solution of LP \textit{relaxation}, as general-purpose techniques. LP relaxation of an MILP problem removes the integrality requirement on the \( x \) variables in the set \( I \). Therefore, the relaxation efficiently and practically transforms MILP problem, as an NP-hard integer-programming problem, into a related LP problem which is polynomially solvable and general-purpose techniques for its solution.

The modelling of a complex problem using linear-programming and formulating its description should be performed systematically \cite{218,332}. This requires a clear distinction between variables and the data of problem instance in the model. In \cite{332}, Wolsey describes and urges the following three-step looped process in order to model MILPs: (i) defining a set of \textit{variables} representing the necessary decision variables in a model; (ii) using decision variables to model the set of \textit{logical constraints}; and (iii) defining the \textit{objective function} using decision variables.

Defining variables and constraints may not always be recognized completely in the course of model construction, and initial set of decision variable may be inadequate. Thus, to
formulate constraints and objective functions adequately, additional or alternative sets of new decision variables may be requisite to be defined iteratively during the construction of model. The correct definitions of decision variables and constraints may not always be straightforward and can be complicated in modelling with integer variables [332].

5.7 QoS-aware Optimization

As discussed in Chapter 1, the notion of feature reflects stakeholder’s requirements and defines service product properties. Configurable process model enables stakeholders to choose among many options on how to use the service with different range of desirable quality.

Nevertheless, an optimum configuration by evaluating all the permissible configuration options and selecting the best configured service product is combinatorial in nature, requiring exhaustive inquiry in configuration space. Assuming simplified instance of a process configuration problem where there are \( n \) mandatory features and \( m \) service candidates operationalizing each features, the total number of configuration options is \( m^n \). Accordingly, we propose a method based on integer programming [332] to achieve possible optimal solution for the reduction of configuration search space. The goal of optimization is to configure process model optimally for an individual stakeholder by adding both functional and non-functional requirements to the list of their preferences. Handpicking the best set of features and corresponding service candidates are two instances of such measures.

In this section, we describe how to transfer and formulate the problem of configuration defined Multi-Staged Process Configuration (MSPC) problem (cf. Section 5.3) into a mathematical optimization problem which can be solved efficiently by COP-based solvers.

We model and formulate the configuration problem as an MILP model which is characterized by four constituents according to definition (15): (1) a set of decision variables \( X \) with (2) a domain \( (D = \{0,1\}) \); (3) a set of constraints \( C \); and (4) objective function \( F \), hereafter also called utility \( U \).

The output of MILP problem is the maximum (or minimum) value of the objective function and the values of variables at this maximum/minimum.

The modeling procedures in details and formulations for the optimization model are explained in following subsections. In Section 5.7.1, we first describe how variability model expressed by feature model is transferred and formalized as linear constraints, which comprises a part of configuration constraint set in MILP model. We then discuss respective
quality aggregation functions and describe how stakeholder quality requirements and relative importance of preferences are weighted and constitute objective function, and provide optimization model.

### 5.7.1 Formalizing Feature Model in MILP Model

The MSPC problem modeled in MILP includes *configuration constraints* which are derived and expressed by a feature model (i.e., variability model). As discussed feature models describe the variant space, i.e., the set of valid configuration spaces and constraints which are defined in terms of relations among features. In this subsection, we explain how to translate the semantics of feature model and formulate it into MILP in our proposed structure, which entails a set of mapping rules and an algorithm for the transformation.

A feature model describes relationships among features. Therefore, we define the feature relationships by linear equality and inequality constraints. Concrete mapping rules to translate a cardinally-based feature model into an MILP is given in Table 5.2. These linear constraints are further added to constraint set of MILP model formulating the optimization model.

**Table 5.2: Mapping feature model to linear constraints for MILP**

<table>
<thead>
<tr>
<th>Relationships</th>
<th>Optional</th>
<th>Mandatory</th>
<th>Or-group</th>
<th>Xor-group</th>
<th>Excludes</th>
<th>Includes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraints</td>
<td>$x_i \leq x_{ip}$</td>
<td>$x_i = x_{ip}$</td>
<td>$x_i \leq x_{ip}$</td>
<td>$\sum_{i \in [1..n]} x_i \leq k' x_{ip}$</td>
<td>$x_i + x_{ij} \leq 1$</td>
<td>$x_i \leq x_{ij}$</td>
</tr>
</tbody>
</table>

The following steps are necessary to transform a feature model into MILP model:

1) For each feature $f_i$ of the feature model, a binary decision variable $x_i$ with a domain $D \in \{0, 1\}$ in an MILP model is created and mapped by a bijective function $\Omega : F \rightarrow X$, where for each $f \in F \subseteq FM$, there is a unique $x \in X \subseteq MILP$ such that $\Omega(f) = x$.

2) each feature relationship is mapped into a constraint corresponding to the relationship type as following:
a) Optional relation: Let \( f_p \) be the parent feature and \( f_c \) the child feature in an optional relation. The equivalent constraint is \( x_{f_c} \leq x_{f_p} \). Because the selection (i.e., presence) of a child feature \( (x_{f_c} = 1) \) relies upon the selection of its parent feature \( (x_{f_p} = 1) \).

b) Mandatory relation: Because the child feature \( f_c \) in mandatory relation is selected if and only if parent feature \( f_p \) is selected, the equivalent constraint is \( x_{f_p} = x_{f_c} \).

c) Or-group relation: Let \( f_p \) be the parent feature in an Or-group relation with cardinality \(<k-k'>\) and \( f_{ci} \) the set of \( n \) child features. The equivalent constraints are defined as:
(i) \( k \cdot x_{f_p} \leq \sum_{i} x_{f_{ci}} \leq k' \cdot x_{f_p} \), because at least \( k \) and at most \( k' \) features require to be included out of the \( n \) features, and (ii) \( x_{f_{ci}} \leq x_{f_p} \), that implies none of child features should be selected if the parent feature is not selected.

d) Xor-group relation: Let \( f_p \) be the parent feature in an alternative Xor-group relation with cardinality \(<k-k'>\) and \( f_{ci} \) the set of \( n \) child features. Because exactly \( k \) features can be included out of the \( n \) features, the equivalent constraint is \( \sum_{i} x_{f_{ci}} \leq k \cdot x_{f_p} \).

e) Exclusion relation: If feature \( f_i \) and \( f_j \) are in an exclusion relationship, the equivalent constraints is expressed as \( x_{f_i} + x_{f_j} \leq 1 \). It implies that both features can not be selected.

f) Inclusion relation: If feature \( f_i \) includes feature \( f_j \), the equivalent constraints is expressed as \( x_{f_i} \leq x_{f_j} \) to fulfill the requirement.

3) The resulted linear constraints from transforming feature model into MILP constitute configuration set, denoted as \( C_{FM} \), which is further added to the optimization model constraint sets.

Algorithm 3 outlines the process of feature model transformation in which feature model is traversed to create and assign decision-variable to each feature and further map feature relations into linear constraints by following mapping rules as described above. A combination of breath and depth-first searches is performed to visit each features and analyze the relations to its child features and possible dependencies with other features in terms of inclusion and exclusion relations in the feature model.
Algorithm 3: Transform\((FM, f, MILP)\). Transform Feature Model to MILP Model.

**Input:** Feature Model \(FM\); feature \(f \in FM\); MILP Model \(P = (X, D, C, F)\)

**Output:** Linear constraints set \(C_{FM} \in C \subset MILP\)

1. \(C_{FM} \leftarrow \{\};\) // Initialize constraints set
2. \(f_p \leftarrow f;\) // labeling input feature as parent feature
3. \(F_c \leftarrow \text{getChildFeatureSets}(f_p);\)
4. \(F_c = \{\hat{F}_c, \hat{F}_c, \hat{F}_c, \hat{F}_c\};\)
5. // Integrity constraints-feature set associated with feature \(f_p\)
6. \(F_{IC} \leftarrow \text{getFeaturesIC}(f_p);\)
7. \(F \leftarrow \{f_p\} \cup F_c \cup F_{IC};\)
8. // Mapping features to binary-decision variables- \(\Omega:F \rightarrow X\)
9. **for all the** \(f \in F\) **do**
10.  
11.  
12.  
13.  
14.  
15.  
16.  
17.  
18.  
19.  
20.  
21.  
22.  
23.  
24.  
25.  
26.  
27.  
28.  
29.  
30.  
31.  
32.  
33.  
34.  
35.  
36. **for all the** \(f_c \in F_c\) **do**
37.  
38.  
39.  
40.  
41.  
42.  
43.  
44.  
45.  
46.  
47.  
48.  
49.  
50.  
51.  
52.  
53.  
54.  
55.  
56.  
57.  
58.  
59.  
60.  
61.  
62.  
63.  
64.  
65.  
66.  
67.  
68.  
69.  
70.  
71.  
72.  
73.  
74.  
75.  
76.  
77.  
78.  
79.  
80.  
81.  
82.  
83.  
84.  
85.  
86.  
87.  
88.  
89.  
90.  
91.  
92.  
93.  
94.  
95.  
96.  
97.  
98.  
99.  
100.  
101.  
102.  
103.  
104.  
105.  
106.  
107.  
108.  
109.  
110.  
111.  
112.  
113.  
114.  
115.  
116.  
117.  
118.  
119.  
120.  
121.  
122.  
123.  
124.  
125.  
126.  
127.  
128.  
129.  
130.  
131.  
132.  
133.  
134.  
135.  
136.  
137.  
138.  
139.  
140.  
141.  
142.  
143.  
144.  
145.  
146.  
147.  
148.  
149.  
150.  
151.  
152.  
153.  
154.  
155.  
156.  
157.  
158.  
159.  
160.  
161.  
162.  
163.  
164.  
165.  
166.  
167.  
168.  
169.  
170.  
171.  
172.  
173.  
174.  
175.  
176.  
177.  
178.  
179.  
180.  
181.  
182.  
183.  
184.  
185.  
186.  
187.  
188.  
189.  
190.  
191.  
192.  
193.  
194.  
195.  
196.  
197.  
198.  
199.  
200.  
201.  
202.  
203.  
204.  
205.  
206.  
207.  
208.  
209.  
210.  
211.  
212.  
213.  
214.  
215.  
216.  
217.  
218.  
219.  
220.  
221.  
222.  
223.  
224.  
225.  
226.  
227.  
228.  
229.  
230.  
231.  
232.  
233.  
234.  
235.  
236. **return** \(C_{FM}\)
Proceeding from feature node of feature model tree (e.g., root feature), all the child features in relationships with input feature (labeled as parent feature $f_p$) are retrieved which constitutes the child feature set $F_g$ including mandatory, optional, Or-group and Xor-group features relations (lines 4-5). This is followed by returning all the features across the feature model in feature set $F_{IC}$, which are associated with integrity constraints with parent feature (line 6). For instance, let’s assume that Figure 5.4 (a) represents a subpart of a feature model. Child feature set $F_g$ of root feature $f_r$ is composed of sets of mandatory features $^\bigstar F = \{f_2, f_3\}$, optional features $^\odot F = \{f_1, f_3, f_4\}$, Or-group features $^\wedge F = \{f_{14}, f_{15}\}$, and Xor-group features $^\triangle F = \{f_{18}, f_{19}\}$. Line 9 ensures each feature is mapped to a binary decision variable created in MILP model; otherwise, new variables are created and assigned.

The rest of the algorithm performs a recursive transformation pattern for each feature based on relation types of parent feature with its child features, ‘and’ siblings and integrity constraints. Each relationship is further transformed into linear constraint which is added to the configuration constraints set $C_{FM}$ of MILP model.

$$C_{FM} : \quad x_0 \leq x_1;$$
$$x_2 = x_0;$$
$$x_0 \leq x_3;$$
$$x_0 \leq x_4;$$
$$x_5 = x_0;$$
$$x_{14} + x_{15} \geq x_0;$$
$$x_{14} \leq x_0;$$
$$x_{15} \leq x_0;$$
$$x_{18} + x_{19} = x_0;$$
$$x_{15} + x_{11} \leq 1;$$
$$x_6 + x_7 + x_8 = x_0;$$
$$x_9 + x_{10} + x_{11} \geq x_0;$$
$$x_9 \leq x_5;$$
$$x_{10} \leq x_5;$$
$$x_{11} \leq x_5;$$
$$x_{12} = x_6;$$
$$x_6 \leq x_{13};$$
$$x_{13} \leq x_4.$$

Figure 5.4: a) A sample feature model. b) Transforming feature model to linear constraints.
variable and shows the results of feature model transformation generating a set of linear constraints.

5.7.2 Optimization Model

The main decisions that have to be made with respect to the stakeholder’s preference are: (i) to determine firstly what features (i.e., abstract services) should be selected to satisfy functional demands for the stakeholder, and (ii) to select the best concrete service from a set of functionally equivalent services that operationalize each feature and thereby best meeting non-functional (i.e., quality) requirements and constraints.

As a basis of the optimization model presented in the following, we first introduce formal notations. Based on the previous definitions, given a reference model $PM$ composed with $n$ activities creates activity set $A = \{a_1, a_2, \ldots, a_n\}$, such that each activity $a_i$ mapped to feature $f_i$.

Regarding branching (conditional or parallel types) in workflow of $PM$, the set of indexed branches is denoted by $B = \{1, \ldots, k\}$, and $b \in B$ represents the branch number within a branching. Thereby, the set $A_B \subseteq A$ represents a set of activities within a branching, and the set of activities within branch $b \in B$ is denoted by $A_{B_b} \subseteq A_B$. The set $A_S = A \backslash (A_{B_b} | b \in B)$ indicates a set of activities which are not placed within a branching. The probability of executing a certain branch $b$ is denoted by $\rho_i$, so conditional branching with $k$ disjoint branches $\sum_{i=1}^{k} \rho_i^b = 1$, where $\rho_i^b$ indicates the probability of execution of $i$th branch. The loop constraints in a process model can be defined to express the number of iterations to execute particular activity. Hence, the probability distribution of every loop with maximum number of iterations $c$ is specified such that $\sum_{i=0}^{c} \rho_i^l = 1$, where $\rho_i^l$ indicates the probability that loop $l$ executed is $i$ times.

For each indexed activity $a_j \in A$ associated with feature $f_i$, a set of service candidates in service repository providing the required operations to execute activity $a_i$, are indexed as $s_{ij}$ and constitute service candidates set $S_{a_i} = \{s_{i1}, s_{i2}, \ldots s_{im}\}$. This set includes binding links to concrete services offering operation implementations with varying non-functional properties (i.e., QoS) for feature $f_i$. Activity $a_i$ is concertized by binding exactly one service $s_j \in S_{a_i}$ with $OP_j$, set of indices of operations implemented by service $s_i$, and $s_{i,o}$ is the invocation of operation $o \in OP_j$. Service $s_{j,o}$ is associated with a QoS vector $Q_s = [q_1^{j,o}, q_2^{j,o}, \ldots, q_n^{j,o}]$, with $n$ quality dimensions.
We introduce following binary decision variables \((x, y, z) \in X, D = \{0, 1\}\), representing “yes-or-no” decisions, to model the selection of features, services, and service operations, respectively:

- \(x_i := \) equals 1 by convention if feature \(f_i \in \mathcal{F}\) is selected, 0 otherwise;
- \(y_{ij} := \) equals 1 if the activity \(a_i\) associated to feature \(f_i\) is executed by service candidate \(s_j \in \mathcal{S}_{a_i}\), 0 otherwise.
- \(z_{ij,o} := \) equals 1 if the activity \(a_i\) is executed by invoking operation \(o \in \mathcal{O}_{j}\) of service \(s_j \in \mathcal{S}_{a_i}\), 0 otherwise.

Including an activity \(a_i\) by selecting counterpart feature \(f_i\) and assigning a qualified service \(s_{ij}\) leads to possible configuration plans (i.e., configuration options). Such configuration plans enable to derive different configured services (i.e., service products) from a reference process model (cf. Figure 3.1).

