



**HAL**  
open science

# Automated support of the variability in configurable process models

Nour Assy

► **To cite this version:**

Nour Assy. Automated support of the variability in configurable process models. Software Engineering [cs.SE]. Université Paris Saclay (COmUE), 2015. English. NNT : 2015SACLL001 . tel-01256612

**HAL Id: tel-01256612**

**<https://theses.hal.science/tel-01256612v1>**

Submitted on 15 Jan 2016

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

NNT : 2015SACLL001

THESE DE DOCTORAT  
DE L'UNIVERSITE PARIS-SACLAY,  
préparée à TELECOM SudParis

ÉCOLE DOCTORALE N°580  
Sciences et technologies de l'information et de la communication

Spécialité de doctorat : Informatique

Par

**Mme Nour Assy**

AUTOMATED SUPPORT FOR CONFIGURABLE PROCESS MODELS

**Thèse présentée et soutenue à Evry, le 28 Septembre 2015 :**

**Composition du Jury :**

|                        |  |                    |
|------------------------|--|--------------------|
| M. Charoy François     | Professeur, Université de Lorraine                           | Président          |
| M. Dumas Marlon        | Professeur, Université de Tartu                              | Rapporteur         |
| M. Seinturier Lionel   | Professeur, Université de Lille 1                            | Rapporteur         |
| M. Mendling Jan        | Professeur, Université d'économie et<br>de gestion de Vienne | Examineur          |
| Mme I. Grida Ben Yahia | Ingénieur en recherche et<br>développement, Orange labs      | Examinatrice       |
| M. Defude Bruno        | Professeur, Telecom SudParis                                 | Directeur de thèse |
| M. Gaaloul Walid       | Professeur, Telecom SudParis                                 | Directeur de thèse |



**Titre :** Automatiser le support de la variabilité dans les modèles de processus configurables

**Mots clés :** Modèles de processus configurables, support automatisée, fragments de processus configurables, configuration guidée, fouille de fragments configurables

**Résumé :** Avec l'évolution rapide des exigences dans les environnements d'entreprises d'aujourd'hui, la modélisation des processus métiers à partir de zéro devient une tâche fastidieuse. Motivé par le paradigme « Conception par Réutilisation », les modèles de processus configurables gagnent récemment de l'élan grâce à leur capacité de représenter explicitement les parties communes et variables de processus similaires en un seul modèle personnalisable. Un modèle de processus configurable doit être configuré en fonction des exigences spécifiques d'une organisation afin de dériver une variante de processus.

Puisque les modèles de processus configurables ont tendance à être larges et complexes, leur conception et configuration sans aucune assistance deviennent des tâches fastidieuses.

Dans cette thèse, nous proposons une approche automatisée d'aide à la conception et à la configuration des modèles de processus configurables. Nous ciblons assister les utilisateurs (i) à concevoir leurs modèles de processus configurables d'une manière fine afin d'éviter des résultats larges et complexes et (ii) à configurer des modèles existants selon leurs besoins spécifiques. Pour ce faire, nous proposons d'apprendre de l'expérience acquise grâce à la modélisation et à la configuration précédentes des processus métiers afin de (i) recommander des fragments de processus configurables qui peuvent être intégrés dans un modèle en cours de modélisation et (ii) recommander des choix de configuration afin de personnaliser un processus configurable existant.

**Title:** Automated support for configurable process models

**Keywords:** Configurable process models, automated support, configurable process fragments, configuration guidance model, configurable process discovery.

**Abstract:** With the rapidly changing demands in today's business environments, modeling business processes from scratch becomes a time-consuming and error prone task. Motivated by the "Design by Reuse" paradigm, configurable process models are recently gaining momentum due to their capability of explicitly representing the common and variable parts of similar processes into one customizable model. A configurable process model needs to be configured to suit the specific requirements of an organization.

Since configurable process models tend to be large and complex, their design and configuration without any assistance become tedious tasks.

In this thesis, we propose an automated approach to assist the design and configuration of configurable process models. Our aim is to assist users (i) to complete the design of their configurable process models in a fine-grained way in order to avoid large and complex results and (ii) to configure existing models according to their specific needs. To do so, we propose to learn from the experience gained through previous process modeling and configuration in order to (i) recommend configurable fragments that can be integrated into an ongoing designed process and (ii) recommend configuration choices to customize an existing configurable process.







*to my family*



# Acknowledgment



I thank God for making all things possible for me and giving me strength to go. A thesis journey is not always easy, but with the supervision, support and encouragement of many people, I have never regretted that I started it.

I would like to thank all members of the jury. I thank Professor Lionel Seinturier and Professor Marlon Dumas for accepting being my thesis reviewers and for their attention and thoughtful comments. I also thank Professor François Charoy, Professor Jan Mendling and Dr. Imen Grida Ben Yahia for accepting being my thesis examiners.

I would like to express my appreciation and gratitude to my supervisor Walid Gaaloul. His valuable advice, enthusiasm and constant support during this thesis allowed me to acquire new understandings and extend my experiences. He was not only an advisor but also a good listener and a friend who showed me just what I was able to achieve even when I did not see it myself. Thank you for taking me on this journey and for your guidance, it has been a true pleasure and I hope that we can continue our collaboration.

I am also grateful to my supervisor Bruno Defude for allowing me to join his team. His wide knowledge and logical way of thinking have been of great value for me.

I owe my deepest gratitude and warmest affection to the members of the computer science department of Telecom SudParis. I would like to thank Brigitte Houassine for her kind help and assistance. A special thank you to my office mates Emna, Rami, Mourad, Chan, Olfa and our new companion Souha for the lovely moments we spent.

I am forever thankful to my family: my father, my mother, my sister and my brothers who were always there for me with encouraging words whenever I started doubting myself. Your encouragement made me go forward and made me want to succeed. I express my deepest gratitude to my loving husband Abdallah. His love, encouragement, understanding and support helped me to get through many difficult times. Thank you so much for having faith in me! I cannot forget our new family member, my beautiful niece Batoul. You supported me without even you know it. I love you so much. I dedicate this thesis to all of you, my wonderful family.





# Abstract

Nowadays, companies are increasingly adopting Process-Aware Information Systems for managing and executing their processes on the basis of process models referred to as business process models. With the rapidly changing demands in today's business environments, modeling business processes from scratch becomes a time-consuming and error prone task. Motivated by the "Design by Reuse" paradigm, configurable process models are recently gaining momentum due to their capability of explicitly representing the common and variable parts of similar processes into one customizable model. A configurable process model needs to be configured to suit the specific requirements of an organization. In this way, new process variants are derived with minimal design efforts.

The *design* and *configuration* of configurable process models is involving more and more many researches in both academics and industry. On the one hand, the manual design of configurable process models is undoubtedly a labor-intensive task. Although automated approaches have been proposed in the literature, they all targeted to *construct an entire configurable process model at once*. This led to *large and complex* processes which are *difficult to reuse*. On the other hand, with an increasing number of configurable elements in the process model and many interdependencies between their configuration choices, the users need means of support to configure the process. Many approaches have been proposed in the literature to build configuration support systems that assist users selecting desirable configuration choices according to their needs. However, *these systems are currently manually created by domain experts which is certainly cumbersome, time-consuming and error-prone*.

In this thesis, we address the above shortcomings by proposing an automated approach for supporting the design and configuration of configurable process models. We target to assist business analysts (i) designing their configurable process models in a *fine-grained* way to avoid complex and large results and (ii) creating their configuration support systems *with a minimal manual effort*. To do so, we realize that previously designed and configured process models contain implicit and useful knowledge for process design and configuration. Therefore, we propose to learn from this past experience in order to automatically (i) *derive configurable process fragments* that are close to business analysts interests and (ii) *extracting configuration guidance models* that guide business analysts in the creation of their configuration support systems. To validate our approach, we (i) develop three proof of concepts as extensions of existing business process modeling tools, (ii) perform experiments on real process models from two large datasets and (iii) conduct a case study with professional and academics. Experimental results show that our approach is feasible, accurate and has good performance in real use-cases.



# Table of contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>  | <b>19</b> |
| 1.1      | Research context . . . . .   | 19        |
| 1.2      | Research problem: <i>How to propose automated support for configurable process models?</i> . . . . . | 23        |
| 1.2.1    | On assisting configurable process design . . . . .   | 24        |
| 1.2.2    | On supporting business process configuration . . . . .   | 24        |
| 1.3      | Motivating example . . . . .   | 25        |
| 1.4      | Thesis principles, objectives and contributions . . . . .  | 31        |
| 1.4.1    | Thesis principles . . . . .  | 31        |
| 1.4.2    | Thesis objectives . . . . .  | 31        |
| 1.4.3    | Thesis contributions . . . . .   | 32        |
| 1.5      | Thesis outline . . . . .   | 34        |
| <br>     |  |           |
| <b>2</b> | <b>Related Work</b>  | <b>37</b> |
| 2.1      | Introduction . . . . .   | 37        |
| 2.2      | On facilitating configurable process design . . . . .  | 38        |
| 2.2.1    | Configurable process modeling . . . . .  | 38        |
| 2.2.2    | Business Process variant retrieval . . . . .   | 42        |
| 2.2.3    | Business process merging . . . . .   | 45        |
| 2.2.4    | Business process mining . . . . .  | 47        |
| 2.2.5    | Synthesis . . . . .  | 48        |
| 2.3      | On supporting business process configuration . . . . .   | 50        |
| 2.3.1    | Domain-based approaches . . . . .  | 50        |
| 2.3.2    | Process-based approaches . . . . .   | 53        |
| 2.3.3    | Synthesis . . . . .  | 55        |
| 2.4      | Conclusion . . . . .   | 56        |
| <br>     |  |           |
| <b>3</b> | <b>Preliminaries</b>   | <b>57</b> |
| 3.1      | Basic Notations . . . . .  | 57        |
| 3.2      | Process Modeling Standards . . . . .   | 58        |
| 3.2.1    | Business Process Model and Notation (BPMN) . . . . .   | 59        |
| 3.2.2    | Configurable BPMN (C-BPMN) . . . . .   | 60        |
| 3.2.3    | Petri Nets . . . . .   | 62        |
| 3.3      | Process Graphs . . . . .   | 64        |
| 3.4      | Event Logs . . . . .   | 66        |

|          |  |            |
|----------|--|------------|
| <b>4</b> | <b>Assisting Process Design with Configurable Process Fragments</b>        | <b>69</b>  |
| 4.1      | Introduction . . . . .   | 69         |
| 4.2      | Aggregated Neighborhood Context Graph . . . . .                            | 70         |
| 4.3      | Deriving Aggregated Neighborhood Context Graphs . . . . .                  | 75         |
| 4.3.1    | Extracting Neighborhood Context Graphs . . . . .                           | 76         |
| 4.3.2    | Clustering Neighborhood Context Graphs . . . . .                           | 78         |
| 4.3.3    | Merging Neighborhood Context Graphs . . . . .                              | 80         |
| 4.3.3.1  | Merging vertices/edges . . . . .   | 81         |
| 4.3.3.2  | Defining edges' labels . . . . .   | 83         |
| 4.3.3.3  | Handling edges sharing their source or target . . . . .                    | 89         |
| 4.3.4    | Behavior Preservation of the Merging Algorithm . . . . .                   | 89         |
| 4.3.5    | Computational Complexity . . . . .   | 90         |
| 4.4      | From Aggregated Neighborhood Context Graph to C-BPMN . . . . .             | 91         |
| 4.5      | Conclusion . . . . .   | 92         |
| <b>5</b> | <b>Supporting Process Configuration with Configuration Guidance Models</b> | <b>95</b>  |
| 5.1      | Introduction . . . . .   | 95         |
| 5.2      | Motivating Example . . . . .   | 96         |
| 5.3      | Configuration Guidance Model . . . . .                                     | 99         |
| 5.4      | Approach Overview . . . . .  | 102        |
| 5.5      | Extracting configuration guidelines from existing process models . . . . . | 103        |
| 5.5.1    | Retrieving process elements' configurations . . . . .                      | 103        |
| 5.5.2    | Apriori-based approach for deriving configuration guidelines . . . . .     | 106        |
| 5.6      | Inferring Configuration Steps order . . . . .                              | 108        |
| 5.7      | Formalizing Configuration Guidelines Dependencies Relations . . . . .      | 110        |
| 5.7.1    | Deriving a transition system from configuration guidelines . . . . .       | 110        |
| 5.7.2    | Deriving a Petri-Net using Theory of Regions . . . . .                     | 113        |
| 5.8      | Conclusion . . . . .   | 114        |
| <b>6</b> | <b>Using event Logs for Configurable Process Design and Configuration</b>  | <b>117</b> |
| 6.1      | Introduction . . . . .   | 117        |
| 6.2      | Deriving Configurable Process Fragments from Event Logs . . . . .          | 120        |
| 6.2.1    | Extracting Log-based Neighborhood Contexts . . . . .                       | 121        |
| 6.2.2    | Mining Configurable Process Fragments . . . . .                            | 123        |
| 6.3      | Mining Ranked Configuration Guidelines . . . . .                           | 125        |
| 6.3.1    | A frequency suffix tree for configuration executions . . . . .             | 125        |
| 6.3.2    | Deriving ranked configuration guidelines . . . . .                         | 128        |
| 6.4      | Conclusion . . . . .   | 131        |

|          |  |            |
|----------|--|------------|
| <b>7</b> | <b>Evaluation and Validation</b>                               | <b>133</b> |
| 7.1      | Introduction . . . . .   | 134        |
| 7.2      | Proof of Concept . . . . .                                     | 135        |
| 7.2.1    | Signavio Extension . . . . .                                   | 135        |
| 7.2.2    | ProM Plug-in . . . . .   | 137        |
| 7.3      | Experimentation . . . . .                                      | 139        |
| 7.3.1    | Configurable Process Design Experiments . . . . .              | 140        |
| 7.3.1.1  | Approach feasibility and parameter impact . . . . .            | 141        |
| 7.3.1.2  | Results quality . . . . .                                      | 143        |
| 7.3.1.3  | Algorithm performance . . . . .                                | 145        |
| 7.3.1.4  | Synthesis . . . . .  | 146        |
| 7.3.2    | Process Configuration Experiments . . . . .                    | 147        |
| 7.3.2.1  | Results quality and parameter impact . . . . .                 | 148        |
| 7.3.2.2  | Approach accuracy and parameter impact . . . . .               | 150        |
| 7.3.2.3  | Synthesis . . . . .  | 151        |
| 7.3.3    | Log-based Experiments . . . . .                                | 152        |
| 7.3.3.1  | Configurable fragments quality . . . . .                       | 153        |
| 7.3.3.2  | Configuration guidelines efficiency . . . . .                  | 155        |
| 7.3.3.3  | Synthesis . . . . .  | 156        |
| 7.4      | Case Study . . . . .   | 156        |
| 7.4.1    | Case Study Objective . . . . .                                 | 157        |
| 7.4.2    | Design, Data Collection and Execution . . . . .                | 157        |
| 7.4.3    | Results Analysis and Findings . . . . .                        | 158        |
| 7.4.4    | Threats to Validity . . . . .                                  | 160        |
| 7.5      | Conclusion . . . . .   | 160        |
| <b>8</b> | <b>Conclusion and Future Works</b>                             | <b>163</b> |
| 8.1      | Contributions . . . . .  | 163        |
| 8.2      | Future work . . . . .  | 166        |
| 8.2.1    | Improving automated support quality . . . . .                  | 166        |
| 8.2.2    | Business process configuration in the cloud . . . . .          | 167        |
|          | <b>Appendices</b>  | <b>169</b> |
| <b>A</b> | <b>Proof for <math>TS_G</math> is a directed acyclic graph</b> | <b>171</b> |
| <b>B</b> | <b>List of Publications</b>                                    | <b>173</b> |
| <b>C</b> | <b>Proof of Concepts</b>                                       | <b>175</b> |
| <b>D</b> | <b>Résumé</b>  | <b>177</b> |