To achieve the optimal configuration, all the possible permissible Configuration Plans need to be evaluated, and the one which maximizes the stakeholder’s preferences while satisfying the imposed constraints is then selected. We formally define the concept of configuration plan used in the reminder of this section as follows:

**Definition 16 (Configuration Plan).** Every configuration instance of a reference process model \(PM\) with \(n\) activities is derived from a configuration plan \(CP_j\), where a set of \(m\) activities \(\{a_1, \ldots, a_m\} \in \mathcal{A}_j \subseteq \mathcal{A}\) is included w.r.t. corresponding selected features such that for each activity \(a_i\) iff one service from its service candidate set \(\mathcal{S}_{a_i}\) is bounded.

Optimization aims to maximize the overall configuration plan’s utility, i.e., as quality characteristics and an measurable objective quantification for desirable quality performance. In consequence, in the course of optimization the utility of each possible configuration plan is calculated and measured by its utility value, and the one with the highest utility value will be selected as the result of optimization.

**Objective Function:** The goal of optimization is to maximize the aggregated value QoS, considering all of the possible configuration plans arising from the composed service specification described by a \(PM\), which is subjected to stakeholders preferences and configuration constraints. Hereafter, objective function is defined to evaluate the global utility of configuration plans (i.e., quality performance) by taking stakeholder preferences about each quality
We apply a MCDM procedure \cite{122} for the utility computation with a Weighted Sum Model (WSM) \cite{146} to construct the objective function. This method is carried out in two phases: 1) \textit{scaling phase} is a pre-processing step normalizing QoS attribute values, and 2) \textit{weighting phase} incorporates the relevant importance of stakeholder preferences regarding QoS criteria. The utility computation involves scaling the QoS attribute values to provide a uniform measurement of the multi-dimensional qualities of services independent of their units, metrics and ranges. Hence, we use normalization to scale the various measurement metrics of quantified QoS attribute values to the same scale.

QoS dimension \((q_j)\) can have tendency property, denoted as \(\tau_{q_j}\), which represent expected tendency of the value from the stakeholder’s perspective. For example, the price of a service is expected to be as low as possible. The type of tendency can be classified into positive (ascending) and negative (descending) dimensions. Positive tendency \(\tau^+\) means a higher value is preferred (e.g., reliability, throughput, security), whereas negative tendency \(\tau^-\) means a lower value is preferred (e.g., price, response time). Thus, QoS with positive and negative tendency need to be maximized and minimized, respectively.

Scaling phase normalizes the values of positive and negative QoS dimensions in interval \([0, 1]\). Given activity \(a_i\), for each service candidate \(s_{ij} \in S_{a_i}\)

\[
\hat{q}_{ij,o}^j = \begin{cases} 
\frac{q_{ij,o}^j - q_{k}^{\text{min}}}{q_{k}^{\text{max}} - q_{k}^{\text{min}}} & \text{if } q_{k}^{\text{max}} \neq q_{k}^{\text{min}} \forall q_{k} : \tau_{q_k} \in r^+; \\
\frac{q_{k}^{\text{max}} - q_{ij,o}^j}{q_{k}^{\text{max}} - q_{k}^{\text{min}}} & \text{if } q_{k}^{\text{max}} \neq q_{k}^{\text{min}} \forall q_{k} : \tau_{q_k} \in r^-; \\
1 & \text{otherwise.}
\end{cases}
\] (5.3)

(5.4)

where \(\hat{q}_{ij,o}^j\) denotes the normalized value of \(k\)th QoS dimension associated with service \(s_{ij,o} \in S_{a_i}\) and its operation, and \(q_{k}^{\text{min}} = \min_{\forall s \in S_{a_i}} q_{ij,o}^j\) and \(q_{k}^{\text{max}} = \max_{\forall s \in S_{a_i}} q_{ij,o}^j\) are the minimum and maximum values of \(k\)th QoS dimension among all services belong to the service candidate set \(S_{a_i}\) associated to activity \(a_i\), respectively. The QoS with positive tendency has higher tends to 1 when the value is higher, whereas negative tendency tends to 1 when the value is lower.
The utility of each service candidate \( s_i \) with its operation \( s_{ij,o} \in S_a \) is formulated and computed uniformly by summing up the product of each normalized value of quality dimension with corresponding assigned weight as follow:

\[
U(s) \triangleq \sum_{k=1}^{n} \hat{q}^{ij,o}_k \cdot w_k \quad \forall k \in Q_s
\]  

(5.5)

where \( w_k \in \mathbb{R}^+ \) is the weight representing the stakeholder’s preferences and priorities about \( k \)th quality criteria. These weights are obtained as the result of the CS-AHP algorithm described in Section 5.5.2.

Thereby, the overall utility of each configuration plan, which ultimately derives an instance of configured process from a reference process model, is evaluated based on the utility of selected candidate services in the course of optimization. As discussed in Chapter 4, QoS aggregation is an important means to calculate the QoS of the overall process model describing service composition. QoS aggregation functions are further used to constitute the objective function, aggregate quality dimensions and evaluate constraints during the optimization process. For instance, when end-to-end QoS constraints and requirements should also be granted and satisfied (e.g., the overall cost requires to be less than given value by stakeholder). In Chapter 4, we summarized some QoS attributes with their aggregation rules by taking into account both variability and composition patterns to compute the maximum and minimum quality ranges, for instance, upper and lower bounds. To evaluate the utility of configuration plans, we need to compute the overall QoS for individual quality dimension with respect to activities in process model and associated candidate services. Respected QoS aggregation functions including service decision variables are summarized in Table 5.3.

The QoS characteristics of process model are associated with a characteristic vector \( \hat{q}_{PM} = [\hat{q}_1, \ldots, \hat{q}_n] \in \mathbb{R}^+ \). Suppose that out of \( n \) QoS criteria, \( \alpha \) criteria with positive tendency need to be maximized while \( \beta \) criteria with negative tendency have to be minimized. The overall utility of an instance of configured process model is formulated and computed as follow:

\[
U(PM) = \sum_{\alpha=1}^{k} \frac{\hat{q}_\alpha - \hat{q}_\alpha^{min}}{\hat{q}_\alpha^{max} - \hat{q}_\alpha^{min}} \cdot w_\alpha + \sum_{\beta=k+1}^{n} \frac{\hat{q}_\beta^{max} - \hat{q}_\beta}{\hat{q}_\beta^{max} - \hat{q}_\beta^{min}} \cdot w_\beta \triangleq \sum_{k=1}^{n} \hat{q}_k \cdot w_k
\]  

(5.6)
where $\hat{q}_k$ indicates the QoS aggregation of $k$th quality dimension of process model based on aggregation functions according to composition patterns as given in Table 5.3, and it includes the corresponding service decision variables. Similarly, the values of quality dimensions with positive and negative tendencies are scaled according to (5.3) and (5.4), respectively. The $\hat{q}_k^{\min}$ and $\hat{q}_k^{\max}$ are minimum and maximum aggregated values of the $k$th QoS dimension, which are constituents of composition patterns of process model, where $\hat{q}_k^{\min} = q_k^{LB}(PM)$ and $\hat{q}_k^{\max} = q_k^{UB}(PM)$. These values are computed by quality range aggregation functions and the Algorithms described in Chapter 4. Note that the minimum (or maximum) possible aggregated values are computed with respect to the activities which are included in a configuration plan. For instance, the maximum execution price ($\hat{q}_k^{pr}$) is computed by summing up the execution price of the most expensive service candidates for each activity. The minimum execution or response time ($\hat{q}_k^{rt}$) is computed by selecting the service candidate with the shortest response time for each activity. Thus the computation time to evaluate the $\hat{q}_k^{\min}$ and $\hat{q}_k^{\max}$ is polynomial, as also discussed in [338]. The normalization produces a normalized QoS vector $\hat{Q}_{PM} = [\hat{q}_1, \ldots, \hat{q}_n]; \forall \hat{q} \in [0, 1]$, and the normalization phase complexity is linear in terms of the number of activities of process model.

Our objective is to maximize the utility value of the configured process model which satisfies all the constraints and restrictions defined in the model. In our problem under study, we distinguish and group constraints into four classes: 1) configuration constraints, formerly denoted as $C_{FM}$ – a set of constraints which are specified as part of the system’s configuration requirements and defined by feature model. As we described above, these constraints set is created by transforming feature model into MILP model; 2) service assignment and dependency constraints, denoted as $C_{SA}$ and $C_{SD}$, respectively – sets of constraints associated with service allocation and dependencies among services specified by service designer; and 3) local and global quality constraints, denoted as $C_{Qf}$ and $C_{Qgc}$, respectively – sets of constraints which are part of application’s requirements and can be specified by either system designer or stakeholder concerning quality needs toward particular features (from local point of view) or global (e.g., end-to-end) constraints. For instance, it includes constraints over total cost, execution time, or specific domain quality characteristic.

Service Assignment Constraints: As one service candidate should be selected from all available services $S_{a_i}$ executing activity $a_i$ associated with feature $f_i$, following selection
CHAPTER 5. QOS-AWARE PROCESS CONFIGURATION & OPTIMIZATION

constraint should be further added to the optimization model:

\[
C_{SA} : \begin{cases}
\sum_{j \in S_{a_i}} y_{ij} = 1, & \forall i \in \mathcal{A}; \\
\sum_{j \in S_{a_i}} \sum_{o \in \mathcal{OP}_j} z_{ij,o} = 1 \ z_{ij,o} \leq y_{ij}, & \forall i; j \in S_{a_i}, \forall o \in \mathcal{OP}_j
\end{cases}
\]

(5.7)

(5.8)

**Service Dependency Constraints.** A set of activities in the process may be invoked by the same service, which imposes service dependency constraint. This type of constraints enables considering both stateless and stateful activities in a process model. Stateless activities do not require sharing any state information with other activities. In contrast, stateful activities do need state information to be shared among them. In consequence, these activities need to be implemented by operations of the same concrete service. In a configuration plan comprising an stateful activity \(a_i\), which includes a set of state-related activities, the optimal selection of service candidates for each activity is influenced by other state-related activities. For instance, let’s assume a stateful activity includes (1) Login, (2) Buy an Item, (3) Payment, (4) Logout, and there are two service candidates (e.g., eBay and Yahoo, to fulfill this stateful activity). If we independently select optimal service candidates for these four state-related activity, the selection results may be: (1) eBay.login(), (2) eBay.buyItem(), (3) eBay.payment(), and (4) eBay.logout(). However, because the state-related activities require to maintain states among consecutive invocations of multiple activities, obviously it is impossible to log in eBay and buy an item from Yahoo service. Hence, the service candidates for the state-related activities of a stateful activity should be provided by the same service provider. Therefore, if two activities \(i_1\) and \(i_2\) have to be executed by the same service, then the following constraint families are introduced and added to the constraint sets of optimization model:

\[
C_{SD} : \begin{cases}
y_{ij_1} = y_{ij_2}, & \forall j \in S_{a_{i_1}} \cap S_{a_{i_2}}; \\
y_{ij_1} = 0, & \forall j \in S_{a_{i_1}} \setminus S_{a_{i_2}} \\
y_{ij_2} = 0, & \forall j \in S_{a_{i_2}} \setminus S_{a_{i_1}}.
\end{cases}
\]

(5.9)

(5.10)

(5.11)
Local and Global Quality Constraints:

Quality of service constraints represents quality requirements, which can be specified at local and global levels. A local (or feature) constraint imposes desirable quality characteristic over particular feature (e.g., the cost of specific feature should be lower than a given threshold). Global constraints impose end-to-end quality requirements for the whole service application, for example, constraints like “the total cost of service application has to be less or equal than a given value” or “the overall execution time has to be less than 5 seconds”. Accordingly, global constraints enforce quality of service restrictions over entire process model, whereas local constraints impose constraints on values of quality attributes of individual activity mapped to feature.

Both constraint types can be expressed as hard (i.e., required) or soft (i.e., optional) QoS constraints. Each hard constraint has to be fulfilled; otherwise, no solution can be found. All the soft constraints (global and local) will be added to optimization model, which can be relaxed by either designer or stakeholders if there is no solution found.

It is noteworthy that the stakeholders are often interested in global constraints because we assume the stakeholder very often has no knowledge of how services are composed. For instance, stakeholder may be concerned about the maximum execution time of the entire service application instead of single feature. Moreover, complex composite services can be encapsulated and hidden to the stakeholder. Either global and local constraints can be expressed in terms of upper (and/or lower) bounds for the aggregated values of the different QoS attributes.

Consequently, each feature \( f_i \) can be subjected to \( m \) local QoS constrains \( C_{Q_f} = \{c_1^R, \ldots, c_m^R\} \), \( 1 \leq m \leq n \), where \( C_{Q_f} \subseteq NF_{f_c} \) according to service definition (1), and \( c_k^R = [c_{k_{LB}}, c_{k_{UB}}] \) is the constraint range vector including the lower and upper bounds for expected QoS range values of \( k \)th quality dimension. Consequently, we include following constraints for each feature constraints \( C_{Q_f} \) in optimization model to guarantee the satisfaction of the selected QoS range values for given feature \( f_i \).

\[
\sum_{j \in \mathcal{S}_{a_i}} \sum_{o \in \mathcal{O}_j} q_{ij}^{o} \cdot z_{ij,o} \leq c_{k_{UB}}^{\text{UB}} \quad \forall i \in \mathcal{A}; \forall k \in C_{Q_{f_c}}
\]

where \( c_{k_{LB}}^{\text{LB}} \) and \( c_{k_{UB}}^{\text{UB}} \) are lower and upper bounds constraint values for \( k \)th quality dimension.
Furthermore, we should include global (i.e., end-to-end) quality constraints as a set of restriction for the aggregated QoS values into constraint sets of optimization model. Hence, in a like manner, the overall process model for particular service application can be subjected to \( m \) global QoS constrains \( \hat{C}_{Q_{gc}} = \{ \hat{c}_1, \ldots, \hat{c}_m \} \), \( 1 \leq m \leq n \) as follows:

\[
\hat{c}_{k}^{LB} \leq \hat{q}_{k}(PM) \leq \hat{c}_{k}^{UB} \quad \forall i \in A; \forall k \in \hat{C}_{Q_{gc}}
\]

where \( \hat{C}_{Q_{gc}} \subseteq NF_{gc} \), \( \hat{c}_{k}^{R} \) is the upper (and/lower) bound constraint values for aggregated QoS of \( k \)th quality dimension, and \( \hat{q}_{k}(PM) : A^R \to \mathbb{R} \), \( A = \{ f_S(q_k), f_L(q_k), f_F(q_k), f_C(q_k) \} \), is the combination of aggregation functions aggregating the value of \( k \)th quality dimension for the entire process model, based on existing composition patterns in a process model (cf. Table 5.3).

All the aforementioned decision constraints describe the constraint sets of optimization model. Accordingly, we formulate the problem of finding the optimal configuration of process model (e.g., MSPC) according to (5.2) as a maximization of objective function, as defined in (5.6), which meets all the constraints specified in model.

\[
\text{Maximize} \quad U(PM) = \sum_{k=1}^{n} \hat{q}_k \cdot w_k
\]

P1. \quad subject to \quad \begin{align*}
C_{FM} \\
C_{SA} \land C_{SD} \\
C_{Q_{fc}} \land \hat{C}_{Q_{gc}}
\end{align*} \hspace{1cm} (5.14)

We devised integer programming-based approach to solve the configuration problem. However, MILP optimizes a linear objective function that is subject to linear equality and inequality constraints. The formulated problem model P1 given in (5.14) constitutes a non-linear optimization problem because the objective function includes non-linear aggregation of decision variables (i.e., multiplication, min/max-operators). Some aggregation rules are non-linear in nature, for instance, the aggregation formula for availability \( q_{av} \) and reliability \( q_{re} \) use the product to aggregate the QoS for a sequence and parallel execution of features. Additionally, the problem model can include non-linear constraint family associated to such
CHAPTER 5. QOS-AWARE PROCESS CONFIGURATION & OPTIMIZATION 182

Table 5.3: QoS attributes and pattern-based aggregation functions.