# List of Tables

|     |   |     |
|-----|---|-----|
| 2.1 | Synthesis on the merging and mining approaches for assisting the configurable process design according to our principles . . . . .  | 49  |
| 2.2 | Synthesis on process configuration support approaches that satisfy our principles . . . . .   | 55  |
| 3.1 | Configuration constraints of configurable gateways . . . . .  | 61  |
| 3.2 | Example of event log for a process that handle fines [1] . . . . .  | 67  |
| 5.1 | An excerpt of the configuration guidelines for the configurable process model in Figure 5.1 . . . . .   | 98  |
| 5.2 | An excerpt of a configuration matrix . . . . .  | 107 |
| 6.1 | $L_1$ : Event log of the process variant in Figure 6.2a . . . . .   | 119 |
| 6.2 | $L_2$ : Event log of the process variant in Figure 6.2b . . . . .   | 119 |
| 6.3 | $L_{a41}^2$ : Log-based event log extracted from $L_1$ in Table 6.1 . . . . .   | 123 |
| 6.4 | $L_{a42}^2$ : Log-based event log extracted from $L_2$ in Table 6.2 . . . . .   | 123 |
| 6.5 | the configuration log $L_{conf}$ derived from $L_{merged}$ . . . . .  | 127 |
| 7.1 | Statistics of the dataset [2] . . . . .   | 141 |
| 7.2 | The overall minimum, maximum and average number of recommended fragments and their compression with different $k^{th}$ -layer values/ with and without clustering . . . . . | 142 |
| 7.3 | Comparison of the compression factor values with existing works . . . . .   | 143 |
| 7.4 | Structural complexity metrics for the configurable fragments and the average of their corresponding merged fragments . . . . .  | 144 |
| 7.5 | The execution time of our proposed algorithm compared with the existing works . . . . .   | 146 |
| 7.6 | Statistics of the clusters and the configurable process models . . . . .  | 148 |
| 7.7 | Number of guidelines for different minimum support threshold values and a minimum confidence threshold $C = 0.8$ . . . . .  | 149 |
| 7.8 | The average number of possible configurations . . . . .   | 155 |
| 7.9 | The average time in man-hour unit spent to build a configuration support system with and without the assistance of configuration guidance models . . . . .                  | 159 |





# List of Figures

|      |  |     |
|------|--|-----|
| 1.1  | Configuration and individualization of a configurable process model . . .  | 21  |
| 1.2  | Configurable process models in the BPM lifecycle (inspired from [3]) . . .   | 22  |
| 1.3  | Research problem . . . . .   | 23  |
| 1.4  | Three process variants of a travel booking process . . . . .   | 26  |
| 1.5  | $BP_R$ : An ongoing design of a travel booking process . . . . .   | 28  |
| 1.6  | A configurable fragment that contains the activity “Select a flight” . . .   | 28  |
| 1.7  | A configurable travel booking process . . . . .  | 30  |
|      |  |     |
| 2.1  | A configurable process model in C-EPC notation (before configuration, after configuration, and resulting EPC [4] . . . . .           | 39  |
| 2.2  | The Provop process variant lifecycle [5] . . . . .   | 41  |
| 2.3  | A feature diagram, a business process model and a mapping between them [6] . . . . .   | 51  |
| 2.4  | An overview of the questionnaire-based approach [7] . . . . .  | 52  |
|      |  |     |
| 3.1  | A configurable activity and its possible configuration choices . . . . .   | 61  |
| 3.2  | Configurable events and their possible configuration choices . . . . .   | 62  |
| 3.3  | An example of a Petri net . . . . .  | 62  |
| 3.4  | An example of a process graph . . . . .  | 65  |
| 3.5  | The process discovered from the event log in Table 3.2 [1] . . . . .   | 68  |
|      |  |     |
| 4.1  | $BP^c$ : An incomplete configurable reference travel booking process . . .   | 70  |
| 4.2  | Two process variants of a travel booking process . . . . .   | 71  |
| 4.3  | Two neighborhood context graphs with 2 layers . . . . .  | 74  |
| 4.4  | $G_{12}^a$ : The aggregated neighborhood context graph resulted from merging the neighborhood context graphs in Figure 4.3 . . . . . | 75  |
| 4.5  | A dendrogram illustrating the result of the AHC algorithm . . . . .  | 80  |
| 4.6  | The aggregated neighborhood context graph after merging the vertices and edges . . . . .   | 83  |
| 4.7  | $\mathbb{F}_1^A$ : One possible alignment of $F_1$ and $F_2$ . . . . .   | 85  |
| 4.8  | $\mathbb{F}_2^A$ : One possible alignment of $F_1$ and $F_2$ . . . . .   | 85  |
| 4.9  | The merging result of the alignments in Figure 4.7 . . . . .   | 87  |
| 4.10 | The aggregated neighborhood context graphs during the definition of the edges’ labels . . . . .                                      | 88  |
| 4.11 | A C-BPMN derived from the aggregated neighborhood context graph in Figure 4.4 . . . . .  | 92  |
|      |  |     |
| 5.1  | $P^c$ : A configurable travel booking process . . . . .  | 97  |
| 5.2  | An excerpt of the configuration guidance model extracted for the configurable process model in Figure 5.1 . . . . .                  | 100 |

|      |  |     |
|------|--|-----|
| 5.3  | An excerpt of the configuration system derived from the configuration guidelines in Table 5.1 . . . . .  | 101 |
| 5.4  | A configured process variant derived from the process model in Figure 5.1 and the selected configurations . . . . .  | 104 |
| 5.5  | (a) An implication graph and (b) its derived optimal spanning tree . .   | 109 |
| 5.6  | An excerpt of a transition system $TS_{\mathbb{G}}$ derived from the configuration guidelines in Table 5.1 . . . . .   | 112 |
| 5.7  | Transition system . . . . .  | 112 |
| 5.8  | Regions . . . . .  | 112 |
| 5.9  | Synthesized petri net . . . . .  | 112 |
| 6.1  | An ongoing design of a travel booking process . . . . .  | 118 |
| 6.3  | The discovered configurable fragment of the activity $a_4$ within 2-layers and the ranked configuration guidelines attached to the configurable elements as text annotations . . . . . | 120 |
| 6.4  | An example of an event log with parallel relations . . . . .   | 122 |
| 6.5  | The fragment corresponding to the event log in Figure 6.4 . . . . .  | 122 |
| 6.6  | The neighborhood context graph of the activity $g$ in the process fragment in Figure 6.7 . . . . .   | 122 |
| 6.7  | The parallel relations in neighborhood context graphs . . . . .  | 122 |
| 6.8  | The process fragment discovered from the logs $L_{a_{21}}^2$ and $L_{a_{42}}^2$ in Table 6.3 and Table 6.4 respectively . . . . .  | 123 |
| 6.9  | The resulted configurable process fragment after applyinh the shared/unshared activity strategy . . . . .  | 124 |
| 6.10 | An excerpt of a frequency suffix tree derived from $L_{merged}$ . . . . .  | 126 |
| 6.11 | The suffix tree $S_{conf}$ of the configuration log $L_{conf}$ . . . . .   | 127 |
| 6.12 | The representation of the three relations $AND$ , $XOR$ and $OR$ using the set theory . . . . .  | 130 |
| 7.1  | A screen-shot of the graphical interface for configurable process modeling in Signavio . . . . .   | 136 |
| 7.2  | A screen-shot of the Signavio graphical interface for visualizing the configuration guidelines . . . . .   | 137 |
| 7.3  | A screen-shot of the Signavio graphical interface for visualizing the configuration guidelines dependencies' in Petri-nets . . . . .   | 138 |
| 7.4  | A screen-shot of the <i>mineFrag</i> plugin in ProM . . . . .  | 138 |
| 7.5  | An activity selected in an existing business process for which a configurable fragment and its configuration guidelines will be discovered . .   | 139 |
| 7.6  | The configuration of the process fragment assisted with the ranked configuration guidelines . . . . .  | 140 |
| 7.7  | Number of configurations per guideline and per element for different minimum support thresholds and a minimum confidence threshold $C = 0.8$ . . . . .                                 | 149 |

---

|     |  |     |
|-----|--|-----|
| 7.8 | The average precision and recall values with different support and confidence thresholds . . . . . | 151 |
| 7.9 | Precision and recall metrics values with different $k^{th}$ -layer values . . .                    | 154 |



# Introduction

## Contents

---

|            |   |           |
|------------|---|-----------|
| <b>1.1</b> | <b>Research context . . . . .</b>   | <b>19</b> |
| <b>1.2</b> | <b>Research problem: <i>How to propose automated support for configurable process models?</i> . . . . .</b> | <b>23</b> |
| 1.2.1      | On assisting configurable process design . . . . .  | 24        |
| 1.2.2      | On supporting business process configuration . . . . .  | 24        |
| <b>1.3</b> | <b>Motivating example . . . . .</b>   | <b>25</b> |
| <b>1.4</b> | <b>Thesis principles, objectives and contributions . . . . .</b>  | <b>31</b> |
| 1.4.1      | Thesis principles . . . . .   | 31        |
| 1.4.2      | Thesis objectives . . . . .   | 31        |
| 1.4.3      | Thesis contributions . . . . .  | 32        |
| <b>1.5</b> | <b>Thesis outline . . . . .</b>   | <b>34</b> |

---

## 1.1 Research context

The increasing pressure from competitive business environments forces companies to provide effective Information Technology (IT) support for achieving excellence and performance in the management and execution of their processes [8,9]. To this end, there has been recently an increasing adoption of Process-Aware Information Systems (PAIS) which are software systems that manage and execute operational processes involving people, applications, and/or information sources on the basis of process models [10]. Examples of such systems are Business Process Management (BPM) systems [11–15].

The key ingredient of a PAIS system is the explicit representation of a process model referred to as *business process model*. A business process model describes the logical and temporal order in which organizational tasks have to be performed to realize a given goal [16]. The PAIS lifecycle (also known as BPM lifecycle) involves repeated steps to carry out continuous improvement of the business process models. It consists of four steps: (1) process design, (2) process implementation, (3) process execution and (4) process diagnosis. In the process design phase, the business

process is modeled using graphical notations proposed in the literature such as Petri nets [17], Yet Another Workflow Language (YAWL) [18], Event-Driven Process Chain (EPC) [18], Business Process Model and Notation (BPMN) [19], UML Activity Diagram [20], etc. In the implementation phase, the designed business process model is implemented and enhanced with technical information that facilitates the enactment of the process by the system. Once implemented, the process can be executed. During execution, data is recorded in log files. Recorded data is then analyzed in the diagnosis phase to identify problems and improve the designed business process model using techniques such as process mining techniques [21].

Although through BPM organizations can derive significant time and cost savings [22], new challenges arise for effective management of business processes in today's fast changing business needs. In such a highly dynamic environment, the business process design, which is the initial and key phase of business process development [11], becomes time-consuming, error-prone, and costly [22, 23]. Therefore, seeking reuse [24] and adaptability [25] is a strong requirement for a successful business process design. On the one hand, it would be inefficient if every time a company engages in modeling or re-designing its process, it did so "from scratch" without any consideration of design experiences, best practices or how other companies perform similar processes. To this end, many efforts on assisting business process design through *reuse* have been proposed such as using process templates [26] or reference processes [24], measuring the similarity between business process models [27, 28] and recommending activities [29, 30]. On the other hand, business processes need to be *flexible* so that they can quickly adapt to new requirements [31, 32]. Owing to this fact, a broad research area has addressed the flexibility and adaptability in business process models [33–40].

Configurable process models introduced in [34, 41] were a step toward enabling a *process design by reuse* while providing *flexibility*. They allow an explicit representation of the common and variable parts of similar processes into one customizable model. A configurable process model is a generic model that integrates multiple process variants of a same business process in a given domain through variation points. These variation points are referred to as *configurable elements* and allow for multiple design options in the process. A configurable process model allows process analysts to have a global view on the commonalities and differences between multiple variants of a business process. It needs to be configured according to a specific requirement by selecting one design option for each configurable element. Once configured, an individualized process variant is derived from the set of selected configurations with a minimal design effort.