<table>
<thead>
<tr>
<th>QoS Dimension</th>
<th>Sequential</th>
<th>Loop</th>
<th>Flow (AND)</th>
<th>Parallel (XOR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price ( (q_{pr}) )</td>
<td>( \sum_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \sum_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \sum_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \sum_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
</tr>
<tr>
<td>Response Time ( (q_{rt}) )</td>
<td>( \sum_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \sum_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \sum_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \sum_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
</tr>
<tr>
<td>Availability ( (q_{av}) )</td>
<td>( \prod_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \prod_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \prod_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \prod_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
</tr>
<tr>
<td>Throughput ( (q_{tp}) )</td>
<td>( \min_{i \in A _p} \left( \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} \right) )</td>
<td>( \min_{i \in A _p} \left( \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} \right) )</td>
<td>( \min_{i \in A _p} \left( \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} \right) )</td>
<td>( \min_{i \in A _p} \left( \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} \right) )</td>
</tr>
<tr>
<td>Reliability ( (q_{re}) )</td>
<td>( \prod_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \prod_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \prod_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
<td>( \prod_{i \in A _p} \sum_{j \in S_{ij}} q_{ij,o}^a \cdot z_{ij,o} )</td>
</tr>
<tr>
<td>Domain Attr. ( (q_{dm}) )</td>
<td>( f_S(q_{dm}) )</td>
<td>( f_L(q_{dm}) )</td>
<td>( f_F(q_{dm}) )</td>
<td>( f_C(q_{dm}) )</td>
</tr>
</tbody>
</table>

quality dimensions. Accordingly, we transform the problem into a linear form by linearizing non-linear aggregation rules. We apply logarithmic function and introduce auxiliary variables to linearize corresponding non-linear aggregation rules in order to satisfy the linearity constraint attached to the IP approach (as shown in similar work [338]).

The seemingly complicated optimization problem is then optimally solved by applying branch-and-bound method and utilizing off-the-shelf MILP solvers. The result is the complete assignment of decision variables indicating the selection of features and further concrete services, which meets constraints specified in model and yields a configured process model for particular service application according to stakeholder’s preferences.

The P1 is NP-complete since it can be considered as an instance of 0-1 knapsack problem as shown in related work [336]. The solution algorithm complexity grows exponentially with the number of binary-decision variables in the model. Nevertheless, as we demonstrate in Chapter 7 by conducting empirical evaluation via numerous experiments that feasible solutions can be computed efficiently in a timely manner for realistic design problems of reasonable size by taking the advantages of current solvers.

5.7.3 Solution Methods

The basic structure of constraint programming to solve CSPs is usually similar. The method defines search of solution space, which includes the enumeration (or labeling) (i.e., the process of generating values of specific domain variables). Many algorithms have been designed
and developed for several decades to solve CSPs and particularly COPs [186]. In general, algorithms to solve CSPs usually fall into two main classes: complete or incomplete [267].

Complete or systematic search algorithms construct a solution by incrementally instantiating the variables of the problem and guarantee to find a solution if one exists. Also, they can be used to prove unsolvability or to find all the solution and are guaranteed to find a provably optimal solution in optimization problems (i.e., COPs) by failing to find a better solution. Backtracking is one of the most common complete search method to solve a CSP in practice. A problem is recursively divided into smaller sub-problems. This is often structured as a search tree including all candidate solutions, where each node represents a variable assignment and each branch represents a potential solution. In essence, the backtrack method, typically known as a constructive method, performs a depth-first search of the space of potentials solutions. It constructs a partial solution by assigning values to variables from their domain until obtaining a complete assignment that satisfies all the constraints in the problem. If all variables are assigned values without violating any constraints, then the solution is found. Otherwise, it needs to backtrack (undo) one or more assignments and choose different values. For instance, chronological backtracking (BT) [47], backjumping (BJ) [114], conflict-directed backjumping (CBJ) [253], dynamic backtracking (DB) [117] are algorithms that check backwards. Several hybrid approaches have also been developed that combine backward with forward search algorithms for CSP (e.g., [252]).

Different heuristics have been proposed to improve the efficiency of solutions methods that have been studied intensively since the early days of AI [85, 268]. One of the key factors of performance and complexity in search algorithms is the heuristic which decides on which variable to branch next and the values for assignment [186, 314]. These heuristics can be categorized into two groups of variable ordering and value ordering heuristics. Variable ordering heuristics aim at addressing the order in which the algorithm assigns the variables, i.e., minimizing backtracking – the minimal width ordering and minimizing the number of backtracks (the minimal bandwidth ordering) [314]. Various dynamic or static variable ordering heuristics have been proposed. In a static way, variables are ordered before search procedure. In contrast, dynamic variable ordering heuristics choose the current variable using information gathered during the search, which have shown better average performance [115]. The value ordering heuristics decide an order on which values will be assigned to a selected variable. It has been shown in many studies that the choice of variable and values have substantial impact on performance of search algorithms (e.g., [32, 115].
Incomplete or non-systematic search algorithms are proposed as practical techniques to cope with the computational intractability of NP-complete combinatorial optimization problems. Local or stochastic search are considered as incomplete algorithms, which include Genetic Algorithms (GAs), simulated annealing and neural network [134]. These classes of algorithm effectively outperform complete (systematic) ones; however, they cannot guarantee to find a solution, and cannot be used to find a provably optimal solution. Such algorithms are often useful at finding an approximation to an optimal solution if one exists, when we simply wish to find a solution quickly. Local search algorithms, such as min-conflict [220], Brekout [279], are utilized for constraint satisfaction and satisfiability problems. However, local search algorithms perform an incomplete exploration of the search space.

Constraint (or consistency propagation) is often an integral part of every constraints programming solvers. Constraint propagation aims at reducing the search space and improving performance by eliminating values from domain of variables that are inconsistent with any constraints. Therefor, the majority of approaches to solve CSPs commonly integrate both backtracking and consistency propagation, which are also combined with heuristics to boost the search. The purpose of this integration has two-fold: 1) It removes inconsistency during the backtracking search that can dramatically prune the search tree by removing many dead-ends and by simplifying the remaining sub-problems. 2) It helps to collect information for variable ordering heuristics to make effective variable ordering decisions.

Branch-and-Bound: The branch-and-bound, also known as branch and relax search which is based on backtracking scheme, is a widely used and very general deterministic algorithm to solve the optimization and MILP problems. This algorithm is implemented by many state-of-the art solvers. This approach provides rigorous lower and upper bounds on the solution, which in turn provide information regarding the optimality of the solution.

The idea is to optimize the search by exploiting the relaxations. A relaxation provides a bound to optimal value of original problem. Hence, we solve a relaxed version of the problem, and then, if some of the relaxed constraints are not satisfied, the problem is split into sub-problems on which this process is repeated. This method thus relies on divide-and-conquer strategy to divide an instance of problem into smaller sub-problems (called branching) until the individual ones can be solved and optimized easily, where combines partial enumeration strategy with relaxation techniques.
The method creates a branching tree in which each node represents a sub-problem. The relaxation is performed at each node of the search to detect the feasibility, prune and reduce the search space. This relaxation step can be carried out by utilizing MILP solvers to verify if the continuous relaxation of a problem is feasible.

As a consequence, the best of all solutions found in the sub-problems result the global optimum. This method enumerates all possible solutions; however, it eliminates large subsets of fruitless candidates by bounding that uses upper and lower estimated bounds of objective values. It uses the knowledge of objective function to guide the search and variable assignment. For instance, if the value exceeds the bound (i.e., the current computed value of objective function), then the sub-tree under the current partial assignment is pruned to avoid exploration.

Algorithm 4 outlines the branch-and-bound procedure. The algorithm initiates a queue $L$ of unprocessed sub-problems with the original problem $P$. We use a binary encoding to represent the solutions (e.g., every solution is represented as a set of binary vectors). We choose a sub-problem $P_j$ from $L$ and solve a relaxed version of the problem, denoted as $P_j^{\text{Relax}}$. In our MILP model, as a binary integer-program model, the linear relaxation of program is the problem that arises by substituting the constraint that each decision variable must be 0 or 1 by a weaker constraint and that each decision variable belongs to the interval $[0,1]$.

If there is a feasible solution, the relaxed version of problem $P_j^{\text{Relax}}$ is solved and the corresponding solution and objective value are yield, which are denoted as $x_j^*$ and $z_j^*$, respectively (Line 4-6).

Feasibility of solutions can be verified by two different ways during traversing search tree. On one hand, the relaxation of the problem is solved at every node (i.e., sub-problem) of search tree to determine whether the node itself is worthwhile to be further divided. Consequently, the node’s relaxation may turn out to be feasible with respect to all of the constraints. On the other hand, integer feasible solutions can also be discovered early by primal heuristics, as part of arsenal of MILP solvers, which are utilized during the search process that is improved to produce a good upper bound early and reduce the size of the branch-and-bound tree. The search backtracks if the optimal value of the relaxation is greater (in case of minimization) or less (in case of maximization) than the value of the best candidate solutions found. We avoid a complete enumeration of all potential solutions of $P_j^{\text{Relax}}$ by bounding (line 7). To this end, a good lower (dual) and upper(primal) bound
must exist to make bounding effective. A relaxation helps to calculate such lower and upper bounds, which are found in the course of branch-and-bound (Lines 8–14).

Algorithm 4: Branch-and-Bound

Input: Minimize/Maximize MILP problem $P = (X, D, C, F)$

Output: The Optimal solution of $x^* \in X$ with value $z^*$ of objective function $F$, or $P$ has no solution

1. Initialize $L \leftarrow \{P\}, z^* \leftarrow \infty$
2. while $L \neq \emptyset$ do
   3. Choose $P_j \in L$, and remove it from list $L \leftarrow L \setminus P_j$
   4. Solve $P^R_j = (X, D, \hat{C}, F)$
   5. if $P^R_j$ is feasible then
      6. Let $x^*_R = \{v_1, \ldots, v_n\}$ be an optimal solution of $P^R_j$ that minimizes/maximizes $F$ w.r.t. $\hat{C}$ with $z^*_R$, its optimal objective function value
      7. if $(z^*_R < z^*$ for minimization $)$ or $(z^* < z^*_R$ for maximization $)$ then
         8. if $x^*_R$ is feasible for $P$ such that $\forall x_i \in X, v_i \in Z$ then
            9. $x^* \leftarrow x^*_R$ and $z^* \leftarrow z^*_R$
         else
            // Split $P_j$ into two new sub-problems
            11. Choose a variable $x_i \in X$ such that $v_i \not\in Z$
            12. $C^{\text{Low}} \leftarrow \hat{C} \cup \{x_j \leq v_j\}$
            13. $C^{\text{Upp}} \leftarrow \hat{C} \cup \{x_j \geq v_j\}$
            14. $L \leftarrow L \cup \{(X, D, C^{\text{Low}}, F), (X, D, C^{\text{Upp}}, F)\}$
   10. else

15. return $x^*$ and $z^*$

If a feasible solution is found, $x^*$ and $z^*$ are updated (line 9), otherwise we split current sub-problem $P_j$ into sub-problems (Lines 11–14). This is performed by choosing a variable $x_j$ that has a fractional value $v_j$ in the relaxation step. Despite the theoretical complexity, the sub-problems become smaller and smaller because of partitioning mechanism, and finally LP relaxation is directly integral for all variable in $I$. The node selection strategy (i.e., the selection of sub-problems $P_j$) and branching strategy (i.e., how we choose the variable $x_j$) determine important decisions of a branch-and-bound method. They are considered as the major factors on how fast optimal solution can be found, and they influence the bounding that cuts off sub-problems and reduces a large search space.
Preprocessing and Cutting planes are two main approaches which are employed to improve the relaxation of the sub-problems. Preprocessing refers to manipulating an MILP formulation to decrease the overall solution time. These manipulations reduce the feasible region in such a way that at least one optimal integer solution remains feasible. In addition to preprocessing, an efficient way to obtain a tighter formulation for an integer problem is to introduce extra constraints in the form of cutting planes. MILP problems also can be solved without branching by simply finding its “right” description of linear-programming.

State-of-the-art MILP solvers leverage hybrid approaches (i.e., branch-and-bound and the cutting plane and variation of the general branch-and-cut scheme) with different heuristics to select branches in the search space that are more likely to lead to solutions efficiently in terms of time. The solution technology has been subject to intense development effort for at least three decades, and extensive research has been conducted for automatic reformulation of integer to reduce overall execution times and improve computational performance. MILP solvers have achieved order-of-magnitude speedups through integrating advanced search strategies, preprocessing and probing techniques, cutting plane algorithms, and primal heuristics. For detailed explanations of these techniques, we refer the reader to [195, 314, 332]. We employ LP computation as a tool by leveraging existing MILP solvers to solve the problem formulated in MILP as described above. It is noteworthy that our approach can be used with a range of backend solvers to enhance the computational time and improve scalability.

Near-optimal Solutions: In a practical scenario, the stakeholder may be interested in selecting a preferred configured service product from a recommended feature set. In this case, we can provide the optimal solution and several near-optimal solutions to form a solution pool for interactive selection. A simplified approach yields suboptimal solution in that after achieving optimal solution, we can define constraints in light of the resultant optimal solution and append it to the original model to construct a modified model. This may result in removing the optimal solution form the solution space. The suboptimal solution can be found by resolving the modified model. Hence, the new alternate solution is not the optimal solution of the original model; however, it will be a sub-optimal; a set of near-optimal solutions can be obtained for further selection of the stakeholder. Our approach can be extended to automatically create such constraints, renew the optimization model, and resolve the new model.
Chapter 6

Evaluation

6.1 Methodology and Experimental Evaluation

In the following sections, the methodology that was adopted for the evaluation of the proposed approach is presented, followed by the experimental design for the system’s evaluation tests (i.e., design parameter and test-cases) as well as experimental results and discussion.

The main objective of our evaluation is to systematically and empirically analyze the performance of the presented approach and algorithms for QoS-aware configuration in the context of a service family. The goal of the evaluation is twofold: 1) evaluating the performance and computational complexity, and scalability of the approach with different model characteristics; and 2) exploring how structural and behavioural characteristics of the process model may significantly impact the computational complexity of configuration.

We employ a simulation modeling technique by the following guidelines similar to those proposed in [169]. Simulation is a widely-used research method to study and analyze complex scenarios and to gain insights into performance and scalability for large-size problem instances. Moreover, it helps to evaluate the generalizability of the results. For this purpose, we developed an experimental platform to generate different test-cases for the evaluation.

In the previous chapters, we described our proposal for configurable process models, where we model and solve the problem as an MILP. We consider the execution time of algorithms as the performance criterion during the course of configuration, and take into account the scalability and performance affected by (1) the size of the reference process models and (2) the structural and behavioural characteristics of the models. With simplicity in mind, we only consider the following four major factors as independent variables which
determine the size of the reference process model for the problem under study:

1. Number of activities in terms of features in the reference process model \( (n_a) \)
2. Number of service candidates available for each activity \( (n_s) \)
3. Number of service operations for each service \( (n_{op}) \)
4. Number of QoS dimensions with corresponding global constraints \( (n_q) \)

The information quantified for analysis in this research is categorized with respect to the level of measurement of the data. Different levels of measurements comprise different amounts of information with regard to whatever data are measured \[280\]. For further analysis, we have used ratio level (i.e., ratio scale), as part of the data classification system which is commonly employed within the framework of empirical software engineering and other disciplines \[51\]. We considered both structural variability and behavioural characteristics of the process model variants, expressed by well-defined variability and composition patterns (cf. Chapter 4 Section 4.4). These patterns are further quantified and characterized in terms of percentile ratios:

- **Composition patterns \( (R_{CP}) \)**
  1. Ratio of sequential pattern (SEQ)
  2. Ratio of AND-parallel pattern (AND)
  3. Ratio of OR-parallel pattern (OR)
  4. Ratio of XOR-parallel pattern (XOR)

- **Variability patterns \( (R_{VP}) \)**
  1. Ratio of mandatory features
  2. Ratio of optional features
  3. Ratio of Or-group
  4. Ratio of Xor-group
  5. Ratio of integrity constraints

### 6.1.1 Experimental Framework

For the evaluation of complex configuration scenarios, we need the ability to construct models with different specifications ranging from small to large-sized models with respect
to diverse structural and behavioral characteristics. For this purpose, we developed a simulation framework allowing automatic generation and the steering of testbeds of large and complex structural and behavioral models.

It is noteworthy that simulation can provide accurate and insightful means to analyze and predict the performance measures. Nonetheless, it should be noted that generating error-prone models whose specifications do not resemble the desirable characteristics can result in erroneous and invalid results in the course of analysis and evaluation.

In this section, we present and summarize the experimental platform and the problem generation technique to create large-size test problems. The simulation framework allows setting up testbeds whose complex behaviour can be achieved by various components.

![Simulation framework architecture](image)

**Figure 6.1: Simulation framework architecture**

Figure 6.1 illustrates a high-level representation of the architecture of the framework that is composed of five main components: 1) process model generator; 2) service generator; QoS generator; 4) feature model generator; and 5) preference and QoS constraints generator. The core of the framework also comprises a set of interfaces and classes to integrate an MILP solver. We utilized IBM ILOG CPLEX [75] in our current implementation. Moreover, the framework supplies a set of transformations which aims at (I) generating mapping between the generated process model and feature model and (II) supporting model-to-model transformation, i.e., to construct a feature model for a generated process model and vice versa. The framework enables guided generation of random models using optimization criteria.
The guided generation implies that a random model is generated to meet desirable specifications (or characteristics) which are specified in advance.

**Process Model Generator:** We require solving an optimization problem to generate highly-customized random process models (or workflows) which comply with predefined specifications, for instance, constructing a model with \( n \) number of activities which includes random composition patterns (e.g., sequential and parallel structures) with specified ratio i.e., sequence (10%), OR-parallel (25%), AND-parallel (40%), and XOR-parallel (25%). To this end, we treat and formulate this problem as a classic knapsack problem – a special case of constraint-based resource allocation. On the account of the fact that it is analogous to a situation in which the process model should be composed of different weights of composition patterns as a series of objects to place in a knapsack. We also formulate this problem as an integer programming problem.

In the following, we review the algorithm generating random process models corresponding to requisite characteristics. The algorithm is based on the idea of the so-called labeling method to generate process models of size \( n \) activities i.e., atomic and compound activities (sub-processes). The algorithm creates a Process Structure Tree (PST) representing a workflow graph of the process model (cf. Figure 4.7), where the nodes of the workflow graph are represented as leaves with no label. The PST is constructed based on a parametrized maximum and minimum of the depth and width of the tree, and the average branching factor. We only set a minimum of 2 and maximum of 50, as a stopping rule for the addition of branches.

The algorithm starts by producing an initial PST with random topology by creating a single root node. At each node a random tree is built recursively. In each iteration, a current node without children is randomly selected, and a random number of child nodes are added by considering the parameter settings as the input constraints. For each constituted subtree, not only is a composition pattern randomly chosen and a corresponding label assigned to the root node, but also the statistics of the tree structure and patterns are computed and updated for the next iterations.

To achieve the optimal solution for the assignment of composition pattern labels (i.e., SEQ, OR, AND, and XOR) to the nodes distributed over the entire tree with respect to the given ratios as constraints, we transfer the PST-labeling problem into a 0-1 knapsack problem. A matrix of binary decision variables is constructed whose columns correspond
CHAPTER 6. EVALUATION

to composition pattern labels, with rows corresponding to the nodes’ labels. If the decision variable is set to one, it indicates that the corresponding composition pattern label is assigned to the node label. Because only one label can be assign to each node, the summation of the decision-variables for the matrix row should be equal to one which is added as allocation constraints to the optimization model. Furthermore, the summation of the decision-variables for the matrix column should not exceed proportionally with respect to the given ratios of the composition patterns, which are also included as constraints to the model. The declared desired ratios of composition patterns for the model are considered as weights incorporated in the utility function that should be maximized. We solve the formulated problem by employing MILP solver, resulting in variable assignment.

Owing to the fact that the knapsack problem is NP-complete, in some test-case generation, an optimal solution for labeling the problem may not be found because of the random topology of the initial generated model as well as imposed constraints in the problem model. Furthermore, the solutions to generate some models may exhibit very high running times.

To remedy these problems to pave the way for optimization, we devised a time-based heuristic to make the generation processes amenable and a distance-based heuristic to yield a near-optimal solution corresponding to the desirable model characteristics. The algorithm is executed \( n \) times (the default value is set to 1000), and an initial execution-time threshold is set incrementally for the large-size model generation based on the history of the execution time of the constructed small-size models. For each execution, the statistics of the generated model and the optimization time are computed, and the threshold is updated. If a feasible solution is not found or the running time exceeds, the execution is revoked.

A distance-based heuristic is based on the minimum Euclidean distance as the heuristic information to select the best model among \( n \) produced models. For each generated model, the algorithm uses the computed statistics to determine the exact ratios of the composition patterns as part of model specification. The Euclidean distance between the specification of the constructed model and the given desirable characteristics is calculated. The algorithm chooses the model which has the minimum distance measure as the final generated model.

**Web Service and QoS generators:** The Web service generator generates services which are delegated and bound to process model activities. For the service generation, we employed the Model-Driven Development (MDD) approach to create a service description with
information about service functionality (i.e., operations) and non-functional behavior specifications (i.e., QoS) based on a service model template. Because there are many QoS dimensions that are worthwhile for different experiments, a QoS generator model is devised which supports QoS ontology and different classes to define attributes with specific metrics and values. The QoS simulator generates synthetic data similar to actual running Web services. Real QoS datasets can also be imported to assign QoS values. If the simulation requires specialized quality attributes, new attributes can be declared, instantiated and randomly initialized according to the specified range of values and given distributions.

In our evaluation, we experimented with real QoS datasets\(^{11, 342}\) as the baseline to generate different distributions and QoS values. These datasets are publicly available and comprise measurements of QoS attributes of real-world Web services collected from public sources on the Web. QoS dataset\(^{342}\) includes the quality evaluation results from 339 users on 5,825 Web services for response time and throughput. QoS dataset\(^{11}\) comprises 2,507 Web services results of the evaluation of one user with the measurements of nine QoS attributes: response time, availability, throughput, likelihood of success, reliability compliance, best practices, latency, and documentation.

**Transformations:** The transformation enables to automatically construct a feature model skeleton represented in a tree structure for a process model and vice versa. Simplicity-wise, the transformation merely provides a one-to-one mapping between activities of the process model and feature constructs in the feature model by taking levels of granularity into account. The process model to feature model transformation (pm2fm) is achieved by applying the following transformation rules:

a) a root feature is created corresponding to the high-level process model;

b) for each activity in a process model a corresponding feature is constructed in the feature model tree.

If the activity is compound (or a subprocess), the procedure is recursively applied in which a child feature is created for each sub-process, and a link to the parent feature is

\(^{1}\) [http://www.wsdream.net/dataset.html](http://www.wsdream.net/dataset.html)  
\(^{2}\) [http://www.uoguelph.ca/~qmahmoud/qws/](http://www.uoguelph.ca/~qmahmoud/qws/)
established. The feature relationships in terms of optional, mandatory, and group relations are further either specified manually or determined by the feature model generator.

In like manner, the transformation of a feature model to a process model (fm2pm) follows similar rules:

3. the creation of a high-level process which is associated with the root feature;

4. the construction of a corresponding activity for each parent feature in The feature model, is such that the atomic child features at the leaves-level build atomic activities – as intermediate features also having children – recursively constructing compound activities (subprocesses) in the tree structure of the process model.

It should be noted that activities can be composed in any combination with respect to composition patterns. Therefore, a workflow of activities regarding sets of gateway labels is randomly generated, followed by labeling the process structure tree by employing a process model generator and optimization engine.

**Feature Model Generator:** The procedure of feature-model generation follows similar algorithms proposed to generate the process models because both process model and feature model are represented by tree structures. Nonetheless, labeling nodes is performed, corresponding to the semantics of the feature model. The feature model generator for the process model also leverages BeTTY[^3] and FaMa Benchmarking Systems (FaMa BS)[^4] which provide support for the visualization, functionality, and performance testing and analysis of feature models with different common formats. Desirable characteristics of the feature model can be specified in advance, and the generator optimizes and constructs a feature model whose specification is close to the optimal solution.

**Preference Model Generator:** The preference model generator produces weighted preference information, including relative importance of QoS. A tree-structure scheme is used to determine the stakeholder’s profile which organizes the preferences of a stakeholder into a hierarchical structure according to the specified criteria concerning QoS. In the course of simulation, the stakeholders profile is established according to predefined QoS criteria. In

[^3]: www.isa.us.es/betty
[^4]: http://www.isa.us.es/fama/?FaMaBenchmarking
our further experiments, random weights are produced and assigned to given QoS criteria, which determine the stakeholder’s preferences.

### 6.1.2 Experiment Design and Method

We considered a repeated-measure design for collecting data. In this method, a single group of models is generated, and independent variables are manipulated to perform further statistical inference.

To empirically evaluate the performance and scalability of our approach for process configuration, we designed different scenarios in which reference process models with different sizes in terms of number of activities and services are generated and configured. In all scenarios, the generated process models include equal ratios of composition patterns. Moreover, for each process model we generated a random feature model including normal distribution and equal ratios of variability patterns. The details and results are described in Section 6.2.

To analyze the impacts of variability and composition patterns on computational cost, we similarly follow a controlled-experimental design. However, the size of the process models are considered fixed, but the specifications of the process models vary in terms of structure and behaviour by changing the corresponding variability and composition pattern ratios. Section 6.3 presents the impact analysis and results.

**Protocol:** The experiments were conducted as follows: (i) running each algorithm 25 times, given the test case with a time limit of 10,000 seconds, for each configuration problem and calculating the average execution time, and (ii) for the optimization, each run is reiterated with a new test-case and constraints in case of unbounded solutions.

### 6.2 Performance Evaluation and Analysis

To assess performance and scalability of our approaches, we conducted a set of experiments based on extensive simulations including different scenarios and settings by real and synthetic QoS data sets. We calculate the average of the computational cost for each setting. The performance evaluation consists of two major parts. The first part focuses on the performance evaluation of quality-aware process configuration approach discussed in Chapter 5 and the later one focuses on the complexity analysis and performance evaluation of quality-range aggregation approach presented in Chapter 4.
The test bed was on an Intel Xeon Dual CPU 2.8 GHz processor with 8 GB of memory with Windows 7 and Java Virtual Machine 1.6. We utilized IBM ILOG Cplex 12.1 tool⁵ which implements a parallel branch-and-cut procedure to solve the resulting MILP models.

6.2.1 Evaluation of QoS-aware Process Configuration

Table 6.1: Characteristics of the generated models

<table>
<thead>
<tr>
<th>Experiment No.</th>
<th>Activity No. $(n_a)$</th>
<th>Service Candidates No. $(n_s)$</th>
<th>Process Model (PM)</th>
<th>Variability Model (FM)</th>
<th>QoS No. $(n_q)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I)</td>
<td>500</td>
<td>[2,128]</td>
<td>Sequence (SEQ) 25%</td>
<td>Mandatory 25%</td>
<td>5</td>
</tr>
<tr>
<td>(II)</td>
<td>600</td>
<td></td>
<td>Parallel (AND) 25%</td>
<td>Optionality 25%</td>
<td></td>
</tr>
<tr>
<td>(III)</td>
<td>1000</td>
<td></td>
<td>Multiple Choice (OR) 25%</td>
<td>Or-group 25%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Exclusive Choice (XOR) 25%</td>
<td>Xor-group 25%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Integrity Constraints 18%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1 shows different characteristics of generated models and experimental settings. We generated reference process model instances with three different sizes $(n_a)$ and random topology: (I) 500, (II) 600, and (III) 1000 activities. An empirical study provides evidence that process models in practice and industrial collections include on average 20 activities [214]. However, we considered the SAP reference model as our baseline which is considered to be a representative of a large-sized reference model. SAP reference model is a collection containing about 600 process models expressed as Event-driven Process Chains (EPCs). This reference model was meant to be used as a blueprint for roll-out projects of SAP’s ERP system. In order to achieve a high internal validity, we generated a very large process model with size of 1000 activities. All the generated models are constituted of equal ratio of random composition patterns; furthermore, for individual process model, a feature model and corresponding mapping model is generated with equal ratios of variability patterns. The integrity constraints or cross-tree constraints (CTC) ratio is set to 18%, which provides the percentage of features that have dependencies or are the dependency target of another features. This ratio value is chosen based on our analysis of feature models from SPLOT⁶ hosting a repository of feature models that are publicly available and distributed

---

⁵http://ibm.com/software/integration/optimization/cplex
⁶http://splot-research.org/
by the software product line community.

Figure 6.2: Computational cost showing average execution time of configuration versus sizes of process model in terms of numbers of activities ($n_a$) and services per each activity ($n_s$), where $t_{total}$ is total execution time for configuration, $t_{opt}$ is optimization time. Graph trendlines – $t_{total}^1: y = 3060.4x - 1410.4, R^2 = 0.978$; $t_{total}^2: y = 2520.8x - 2629.7, R^2 = 0.990$; $t_{total}^3: y = 859.54x + 2200.8, R^2 = 0.983$.

To reduce the time in experimental testing, we only considered five QoS dimensions (cost, response time, throughput, reliability and availability).

The QoS values of service candidates have been randomly generated, and we assumed that the quality information has Gaussian distribution according to the literature [23, 61, 315, 338]. We also consider the distribution of real QoS datasets [11, 342] for the generation.

Because datasets did not include the cost of service invocations, maximum and minimum price values were randomly generated – assuming uniform distributions in the interval 10 to 500. Response time values were generated with Gaussian distribution ($\mu = 383.83, \sigma = 564.36$), where $\mu$ and $\sigma$ indicate the mean and standard deviation. In the similar manner, throughput, reliability and availability were determined, considering the distribution ($\mu = 44.10, \sigma = 71.37$), ($\mu = 69.78, \sigma = 8.57$), ($\mu = 81.14, \sigma = 18.70$), respectively.

To evaluate the performance and scalability of proposed approach, we varied the number of service candidates ($n_s$) ranging from 2 to 128 per each activity of individual generated
CHAPTER 6. EVALUATION

198

process model (I, II, and III). The range number of services is increased on the logarithmic
bases. We assumed one operation for each service, i.e., \( n_{op} = 1 \).

The overall results for the different settings according to Table 6.1 are illustrated in
Figure 6.2. We measured the computational cost of configuration in seconds for the total
execution time to seek possible optimal solution for each problem instance. The total
time \( (n_{total}) \) depicted in Figure 6.2 includes the computational cost of QoS aggregation
and computation \( (n_{agg}) \), optimization \( (n_{opt}) \) (i.e., solving MILP model), and input/output
operations which are not reported. Furthermore, the computational cost of optimization
are shown by dotted lines to solely represent the optimization time. The detail statistics
are given in Table 6.2.

Table 6.2: Statistics of computational cost for different process model size.