To understand how a configurable process model is configured and individualized, we illustrate a simple example in Figure 1.1. On the left-hand side of the figure, a configurable process modeled with the Configurable BPMN (C-BPMN) notation is depicted (More details on BPMN and C-BPMN are discussed in Chapter 3). Briefly, BPMN consists of three main elements for modeling the control-flow in a business

process: event, activity, and gateway. An event is represented with a circle and denotes something that happens. An activity describes the kind of work which must be done and is graphically modeled with a rectangle. Three main types of gateways, *OR* (inclusive choice), *XOR* (exclusive choice) and *AND* (parallel flow) are used to model the splits and joins in the model. C-BPMN allows the control-flow elements to be configurable. Configurable elements are graphically modeled with thick lines. Returning back to our example, the configurable process process contains 5 activities:

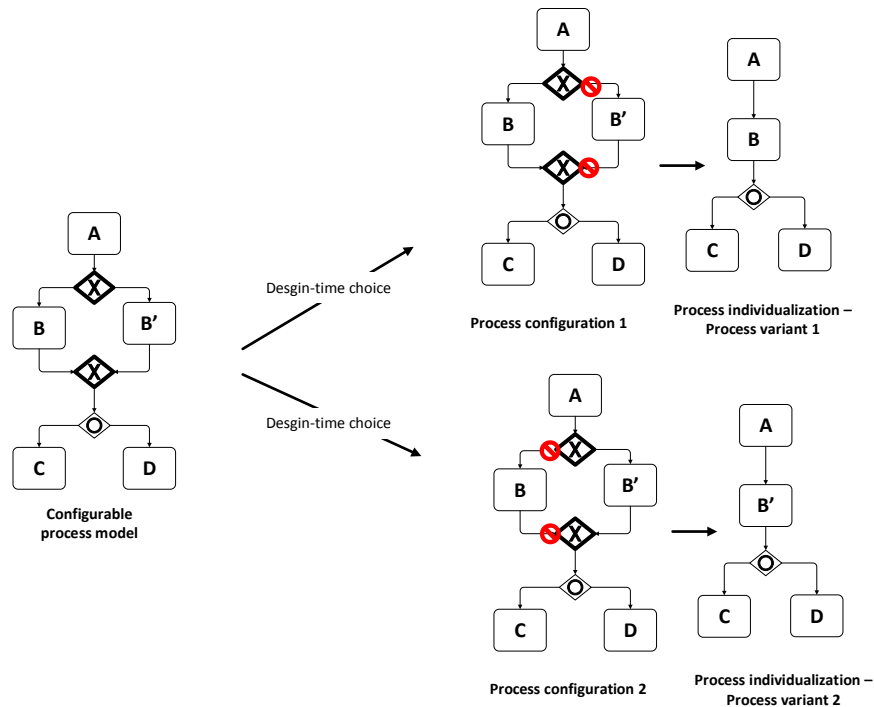


Figure 1.1: Configuration and individualization of a configurable process model

$A$ ,  $B$ ,  $B'$ ,  $C$  and  $D$ .  $B$  and  $B'$  are connected through a configurable *XOR* (denoted as  $XOR^c$ ).  $C$  and  $D$  are connected through a “normal” *OR*. Unlike a “normal” BPMN gateway, the  $XOR^c$  does not represent a run-time decision. Instead, it represents a design choice that will need to be made by an analyst to adapt the configurable process model to a particular setting, such as a project or an organization. For instance, one may choose to exclude the functionality implemented by the task  $B'$ . In terms of configuration, this corresponds to blocking the path of  $XOR^c$  leading to  $B'$  (see *Process configuration 1* in Figure 1.1). Once configuration choices are selected, the individualization phase consists of (i) deriving a process variant from the configured process that does not contain the elements excluded during configuration and (ii) mapping the configurable elements to normal ones. For example, in Figure 1.1, the derived variant (see *Process individualization 1 - Process variant 1*) does not contain



$B'$  and the configurable *XOR* gateways are mapped onto sequences. This variant does not contain any configurable element and therefore can be executed by the PAIS.

Recent research activities on configurable process models have led to the specification of many configurable modeling notations as for example configurable EPC (C-EPC) [34] and configurable YAWL (C-YWAL) [41] that extend the EPC and YAWL notations respectively with configurable elements. Since then, the issue of building and configuring configurable process models has been investigated. As configurable process models tend to be very complex with a large number of configurable elements [42], many automated approaches have been proposed to assist their design [43–47]. These research results highlight the need for means of support to configure the process. Therefore, many approaches have been proposed to build a configuration support system for assisting end users selecting desirable configuration choices according to their requirements [6, 7, 48–50].

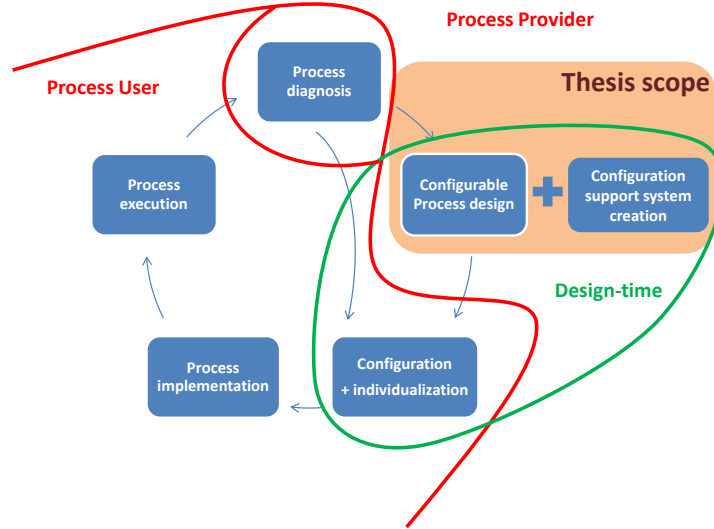


Figure 1.2: Configurable process models in the BPM lifecycle (inspired from [3])

To resume, we illustrate in Figure 1.2 the configurable process development steps in the BPM lifecycle that is inspired from [3]. Besides the traditional BPM lifecycle phases, the *configuration+individualization* phase is added. Moreover, the process design phase is replaced with *configurable process design* (detailed in Section 1.2.1) and it is enhanced with the creation of a configuration support system phase (detailed in Section 1.2.2). The configurable process design and the creation of a configuration support system are the scope of this thesis work. The three phases *configurable process design*, *configuration support system creation* and *configuration+individualization* are

performed at design-time. The lifecycle steps are split into two roles (i) *the process provider* who is responsible of designing the configurable process, building a configuration support system and performing the diagnosis of the designed configurable process, and (ii) *the process user* who is responsible of configuring and individualizing the configurable process assisted with the configuration support system and then executing and performing the diagnosis phases.

## 1.2 Research problem: How to propose automated support for configurable process models?

As we mentioned in Section 1.1, the design and configuration of configurable process models has been an active research area over the last years. Configurable process models tend to be very complex with a large number of configurable elements and many interdependencies between their configuration choices. Therefore, the process providers need means of support in order to create their configurable process models and configuration support systems. This research problem is illustrated in Figure 1.3.

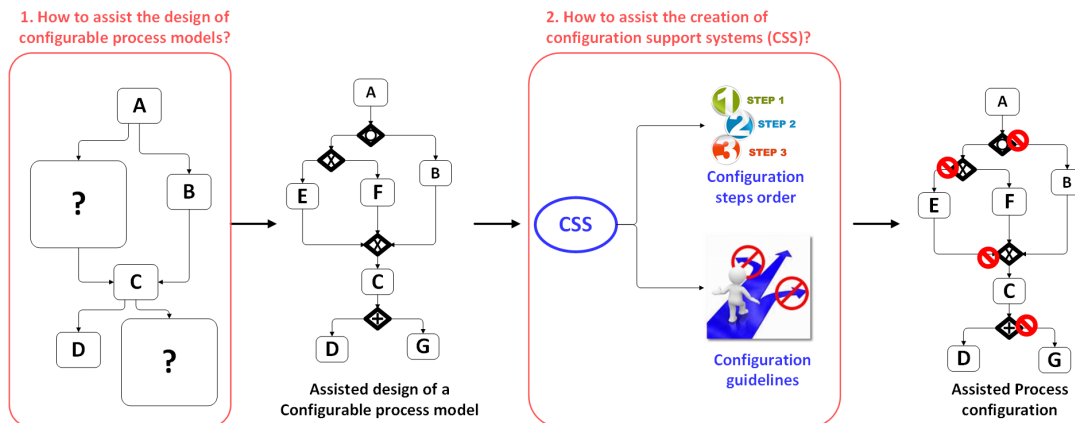


Figure 1.3: Research problem

First, the design from scratch of a configurable process model that includes all possible functionalities in a given domain is undoubtedly a tedious, if not impossible task. Therefore, automated approaches should be proposed to assist their design (see Section 1.2.1). Second, once designed, the model is provided to the process users (e.g. organizations, companies, departments, etc.) who are responsible of configuring it according to their specific needs. However, the manual configuration of a process model without any support is an error-prone and time consuming task. Therefore, process users should be assisted with configuration support systems in order to have recommendations on the suitable configuration choices (see Section 1.2.2).

### 1.2.1 On assisting configurable process design

Motivated by the “Design by Reuse” paradigm, many approaches have proposed to take into consideration the previous design experience, best practices and how other companies perform similar processes in order to assist the design of configurable process models.

To this end, several works have proposed to merge similar process models into configurable one [43–45, 51–53] or mine a configurable process from a collection of execution logs [46, 47, 54]. However, they all targeted to *merge or mine entire processes* which result in *large and complex models* that are difficult to understand and reuse [42]. Moreover, merging and mining entire process models *cost much computation time* especially when there exists a high number of large input models. In some cases a compromise between the computational complexity and the quality of results [28, 55, 56] needs to be found.

On the other hand, recommending entire configurable processes provides only the possibility to *configure and (re)use the entire process model*; while in some circumstances, the process providers may be interested in only some parts of the process model. For instance, a process provider may look for *specific process fragments* that are suitable to fill a missing part (e.g. see the process in the left of Figure 1.3) or that can replace some parts causing efficiency degradation in his process. In this case, assisting the process provider with an entire configurable business process is not helpful. Instead, *fine-grained and focused configurable fragments* are more suitable and straightforward.

In light of these limitations, our first objective is to facilitate the design of configurable process models without confusing the process providers with large and complex results. We aim at recommending configurable fragments that are relevant to selected positions of an ongoing designed process. We follow the “Design by Reuse” paradigm by using existing data (previous process models, execution logs) to recommend focused and comprehensible results. We also want to avoid the computational complexity problem. To address this research problem, we need to answer the following questions:

1. How to identify fragments that are close to process provider interests?
2. How to derive configurable fragments?
3. Can execution logs be useful? and how?
4. How efficient our approach is (in terms of configurable fragments complexity and computation time)?

### 1.2.2 On supporting business process configuration

As configurable process models should be carefully configured to derive correct variants [57], many approaches have been proposed to preserve the structural and behav-

ioral [58, 59] correctness during configuration and to derive valid variants considering specific domain constraints (e.g. using Questionnaire models [7], rules [49], feature models [6, 50], etc.).

While structural and behavioral based approaches are automated, the domain based approaches still *require a significant manual work*. Indeed, *domain experts are actually in total charge of creating configuration support systems* that assist process users selecting valid configuration choices according to their specific needs and to the domain constraints. Most of these systems are business-oriented. They abstract from the process technical details and recommend (1) the order in which the configuration steps are performed and (2) the suitable configuration choices according to a set of identified domain constraints (see CSS in Figure 1.3).

Unfortunately, manually ordering the configuration steps through dependencies' relations according to the expert knowledge is far from trivial, especially for complex process models with a large number of configurable elements and many interdependencies between their configuration choices. In addition, manually identifying all possible domain constraints is undoubtedly a tedious and error prone task. Finally, in today's dynamic and fast changing requirements, configurable process models may be subject to dynamic changes [3]. Therefore, each change in the configurable process model requires the intervention of the domain expert in order to update the configuration support system according to the process model modifications. This task manually performed is challengeable and unrealistic and may affect the configuration performance.

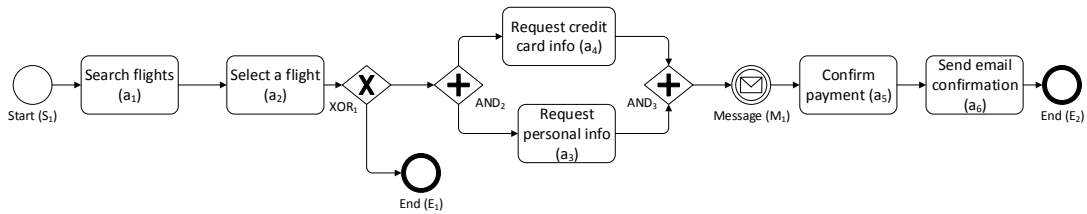
In light of these limitations, our second objective is to assist the creation of configuration support systems. We aim at learning from the experience gained through previous process configurations, in order to extract implicit and useful knowledge for the configuration decision making. By doing so, we target to integrate, for the first time, the process users' experience in the creation of configuration support systems which has been recognized as successful for the configuration experience [3, 34]. To address this research problem, we need to answer the following questions:

1. How to assist the creation of configuration support systems?
  - How to assist the identification of configuration steps order?
  - How to assist the identification of domain constraints?
2. Can execution logs be useful? and how?
3. How efficient our approach is (in terms of results quality and accuracy)?

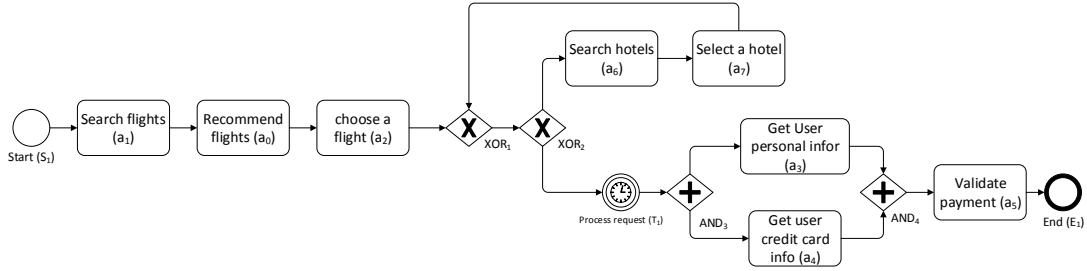
### 1.3 Motivating example

We present in the following a scenario to illustrate and motivate our approach. It is also used to explain our approach in the next chapters.

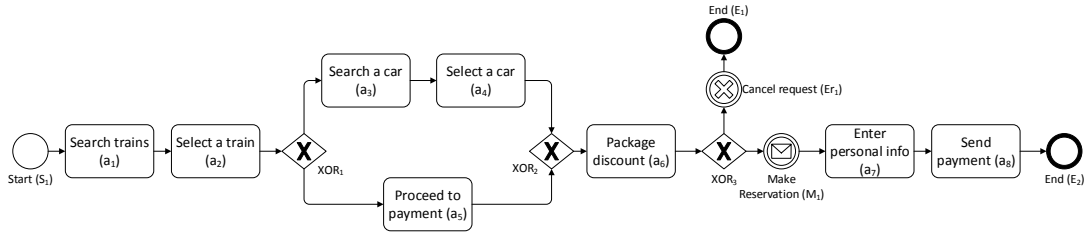
We consider a travel booking agency that has multiple branches in different cities and countries. These branches execute different variants of the same process that may differ in their structure and behavior according to the country and customers needs. In Figure 1.4, we show three variants of the travel booking process executed by three different branches:  $branch_1$ ,  $branch_2$  and  $branch_3$  which are modeled with BPMN 2.0. Please note that, although our processes are modeled with BPMN, the *de-facto* process modeling notation, our work can be easily extended to other *graph-based* business process modeling notations such as EPC.



(a)  $BP_1$  of  $branch_1$ : A flight booking process variant



(b)  $BP_2$  of  $branch_2$ : A flight and hotel booking process variant



(c)  $BP_3$  of  $branch_3$ : A train and car booking process variant

Figure 1.4: Three process variants of a travel booking process

The first variant  $BP_1$  (in Figure 1.4a) corresponds to a simple flight booking process. The traveler starts by searching for available flights (activity  $a_1$ ) according to a set of selected criteria (e.g. date, departure city, arrival city, etc.). He can select a one (activity  $a_2$ ) or cancel the request (end event  $E_1$ ) and terminates the processes. In case of a selected flight, the traveler proceeds to the payment step. He enters his personal and credit card info (activities  $a_3$  and  $a_4$ ) and confirms the payment (activity  $a_5$ ). A confirmation email is then sent (activity  $a_6$ ) and the process terminates. The

second variant  $BP_2$  (in Figure 1.4b) corresponds to a flight booking process with the option of booking a hotel. The steps are approximately the same as the first variant with the addition of the recommendation functionality (activity  $a_0$ ) and the hotel reservation related activities (activities  $a_6$  and  $a_7$ ). The third variant  $BP_3$  (in Figure 1.4c) corresponds to a train booking process with the option of renting a car. The traveler starts by searching and then selecting a train (activities  $a_1$  and  $a_2$ ). Thereafter, he has the option to rent a car (activities  $a_3$  and  $a_4$ ) and can benefit from a discount offer (activity  $a_6$ ). If he doesn't like the offer, he can cancel the request (cancel event  $Er_1$ ) and terminate the process. Otherwise, he proceeds to enter his personal information (activity  $a_7$ ) and finally he sends the payment (activity  $a_8$ ) and the process terminates.

Suppose now that  $branch_1$  notices that its customers often search for a hotel after booking their flights while  $branch_2$  notices that its customers often search for a car after booking their flights and hotels. Since each of these branches' processes do not support the combination of the aforementioned functionalities, the customers search for the requested services in an ad-hoc manner. In order to answer the new business needs and develop more and more value added processes, the process provider, decides to (1) modify  $BP_1$  in order to integrate the hotel booking functionality and (2) modify  $BP_2$  in order to integrate the car rental functionality. After a while, the agency decides to open a new branch "branch 4" that needs to define its own process. The general requirements for the new branch process are transmitted to the process provider. In his turn, the process provider notes that the new process should allow for a flight booking, a hotel reservation and a car rental service. The process should also offer promotions and discounts to the clients. Therefore, he decides to combine these functionalities in a new process  $BP_4$ .

In such a changing business environment, *an ad-hoc process design becomes time-consuming and costly*. Indeed, as the business requirements evolve, the process provider is engaged in modeling a new business process or adapting an existing one in order to answer the varying customers' needs and remain competitive in market and business survival. Therefore, the process provider decides to consolidate its expertise in travel booking by designing a generic configurable reference model that can be customized and used by different process users (i.e. the agency branches) according to their specific needs.

At this stage, the process provider needs some assistance in order to design the process. He could use existing approaches such as process merging and process mining to create a configurable process model from the existing ones. However, the returned models can be very large and complex when they are derived from a high number of process models. This can make the process provider confused and unable to control and manage the variable parts in his process.

Therefore, the process provider decides to rely on his experience to rapidly sketch-out the reference travel booking process with some basic activities as given in Figure 1.5. He marks the unknown parts with a '?' symbol. Mainly, the sketched

activities are those that are well known and common for all travel booking processes while the unknown parts are those that represent alternatives such as booking a hotel or a train, renting a car, offering discounts, etc.

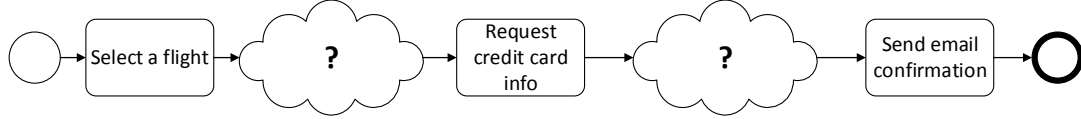


Figure 1.5:  $BP_R$ : An ongoing design of a travel booking process

At this stage, he would like to know how the outlined activities are connected to others in previously modeled processes. Using our approach, we propose to recommend *configurable process fragments* that are close to the process provider interests and that inspire him to complete the missing parts. Concretely, the process provider selects an activity for which he desires to have propositions on how it is connected to the missing parts in the process. By capturing the activities' relations to their closest neighbors in a fragment-based structure referred to as *neighborhood context graph* [2], our approach can *extract*, *cluster* and *merge* all fragments from different process models that contain an activity similar to the selected one into configurable fragments. For example, suppose that the designer selects the activity “Select a flight” in the process in Figure 1.5. We detect that the activities “Select a flight” and “choose a flight” in  $BP_1$  and  $BP_2$  in Figure 1.4 have a similar functionality. Therefore, based on the *neighborhood context graph* definition, we (i) extract the fragments that include these activities and their relations to their closest neighbors, (ii) cluster them based on their similarity and (iii) merge the created clusters into configurable fragments. An example of the resulted configurable fragment is shown in Figure 1.6. It contains the activity selected by the designer (i.e. “Select a flight”) which is the result of merging the activities “Select a flight” and “choose a flight”, and its relations to its activities' neighbors through configurable elements.

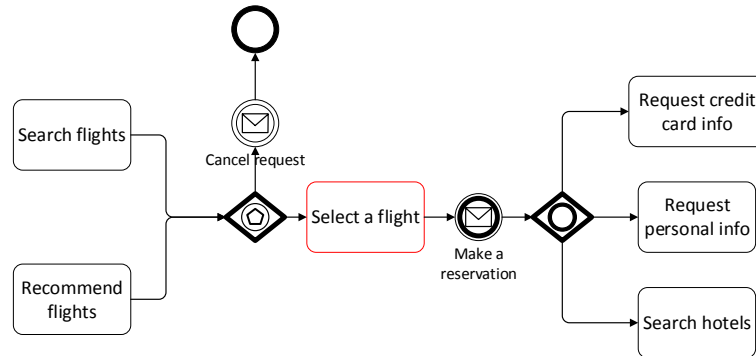


Figure 1.6: A configurable fragment that contains the activity “Select a flight”

By recommending configurable fragments to specific positions in the process, pro-

cess providers can interactively complete the design of their processes. They can select activities from the recommended fragments in order to have more recommendations and extend the ongoing designed processes. Also, recommending configurable fragments give them the hand to specify the configurable parts in their processes for a better variability control and management.

Suppose that the process provider completes the design of the configurable travel booking process as shown in Figure 1.7. This process includes four main functionalities: (1) flight booking with alternatives (i.e. the process flow in the green dashed rectangle), (2) recommendation (i.e. the process flow in the red dashed rectangle), (3) discount offer (i.e. the process flow in the red dashed rectangle) and (4) payment (i.e. the process flow in the orange dashed rectangle).

This process is shared between different process users. It is configured according to their specific needs. However, the process provider receives complaints from the process users as they encounter difficulties during the configuration of the process. They claim that they had to analyze and understand the large and complex configurable process model in order to (i) *detect how the configuration choices are interrelated in the process* and (ii) *select those that suit best their needs following a logical configuration steps order*. The dependencies between different configuration choices may come from specific domain constraints. For example, in the travel booking domain, one may identify that the discount offer is frequently proposed if the process includes a recommendation functionality. Therefore, in the configurable process in Figure 1.7, the configuration of the elements in the discount offer part depends on those in the recommendation functionality part. For instance, if the process user blocks the configurable activity “Get Package discount” (i.e. exclude it from the process), then it is recommended that he also blocks the outgoing flow of the configurable gateway  $XOR_1^c$  starting with the “Recommendation functionality” part.

In order to ease the process configuration experience, the process provider decides to build a *configuration support system* that assists the process users during the configuration of the process model. The configuration support system should guide the process users step by step to configure the process by (1) *presenting the order in which the configurable elements are configured* and (2) *recommending suitable configuration choices for each configurable element taking into account the already chosen ones*. However, relying solely on the process provider knowledge to create the configuration support system is error-prone and inefficient. In our approach, we propose to assist process providers building their configuration support systems. We realize that existing process models in the same domain contain implicit and useful information for process configuration. Therefore, we propose to learn from the experience gained through previous process modeling and configuration in order to recommend process providers relevant information for the creation of configuration support systems. Basically, we recommend them a *plan for the configuration steps order* and (ii) *a set of configuration guidelines that can be mapped to domain constraints*.



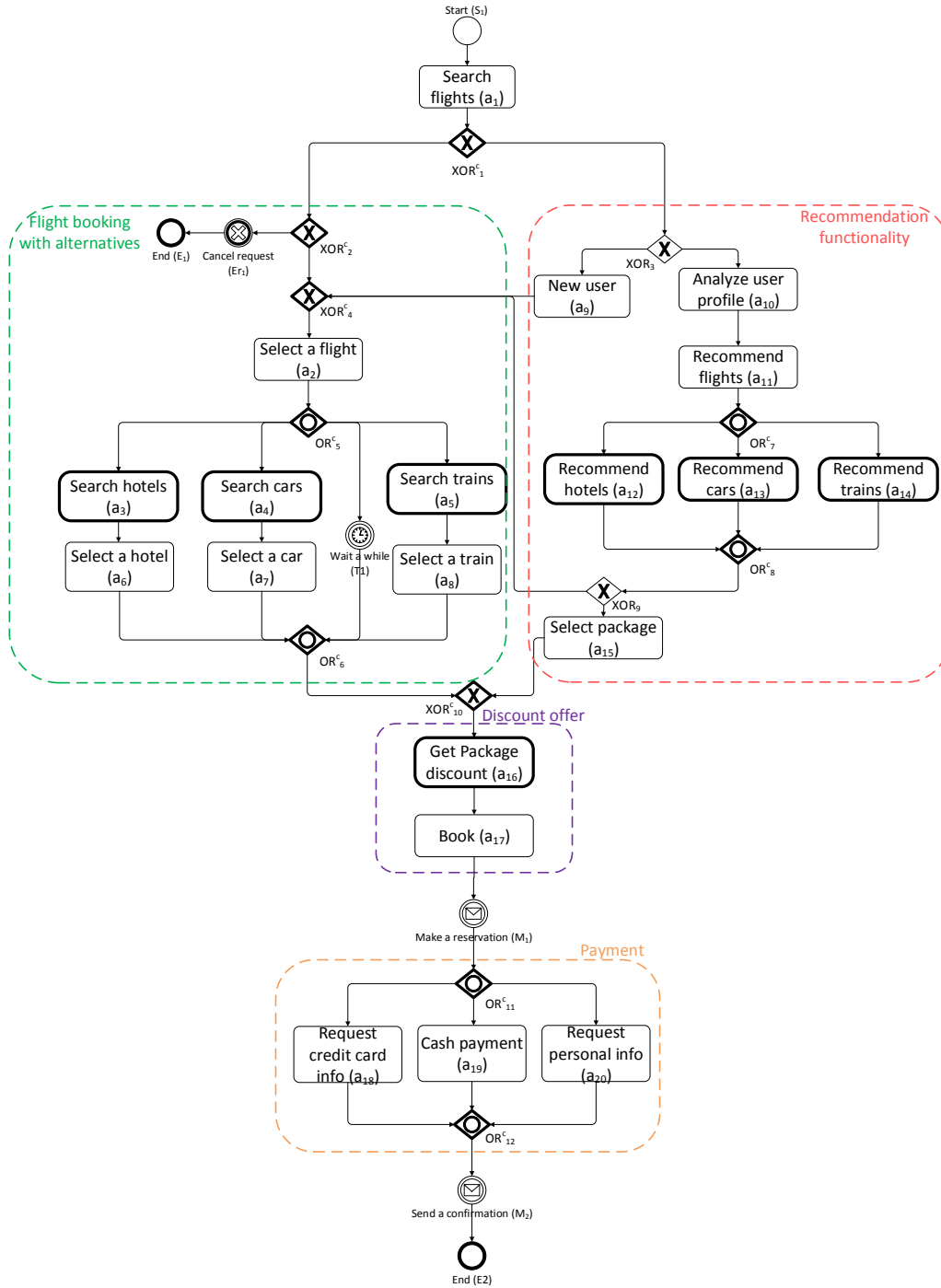


Figure 1.7: A configurable travel booking process

## 1.4 Thesis principles, objectives and contributions

### 1.4.1 Thesis principles

In our approach we consider the following principles:

- **Automation:** The approach should propose automated techniques in order to support the design and configuration of configurable process models.
- **Implicit knowledge exploitation:** The approach should be driven by the “Design by Reuse” and “Configuration by Reuse” paradigms. Consequently, it should extract and utilize implicit knowledge hidden in existing and accessible PAIS data such as designed process models and execution logs.
- **Focused results:** To not confuse the process providers, the approach should recommend focused results that are close to their interest.
- **Balanced computation:** The approach should make a compromise between the computational complexity and the quality of results.

It is noteworthy that the proposed work in this thesis needs to be (i) validated through proof of concepts and (ii) evaluated through different experiments on real datasets and returned users’ feedback. Therefore, the implementation, experiments, and case study results with end users should be detailed.

### 1.4.2 Thesis objectives

In this thesis, we aim at **proposing automated support for configurable process models**. Our objective is twofold: (i) assist the design of configurable process models by proposing fine-grained results that are close to process providers’ interests and (ii) assist the creation of configuration support systems that assist the business process configuration.

To achieve the first objective, we propose to learn from the experience gained through past process modeling in order to assist process providers with *configurable process fragments*. The recommended fragments are close to process providers interests and inspire them to complete the missing parts in their ongoing designed processes.

To achieve the second objective, we realize that previously designed and configured process models contain implicit and useful knowledge for process configuration. Therefore, we propose to benefit from the past experience in order to automatically derive relevant and useful information for the creation of configuration support system.

Since process models are not always explicitly modeled (as for example in hospital information systems [60]) and do not always provide the real behavior of their executions, we realize that we could use the execution logs (referred to as *event logs*) which

are available in all today's information systems in order to achieve our objectives. Therefore, we also propose to recommend configurable process fragments along with a configuration assistance using available event logs.

### 1.4.3 Thesis contributions

To recommend configurable fragments, we define a process fragment as the *neighborhood context graph* [2] of an activity that consists of relations between the associated activity and its neighbors. This definition is focused and granular so that it enables process providers to view the possible interactions of an activity to its closest neighbors. For an activity selected by the process provider, we propose to discover the neighborhood context graphs around the activities having a functionality similar to the selected one in different processes. Since the neighbor context presents the behavior of the associated activity within the process, we expect that similar activities show many similarities between their neighborhood context graphs. Therefore, these graphs are *extracted, clustered and merged* into configurable process fragments. The resulted configurable fragments contain the selected activity and its relations to its closest neighbors in different processes through configurable elements.