<table>
<thead>
<tr>
<th>PM</th>
<th>( n_s )</th>
<th>Min (sec.)</th>
<th>Max (sec.)</th>
<th>Mean (sec.)</th>
<th>Std. Deviation (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( t_{total} )</td>
<td>( t_{opt} )</td>
<td>( t_{agg} )</td>
<td>( t_{total} )</td>
</tr>
<tr>
<td>500</td>
<td>[1,2]</td>
<td>5.99</td>
<td>0.95</td>
<td>2.22</td>
<td>27.24</td>
</tr>
<tr>
<td></td>
<td>[3,4]</td>
<td>8.49</td>
<td>1.89</td>
<td>3.17</td>
<td>10.35</td>
</tr>
<tr>
<td></td>
<td>[5,8]</td>
<td>9.35</td>
<td>2.22</td>
<td>3.50</td>
<td>14.64</td>
</tr>
<tr>
<td></td>
<td>[9,16]</td>
<td>14.09</td>
<td>4.17</td>
<td>3.95</td>
<td>24.16</td>
</tr>
<tr>
<td></td>
<td>[17,32]</td>
<td>21.38</td>
<td>8.49</td>
<td>6.07</td>
<td>40.17</td>
</tr>
<tr>
<td></td>
<td>[33,64]</td>
<td>44.64</td>
<td>21.65</td>
<td>7.49</td>
<td>100.26</td>
</tr>
<tr>
<td></td>
<td>[65,128]</td>
<td>101.07</td>
<td>62.52</td>
<td>11.67</td>
<td>241.45</td>
</tr>
<tr>
<td>600</td>
<td>[1,2]</td>
<td>5.90</td>
<td>0.50</td>
<td>2.45</td>
<td>11.13</td>
</tr>
<tr>
<td></td>
<td>[3,4]</td>
<td>9.19</td>
<td>2.31</td>
<td>3.42</td>
<td>32.82</td>
</tr>
<tr>
<td></td>
<td>[5,8]</td>
<td>13.05</td>
<td>5.57</td>
<td>3.70</td>
<td>58.34</td>
</tr>
<tr>
<td></td>
<td>[9,16]</td>
<td>17.90</td>
<td>5.58</td>
<td>4.17</td>
<td>86.60</td>
</tr>
<tr>
<td></td>
<td>[17,32]</td>
<td>27.56</td>
<td>11.68</td>
<td>7.04</td>
<td>175.90</td>
</tr>
<tr>
<td></td>
<td>[33,64]</td>
<td>53.94</td>
<td>29.06</td>
<td>9.42</td>
<td>317.79</td>
</tr>
<tr>
<td></td>
<td>[65,128]</td>
<td>84.32</td>
<td>38.05</td>
<td>12.42</td>
<td>1207.56</td>
</tr>
<tr>
<td>1000</td>
<td>[1,2]</td>
<td>10.36</td>
<td>2.73</td>
<td>3.12</td>
<td>45.56</td>
</tr>
<tr>
<td></td>
<td>[3,4]</td>
<td>12.39</td>
<td>3.75</td>
<td>4.06</td>
<td>99.30</td>
</tr>
<tr>
<td></td>
<td>[5,8]</td>
<td>17.17</td>
<td>5.86</td>
<td>4.34</td>
<td>171.52</td>
</tr>
<tr>
<td></td>
<td>[9,16]</td>
<td>25.50</td>
<td>9.71</td>
<td>6.93</td>
<td>440.47</td>
</tr>
<tr>
<td></td>
<td>[17,32]</td>
<td>42.74</td>
<td>17.31</td>
<td>7.61</td>
<td>271.85</td>
</tr>
<tr>
<td></td>
<td>[33,64]</td>
<td>84.35</td>
<td>42.26</td>
<td>11.30</td>
<td>2874.52</td>
</tr>
<tr>
<td></td>
<td>[65,128]</td>
<td>198.72</td>
<td>111.31</td>
<td>20.12</td>
<td>3890.78</td>
</tr>
</tbody>
</table>

The graph shows the performance and scalability. It can be observed that the
computational cost for large and very large process model in terms of sizes are not high; for
instance, the averages are less than 20 and 40 seconds for the small to medium scales in
aspect of number of service candidates per each activity ranging of [3,4] and [7,8] services,
respectively. The computational cost increases proportionally with the number of activities
and service candidates per each activity. It is noteworthy that we examined the impact of different model sizes in terms of very large number of service alternatives per activities \((n_s)\) (e.g., range of \([64, 128]\)) from theoretical point of view in order to demonstrate the scalability of approach. Nevertheless, such large sets of service candidates and settings are not encountered normally in industrial practice and have been considered to be worst-case scenario in our evaluation. For instance, the minimum and maximum computational cost to find the optimal or feasible solutions for solving MILP models \((n_{opt})\) for following settings \(n_a = 1000, n_s = [65, 128]\) is 111.31 and 3890.78 seconds, respectively (cf. Table 6.2). However, the best-average solution time obtained by the CPLEX mixed-integer solver is 317.35 seconds. It should be noted that finding sets of feasible solutions for MILP models relies on problem and given constraints. We employed MILP solver to implement branch-and-bound and cutting plan algorithms and heuristics.

6.2.2 Evaluation of QoS-range Aggregation

Revisiting our original formulated challenge, the goal of QoS range aggregation is to ensure that the required quality levels are achieved for SOSPL architecture for each product in the family. The proposed quality aggregation model considers variability patterns from both structural and behavioral perspectives.

**Complexity Evaluation:** The proposed QoS aggregation model includes the following three high-level steps: (1) quality-range aggregation of features for feature models; (2) process structure tree construction related to each feature and finding canonical process components based on composition patterns; and (3) aggregation of QoS of process components w.r.t. aggregation rules and their propagation over the feature model.

The size of the state-space in a feature model depends primarily on the size of the given feature-model graph \(G_{FM}(V, E)\). Backtracking of a feature model produces an ordering of the features in which the parent nodes are placed in post-order of their ancestors. The traversal of a feature model requires \(O(|V_{FM}| + |E_{FM}|)\), which has linear-time complexity. Given the presence of integrity constraints (includes and excludes relations) in a feature model, the size of the resulting graph will be proportional to the number of constraints. This step could have an exponential time complexity for feature models with a large number of constraints. However, we show below that this is usually not the case.

In the second step, the modular decomposition of business processes and the construction
of PST is performed in linear-time proportional to the size of the directed graph of the business process (cf. \[207\]). The third step of the algorithm for computing and aggregating quality-range values is achieved by parsing the PST of the business process model $G_{BP}(V, E)$ and control flow analysis via alternative post-order depth-first traversals. This step requires $O(|V_{BP}| + |E_{BP}|)(C + S)$, where $C$ and $S$ denote the execution time of control flow analysis for each process component; and computing quality-range values according to both the aggregation rules and the size of candidate service lists for each activity. The time required for grouping features based on variability patterns and capturing quality values does not add any computational complexity and can hence be ignored.

As a result, given that the worst-case time complexity of the first step can be in some cases exponential, the time complexity of the entire algorithm can be exponential in the worst-case. However, it is important to note that the aggregation algorithm has a linear-time complexity when the number of integrity constraints is smaller than the number of features in a feature model, i.e., the number of integrity constraints to the number of features ratio is approximately 18%.

**Evaluation Results:** We continued with yet another set of experiments after the first round which were aimed at the evaluation of QoS aggregation algorithms described in Chapter 4. We calculated the computational cost for aggregation of five quality dimensions as formerly listed. Figure 6.3 reports the average computational cost of quality aggregation and impacts of process model size varying in number of activities and service candidates per activity. For the formation of fairly large process model (i.e., 1000 activities with a range of 64 to 128 service candidates per activity), it can be observed the algorithm performs quality aggregation in reasonable time, i.e., the average computational cost being 26.38 seconds (cf. Table 6.2). As we discussed above, the time complexity of QoS aggregation algorithm is effected by the size and topology of the process model and feature model as well as specified integrity constraints, which can be exponential in worst-case scenario.

Based on experiments, the trend line reveals a moderately strong linear relationships between computational cost and the sizes of process model and feature model with given distributions (cf. Table 6.1), considering very large-size models.
6.3 Impact Analysis of Variability and Composition Patterns on Computational Cost

We conducted another set of experiments to explore the computational cost impacts of variability and composition patterns expressing structural changes and behavioural characteristics of a reference process model. We only focused and measured optimization time of the MILP solver, as a performance indicator for computational cost during the experiments. Our main objectives are to analyze the effects of variability and composition patterns individually and independently of each other on computational cost which is considered as variable of interest in the course of experiments. We refined our objectives into the following questions that we tested through a series of experiments:

i) Is computational cost to find the possible optimal solution during optimization significantly impacted by the magnitude ratio of the individual pattern?

ii) Are there significant differences among variability/composition patterns on computational cost?
iii) Do variability and composition patterns independently have the same impact?

To analyze the performance of the proposed approach and test differences between experimental conditions, we carried out a series of statistical tests. Accordingly, we conducted tests to examine significant differences within-group (i.e., by considering individual patterns partitioned into groups varying in ratios of pattern) and between-group (i.e., by taking into account difference among variability/composition patterns).

The analysis of variance (ANOVA) was used as a general statistical procedure to partition and analyze the variation in a continuous response variable. All the statistical computations have been performed by the SPSS statistical software package. We found that data used in validation and verification experiments does not have normal distributions because of their inherent random structure.

To validate the normality assumption, we prioritized the conduction of a set of normality tests. A comprehensive review of which is presented in [305]. Kolmogorov-Smirnov and Shapiro Wilk tests which are among the most popular and widely-used tests of normality indicated that the conditions of normality were not fulfilled. With the conditions of normality on data not satisfied, non-parametric statistical tests for further statistical analysis were utilized. Non-parametric tests particularly do not rely on precise assumptions about the distribution of variables and normality. We also employed Friedman test [280] throughout further series of the experiments with the corresponding post-hoc tests for comparisons of grouped multiple models.

The Friedman test – a non-parametric statistical test utilizing ranks – is credited as one of the best known procedures to test the differences among more than two related samples [116, 142], and is comparable with two-way repeated measures Analysis of Variance (ANOVA) [280]. If the result of the Friedman’s two-way analysis of variance by ranks is significant, it indicates that there is a significant difference between at least two of the sample medians in the set of k medians.

Let’s assume N sample models, known as blocks, and k dependent groups each of which includes the computational cost of the algorithm with different ratios of variability patterns in process models. For each block, \( r^j_i \) is the rank of the \( j \)-th of \( k \) groups in the \( i \)-th of \( N \) data sets. The Friedman test compares the average ranks of groups, \( R_j = \frac{1}{N} \sum_{i=1}^{N} r^j_i \). The total sum of ranks for all \( N \) blocks and all \( k \) groups is \( Nk(k+1)/2 \). The expected value of the rank sum is \( N(k+1)/2 \) for every group if there is no difference between groups (e.g., under
CHAPTER 6. EVALUATION

For each group, the difference between the rank sum and its expected value can be computed and squared. The test statistic of the Friedman test denoted by $\chi^2_r$ is the sum of these squared differences. We calculate the Friedman test statistic from the following formula:

$$
\chi^2_r = \frac{12}{Nk(k+1)} \sum_{j=1}^{k} \left( R_j - \frac{k+1}{2} \right)^2 = \left( \frac{12}{Nk(k+1)} \sum_{j=1}^{k} R_j^2 \right) - 3N(k+1)
$$

(6.1)

where $(k+1)/2 = \sum_{i=1}^{n} \sum_{j=1}^{k} r_j^i / nk$ is the average rank assigned via this within-blocks ranking scheme. The chi-square ($\chi^2$) distribution is utilized to approximate the statistic of the test with $k-1$ degrees of freedom (d.f.) if $N > 10$ (cf. \[142\] for indications of proof). Using this distribution, the corresponding $p$-value is $p = P[\chi^2_r \geq \chi^2_{k-1,\alpha}]$, which helps to statistically infer if there is a significant difference among groups at the $\alpha$ level of significance. We chose the significance level to be 95% ($\alpha = 0.05$) as a standard benchmark. Accordingly, $p$-values $< \alpha$ were considered statistically significant.

For pairwise statistical comparisons, we used Wilcoxon signed-ranks test – a safe and robust non-parametric test which is alternative to the pairs – as a post-hoc procedure based on the standardized mean ranks to decide which groups are significantly different from each other \[116, 142\].

6.3.1 Structural Variability Analysis

To explore the effects of structural variability of the process model on computational cost for configuration, we generated $N = 100$ process models with random topology without cycle whose characteristics include equal ratio of composition patterns, $n_a = 300$ activities, and $n_s = 128$ service candidates per activity. Each experiment set focuses on a primary individual variability pattern, i.e., optionality, mandatory, Or-group, and Xor-group as well as integrity constrains. For each variability pattern, we generated four random feature models for each process model with different percentile ratio values increased incrementally by 25%, 50%, 75% and 100%. For each set of experiments corresponding to variability
patterns, we fixed the percentage ratio for the integrity constraints at 18%. Accordingly, for each experiment, we partitioned process models with their associated feature models including the equivalent ratio for each variability pattern into four groups (I, II, III, and IV). Then, the configuration algorithms for each group were executed to find the optimal solution, and the performance of MILP solver for each process model configuration was measured in terms of computational cost. Table 6.3 gives the detailed and descriptive statistics of the experiments for different variability patterns varying in ratios.

We fulfilled the statistical tests to determine “whether or not the computational cost to find a possible optimal solution is significant for different process model instances associated with different feature models varying in variability patterns”.

We carried out a series of Friedman tests in respect of average ranks among computational costs in terms of optimization time for different variability patterns (c.f. Table 6.4). For the optionality pattern, the calculated values of the statistical test \( \chi^2 = 0.980, p > 0.05 \) shows that for the within-group process models whose feature models include different ratios of optionality, there are no significant differences in optimization time. As a result, based on the statistics we can predict that the computational cost to configure very large-size models and to find a possible optimal solution cannot be likely impacted by the ratio of optionality pattern.

For the mandatory pattern, an additional test indicates that the optimization time for four groups are significantly different \( \chi^2 = 269.916, p < 0.001 \). This is due to the fact that the escalation of ratio of mandatory features in a feature model imposes and adds more corresponding constraints into the optimization model, which may result in reducing search space to find the optimal solution among feasible ones throughout the decision tree for large-size process models. Another test reveals there are differences among groups where feature models contain variation of ratios of Or-group relations, \( \chi^2 = 20.604, p < 0.001 \). Thus, it appears that the computational cost can also be affected by varying this pattern.

Similarly, the optimization time responded differently by changing the ratio of the Xor-group pattern. The Friedman test found significant variations among groups, \( \chi^2 = 41.124, p < 0.001 \).
Table 6.3: Descriptive statistics of computational cost (optimization time) corresponding to
different ratio values of variability patterns.

<table>
<thead>
<tr>
<th>Variability patterns</th>
<th>Ratio (%)</th>
<th>Mean (sec.)</th>
<th>Std. Error of Mean (sec.)</th>
<th>Median (sec.)</th>
<th>Std. Deviation (sec.)</th>
<th>Minimum (sec.)</th>
<th>Maximum (sec.)</th>
<th>Percentiles (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>Optional</td>
<td>25%</td>
<td>34.690</td>
<td>5.298</td>
<td>19.967</td>
<td>52.978</td>
<td>7.432</td>
<td>316.479</td>
<td>14.121</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>28.258</td>
<td>3.893</td>
<td>17.774</td>
<td>38.926</td>
<td>4.764</td>
<td>250.580</td>
<td>13.030</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>58.417</td>
<td>18.539</td>
<td>19.134</td>
<td>185.391</td>
<td>3.455</td>
<td>1372.820</td>
<td>12.334</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>53.471</td>
<td>11.930</td>
<td>15.933</td>
<td>119.302</td>
<td>3.976</td>
<td>826.763</td>
<td>12.283</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>13.494</td>
<td>1.440</td>
<td>9.655</td>
<td>14.403</td>
<td>2.155</td>
<td>105.684</td>
<td>4.978</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>2.421</td>
<td>0.214</td>
<td>1.828</td>
<td>2.142</td>
<td>1.015</td>
<td>18.859</td>
<td>1.487</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>0.710</td>
<td>0.029</td>
<td>0.587</td>
<td>0.287</td>
<td>0.500</td>
<td>1.584</td>
<td>0.551</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>34.165</td>
<td>6.080</td>
<td>19.838</td>
<td>60.803</td>
<td>5.688</td>
<td>430.708</td>
<td>13.568</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>43.544</td>
<td>10.173</td>
<td>12.541</td>
<td>101.734</td>
<td>7.287</td>
<td>694.728</td>
<td>10.555</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>82.684</td>
<td>20.112</td>
<td>12.137</td>
<td>201.121</td>
<td>8.817</td>
<td>1383.357</td>
<td>10.764</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>98.567</td>
<td>18.900</td>
<td>12.785</td>
<td>189.004</td>
<td>8.751</td>
<td>1158.536</td>
<td>11.549</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>29.264</td>
<td>3.391</td>
<td>20.400</td>
<td>33.915</td>
<td>5.830</td>
<td>285.760</td>
<td>13.196</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>11.839</td>
<td>0.937</td>
<td>10.012</td>
<td>9.374</td>
<td>2.030</td>
<td>64.200</td>
<td>5.299</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>4.795</td>
<td>0.318</td>
<td>3.607</td>
<td>3.181</td>
<td>1.580</td>
<td>19.780</td>
<td>2.578</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>3.000</td>
<td>0.193</td>
<td>2.429</td>
<td>1.927</td>
<td>1.440</td>
<td>14.080</td>
<td>1.966</td>
</tr>
</tbody>
</table>

Table 6.4: The mean ranks and test statistics of the optimization time corresponding to
different ratio of variability patterns.