To support the creation of configuration support systems, we introduce the new concept *configuration guidance model* which provides information on (i) the configuration guidelines for selecting desirable configuration choices in a configurable process and (ii) the configuration steps order to be followed. We propose an automated two-step approach to extract configuration guidance models from existing business process repositories. The first step consists of *extracting configuration guidelines* from existing business process models. These guidelines reveal how the configuration decisions are interrelated in a configurable process model. To do so, we propose to use Data Mining techniques [61], in particular Association Rule Mining [62].

We notice that the derived configuration guidelines *should be carefully and correctly applied to avoid inconsistent configuration results*. Therefore, we push farther our work and propose to formalize the configuration guidelines dependencies' using Petri nets [63]. We identify three main dependencies' relations that may exist between different configuration guidelines, mainly *causality* (i.e. in which order the guidelines can be applied), *concurrency* (i.e. which guidelines can be applied in parallel) and *exclusivity* (i.e. which guidelines exclude the application of each others). These relations are automatically derived using the Theory of Regions [64].

The second step consists of *inferring the order in which the configuration steps are performed*. To do so, we propose to infer a partial order between the configurable elements of the process model. We notice that the process structure imposes a partial order between the configurable elements, however this latter does not reflect their dependency from a configuration point of view. Therefore, we propose another approach that takes into account the *dependencies between the elements' configuration choices* and constructs a tree-like structure consisting of configurable elements in

parent-child relations (i.e. the parent element is configured before the child element). We use Graph Theory techniques and map the problem to the derivation of optimal spanning trees [65].

Finally, we propose a log-based approach for assisting the design and configuration of configurable process models using process mining techniques. We propose to *discover configurable process fragments from existing logs* using a log-based definition of an activity neighborhood context. We also propose to *discover ranked configuration guidelines* for assisting the configuration of the discovered configurable fragment. These guidelines take into account the importance of activities' execution which is reflected by their occurrence in the event logs. We use suffix trees [66] and Set Theory to derive the guidelines and their probabilities of occurrence expressed in terms of rankings.

We validated our approach in three steps. Firstly, we developed three proof-of-concepts *FragMerg*, *ConfRule* and *MineFrag* as extensions of Signavio process editor [67], a web-based process modeling tool and ProM [68], an extensible framework for process mining tools. *FragMerg* is an extension of Signavio and recommends configurable process fragments for a selected activity in a business process. *ConfRule* is also an extension of Signavio and extracts a configuration guidance model for a designed configurable process. *MineFrag* is a plugin of ProM and recommends configurable process fragments with ranked guidelines for a selected activity in an existing business process.

Secondly, we performed experiments on two large datasets of process models from IBM [69] and the SAP reference model [70]. We evaluated the *feasibility*, *efficiency* and *accuracy* of our proposed solutions. We made statistics on the results to estimate their quality, computed the Precision and Recall values and measured the performance of our algorithms based on the computation time. We also analyzed the parameters that impact our results quality.

Thirdly, we carried-out a case study with professional and academics in order to show the practical usefulness of a frequency-based approach for process configuration. Through this case study, we aimed to assess the usefulness of our configuration guidance models when process providers build their configuration support systems using existing manual approaches.

In summary, our contributions in this thesis are as followings:

1. **An automated approach to assist the design of configurable process models with configurable process fragments:**
  - An algorithm that *extracts, clusters and merges* process fragments for selected positions in a business process into configurable ones.
2. **An automated approach to support the creation of configuration support systems:**

- The new concept *configuration guidance model* that provides information on the (i) configuration guidelines and (ii) the configuration steps order which should be implemented by a configuration support system;
- A *data mining based approach* to extract configuration guidelines from previously modeled and configured process models;
- A *Theory of Regions based approach* to formalize the dependencies' relations between the configuration guidelines.
- A *Graph Theory based approach* to infer the configuration steps order between the configurable elements;

### 3. A log-based approach to assist the design and configuration of configurable process models:

- An *algorithm for discovering configurable process fragments* from existing event logs.;
- A *frequency-based approach using suffix-trees and Set Theory* to mine ranked configuration guidelines for assisting the configuration of the discovered fragment.

### 4. A three-step validation approach:

- *Three proof-of-concepts* for each contribution implemented as extensions of Signavio process editor and ProM framework;
- *Experiments on two large datasets from IBM and the SAP reference model* to demonstrate the feasibility, efficiency and accuracy of our proposed solutions;
- A *case-study conducted with professionals and academics* to show the practical usefulness of a frequency-based approach for process configuration.

## 1.5 Thesis outline

This thesis is organized as follows: Chapter 2 presents a background on our research context. It starts by presenting the concept of variability management that has been widely studied in the context of Software Product Line Engineering and then in the context of Business Process Management. We then present the different proposed configurable process modeling approaches for enabling a design-time variability modeling in business processes. Next, we study different approaches for supporting the design and configuration of configurable process models. We introduce their models and analyze their solutions. This analysis allows us to justify the need for *proposing an automated support for configurable process models*.

Chapter 3 presents some concepts' definitions used throughout the thesis. We give some basic mathematical notations, the different modeling formalisms that we

use (mainly BPMN, C-BPMN and Petri Nets) and an abstract representation of process models using process graphs. We also give some definitions related to event logs.

Chapters 4, 5 and 6 are the core of our thesis which elaborate our approach to support the design and configuration of configurable process models.

In Chapter 4, we present our solution to assist the design of configurable process models with configurable process fragments. We present the definition of a process fragment based on the notion of *neighborhood context graph*. Then, we present our approach for deriving configurable process fragments. We propose an algorithm for extracting, clustering and merging process fragments into configurable fragments. We show that, by construction, the resulted configurable fragment preserves the behavior of the merged ones.

In Chapter 5, we present our automated approach for supporting the creation of configuration support systems. We introduce the new concept *configuration guidance model* that provides information on (i) the configuration guidelines and (ii) the configuration steps order that a configuration support system has to include. Then, we propose an automated approach for extracting configuration guidance models from existing business process repositories.

In Chapter 6, we present a log-based approach for assisting the design and configuration of configurable process models using process mining techniques. We examine the available execution logs in order to derive configurable process fragments. Then, we present an approach to mine ranked configuration guidelines that assist the configuration of the discovered fragment.

In chapter 7, we present the proof of concepts that we implemented, the experiments that we performed and the case study that we conducted to validate our approach.

Finally, Chapter 8 concludes this thesis by summarizing the work presented and discussing possible extensions.



# Related Work

## Contents

---

|            |   |           |
|------------|---|-----------|
| <b>2.1</b> | <b>Introduction</b>                                 | <b>37</b> |
| <b>2.2</b> | <b>On facilitating configurable process design</b>  | <b>38</b> |
| 2.2.1      | Configurable process modeling                       | 38        |
| 2.2.2      | Business Process variant retrieval                  | 42        |
| 2.2.3      | Business process merging                            | 45        |
| 2.2.4      | Business process mining                             | 47        |
| 2.2.5      | Synthesis   | 48        |
| <b>2.3</b> | <b>On supporting business process configuration</b> | <b>50</b> |
| 2.3.1      | Domain-based approaches                             | 50        |
| 2.3.2      | Process-based approaches                            | 53        |
| 2.3.3      | Synthesis   | 55        |
| <b>2.4</b> | <b>Conclusion</b>                                   | <b>56</b> |

---

## 2.1 Introduction

In this chapter, we review the existing works in the literature relevant to the topic of supporting the variability in configurable process models by means of design and configuration. In Section 2.2, we study existing solutions for facilitating the design of configurable process models. We classify them into four categories: (i) configurable process modeling, (ii) business process variant retrieval, (iii) business process merging and (iv) business process mining. Next, in Section 2.3, we discuss existing works on supporting the configuration of process models. We review the proposed approaches for guiding the configuration process. These approaches can be classified into two main categories: (i) domain-based approaches and (ii) process-based approaches. We present shortcomings of the related approaches, identify the difference and bring out the advantages of our approach.



## 2.2 On facilitating configurable process design

Configurable process models allow to explicitly represent the commonalities and differences between different variants of a business process. Their design requires two main steps: (i) identify the different variants that may exist for a specific business process and (ii) aggregate the identified variants into one customizable process model. Many configurable process modeling approaches and languages have been proposed to facilitate the design of configurable process models. However, the experience revealed that manually constructing a configurable process is a time-consuming and error-prone task. Therefore, automated approaches have been proposed in the literature to assist their design by learning from the experience gained through previous process modeling.

In this section, we review existing approaches for facilitating the configurable process design and classify them into four categories: (i) configurable process modeling (Section 2.2.1), (ii) business process variant retrieval (Section 2.2.2), (iii) business process merging (Section 2.2.3) and (iv) business process mining (Section 2.2.4). In Section 2.2.5, the proposed approaches are evaluated against our *four principles* presented in Section 1.4.1: (i) automation, (ii) implicit knowledge exploitation, (iii) focused results and (iv) balanced computation.

### 2.2.1 Configurable process modeling

Business process modeling allows to represent business processes by means of suitable graphical notations [12]. Explicitly designing process models allows to filter out the complexity of the real world so that efforts can be directed toward the most important parts of the system [71]. Over the last decade, many process modeling languages have been proposed to describe business processes such as Unified Modeling Language (UML), Event-driven process chain (EPC), Business Process Model and Notation (BPMN), Yet Another workflow Language (YAWL), XML Process Definition Language (XPDL), Extended Business Modeling Language (xBML), and so on. However, these languages are not able to capture the variability in business processes for the purpose of modeling configurable processes. Therefore, various configurable process modeling languages have been proposed over the recent years to facilitate the configurable process design [5, 33, 34, 41, 72–82]. Most of them extend existing languages such as EPC [83] (e.g. [34, 72–74]), BPMN [19] (e.g. [5, 33, 75–80]) and UML [20] (e.g. [81]) with variable elements. For a comprehensive survey, please refer to [4].

Rosemann et al. [34, 72] propose a Configurable EPC (C-EPC) notation which extends EPC with variable elements in order to improve the configurability of Enterprise systems and reference models such as SAP R/3 reference model. Basically, the EPC notation consists of three main control-flow elements: *event*, *function* and *gateway*. An event can be seen as a pre- and/or post-condition that triggers a function. A function describes the kind of work which must be done. Three types of

connectors, OR, exclusive OR (XOR) and AND are used to model the splits and joins. C-EPC adds two constructs to the EPC language: *configurable nodes* and *configuration requirements and guidelines*. An example of a configurable process model in the C-EPC notation is illustrated in Figure 2.1. The configurable nodes are used to explicitly model the differences among the variants. They are the active elements of the EPC notation, i.e. functions and connectors. They are graphically modeled with a thick line. For example, in Figure 2.1, the functions *A*, *D* and *E* and the connector *OR* (represented as  $\vee$ ) are configurable. The configuration requirements and

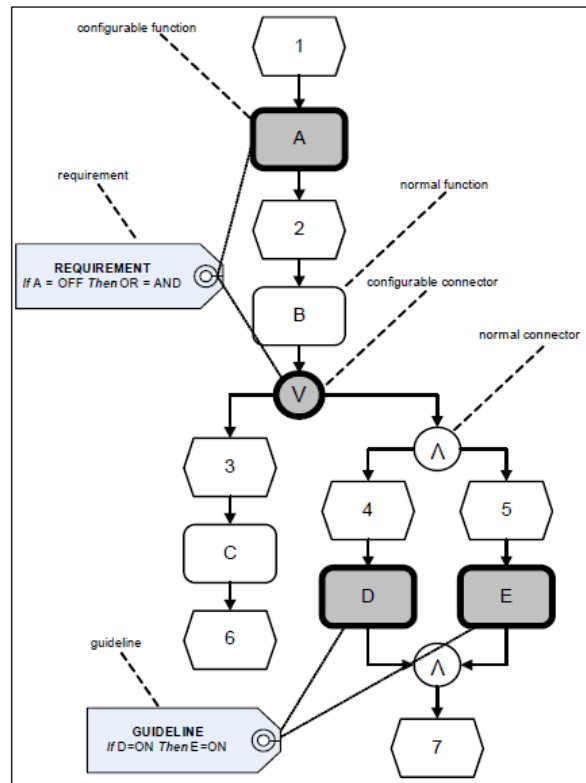


Figure 2.1: A configurable process model in C-EPC notation (before configuration, after configuration, and resulting EPC [4])

guidelines assist the users selecting the right configuration choices. The configuration requirements can be seen as hard constraints while the configuration guidelines are soft constraints. Both requirements and guidelines are expressed as logical predicates and are depicted as tags attached to the involved nodes. For example, in Figure 2.1, the requirement “*if A = OFF then OR = AND*” means that whenever the user selects the configuration choice *OFF* of the function *A*, the configuration choice *AND* of the configurable *OR* is recommended.

The C-EPC notation is extended in various ways. La Rosa et al. [73] propose to

take into account the configuration at the resource and data perspectives. The authors propose to associate the process functions to a variable number of resources and data objects through configurable connectors having a range parameter. The range allows to specify the minimal and maximal elements to be selected in a configuration choice. Vervuurt et al. [74] evaluate existing business process modeling notations, namely extended EPC, C-EPC and BPMN, based on a set of variability modeling criteria. In light of the identified limitations, different process variability modeling alternative solutions have been suggested such as: Feature-EPC which combines C-EPC with feature diagrams [84] and configuration rules, COV-EPC which extends C-EPC with Change-Oriented Versioning [85], PCL-EPC which utilizes Proteus Configuration Language [86] that in turn models the configurations and their structural variability with an object-oriented language.

Gottschalk et al. [87] present a theoretical approach for process configuration. They introduce the hiding and blocking operators to enable configurable workflow modeling using Labeled Transition Systems (LTSs). The hiding and blocking operators can be applied on the LTS edges which are its active elements. Blocking an edge means that the corresponding path in the LTS cannot be taken anymore. Hiding an edge means that the corresponding path is a *silent* one, i.e. it is traversed but it is unobserved. In [41], the Configurable YAWL (C-YAWL) language that extends YAWL with hiding and blocking operators for the activities has been developed.