<table>
<thead>
<tr>
<th>Variability pattern</th>
<th>(I) 25%</th>
<th>(II) 50%</th>
<th>(III) 75%</th>
<th>(IV) 100%</th>
<th>Statistic ($x^2$)</th>
<th>Overall $p$-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>2.550</td>
<td>2.410</td>
<td>2.570</td>
<td>2.470</td>
<td>0.980</td>
<td>0.805</td>
</tr>
<tr>
<td>Mandatory</td>
<td>3.780</td>
<td>3.170</td>
<td>2.030</td>
<td>1.020</td>
<td>269.910</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Or-group</td>
<td>2.860</td>
<td>2.110</td>
<td>2.340</td>
<td>2.690</td>
<td>20.600</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Xor-group</td>
<td>2.110</td>
<td>2.160</td>
<td>2.590</td>
<td>3.140</td>
<td>41.120</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Integrity constraints</td>
<td>3.760</td>
<td>2.960</td>
<td>1.980</td>
<td>1.300</td>
<td>210.570</td>
<td>&lt;0.001</td>
</tr>
</tbody>
</table>
In addition, we performed a test to observe how the presence of integrity constraints affects computational cost. For this purpose, in the course of generating random feature models, we consider equal ratios (25%) for variability patterns of optionality, mandatory,
Or-group and Xor-group, as a fixed variable for all generated feature models, but we varied the ratio of integrity constraints for four groups, i.e., (I:25%), (II:50%), (III:75%), and (IV:100%). The statistical test ($\chi^2 = 210.576, p < 0.001$) confirms that the integrity constraints in the feature models can significantly impact optimization time. Similar to a mandatory pattern, the integrity constrains produce constraints sets which are further added to the optimization model. Such constraint can reduce the search space to obtain a possible optimal solution if one exists. Figure 6.4 and 6.5 displays the box plots for variability patterns.

![Box plots for the optimization time of process model configuration by increasing the ratio values of integrity constraints.](image)

Figure 6.5: Box plots for the optimization time of process model configuration by increasing the ratio values of integrity constraints.

Results of the tests revealed statistical significance in computational cost for mandatory, Or-group, and Xor-group patterns, as well as integrity constraints. Hence, we carried out analyses for pairwise comparisons to test the significance of the relationships within each group. By applying Wilcoxon signed-ranks tests, we proceeded with the post-hoc procedures in order to find the particular pairs of ratios which may produce differences in computational cost. Pairwise statistical procedures perform individual comparisons between two ratios of patterns, obtaining a $p$-value independent from another pair in each case.

Table 6.5 presents the results of the tests for four groups corresponding to variability patterns with respect to their given ratios. For a mandatory pattern, there is no significant variation between the average ranks of computational costs of groups (I) and (II) where models include 25% and 50% ratios of mandatory features respectively whereas significant
Table 6.5: Pairwise comparison analysis of variability patterns according to groups (I-IV) including different ratio by their $p$-values.

<table>
<thead>
<tr>
<th>Variability pattern</th>
<th>(I) 25%</th>
<th>(II) 50%</th>
<th>(III) 75%</th>
<th>(IV) 100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(II) 50%</td>
<td>0.005</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(III) 75%</td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
<td></td>
</tr>
<tr>
<td>(IV) 100%</td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Or-group</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(II) 50%</td>
<td></td>
<td>&lt;0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(III) 75%</td>
<td>0.026</td>
<td>1.000</td>
<td>0.090</td>
<td>0.331</td>
</tr>
<tr>
<td>(IV) 100%</td>
<td>1.000</td>
<td>0.051</td>
<td>0.111</td>
<td></td>
</tr>
<tr>
<td>Xor-group</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(II) 50%</td>
<td></td>
<td></td>
<td>&lt;0.001</td>
<td></td>
</tr>
<tr>
<td>(III) 75%</td>
<td></td>
<td></td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>(IV) 100%</td>
<td></td>
<td></td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Integrity Constraints</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.6: The mean ranks and the test statistic for group analysis of variability patterns.

<table>
<thead>
<tr>
<th>(I) Optional</th>
<th>(II) Mandatory</th>
<th>(III) Or-group</th>
<th>(IV) Xor-group</th>
<th>(V) Integrity Constraints</th>
<th>Statistic ($\chi^2$)</th>
<th>$p$-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.600</td>
<td>1.970</td>
<td>2.230</td>
<td>3.890</td>
<td>2.300</td>
<td>442.567</td>
<td>&lt;0.001</td>
</tr>
</tbody>
</table>

Differences are detected for the other paired comparisons. One observation is that the computational cost is lessened for the model with a large size when the ratio of mandatory patterns is more than 50%. For Or-group pattern, the test detected that there is only statistical difference when the ratio of pattern rises 25% to 50%; otherwise, there are no considerable differences among other pairs. The pairwise comparisons showed that the computational cost differs from group (IV) when a XOR-group pattern varying significantly in ratios for groups (I), (II), and (III) is present. The results also exhibit that there are significant variations in computational costs among all pairs for the integrity constraints.

The test gives lower $p$-values for all the paired comparisons and is more likely to reject, for instance, the null-hypothesis about the increase or decrease of integrity constraints in the feature model which have no influence on computational costs.

The last series of tests were performed for group analysis of variability patterns to investigate and identify whether or not these patterns have equal impacts on optimization
CHAPTER 6. EVALUATION

Table 6.7: Pairwise comparisons of variability patterns by their $p$-values.

<table>
<thead>
<tr>
<th>Variability patterns</th>
<th>(I) Optional</th>
<th>(II) Mandatory</th>
<th>(III) Or-group</th>
<th>(IV) Xor-group</th>
</tr>
</thead>
<tbody>
<tr>
<td>(II) Mandatory</td>
<td>&lt;0.001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(III) Or-group</td>
<td>0.100</td>
<td>&lt;0.001</td>
<td></td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>(IV) Xor-group</td>
<td>0.092</td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
<td></td>
</tr>
<tr>
<td>(V) Integrity Constraints</td>
<td>&lt;0.001</td>
<td>0.037</td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
</tr>
</tbody>
</table>

computational cost. The test reports that the average ranks of computational cost for different patterns are widely different (cf. Table 6.6).

We proceeded with the Wilcoxon signed-rank test to find out which paired patterns actually differ and compare their effects on computational cost. The pairwise comparison matrix is calculated and tabulated in Table 6.7. The statistics indicate that there are no significant differences between optionality and Or-group patterns, and optionality and Xor-group patterns. Similarly, the test showed statistical difference between mandatory patterns and integrity constraints. The results are deemed statistically significant (at least $p < 0.001$) for other paired patterns.

Accordingly, it may yield evidence to estimate that the presence of patterns which statistically have no significant differences can likely have the same impact on the entire computational cost to find the optimal solutions in the course of optimization of a large-size model (e.g., the number of optional features and features in Or-group relations). Figure 6.6 demonstrates the box plots of the analysis of the variability patterns.

6.3.2 Analysis of Composition Patterns

We conducted a series of experiments to statistically analyze the effects on computational cost of composition patterns (i.e., workflow patterns) expressing the behaviour of the process model by solely focusing on optimization time. We aimed at exploring whether the ratio of individual composition patterns of a process model has significant effects on computational cost of optimization and if composition patterns have equal effects. For this purpose, the experimental design involved the generation of process models with different ratios of composition patterns. We considered only main sequential and parallel patterns related to the control-flow of the business process model, i.e., sequence (SEQ), parallel (AND), multiple choice (OR), and exclusive choice (XOR). We considered percentile ratio values of
25%, 50%, 75%, and 100%, referred to as groups I-IV for individual composition patterns. For each group, we generated $N = 100$ process models with random topology, including $n_a = 300$ activities and $n_s = 128$ service candidates per activity. We executed the configuration algorithms for each group to find the optimal solution and measured the performance of the MILP solver in terms of computational cost for each process model configuration. Table 6.8 presents details of the statistics of the computational cost of optimization (MILP) for composition patterns as the result of variable ratios for individual patterns.

For each experiment, we examined the influence of each composition pattern. Table 6.9 summarizes the results of the tests. For the individual composition pattern, the statistics reveals that there are significant differences among computational costs as the result of changing variable ratios of a pattern partitioned into the four inner groups (I-IV). Accordingly, the presence of any composition pattern with different distributions in the process model can have effects on the computational cost of optimization in the course of process model configuration. Figure 6.7 depicts the box plots representing the impacts of different composition patterns on computational cost.
Table 6.8: Descriptive statistics of optimization time corresponding to different ratio values of composition patterns.

<table>
<thead>
<tr>
<th>Composition patterns</th>
<th>Ratio (%)</th>
<th>Mean (sec.)</th>
<th>Std. Error of Mean (sec.)</th>
<th>Median (sec.)</th>
<th>Std. Deviation (sec.)</th>
<th>Minimum (sec.)</th>
<th>Maximum (sec.)</th>
<th>Percentiles (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25%</td>
<td>133.287</td>
<td>21.920</td>
<td>33.705</td>
<td>9.420</td>
<td>486.370</td>
<td>17.022</td>
<td>33.705</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>78.063</td>
<td>12.884</td>
<td>18.689</td>
<td>10.960</td>
<td>633.920</td>
<td>14.060</td>
<td>18.689</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>10.124</td>
<td>0.187</td>
<td>10.253</td>
<td>4.920</td>
<td>17.550</td>
<td>9.631</td>
<td>10.253</td>
</tr>
<tr>
<td>Sequence (SEQ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>221.771</td>
<td>31.783</td>
<td>66.362</td>
<td>25.823</td>
<td>1748.380</td>
<td>25.823</td>
<td>66.362</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>550.571</td>
<td>67.247</td>
<td>344.727</td>
<td>90.018</td>
<td>3527.950</td>
<td>90.018</td>
<td>344.727</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>341.708</td>
<td>47.452</td>
<td>225.895</td>
<td>130.032</td>
<td>4108.560</td>
<td>130.032</td>
<td>225.895</td>
</tr>
<tr>
<td>Parallel (AND)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>89.134</td>
<td>15.655</td>
<td>22.653</td>
<td>19.200</td>
<td>156.546</td>
<td>13.968</td>
<td>22.653</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>118.211</td>
<td>81.552</td>
<td>20.820</td>
<td>25.823</td>
<td>815.525</td>
<td>25.823</td>
<td>20.820</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>29.094</td>
<td>6.743</td>
<td>12.901</td>
<td>11.028</td>
<td>67.434</td>
<td>11.028</td>
<td>12.901</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>10.049</td>
<td>0.291</td>
<td>9.639</td>
<td>7.945</td>
<td>2.523</td>
<td>7.945</td>
<td>9.639</td>
</tr>
<tr>
<td>Multiple Choice (OR)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>77.093</td>
<td>42.633</td>
<td>13.801</td>
<td>11.028</td>
<td>426.327</td>
<td>11.028</td>
<td>13.801</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>3.085</td>
<td>0.164</td>
<td>2.523</td>
<td>2.452</td>
<td>2.523</td>
<td>2.452</td>
<td>2.523</td>
</tr>
<tr>
<td>Exclusive Choice (XOR)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.9: The mean ranks and the test statistics of the optimization time corresponding to different ratio of composition patterns.

<table>
<thead>
<tr>
<th>Composition pattern</th>
<th>(I) Statistic</th>
<th>(II) Statistic</th>
<th>(III) Statistic</th>
<th>(IV) Statistic</th>
<th>Overall p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence (SEQ)</td>
<td>3.410</td>
<td>3.010</td>
<td>2.350</td>
<td>1.230</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Parallel (AND)</td>
<td>1.790</td>
<td>2.250</td>
<td>3.030</td>
<td>2.930</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Multiple Choice (OR)</td>
<td>3.390</td>
<td>3.010</td>
<td>2.260</td>
<td>1.340</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Exclusive Choice (XOR)</td>
<td>3.460</td>
<td>2.990</td>
<td>2.550</td>
<td>1.000</td>
<td>&lt;0.001</td>
</tr>
</tbody>
</table>
Table 6.10 shows the results of pairwise comparisons for each composition pattern according to ratios. The tests resulted in significant differences at a significance level of $\alpha = 0.05$ for all the paired patterns except for the parallel (AND) pattern in group (III) and (IV), where the variable ratio of patterns varies from 75% to 100%. For this particular pattern at the given ratios, the test cannot help us to infer the causality of confirmed non-significant result.
CHAPTER 6. EVALUATION

According to overall tests results, we can expect that the underlying ratios of any composition patterns for large-size business process models can impact the computational cost of optimization.

Table 6.10: Pairwise comparison of composition patterns according to within-groups (I-IV) representing different ratio by their p-values.

<table>
<thead>
<tr>
<th>Composition pattern</th>
<th>(I) 25%</th>
<th>(II) 50%</th>
<th>(III) 75%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence (SEQ)</td>
<td></td>
<td>0.028</td>
<td></td>
</tr>
<tr>
<td>Parallel (AND)</td>
<td></td>
<td></td>
<td>&lt;0.001 &lt;0.001</td>
</tr>
<tr>
<td>Multiple Choice (OR)</td>
<td></td>
<td>0.0460 &lt;0.001</td>
<td></td>
</tr>
<tr>
<td>Exclusive Choice (XOR)</td>
<td></td>
<td>&lt;0.001 &lt;0.001</td>
<td>0.0120</td>
</tr>
</tbody>
</table>

Table 6.11: The mean ranks and the test statistic for group analysis of composition patterns.

<table>
<thead>
<tr>
<th>(I) Sequence (SEQ)</th>
<th>(II) Parallel (AND)</th>
<th>(III) Multiple Choice (OR)</th>
<th>(IV) Exclusive Choice (XOR)</th>
<th>Statistic ($\chi^2$)</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.430</td>
<td>3.310</td>
<td>2.250</td>
<td>2.020</td>
<td>231.759</td>
<td>&lt;0.001</td>
</tr>
</tbody>
</table>

Table 6.12: Pairwise comparisons of composition patterns by their p-values.

<table>
<thead>
<tr>
<th>Composition patterns</th>
<th>(I) Sequence (SEQ)</th>
<th>(II) Parallel (AND)</th>
<th>(III) Multiple Choice (OR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(II) Parallel (AND)</td>
<td>&lt;0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(III) Multiple Choice (OR)</td>
<td>0.0460 &lt;0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(IV) Exclusive Choice (XOR)</td>
<td>&lt;0.001 &lt;0.001</td>
<td></td>
<td>0.0120</td>
</tr>
</tbody>
</table>

Furthermore, we performed group analysis between composition patterns to determine if they significantly impact computational cost. Tables 6.11 and 6.12 show the results of the tests and associated post-hoc pairwise comparison, which indicate statistically significant differences among composition patterns.
Figure 6.8 illustrates the results of the measurement of optimization time for the presence of different composition patterns grouped by variable ratios.

Figure 6.8: Box plots for the optimization time among composition patterns corresponding to different ratios.

Figure 6.9: Box plots for the optimization time for grouped models.

In the last series of the experiments, we examined if variation in the structure of the
business process model impacts computational cost in comparison with the process model without structural variability. For this purpose, we designed an experiment where we partitioned all the generated process models into two groups. The constructed process models with associated variability models (i.e., feature models) assigned to group (I), and group (II) comprises process models without variability models. Each group includes $N = 1600$ process models in total. The Friedman test statistic ($\chi^2_1 = 1585, p < 0.001$) marks statistical significance. The box plots are displayed in Figure 6.9. The computational cost for process models with no structural variability can be high for some test-cases. This is because of the fact that a process model without structural variability can be viewed as a process model where all the activities are mandatory features which impose constraints for any optimization model. In consequence, it may pose more computational burden on performing multi-objective optimization to obtain either feasible or optimal solutions.