Hallerbach et al. [5,75] introduce Provop (PROcess Variant by OPTions) to manage and model process variants. Different from [34,72] which derive process variants by restricting the model behavior, the Provop method is based on deriving a process variant from a reference model referred to as *base model* by applying a set of change operations (INSERT/DELETE/MOVE fragment, MODIFY attribute). Figure 2.2 illustrates the process lifecycle with Provop which consists of continuous and repeated steps of *Process modeling - Configuration of variants - Process execution - Process Optimization*. In the process modeling phase, a base model is designed based on one of five policies: (i) it could be the standard or reference process, (ii) the most frequently used process, (iii) a process model that is the minimal average distance between itself and all its variants [88], (iv) the result of merging all the process variants or (v) the intersection of the common parts of all process variants. *Adjustment points*, where change operations can be applied, are explicitly annotated in the base model to allow configuration. Since the number of variants that are derived from the change operations may be very large, Provop allows to group a set of change operations (e.g. those that co-occur frequently together) into *Options*. It also allows to specify *option constraints* that are similar to the configuration guidelines and requirements proposed by [34,72]. These constraints depict five relations between the options: (i) implication (i.e. the selection of an option implies another), (ii) mutual exclusion (i.e. the selection of an option excludes another), (iii) application order (i.e. the order in which the options are applied), (iv) hierarchy (i.e. allows to group the implication and implication order relations) and (v) at most n-out-of-m options (i.e. a process variant

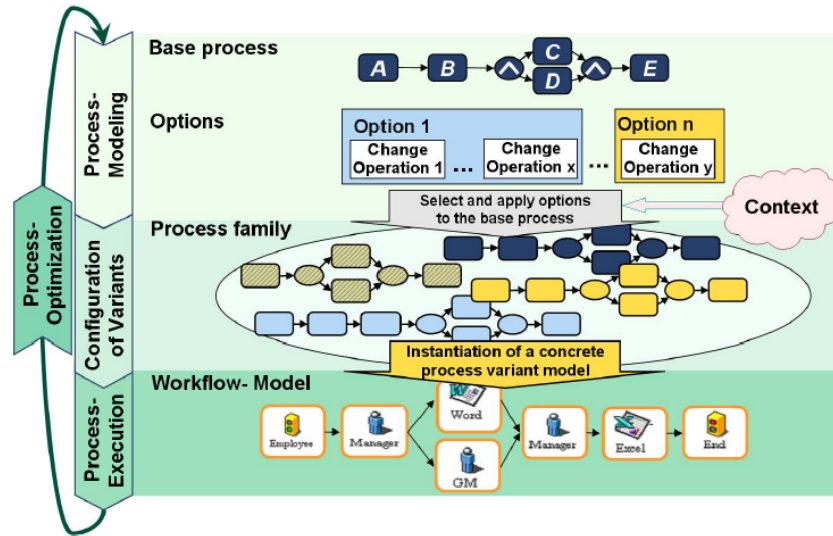


Figure 2.2: The Provop process variant lifecycle [5]

is created by applying between  $n$  and  $m$  options). In the configuration phase, the user selects a sequence of options to configure the process and derive the desired variant. In the execution phase, the configured process variant is deployed and executed by a Workflow Management System. Finally, the process variants that evolve over time as a result of configuration are analyzed for possible optimization of the base model.

Despite the great support of various configurable process modeling approaches and languages, the design of configurable process models from scratch is a well-known tedious and complex task. Indeed, a configurable process model contains the behavior of several process variants. These variants are not new but rather operational in different organizations [3]. Therefore, the starting point for building a configurable process model should be the information available about these best-practice process variants [3, 43]. In our work, we exploit such information in existing repositories of business process models and execution history. We propose to incrementally assist the process designer to complete the design of his configurable process by proposing *configurable process fragments* for selected positions in the process.

To represent our configurable process fragments, we choose the approaches that use *configurable nodes* [34] for two reasons. First, this approach has solid fundamental since it is built on top of the theoretical study on process model configuration conducted in [87]. Second, as it is highlighted in [87], configurable nodes based approaches allow a *generic-monolithic* approach for model re-use. That means, a “configurable model must be able to provide a complete, integrated set of all possible process configurations”. This is indeed a desirable property for the Off-the-shelf packages such as SAP that need to be configured by restricting their behavior to suit the specific requirements of an organization.

## 2.2.2 Business Process variant retrieval

Traditional approaches for separate modeling of process variants create many redundancies as these variants share many commonalities and the newly modeled ones are often a result of copy-pasting [89]. Therefore, the foundational base of process configuration approaches is that having one consolidated process is better than many process variants' versions [43, 90]. Automatic comparison and detection of similarities between process models is essential for retrieving the different variants of a given business process. It is considered as a prerequisite for aggregating the discovered process variants and constructing a configurable process [43]. Similarity search includes two main research streams: (i) the development of *similarity measures* for computing the similarity between process models [55, 56, 91–99] and (ii) the development of *efficient algorithms* for retrieving process (fragment) models that are similar to a given one [27, 28, 89, 100, 101].

Similarity measures can be classified into three categories [91]: (i) *labels' similarity metrics* that are based on the comparison of the activities' labels in the business processes, (ii) *structural similarity metrics* that are based on the comparison of the business processes' graph structures and (iii) *behavioral similarity metrics* that are based on the comparison of the behavior obtained from process executions.

The authors in [91] compute the similarity between two process models using a combination of metrics from the three categories. The similarity between the activities' labels is computed based on the string edit distance [102] which counts the minimal number of atomic character operations (insert, delete, substitute) needed to transform one string into another. To compute the structural similarity [55], they represent the business process models as directed attributed graphs and adopt the graph-edit distance [103] which counts the minimal number of atomic graph operations (substitute node/edge, insert/delete node/edge) needed to transform one graph into another. For the behavioral similarity, they define causal footprint vectors [99] which represent the execution orders of the activities in process models. Then, the similarity between footprint vectors is computed using vector space model, i.e. the cosine value of the angle created by the corresponding vectors.

A structural similarity based on high-level change operations is proposed by Li et. al. [95]. Different from [91] which measures the difference based on the number of deletions/insertions/substitutions of nodes, the authors take into account the execution orders between activities and measure the difference based on the deletions/insertions/movements of activities. They target to keep the execution orders when transforming one process model to the other to guarantee the soundness of the business process.

Yan et. al. [28] propose to compute the similarity between two process models based on the labels and structural similarity metrics. Similarly to [91], they use the string edit distance [102] to compute the activities' labels similarity. Regarding the structural similarity, they define the structural features of a process model as its

connection elements including start, end, sequence, split and join. They compute the similarity between two structural features by computing the average number of input and output paths of the corresponding connection elements.

Ehrig et. al [96] propose to measure the similarity between Petri-nets process models semantically modeled with the Web Ontology Language (OWL) [104]. They use a combination of labels and structural similarity measures. The similarity between the process elements' labels is computed based on syntactic (using string edit distance [102] and ontology based similarity [105]) and linguistic (using WordNet database [106] and specific UML profile [107]) metrics. For the structural similarity, they make use of the hierarchical ontology structure and take into account the context of the concept instances represented by their properties' values. They model the concepts instances and their context in a tree-like structure and compute the similarity by matching the tree elements.

The activities' labels matching used by the aforementioned approaches are improved in two ways. Mueller et al. [97] propose a new metric to increase the recall of process model matching. The authors propose to treat each activity label as a bag of words and apply word stemming techniques [108,109] for better comparability. Then, the words are pruned from the longer label and the similarity of two labels is computed based on the pruned words. A combination of syntactic (string edit distance [102]) and linguistic (based on Lin [110] metric) metrics is used to compute the overall similarity. The ICoP framework proposed by Weidlich et. al [56] overcome the limitations of existing labels' similarity metrics that allow only for 1:1 matching. This framework is tailored to deal with complex 1:n matches, i.e. each activity can be matched to an arbitrary number of other activities.

A behavioral-based metric is proposed in [93] to compare two business process models based on their execution semantics. The authors use Petri-nets to model the business processes. The executions of these models are recorded as sequences of activities, called log traces, and their frequencies. The similarity between two business process models is evaluated with respect to the precision and recall metrics. These metrics are computed based on the fitness between the log traces of the two processes and their frequencies.

The behavioral metrics proposed in [93,99] rely on exhaustive searching, i.e. the query model is compared with each model in the repository. In order to overcome this issue, Kunze et al. [92] propose a behavioral similarity metric based on the behavioral profiles [111] which satisfies the triangle inequality [112], i.e. the minimum and maximum distances of two objects can be determined without calculating it, if their pairwise distance to a third object is given. A behavioral profile is an abstract representation of a process model. It is defined as an  $n \times n$  matrix where  $n$  is the number of activities in the process. Each cell contains one out of three relations based on the activities' execution order: strict order, exclusive order or interleaving. For each of the identified relations, they define a corresponding similarity based on the Jaccard coefficient. The overall similarity between two behavioral profiles is one

minus the weighted sum of their relations' similarities.

The behavioral metric proposed by [92] may mishandle several types of constructs such as silent transitions, duplicate tasks, and cycles. Therefore, in [98], a behavioral metric that computes the similarity between two process models and describe their differences via textual statements is proposed. The behaviors of two process models are canonically represented using Asymmetric Event Structure [113]. Then, the behavioral equivalence is computed based on visible-pomset equivalence [114].

The similarity search helps to identify and retrieve process variants that are similar to a given process. It assists process designers to rapidly search for business processes that can be integrated into a configurable process model. However, this may be efficient with small-size business processes, i.e. with few activities and operations. In contrast, the large-size business processes, e.g. consist of hundreds of activities and operations, may consume much computation time. Moreover, they may make process designers confused and hard to detect how the business processes are similar and which parts should be used for the design of the configurable process model. In addition, the matching of the whole business processes often leads to the graph-matching problem, which is NP-complete [115], and they, e.g. [28, 55, 56], have to deal with the trade-off among the complexity, accuracy (efficiency) and system performance. In our approach, we focus partially on the business process and take into account only the different variant fragments related to an activity neighborhood context for retrieval. Consequently, we retrieve the corresponding fragments without facing the complexity problem.

Efficient algorithms for querying large repositories of process models to retrieve exact or approximate process (fragment) models are currently being developed. Dumas et al. [101] seek to address the problem of many duplicates referred to as *exact clones* in large repositories of process models. The authors present an indexing structure called *RPSDAG* [116] that supports fast detection of clones in large repositories of process models for the purpose of refactoring into separate sub-processes. The *RPSDAG* index also allows them to efficiently answer fragment queries. Their method is based on Refined Process Structure Tree (RPST) and code-based graph indexing. The RPST represents the process models taken as input in a tree of single entry single exit (SESE) fragments. Then the process models are indexed and duplicate SESE fragments (clones) are identified. Ekanayake et al. [89] identify approximate clones, i.e. similar SESE fragments based on the same index structure. A matrix storing the similarity between each pair of SESE fragments is constructed and used to cluster the SESE fragments. The fragments within the same cluster are considered as approximate clones. The identified clusters of approximate clones are then refactored to their medoid. Detecting approximate clones help to retrieve similar process fragments for possible configurable process fragment design. Different from them, in our approach we do not restrict ourselves to SESE fragments. Our neighborhood context definition as a process fragment model allows for more flexibility since it does not impose the well structuredness of the process models.

### 2.2.3 Business process merging

In case a collection of process models exist, configurable process models can be (semi) automatically constructed by merging the similar process models in the process repository. The original models used as inputs correspond to configurations of the resulted configurable model. The model merging has been addressed in several works [43–45, 51, 53, 82, 88, 117–119].

Li et al. [88] develop a method that merges variants into one reference model. Their method creates a reference (generic) process model from a given set of similar process variants. This process model is constructed in such way that the change distance (for example insert, delete or move actions) is minimal between the reference model and the process variants used as inputs. However, their approach only works for block-structured process models with AND and XOR blocks.

Sun et al. [118] describe the problem of merging block-structured workflow nets. The algorithm first finds the mapping pairs (i.e. points in the workflow to be merged) and then merges the models by applying a set of “merge patterns” (sequential, parallel, conditional and iterative). However, the resulted merged workflow is not configurable but rather a combined representation of the input process variants. The merge can be lossless or lossy. The last one refers to the fact that it is not guaranteed that all tasks of initial models remain in the merged model. Therefore, the behavior-preservation property is not guaranteed. Another drawback is that the proposed method is not fully automated.

The methods of Li et al. [88] and Sun et al. [118] do not guarantee the behavior preservation property. Gottschalk et al. [45] address this limitation. In their method, one can see the behavior of the input process models and also additional possible behaviors of the process. Their method, which is based on EPC, works in three phases. In the first phase, the input EPC process models are reduced to their active behavior (reduction of an EPC by removing the gateway nodes and adding them on the arcs connecting functions) and represented as functional graphs. The resulting functional graphs are then merged into a new function graph that shows the combined behavior of the input EPC models. Finally, the merged functional graph is converted back to EPC. The resulted merged EPC is not configurable and does not allow the process designer to explicitly see the commonalities and differences between the EPC input models. The function graph used in this approach is similar to our process fragment model. However, their model does not support the full expressiveness of process models since it does not allow for a chain of connectors between functions.

Mendling et al. [51] propose a method for merging two process models (EPC) that represent the same business process but from different views. The resulted model is also not configurable but a combined representation of different views. Different views could be two EPC describing the process of receiving customer inquiry. One of the processes is from the Project Management branch and the other from Sales and Distribution branch. These two EPC represent similar processes and share common parts.



The proposed method consists of three steps. The first step is to identify the semantic relationship of the two input EPC process models. This step is manually performed by the process designer who identifies the equivalence and the sequential order of functions and events in the two EPC models. The similarity can only be defined in terms of functions and events; connectors and more complex graph topologies are not taken into account. Then, as a second step, an integrated EPC is created from the two input EPCs. This is achieved by first creating an integrated EPC where all the elements of the two input EPCs are included. Then, nodes that capture the same thing in the process are merged into one node and the incoming and outgoing arcs are managed with split and join connectors. In the last step a set of restructuring rules that cleans the integrated EPC model from unnecessary structures by removing redundant arcs and eliminating connectors with only one input and one output arc are applied.

To overcome the limitations related to the lack of behavior preservation and explicit modeling of the variability in the merged model, La Rosa et al. [43] propose a process model merging method that at the same time allows process analysts to trace back the input process models from the merged model. The algorithm first extracts the common parts of the input process models and creates a copy in the configurable process model. Then, the remaining parts in the input models (i.e. the different parts) are added to the configurable process and connected to the common parts through configurable connectors. Finally the algorithm cleans the process model by eliminating the redundant elements and *useless* connectors (i.e. connectors with one input and one output branch).

The merging algorithm proposed by La Rosa et al. [43] and Gottshalck et al. [45] allows to merge a pair of process models and can be iteratively applied to merge multiple process models. Derguech et al. [44] make a new contribution by proposing an algorithm that enables merging a set of process models at once. However, their experiments show that this approach is efficient for a few number of process models. The matching problem becomes more complex when trying to match a large number of process models. In our approach, we do not face such problem since we merge small fragments that, in most cases, include only the relation of an activity to its closet neighbors, we can deal with a relatively high number of process fragments.

Schunselaar et al. [82] leverage the problem of deriving configurable process models that produce sound process variants. Furthermore, they must be fully reversible, i.e., the input process variants should be instantiations of the configurable process model. They propose fulfilling the above stated requirements by introducing CoSeNet process models. CoSeNets is a tree-like block structured process models that capture the business processes. The CoSeNet process models are read from left to right with each leaf representing a task and each parent node representing an operator (sequence, logical connectors OR, AND, data-driven XOR and event-driven XOR). The parent connectors are linked through VOID nodes (linked with edges to the parent nodes). The configuration of CoSeNet process models is achieved by blocking and/or hiding

VOID nodes.

Kuster et al. [117] introduce a method for assisting process designers in the merging procedure. Their approach is divided into three steps. In the first step, the differences between models are detected, using correspondences between process models and the SESE fragment technique they present in [120]. In the second step, they visualize the differences, and in the last step, the process models are iteratively merged based on the process designer's input.

To summarize, the merging approaches proposed in [45, 51, 88, 118] do not explicitly represent the variability in the resulted merged model. Some of the approaches ensure that the merged model subsumes the behavior of all the input models [43–45] while others do not guarantee the behavior-preservation requirement [51, 88, 118]. The approaches in [43, 44] allow to trace back the input models from the merged models. The approaches by [82, 118] work only with block-structured processes. All of the proposed approaches target to merge whole process models and can therefore encounter the problem of managing the complexity of merged models when input models are large and varied [42]. In our approach, we explicitly represent the variable parts in the merged fragments using configurable nodes. We guarantee the behavior preservation property but do not address the traceability requirement. This is because, different from existing approaches which target to maintain large repositories of process models by refactoring similar variants, our objective is to assist the design of new configurable process models. Therefore, the traceability property is not important in this context. We also do not impose the structuredness of the input models. And last, we do not face the problem of computation and model complexity since (i) we merge small fragments instead of entire process models and (ii) our fragment model represents a functional aggregated representation which allows us to speed up the matching phase.

## 2.2.4 Business process mining

Today's information systems, such as workflow management systems (e.g. Staffware), ERP systems (e.g. SAP), case handling systems (e.g. FLOWer), PDM systems (e.g. Windchill), CRM systems (e.g. Microsoft Dynamics CRM), middle ware (e.g. IBM's WebSphere), hospital information systems (e.g. Chipsoft), etc., record their business transactions as event logs [60]. These logs back up not only the business execution but also the knowledge related to the a-priori business process models. The goal of process mining [21] is to extract information from these logs in order to exploit the hidden knowledge that may be helpful for the business analysis for discovering the process models [121–127] from an enormous set of log traces. Process mining can be used to mine the business constraints to check the conformance of a-priori models [93, 128]. It can also detect execution errors [129, 130], observe social behaviors between groups of users [131], etc.

Business process mining is a discipline that sits between machine learning and data

mining on the one hand and process modeling and analysis on the other hand [132]. It was firstly introduced in 1995 [133] and was proven as a powerful technique to discover behaviors observed from event logs. It has involved the development of many tools and techniques that support mining event logs [134–137].

Business process mining techniques can be used to mine a configurable process model from a collection of event logs. Gottschalk et al. [47] presented two different approaches for mining configurable process models but these were not supported by concrete discovery algorithms. Buijs et al. [46] proposed four approaches for mining configurable models with concrete discovery algorithms from which the first two approaches are those proposed by [47]. In their work, the authors propose to use a tree-like representation to create a configurable process tree. The first approach is the merging of individually discovered process models. As a result, all process models are rooted in the configurable process tree, where a simple choice between the children of the root derives a process model. The second approach merges all event logs in a single event log, and then discovers a process model reflecting the common behavior. Afterwards, each log and the common process model are used to discover individual process models that later on are merged to create a configurable process tree using [82]. The third approach merges event logs and then creates a configurable process model. Thus, to derive process models for each variant, it uses the corresponding event log and the configurable process model; however, the model has low precision. The fourth approach is a novel technique that takes all event logs and then discovers at the same time a configurable process model and a single model for each original event log. All approaches use discovery algorithms that are based on the Evolutionary Tree Miner (ETM) [138].

Oirschot [139] uses trace clustering to create a configurable process model using a process tree representation. First, groups of behaviorally similar traces using hierarchical trace clustering are found. In the second step insights are provided to the end-user, and finally a configurable process tree using selections of groups of traces in the hierarchical clustering is created. This work offers a new technique that uses a heuristic way to discover configurations.

In our approach, we also use process mining techniques to assist the design of configurable process models. However, our target is not to discover an entire configurable process but to propose a configurable fragment that is close to designers' interests. We propose to project a collection of event logs on the neighborhood context of an activity and mine a configurable fragment from the projected logs. We also propose a frequency-based approach to derive ranked configurations from the event logs.

### 2.2.5 Synthesis

Many automated approaches have been proposed to assist the design of configurable process models [43–47, 47, 51, 53, 82, 88, 117, 118]. They merged similar process variants into one consolidated process model [43–45, 51, 53, 82, 88, 117, 118] and mined config-

urable process models from a collection of event logs [46, 47, 47]. All of them dealt with **merging or mining the whole business process model**. Therefore, they encountered the problem of managing the complexity of the resulted models when input models are large and varied [42]. They had to deal with the **NP-complexity problem** of the graph matching when searching for similar process variants and they needed to find a trade-off between the computational complexity and the quality of results or implement other strategies.

Table 2.1 shows a synthesis on the presented approaches in terms of the four principles identified in Section 1.4.1: (i) automation, (ii) focused results, (iii) implicit knowledge exploitation, and (iv) balanced computation. We further decompose the *implicit knowledge exploitation* into two sub-principles: *(iii-1) explicit variations* and *(iii-2) behavior preservation*. The *explicit variations* principle stands for the explicit representation of the configuration in the recommended results. The behavior preservation stands for the fact that the proposed merging or mining algorithms generate configurable process models that preserve the behavior of the input models. ‘+’ indicates that the corresponding principle is fulfilled by the corresponding approach, ‘-’ indicates that the corresponding principle is not fulfilled and ‘+/-’ indicates that the corresponding principle is partially fulfilled.

| Approaches                     | Principles |                 |                     |                       |                      |
|--------------------------------|------------|-----------------|---------------------|-----------------------|----------------------|
|                                | Automation | focused results | explicit variations | Behavior preservation | Balanced computation |
| [43] [44] [46] [47] [139] [82] | +          | -               | +                   | +                     | +/-                  |
| [45]                           | +          | -               | -                   | +                     | +/-                  |
| [51]                           | +/-        | -               | -                   | +/-                   | +/-                  |
| [88] [118]                     | +          | -               | -                   | +/-                   | +/-                  |
| [53]                           | -          | -               | -                   | +                     | +/-                  |
| [117]                          | +/-        | -               | +                   | +/-                   | +/-                  |

Table 2.1: Synthesis on the merging and mining approaches for assisting the configurable process design according to our principles

The configurable process modeling and process variant retrieval approaches are not shown in the table since they are used as prerequisite by most of the merging and mining approaches to derive configurable process models.

In our approach, we focus on specific parts of the business process model. We aim at assisting the process designers completing the design of their configurable process models. We define the neighborhood context graph around an activity as a process fragment model and propose to construct a configurable fragment for a selected activity in the process. The configurable fragment inspires the designer to complete the design of his process by showing him the relations of the selected activity to its closet neighbors. We exploit the existing business process models or the execution logs to automatically derive the configurable fragment. Our approach does not face the complexity problem as (i) we deal with specific and small parts of the process and (ii) our fragment model based on the neighborhood context definition allows us

to efficiently match the input fragments.

## 2.3 On supporting business process configuration

While configuration facilities in a configurable process model allow for an easy adaptation to individual needs, the configuration decisions cannot be taken freely [57]. In fact, a derived process variant needs to be correct from a *structural*, *behavioral* and *domain* perspective [3]. The structural and behavioral correctness [58,59] ensure that after individualization of the configured process model, the derived process variant is technically executable, i.e. the model does not contain disconnected nodes and is deadlock free. The domain-based correctness ensures the validity of the configuration choices regarding specific requirements. For instance, the configuration decisions to implement the process for a travel agency are different from the decisions made for a travel booking website in the internet. While structural and behavioral correctness approaches are completely automated, most of the domain-based approaches still need a considerable manual effort from domain experts.

In this section, we review existing approaches for guiding the configuration of process models according to domain constraints. We classify them into two categories: (i) domain-based approaches which propose a configuration guidance that abstracts from the technical details of the process models (Section 2.3.1) and (ii) process-based approaches which propose a direct configuration guidance on the designed process model (Section 2.3.2). In Section 2.3.3, the approaches are evaluated against our four principles presented in Section 1.4.1: (i) automation, (ii) implicit knowledge exploitation, (iii) focused results and (iv) balanced computation.

### 2.3.1 Domain-based approaches

Domain-based approaches for process configuration [6, 7, 48, 50, 76, 140–144] propose to abstract from the process technical details. They are motivated by the fact that the configuration decision-making is of the business experts responsibility who are not aware of the process modeling technical details. Most of these approaches have been inspired from configuration management in Software Product Line Engineering (SPLE) [145]. They propose to model the process variability in a domain-based model and perform the configuration on it. A mapping between the process and the domain model should be established so that the domain-based configuration decisions are translated into process-based configuration decisions.

SPLE based approaches have been used by [6, 50, 141, 142] to support the configuration of business process models. Within the domain of SPLE, product variability is managed with the use of feature models [146]. A feature model consists of one or more feature diagrams that are represented as a tree structure. At the top of the tree, high level features are depicted. Then they are decomposed into sub-features. The features can be either mandatory or optional. Constraints among sub-features are graphically

represented in a diagram. The foundational relations in a feature diagram are AND (all the sub-features must be selected), XOR (only one feature can be selected) and OR (one or several of the sub-features can be selected). A process model is configured by selecting/deselecting features from a feature model. In order to do so, a mapping should be established between the features on the one hand, and the variants of the variation points in the process model on the other hand. An example of a feature diagram, a process model and a mapping between them is illustrated in Figure 2.3. Once a feature configuration has been completed, an algorithm uses this mapping to select the right variant(s) for each variation point of the process model. Then an individualization algorithm, if available, is triggered to individualize the customized process model.

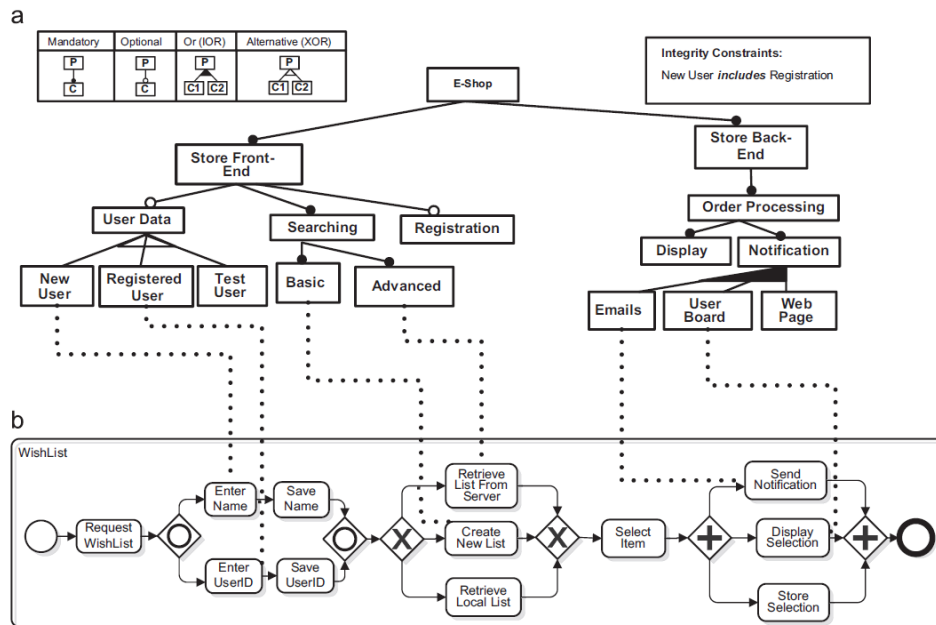


Figure 2.3: A feature diagram, a business process model and a mapping between them [6]

The SPLE based approaches require the domain experts to be familiar with the feature modeling. La Rosa et al. [7, 140] address this issue by proposing a questionnaire-driven approach for configuring reference models. They describe a framework to capture the system variability based on a set of questions defined by domain experts and answered by designers (Figure 2.4). The process model variability is captured with Boolean domain facts at each configurable node. A questionnaire model is built that is connected with the facts and the nodes. The user answers questions by choosing from alternative responses. These responses are in turn connected with facts and nodes. Depending on the answers, the configuration (i.e actions) is triggered by

configuring the respective node with the selected alternatives and removing irrelevant paths.

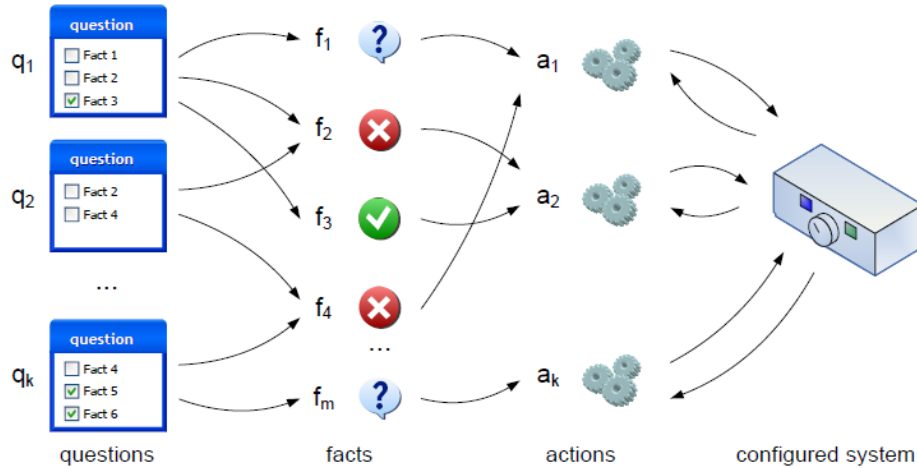


Figure 2.4: An overview of the questionnaire-based approach [7]

Lapouchnian et al. [144] propose a goal-driven configuration approach. They employ goal models to capture business goals and to analyze the variability (the various ways these goals can be attained) in the business domain. In order to configure the process, quality attributes such as customer satisfaction are used as a selection criteria for choosing among the business process alternatives induced by the goal models. These high-level variability goal models are then used in a semi-automatic variability-preserving transformation to generate configurable executable business processes.

Huang et al. [48] propose an ontology-based framework for specifying configuration guidelines using Semantic Web Rule Language (SWRL). They use two types of ontologies: a business rule ontology which is specified by a domain expert, and a process variation points ontology based on the C-EPC language. Using these ontologies, they derive SWRL rules that guide the configuration process.