6.4 Discussion

Our solution to support automatic QoS-aware configuration of business process variants is based on modeling and solving the configuration problem by Integer Programming (IP). Our approach is based on local and global constraints optimization by taking into account structural variability in the process. The results of experimental evaluations show the performance and prove the scalability of our proposed approach. The scalability analysis of the approach, including optimization and QoS aggregation, has been performed by considering a large set of randomly-produced problem instances, where characteristics of the process models have been varied. In realistic scenarios at the enterprise level, we argue that the approach is practical for medium and large-size process models. By taking our modest experimental setup and computing power into account, the computational cost required to solve the MILP optimization models by the solver and deploy a configured composite service was about six minutes on average and one hour in maximum for very large problem instances. In some cases, the MILP optimization model could not be solved in reasonable execution time because of constraints imposed on the models.

We employed the branch-and-cut technique implemented by CPLEX, which has a worst-case exponential-time complexity. However, CPLEX is one of the highly efficient commercial off-the-shelf MILP-solvers leveraging state-of-the-art implementations of simplex and
branch-and-bound search algorithms with cuts and heuristics for integer programming problems. Modern computing environments and advanced solvers with hybrid approaches mitigate the time complexity and computational cost to determine the optimum solution in the course of configuration. Furthermore, the linear programming models have polynomial-time complexity. One can see that solving the configuration problem instances with extremely large-size variable sets and constraints lead to reasonable performance. Nevertheless, optimization is subject to the trade-offs between performance and scalability, and between solution complexity and feasibility of the models.

As we discussed in Chapter 2, there is a large body of ongoing research which addresses business process optimization, particularly QoS-aware service selection. Our approach for the optimization is inspired by the seminal work presented by Zeng et al. [338], where the dynamic service composition problem has been modeled by means of Integer Programming (IP). In recent works, similar approaches have been proposed for business process adaptation and optimization [15, 23, 24, 59, 126]. The complexity of problems and mathematical optimization are analyzed, and their results also indicate the applicability of IP-based approach. Notwithstanding, the majority of previous studies focused on run-time self-adaptation of the SOA-based applications, dynamic generation of service composition plans (i.e., workflow restructuring), and run-time service selection in a dynamic environment where the QoS services may undergo changes during the execution of a composite service. Moreover, most of the proposed approaches have considered only the sequential composition models by assuming the fact that non-sequential constructs in a process model are transformed into sequential ones. Nonetheless, to the best of our knowledge, none of the current approaches take into account design-time variability and constraints in process optimization. Service providers very often plan for variability at the design time which has been also our focus in the context of SOSPL.

The impact analysis of variability and composition patterns shows the effects of structural variability and behavioural characteristics of the process model on optimization and overall performance. The results indicate that the optionality pattern has no significant effect on the computational cost of optimization. Notably, statistically significant differences were observed for other patterns. In consequence, we can expect different performances or change in computational cost in the presence of such patterns in a business process model in the course of configuration and optimization.

It should be noted that QoS constraints defined by a stakeholder specify limitations on
process model configuration, which in turn lead to an NP-hard problem to guarantee QoS constraints compliance. Accordingly, increasing the number of QoS constraints in MILP model obviously affects computational cost in finding the optimal solution.

Limitations: In our approach, we used structured process models which provide a feasible approach for the QoS aggregation by considering that most process modeling tools and BPEL—as a standard workflow language—support structured process models. Accordingly, we did not cover unstructured process models. Although unstructured process models can be more expressive from a modelling perspective, they can be more prone to error. Unstructured process models can be transformed into structured constructs based on the alignment of the considered composition (workflow) patterns [5]. We considered only acyclic process models in our experiments, i.e., in these models, there were no paths that created cycles. However, arbitrary cycle in the model can be handled by adopting a loop pealing mechanism prior to starting the optimization process, where cycles are unfolded and iterations are transformed into sequential branches according to the number of iterations and probabilities of executions.

A potential limitation of our approach using mixed-integer linear programming is the overhead of linearization of objective functions, QoS dimensions, and related operations which are characteristically non-linear in nature. Adopting meta-heuristic search-based methods, like genetic algorithms (GA), can help cope with non-linear constraints needless of linearization. Several approaches based on GA have been proposed for the process optimization in the literature [56, 113, 154, 197, 335, 339]. In our previous work [235], we also utilized GA to model and solve the optimization problem of business process configuration. It is noteworthy that the GA-based approaches do not guarantee an optimal solution. Furthermore, one practical difficulty of using GA for multi-objective optimization is to determine the parameters, fitness functions, and stopping criterion for the iterative optimizer due to the fact that GA convergence is problem-dependent (e.g., [197, 235]). In [57], as one of the first attempts, the convergence times of GA and Integer Programming for the same achieved solution are compared through limited numerical simulation, using an open-source MILP-solver[^7]. The results indicated that Integer Programming outperforms GA because problem instances are comparatively smaller, and GA performance surpasses if the combinatorial size is too large. However, it should be noted that results of any solver benchmark

is inherently limited by the temporal nature of the solver itself. In addition, commercial
solvers significantly outperform non-commercial ones due to leveraging algorithmic advances
in the field of computational Integer Programming and employing a variety of techniques
that exploit the characteristics of the problem to reduce the amount of required computa-
tions. These may include the constraints reformulation and derivation of an alternative and
more compact linear-programming relaxation of the problem.
Chapter 7

Conclusions and Future Research

This chapter concludes the thesis and provides an outlook on future work. We present a summary of the key contributions of this thesis and answer the research questions. We also describe ongoing related research and an outlook on some further research questions that we did not answer in this thesis.

7.1 Summary and Main Contributions

This thesis addresses the problem of variability management and high-level configuration of service-based systems, particularly focusing on the business process layer in SOA which hinders to address efficient and automated user-centered approaches facilitating the tailoring of service products to individual stakeholders.

Process models are used to describe service compositions in practice; however, they lack systematic reuse and decision support to enable configuration and customization of service-based applications based on individual stakeholder’s requirements and preferences. The concept of a configurable process model has been proposed to address these limitations. A configurable process model can be viewed as a family of services, which is an integrated representation of multiple variants of a process model defining service composition in a given domain and serving as a basis to enable the reuse and configuration of service products. We have observed that existing approaches to model and manage variability for configurable process models face three major shortcomings and challenges.

First and foremost, the absence of systematic approaches to capture and manage variability to construct configurable (or customizable) business process models is a notable
drawback. The majority of current surveyed approaches are characterized as extending conventional process (workflow) modeling languages with the notion of variation points and adding constructs to describe variability and support configuration options (decisions). Variations points are usually associated with specific process model elements (i.e., workflow elements such as service activities, data objects, and resources), which can be linked to elements in a domain model so as to facilitate model configuration. Hence, different mechanisms are consequently proposed to incorporate variability into (reference) process modeling languages; however, the shortcomings affecting the existing approaches are continually increasing the complexity of process models and the lack of expressiveness to model and implement a systematic management of variability for both functional and non-functional (quality) aspects. Defining variability as an integrated part of development software artifacts (i.e., process models) in current approaches has also significant drawbacks. The variability information is scattered across software artifacts and makes it infeasible to manage them in a consistent manner; the configuration of large process models leads to unmanageable complexity, as a consequence. It is also too intricate to determine the influence of required variability information on design, implementation, or test artifacts.

To address these issues, the first part of this thesis focused on devising a general approach and conceptual foundation for configurable business process models. We adopted feature-oriented approach based on software product-line, because of the fact that variability management has been extensively researched in the field of software product line engineering. In particular, the objective was to define service-oriented software product line (SOSPL) by combining software product line and service-orientation to achieve configurable process models, manage process variability, and support automated reasoning for the selection of configuration options.

Through the analysis of the state-of-the-art, we realized that configuration or customization requires an approach to explicitly model and systematically manage process variants for service configuration and the capabilities that are required to address these challenges. We found that configurability needs concepts and mechanisms that require to be integrated into the “design process” to provide a generic solution. The fact is that configurable process models realizing SOSPLs have to be implemented systematically and efficiently in order to facilitate reuse, variation, and automated quality-aware service-product configuration. We thus followed this approach by providing general concepts that enable native support for configurability. We extended a conventional software product line life-cycle to support
modeling and managing process model variants, which contributes to the development of variant-rich service-oriented applications and the derivation of stakeholder-tailored services. In our approach, based on requirement analysis in domain engineering phase, a reference process model describing service composition is designed and developed in a consolidated manner which captures multiple variants of a process (common and variable services). We provided insights on how to capture variation points in a language-independent manner for the purpose of representing configurable process models. To manage variability, we model and express variability relations and configuration dependencies of a reference process model by a variability model.

Decoupling the variability model from the process model leads to lower complexity, facilitates business process management and maintenance, and further enables automatic variability analysis and configuration. Complexity emerges from a typically large number of variation points and variants defined in the business process layer. We used feature models, as an orthogonal variability model encapsulating the configuration knowledge, where features abstract over commonality and variability pertaining to functional and non-functional properties of services in a (reference) process model. The link between feature model and reference process model was achieved by mapping common and variant services in process to corresponding features. We utilized the feature model as the baseline to configure process model by supporting automatic feature selection in the course of process configuration.

The second issue stems from a lack of appropriate mechanisms in existing approaches to model and integrate QoS and evaluate supported quality range by considering variations in both functional and non-functional (quality) properties. The quality evaluation becomes crucial as it allows for the examination of whether the final service product satisfies and guarantees quality requirements within the envisioned scope of the product line. The second part of this thesis has focused on proposing a QoS model and a framework for QoS aggregation and computation to address the evaluation of quality ranges in SOSPLs, which help assess the potential quality of the developed SOSPL and are further used as an essential part of the quality-aware service-product derivation, decision-making, and process optimization. We identified a set of variability patterns which may occur within composition patterns and proposed aggregation rules and a method for quality-range computation for holistic evaluation and prediction of quality ranges of an SOSPL based on configurable process model.
Last but not least, the third shortcoming is that there is a lack of decision-making support for automatic configuration and optimization of process model variants by considering stakeholders’ preferences concerning quality. Studies have shown that configuration and service product derivation have to be automated as far as possible since stakeholders need guidance for decision-making [256].

The last part of this thesis focused specifically on quality-aware configuration of SOS-PLs by incorporating the relevant importance of stakeholder’s preferences. To this end, we have proposed and developed a preference-based configuration framework, where QoS is leveraged as an important means to evaluate the quality of service and to provide decision criteria for the configuration and derivation of stakeholder-tailored services. We described how to encode a process configuration problem into a Constraint Optimization Problem (COP) in order to exploit Constraint Programming to implement an automatic or interactive configuration system base. We showed how feature configuration (feature selection) can be formulated by Integer Programming (IP), suggesting how to translate the semantics of feature model and formulate it into Mixed-Integer Linear Programming (MILP). Our proposed configuration approach provides decision-making support during the course of configuration to achieve the optimal solutions within the constraints boundaries specified by stakeholders and configurable limits. One of the observations of conducted studies shows that the vast majority of approaches of configurable business process models have not been fully validated because of required efforts for constructing such process models [265].

We developed a simulation framework and conducted a series of experiments to construct, analyze, and study complex process models in terms of different control-flow structure and variability patterns to gain insights into performance and scalability for large-size problem instances. The results of experimental evaluations indicate the acceptable performance in terms of computational cost and prove the scalability of our proposed approach.

7.2 Assessment of the Research Questions

In this section, we assess the proposed methods in this thesis based on the research questions described in Chapter 1.

Research Question RQ1: In Chapter we conducted a study to explore the available evidence regarding the integration or adoption of both software product line and service orientation principles and approaches. We used a systematic mapping study to constitute a
classification of research objectives and overview of existing approaches. The results show
that the combination of SO and SPLE is promising. It not only enables architectures to be
reused in different instances but also supports variability management, configuration and
customization, where benefits stem from SPLE principles. It helps develop architectures
for adaptive systems in responding effectively to dynamic functional and non-functional re-
quirements and to construct dynamic software product lines, where the advantages stem
from SO principles as well. The results mark an escalation of adaptation of SPL-based
approaches in service-oriented design and development; it is an emerging field, nonetheless,
the proposed approaches are still at an early stage and gaining maturity. Several consider-
able conceptual proposals have been presented in this area. We also observed that the vast
majority of approaches employed variability models – most notably using feature models –
in order to address variability management in different stages of the development life-cycle
of service-oriented systems.

**Research Question RQ2:** In order to support the full development life-cycle of an SOSPL,
in Chapter 3, we first presented a holistic comparison of SPL and SOA by focusing on reuse,
architectural, and variability aspects of the two paradigms. To compare SPL and SOA,
we consider four main aspects including development processes, reusability notions, archi-
tectural styles, and variability modeling and management. Subsequently, we proposed a
method and outlined activities to guide the development of configurable process models
by adopting and extending a traditional SPLE life-cycle. Our approach is a twofold ex-
tension of the existing SPL development process. First, it extends the domain engineering
life-cycle to support both specificities of SOSPLs reflecting their service-oriented nature
and supporting non-functional (quality) requirements as needed in the course of configura-
tion of SOSPL. Second, our approach in application engineering provides decision-making
support for quality-aware SOSPL configuration on business process layer. Accordingly, we
proposed a top-down method following a two-life-cycle approach that separates two core
activities related to service-domain engineering and service-application engineering. The
service-domain engineering focuses on analysis and design of configurable process model
realizing SOSPL by analyzing the requirements and scoping the product line as a whole and
producing any common, reusable business processes and services which are integrated in a
configurable reference process model. The service-application engineering focuses on con-
figuration and deriving stakeholder-tailored services by taking into account stakeholders’
preferences. The proposed method follows a top-down approach for constructing configurable business process models; however, it can be applied bottom-up where a variability of business process model can be captured and expressed by orthogonal variability modeling.

**Research Question RQ3:** In order to support the full services life-cycle in an SOSPL, we proposed an extensible QoS model (Chapter 4) to coherently integrate quality information in one common model at various levels of abstraction (requirements specification, business process, and service layers). We also provided the foundation for SOSPL-quality evaluation which enables to predict and assess the QoS-range supported by configurable process models being developed.

The modeling aspect is specifically addressed in the business process layer by providing expressive and extensible submodels to define QoS and specify relations, dependencies, and preferences about quality properties from the different decision-makers perspectives. The QoS model supports the definitions of QoS-range and the relative importance of QoS from both system and stakeholders’ views which are a prerequisite and used in the decision-making and reasoning over QoS properties and requirements as well as for the quality-driven process configuration and optimization.

A systematic method proposed for the QoS aggregation and computation provides an effective solution to address the evaluation of QoS-range values in the presence of variability (functional and quality variants). We identified and introduced a set of possible variability patterns that occur in the structure of business process models, used to define execution logic of composite services. We provided the formalization of a computational model for quality evaluation, which takes both variability and composition patterns into account and allows for a trade-off analysis and architectural decision-making among options that provide similar functional properties but different quality levels. Our evaluation demonstrates acceptable performance and scalability of the approach for very large process models. This work can be considered to support variability of enterprise services, and facilitate the quality management and configuration of multi-tenant cloud applications.

**Research Question RQ4:** We proposed an efficient feature-based configuration framework to support decision-making and automating configuration of a configurable business process model to devise service products in an SOSPL with respect to quality requirements.
and the constraints boundaries specified by the system and stakeholders (Chapter 5). We focused on high-level configuration of business process models by considering both functional and quality requirements at the level of design and implementation provided by services offering the same functionality and varying in QoS properties. We defined the QoS-aware configuration problem as the selection of the most desirable features, which capture aspects of functional value for stakeholders, and further the selection of appropriate service implementations with the right level of QoS that maximize the overall quality level with the consideration of stakeholders’ preferences and constraints defined in the system. To this end, we propose a Multiple Criteria Decision-Making (MCDM) approach enabling to achieve the optimal selection of appropriate features and best service candidates where functional and non-functional requirements are satisfied within the constraints boundaries specified by stakeholders. We modeled the configuration problem as a constraints optimization problem and an integer programming problem formulated as MILP solved by well-developed mathematical programming techniques with off-the-shelf solvers. Our approach enables to aggregate multiple objective functions defining stakeholders’ preferences and system utility into one overall function which can be maximized or minimized further. We provided an algorithm which transforms feature model including variability relations and configuration knowledge of configurable process model, stakeholders’ preferences, and specified local and global constraints into MILP model.

Preference-based product configuration in SPLE is still rather immature and requires further research. Different approaches have been devised to address the complexity of variant configuration in SPL. The majority of related work, however, focuses on how a configuration problem can be formalized as a constraint satisfaction problem rather than an optimization problem and do not address how to perform application configuration with stakeholders’ preferences considered and optimal (or near-optimal) solutions sought.

7.3 Outlook and Future Research

There are several research directions and possible extensions to this body of research. As we showed and discussed, the combination or integration of software product line and service-oriented software engineering is an emerging research field, and there is room for extensive research and development in areas such as methodology, variability management, and service configuration, to name a few.
Configuration and customization validation: Configuration and customization can be performed at the different layers of SOA ranging from business process to service-component layers. Hence, validation mechanisms are required for the consistency checking to guarantee the correctness and completeness of the configuration of final customized service-application. Verification allows correctness to be checked at each intermediate step of the configuration or customization procedure and ensures that the final configured process model being deployed in the run-time environment can be properly executed. For instance, corresponding process instances guarantee compliance requirements and soundness (i.e., always complete in a well-defined and proper state); they conform to the grammar rules as defined by the process modeling language used, and the semantics between control- and data-flow are consistent.

In this thesis, we did not address the configuration validation as it was beyond the scope of this work. We have already started developing a formal framework to support configuration and customization verification by considering goal models, which enables traceability on the continuum between high-level strategic concerns and low-level technical ones in the course of configuration or customization [86, 87].

One direction for future research is investigating how to preserve the soundness of customized processes [4, 7, 130, 261]. There is notable literature on verification approaches and process correctness topics guaranteeing structural and behavioural soundness of a single model. Nevertheless, characterizing the correctness of a family of processes (a configurable process model) is a more complex task. For example, individual stakeholders may require changes to a configurable process model which has already been designed and developed, resulting in unknown process structures or imposing restrictions. As a consequence, the soundness and semantics of a customized process cannot be guaranteed. It is important, highly beneficial, and convenient to check the soundness during configuration and customization; hence, validation mechanisms are imperative within the entire framework.

Quality management and probabilistic evaluation: We proposed a quality evaluation framework for SOSPL; however, there are several aspects of this work that could be improved and expanded, and further investigation can help make the proposed framework more practical. There is a need to study the implications when non-hierarchical dependency relationships exist among services in a configurable process model in terms of integrity constraints and complex relations. We identified that the integrity constraints among services
may not be completely independent; the service interaction, selection or presence of a particular service in the final configured service product may therefore impose an intricate chain of constraints which influence a range of quality characteristics for the entire business process. The consequence is that measuring and determining accurate values of functional impacts on a quality characteristic is usually not possible. Some effort need to be dedicated to studying feature interaction and dependency analysis in software product lines research in order to address this issue. For instance, heuristic-based approaches can be utilized to detect service interactions.  

In this work, even though we assumed that the QoS information for service components forming a composite service in a process model is static and pre-existing, QoS of each service may change dynamically because services are operated in heterogeneous environments, which impact the overall quality evaluation of an SOSPL in scenarios where real-time QoS information is crucial. Another direction in which our work can be extended is to leverage probabilistic modelling approaches for the QoS management, probabilistic analysis and assessment of quality properties which are expressed in a probabilistic form to reason on QoS information under uncertainty. Probabilistic modelling is widely used in the field of performance evaluation and can be utilized to address this issue; namely, Bayesian Network, Discrete-Time Markov Chains (DTMCs), Markov decision processes (MDPs), and Continuous Time Markov Chains (CTMCs).

**Design and run-time variability management:** In this thesis, we only focused on functional and quality variability management for the business process (service composition) layer at design time by considering services as core assets and variability in control-flow. Variability management should also consider other perspectives of the process models; noteworthy examples are variability in data objects, resources, and operational facets addressing the technical aspects of deployment and execution, which remain an open avenue for future research. As discussed in Chapter 1 in layered SOA architecture, variability can exist at the business process, service, or service component layers. As a result, explicit variability management should be accomplished at the different layers to fully support the development life-cycle and customization of service-oriented applications for SOSPLs and to enhance the reusability and adaptability of SOA-based solutions.

In SOSPL, service-oriented applications not only need to be configurable before deployment by creating predefined service-product variants but are also required to be adaptable
at run-time (e.g., dynamic service binding and replacement). In consequence, variability transformation at run-time and management are becoming key requirements to adapt their configuration to a changing context and environment. Dynamic software product lines (DSPL) built upon services and SOA is becoming another active research direction which aims at investigating mechanisms to address run-time variability and dynamic reconfiguration (cf. Chapter 2).

**Interactive configuration and recommendation supports:** The proposed configuration framework provides staged configuration by considering the stakeholder’s preferences and allows the stakeholder to make decision about desired features related to the final service product. We advocate that fully-automated approaches to configuration and customization are likely inefficacious because different stakeholders may be involved in the procedure, or stakeholder’s preference can be altered in the course of configuration \[[256, 265]\]. The approach we presented supports automatic feature configuration which can encode partial configurations into constraint optimization problems and compute the consequences on stakeholder choices. We enable decision makers (system or stakeholders) to add new constraints to the model under configuration. We proposed preference-based configuration and supporting stakeholders to find the most suitable set of options for their service application by firstly prioritizing (ranking) all alternatives based on their preferences and objectives. We also utilized a method to determine the rank of stakeholders’ preference regarding the prioritization of QoS criteria and optional features based on Analytical Hierarchy Process (AHP), a widely-used technique to model subjective decision-making. However, the crisp pair-wise comparison in the conventional AHP seems too insufficient and imprecise in accurately capturing the judgments of the decision-maker(s) because of vagueness and uncertainty of the decision-maker(s) judgments, so we plan to incorporate AHP along with fuzzy approach and investigate alternative methods to address this deficiency. Moreover, the stakeholder may encounter difficulties such as inconsistency because of the conflicts with previous decisions during the course of configuration. Our approach still requires performing consistency checking among stakeholder’s requirements, which remains for future work.

Interactive configuration and guidance can be useful to recommend particular features and also to make decision-makers aware of important dependencies. Furthermore, it is significantly useful to provide stakeholders with recommendations about which choice might be the best, under particular conditions. We envision that incorporating machine-learning
techniques is an interesting future research direction in order to develop interactive recommender systems for service configuration and customization.

Multi-tenant cloud applications: Today, many cloud applications serve large numbers of customers whose versatile demands on resources and application performance have to be met effectively and opportune. The widespread adoption of multi-tenant cloud systems is like a gate to a world of possibilities to utilize configurable process models as artifacts to derive the configuration and customization of such systems. Currently, the configuration of multi-tenant SaaS applications is mostly performed manually and resource-intensive because of the large number of configuration options offered by such applications. Notwithstanding the proposal of initial visions for multi-tenant system configuration based on configurable process models [104], the realization of these visions remains for future research. This work can be considered in the facilitation of the management and configuration of multi-tenant cloud applications [1, 2, 217].
Appendix A

Mapping Study

Table A.1: List of related search terms.

<table>
<thead>
<tr>
<th>Search Query (SQ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQ1: “Service-Oriented Product Line” OR “Service-Oriented Software Product Line” OR [SQ2 AND SQ3]</td>
</tr>
<tr>
<td>SQ2: (“software product line” OR SPL) OR (“product line” OR “product family”) OR “process family” OR “product line engineering” OR “domain engineering” OR “application engineering” OR “variability” OR “variability modeling” OR “variability management” OR “variability analysis” OR “feature modeling” OR “feature analysis”</td>
</tr>
<tr>
<td>SQ3: (“service-based OR service-oriented” OR “service orientation”) OR (“service-oriented architecture” OR ‘SOA”) OR “service computing” OR “service engineering” OR “service development” OR “web services” OR (“software reuse services” OR “service reuse”) OR “service variability” OR “service customization” OR “service identification” OR (“service orchestration” OR “service composition”)</td>
</tr>
</tbody>
</table>
Figure A.1: Distribution of workshops series

Figure A.2: Distribution of publication fora

\[1\] The book sections and technical reports are succinctly named “others”.
## APPENDIX A. MAPPING STUDY

Table A.2: List of journals.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Journal title</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDBI</td>
<td>CBDI Journal</td>
</tr>
<tr>
<td>IEEE Softw.</td>
<td>IEEE Software</td>
</tr>
<tr>
<td>IJBPM</td>
<td>Journal of Business Process Integration and Management</td>
</tr>
<tr>
<td>IJCS</td>
<td>Journal of Cooperative Information Systems</td>
</tr>
<tr>
<td>IIIDS</td>
<td>Journal of Intelligent Information and Database Systems</td>
</tr>
<tr>
<td>IJSEKE</td>
<td>Journal of Software Engineering and Knowledge Engineering</td>
</tr>
<tr>
<td>IJSSSE</td>
<td>Journal of Systems Science and Systems Engineering</td>
</tr>
<tr>
<td>IJWIS</td>
<td>Journal of Web Information Systems</td>
</tr>
<tr>
<td>JDM</td>
<td>Journal of Database Management</td>
</tr>
<tr>
<td>JRPIT</td>
<td>Journal of Research and Practice in Information Technology</td>
</tr>
<tr>
<td>JSS</td>
<td>Journal of Systems and Software</td>
</tr>
<tr>
<td>JUICS</td>
<td>Journal of Universal Computer Science</td>
</tr>
</tbody>
</table>

Table A.3: List of workshops.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Workshops title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACoTA</td>
<td>International Workshop on Automated Configuration and Tailoring of Applications</td>
</tr>
<tr>
<td>BPMW</td>
<td>Business Process Management Workshops</td>
</tr>
<tr>
<td>DSPL</td>
<td>International Workshop on Dynamic Software Product Lines</td>
</tr>
<tr>
<td>FOSD</td>
<td>International Workshop on Feature-Oriented Software Development</td>
</tr>
<tr>
<td>IWPSE</td>
<td>International Workshop on Principles of Software Evolution</td>
</tr>
<tr>
<td>MAPLE</td>
<td>International Workshop on Model-Driven Product Line Engineering</td>
</tr>
<tr>
<td>MRT</td>
<td>International Workshop <a href="mailto:Models@run.time">Models@run.time</a> at Models</td>
</tr>
<tr>
<td>PESOS</td>
<td>International Workshop on Principles of Engineering Service Oriented Systems</td>
</tr>
<tr>
<td>RAM-SE</td>
<td>International Workshop on Reflection, AOP and Meta-Data for Software Evolution</td>
</tr>
<tr>
<td>SCArVeS</td>
<td>International Workshop on Services, Clouds, and Alternative Design Strategies for Variant-Rich Software Systems</td>
</tr>
<tr>
<td>SDOA</td>
<td>International Workshop on Systems development in SOA environments</td>
</tr>
<tr>
<td>SOAPL</td>
<td>International Workshop on Service-Oriented Architectures and Software Product Lines</td>
</tr>
<tr>
<td>SOCCER</td>
<td>International Workshop on Service-Oriented Computing: Consequences for Engineering Requirements</td>
</tr>
<tr>
<td>SWSESE</td>
<td>International Workshop on Semantic Web enabled Software Engineering</td>
</tr>
<tr>
<td>TSOA</td>
<td>International Workshop on Telecom Service Oriented Architectures</td>
</tr>
<tr>
<td>VaMOS</td>
<td>International Workshop on Variability Modelling of Software-Intensive Systems</td>
</tr>
<tr>
<td>WESOA</td>
<td>International Workshop on Engineering Service-Oriented Applications</td>
</tr>
<tr>
<td>WWV</td>
<td>International Workshop on Automated Specification and Verification of Web Systems</td>
</tr>
</tbody>
</table>
Table A.4: List of conferences and symposiums.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Conferences title</th>
</tr>
</thead>
<tbody>
<tr>
<td>AICCSA</td>
<td>ACS/IEEE International Conference on Computer Systems and Applications</td>
</tr>
<tr>
<td>APSCC</td>
<td>Asia-Pacific Services Computing Conference</td>
</tr>
<tr>
<td>APEC</td>
<td>Asia-Pacific Software Engineering Conference</td>
</tr>
<tr>
<td>BIS</td>
<td>International Conference on Business Information Systems</td>
</tr>
<tr>
<td>CAiSE</td>
<td>International Conference Advanced Information Systems Engineering</td>
</tr>
<tr>
<td>CBSE</td>
<td>International Conference on Component-Based Software Engineering</td>
</tr>
<tr>
<td>CEC</td>
<td>International Conference on E-Commerce Technology</td>
</tr>
<tr>
<td>CloudCom</td>
<td>International Conference on Cloud Computing</td>
</tr>
<tr>
<td>COMPSAC</td>
<td>International Computer Software and Applications Conference</td>
</tr>
<tr>
<td>ECBS</td>
<td>International Conference on Engineering of Computer Based Systems</td>
</tr>
<tr>
<td>ECWS</td>
<td>European Conference on Web Services</td>
</tr>
<tr>
<td>ECSA</td>
<td>European Conference on Software Architecture</td>
</tr>
<tr>
<td>FSE</td>
<td>International Symposium on Foundations of software engineering</td>
</tr>
<tr>
<td>GCC</td>
<td>International Conference on Grid and Cooperative Computing</td>
</tr>
<tr>
<td>HICSS</td>
<td>Annual Hawaii International Conference on System Sciences</td>
</tr>
<tr>
<td>ICACT</td>
<td>International Conference on Advanced Communication Technology</td>
</tr>
<tr>
<td>ICCIT</td>
<td>International Conference on Computer and Information Technology</td>
</tr>
<tr>
<td>ICEBE</td>
<td>International Conference on E-Business Engineering</td>
</tr>
<tr>
<td>ICHIT</td>
<td>International Conference on Convergence and Hybrid Information Technology</td>
</tr>
<tr>
<td>ICIS</td>
<td>International Conference on Computer and Information Science</td>
</tr>
<tr>
<td>ICSOC</td>
<td>International Conference on Service-Oriented Computing</td>
</tr>
<tr>
<td>ICISR</td>
<td>International Conference on Software Reuse</td>
</tr>
<tr>
<td>ICWS</td>
<td>International Conference on Web Services</td>
</tr>
<tr>
<td>ISCID</td>
<td>International Symposium on Computational Intelligence and Design</td>
</tr>
<tr>
<td>ITNG</td>
<td>International Conference on Information Technology</td>
</tr>
<tr>
<td>PROFES</td>
<td>International Conference on Product Focused Software Process Improvement</td>
</tr>
<tr>
<td>SAC</td>
<td>ACM symposium on Applied Computing</td>
</tr>
<tr>
<td>SBCARS</td>
<td>Brazilian Symposium on Software Components, Architectures and Reuse</td>
</tr>
<tr>
<td>SC</td>
<td>International Conference on Software Composition</td>
</tr>
<tr>
<td>SCC</td>
<td>International Conference on Services Computing</td>
</tr>
<tr>
<td>SEKE</td>
<td>International Conference on Software Engineering &amp; Knowledge Engineering</td>
</tr>
<tr>
<td>SERVICES</td>
<td>World Congress on Services</td>
</tr>
<tr>
<td>SOCA</td>
<td>International Conference on Service-Oriented Computing and Applications</td>
</tr>
<tr>
<td>SPLC</td>
<td>International Software Product Line Conference</td>
</tr>
<tr>
<td>TEAA</td>
<td>International Conference on Trends in enterprise application architecture</td>
</tr>
<tr>
<td>WWW</td>
<td>International Conference on World Wide Web</td>
</tr>
</tbody>
</table>
Bibliography


BIBLIOGRAPHY


250


Mapping Study References


MAPPING STUDY REFERENCES