The aforementioned approaches require considerable manual steps from a domain expert to create the domain model. Schunselaar et al. [143] overcome this issue by proposing an automated approach to derive a domain model referred to as *consistency graph* from a business process model. A consistency graph consists of a *concept graph* and a set of rules to ensure the consistency of that graph. The concept graph is a domain abstract representation of the activities in a business process model. It is constructed by decomposing the process activities into concepts (business objects, actions, and business object modifiers) and link them based on their ordering. Afterwards, the user selects which concepts and relations are to be taken into account in the configuration process. However, the consistency graph does not provide recom-

mendations for taking the configuration decisions.

In summary, domain-based approaches for assisting the configuration of process models require an expensive manual work from experts. This can be a labor-intensive task especially for large process models that have an exponential number of possible configurations. In addition, these approaches are only based on the expert knowledge while, as highlighted in [3, 34], a successful process configuration has to integrate the experience gained through previous configurations. Therefore, in our work, we address this research gap by proposing an automated approach for assisting the configuration of process models using previously configured processes.

### 2.3.2 Process-based approaches

Direct configuration guidance on the designed process model has been proposed by many approaches [5, 34, 43, 44, 46, 49, 54, 72, 75, 139]. Some of them require a domain expert to define configuration constraints [5, 34, 49, 72, 75] while others proposed to retrieve process configurations from previously configured process models [43, 44, 46, 54, 139].

Rosemann et al. [34] defined the requirements for a configurable process modeling technique before proposing the C-EPC notation. They highlighted the need for configuration guidelines that guide the configuration process. These guidelines should clearly depict the interrelationships between the configuration decisions and can include the frequency information. Based on the identified requirements, they developed the C-EPC notation which extends EPC with configurable nodes and *configuration guidelines and requirements* (Section 2.2.1). These guidelines are logical expressions over the configuration choices of configurable elements and are defined by domain experts.

The Provop approach [5, 75] (Section 2.2.1) allows a similar guidance but w.r.t. change operations instead of configurable nodes. The authors defined the notion of *option constraints* that consist of inclusive, exclusive, application order, hierarchy and most n-out-of-m relations between the change operations. Particularly, the application order relation overcomes the drawback of the configuration guidelines proposed by Rosemann et al. [34] since it allows to define the order in which the constraints should be applied to avoid inconsistent configurations. A graphical interface is also provided to ease the modeling of such relations.

Templates and configuration rules are proposed by Kumar et al. [49] in order to configure a reference process template using the configuration rules. The rules can be used to configure the template by restricting or extending its behavior via change operations. Change operations affect (i) the control-flow perspective (by deleting, inserting, replacing or moving a single task or a process fragment), (ii) the resource perspective (by assigning a role to a task), and (iii) the data perspective (by assigning a value to a data attribute or changing the value of a role's property or of a task's input or output data). It is also possible to change the status of a process among four



predefined values (normal, expedite, urgent and OFF). Rules associate change operations with a Boolean condition, so that if the condition is satisfied, the corresponding change operation is applied onto the process template. Depending on the type of operation, the approach differentiates between control-flow rules, data rules, resource rules and hybrid rules (the latter incorporating multiple process perspectives). These rules are defined and validated by domain experts and have priority numbers in order to specify the order in which they are applied.

Different from [5, 34, 49, 72, 75], some approaches propose to aid the configuration by recommending a previously configured process variant satisfying some requirements. The merging approaches proposed by [43] and [44] generate configurable process models in which the configurable nodes are annotated with a multiset of (*process variant id*, *configuration choice*) pairs. Each pair depicts the configuration choice in one process variant. A user specifies a process variant identifier for which the corresponding configuration choices are recommended.

The approaches proposed in [46, 54, 139] use process mining techniques instead of business process models to assist the configuration process. Buijs et al. [46] and Oirschot [139] propose a genetic algorithm that, given a configurable process tree and a collection of event logs, derive the configuration choices for each event log. Different from them, Jansen-Vullers et al. [54] propose an approach to derive the frequently executed configuration (i.e. an EPC model) given that a C-EPC exists and a log containing only data on the frequency of executed activities. They formulate the problem as an Integer Linear Programming in order to find the best configuration.

Process-based approaches allow a configuration guidance directly applied on the process. They release the process and domain experts from the burden of mapping the business oriented domain models to the technical oriented process models (as in the domain-based approaches). They can also be automated by learning from the previous experience in process configuration. However, they are difficult to use by the users who are not aware of process modeling technical details. We believe that such approaches are tailored to assist the process analysts analyzing and understanding the variability in their processes. They may be a starting point for a diagnosis and optimization step of the guidance models provided by the domain-based approaches. Moreover, since they operate on a fine-grained level (i.e. at the level of process elements), the manual approaches [5, 34, 49, 72, 75] may become quickly infeasible when process models are large and contain many configurable elements. On the other side, the existing automated approaches [43, 44, 46, 54, 139] are not suited for a configuration guidance as they return complete process variants instead of interactively assisting the user deriving the elements' configurations.

In our work, we propose a process-based approach that combines the features of process-based and domain-based approaches. We propose to *automatically* extract a configuration guidance model from previously configured process variants that *interactively assists* the user during the configuration of his process model. We explore the implicit configuration knowledge in existing process models in order to infer (i)

a plan for the configuration steps order and (ii) a set of configuration guidelines for assisting the configuration decision-making.

### 2.3.3 Synthesis

Many approaches have been proposed for supporting the configuration of business process models [5–7, 34, 43, 44, 46, 48–50, 54, 72, 75, 76, 139–144]. Some of them proposed to guide the configuration on the designed process [5, 34, 43, 44, 46, 49, 54, 72, 75, 139] while others propose to abstract from the process technical details and guide the configuration on a domain-based model [6, 7, 48, 50, 76, 140–144]. Many of them **rely on the domain expert knowledge** and require an **expensive manual work** to build the guidance model. However, as highlighted in [3, 34], a successful process configuration has to integrate the experience gained through previous configurations. Some automated approaches as for example [43, 44, 46, 54, 139] take this requirement into account and derive process configurations from existing process models and event logs. However, these approaches fail to guide the user step by step during the configuration process. They return entire process models that may confuse the user and require him to re-verify the configuration decisions that have been taken.

Table 2.2 shows a synthesis on the presented approaches in terms of our principles identified in Section 1.4.1: (i) automation, (ii) implicit knowledge exploitation (iii) focused results and (iv) balanced computation. We further decompose the *focused results* principle into two sub-principles relevant to the configuration context which are *(ii-1) configuration steps guidance* and *(ii-2) configuration decision guidance*. The *configuration steps guidance* sub-principle refers to the approaches that provide an interactive step-by-step guidance to the end users. The *configuration decision guidance* sub-principle refers to the approaches that provide recommendations on the suitable configuration choices to the end users. These two sub-principles refer to the guidance granularity provided by existing approaches.

| Approaches                   | Principles |                    |                              |                                 |                      |
|------------------------------|------------|--------------------|------------------------------|---------------------------------|----------------------|
|                              | Automation | Implicit knowledge | Focused results              |                                 | Balanced Computation |
|                              |            |                    | Configuration steps guidance | Configuration decision guidance |                      |
| [7] [140]                    | -          | -                  | +                            | +                               | manual               |
| [141] [142] [6]<br>[50] [76] | -          | -                  | +/-                          | -                               | manual               |
| [143]                        | +/-        | +                  | +/-                          | -                               | manual               |
| [48] [34] [72]               | -          | -                  | -                            | +                               | manual               |
| [144]                        | -          | -                  | -                            | +                               | manual               |
| [75] [5] [49]                | -          | -                  | +                            | +                               | manual               |
| [43] [44]<br>[46] [139] [41] | +          | -                  | -                            | +                               | +/-                  |

Table 2.2: Synthesis on process configuration support approaches that satisfy our principles

The results show that any of the approaches meet all of our identified principles.

The automation principle is met by only few works, namely the process-based approaches for merging [43, 44] and mining [41, 46, 139] configurable process models and the approach by Schunselaar et al. [143]. This latter approach exploits the implicit knowledge in the activities of the configurable process model to derive a domain-based abstract model. Different from them, in our approach we exploit the implicit knowledge hidden in existing business process models to infer useful information for process configuration. The results show also that all of the approaches that meet our two sub-principles, configuration steps guidance or configuration decision guidance, are manual.

In our approach, we address this research gap by proposing an automated approach for automatically deriving configuration guidance models that meet the four identified principles. Our configuration guidance models assist the users step by step during the configuration of the process by presenting them (i) the order in which the configuration steps are performed and (ii) the configuration decisions that are suitable taking into account the already chosen ones. We use the implicit knowledge on configuration decisions inferred from previously configured processes as well as event logs.

## 2.4 Conclusion

In this chapter we presented different approaches that support the design and configuration of configurable process models. We classified the design-based approaches into four categories: configurable business process modeling, business process variant retrieval, business process merging and business process mining. For the configuration approaches, we classified them into two categories: domain-based approaches and process-based approaches. We briefly introduced these approaches and identified their principles. We showed that most of the existing design approaches are automated but do not generate focused results and encounter the computational complexity problem. Regarding the existing configuration approaches, we showed that most of them are currently manual and that the few automated ones still miss guidance features to facilitate the process configuration. We also present the difference between current approaches and our approach.

We start presenting in detail our approach in the next chapters. In chapter 3, we present some formal definitions related to the process modeling languages used in this thesis. We also present the definition of event logs. In chapter 4, we elaborate on how we assist the design of configurable process models with configurable process fragments derived from existing business process models. In chapter 5, we present our approach for supporting the process configuration with configuration guidance models. In chapter 6, we show how we can use event logs to assist the design and configuration of configurable process models.

# Preliminaries

## Contents

---

|  |           |
|--|-----------|
| <b>3.1 Basic Notations</b> . . . . .                       | <b>57</b> |
| <b>3.2 Process Modeling Standards</b> . . . . .            | <b>58</b> |
| 3.2.1 Business Process Model and Notation (BPMN) . . . . . | 59        |
| 3.2.2 Configurable BPMN (C-BPMN) . . . . .                 | 60        |
| 3.2.3 Petri Nets . . . . .                                 | 62        |
| <b>3.3 Process Graphs</b> . . . . .                        | <b>64</b> |
| <b>3.4 Event Logs</b> . . . . .                            | <b>66</b> |

---

This chapter presents the preliminaries used in the remainder of this thesis. In Section 3.1, we introduce some basic mathematical notations. In Section 3.2, we present different graph based (configurable) process modeling standards used to illustrate this work. Then, in Section 3.4 we introduce event logs, the data input for process mining techniques.

## 3.1 Basic Notations

In this section, we introduce the basic notations, for sets, multisets, sequences and functions.

We define sets as follows:

**Definition 3.1.1** (Sets). *A set  $S$  is a possible infinite collection of elements. The elements in the set are listed between braces, e.g. ,  $S = \{a, b, c\}$ . The empty set is represented by  $\phi$ .  $|S|$  denotes the size of the set. For example,  $|S| = 3$ .  $\mathcal{P}(S)$  denotes the powerset of  $S$ , i.e. the set of all subsets of  $A$ , including the empty set and  $S$  itself. For example  $\mathcal{P}(S) = \{\phi, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$ .*

Three operations, *union*, *intersection* and *difference* are defined over sets. Let  $S_1 = \{a, b, c\}$  and  $S_2 = \{a, b, d\}$  be two sets. The *union* of  $S_1$  and  $S_2$  denoted as  $S_1 \cup S_2$  is a set containing all the elements of  $S_1$  and  $S_2$ . For example  $S_1 \cup S_2 = \{a, b, c, d\}$ . The *intersection* of  $S_1$  and  $S_2$  denoted as  $S_1 \cap S_2$  is a set containing the common elements between  $S_1$  and  $S_2$ . For example,  $S_1 \cap S_2 = \{a, b\}$ . The *difference* between

$S_1$  and  $S_2$  denoted as  $S_1 \setminus S_2$  is a set containing all the elements in  $S_1$  that are not in  $S_2$ . For example  $S_1 \setminus S_2 = \{c\}$ .

Multisets, also known as *bags* are defined as follows:

**Definition 3.1.2.** *A multiset  $M$  over a set  $S$  is a possible infinite collection of elements of  $S$ , where each element may appear more than once. The elements in the multiset are listed between square brackets. An example of a multiset  $M$  over  $S = \{a, b\}$  is  $M = [a, a, b]$  also denoted as  $M = [a^2, b]$ . We denote by  $M(a)$  the number of times the element  $a$  appears in  $M$ . For example,  $M(a) = 2$ ,  $M(b) = 1$  and  $M(c) = 0 \forall c \notin S$ . A set  $A \subseteq S$  can be viewed as a multiset where each element occurs once. The empty multiset is denoted as  $[\ ]$ . We denote by  $\mathbb{M}(S)$  the set of all multisets over  $S$ .*

The union, intersection and difference operations defined over sets are also applicable for multisets. For example  $[a, b] \cup [a, a, c] = [a, a, a, b, c]$ ;  $[a, b] \cap [a, a, c] = [a, a]$ ;  $[a, b] \setminus [a, a, c] = [b]$  and the size  $|[a, b]| = 2$ .

We define sequences and projection on sequences as follows:

**Definition 3.1.3** (Sequence, Projection). *Let  $S$  be a set. A sequence  $\sigma = \langle s_1, s_2, \dots, s_n \rangle \in S^*$  is an ordered list of elements  $s_i \in A, 1 \leq i \leq |S|$ . The empty sequence is denoted as  $\langle \rangle$ . The projection of  $\sigma$  on a subset  $S' \subseteq S$  denoted as  $\sigma_{\downarrow S'}$  is a subsequence of  $\sigma$  containing only the elements of  $S'$ . For example  $\langle a, a, b, d \rangle_{\downarrow \{a, d\}} = \langle a, a, d \rangle$ .*

We define functions as follows:

**Definition 3.1.4** (functions). *Let  $S_1$  and  $S_2$  be two non-empty sets. A function  $f$  from  $S_1$  to  $S_2$ , denoted as  $f : S_1 \rightarrow S_2$ , is a relation from  $S_1$  to  $S_2$ , where every element of  $S_1$  is associated with an element of  $S_2$ . We denote by  $\text{dom}(f)$ ,  $\text{cod}(f)$  and  $\text{Rng}(f)$  the domain, codomain and range of  $f$  respectively.*

## 3.2 Process Modeling Standards

Process models allow to explicitly represent the behavior of a business process according to its three perspectives: (i) control flow which describes the logical order between the process tasks (ii) resource flow which describes the physical objects and human performers required to accomplish a task and (iii) data flow which describes the data exchanged between the tasks [147]. In this thesis we mainly focus on the control flow perspective of the processes, and therefore the models are used to capture the ordering between the process tasks.

A wide range of graphical process modeling languages has been proposed over the last decade to represent a business process such as BPMN, EPC, YAWL, UML activity diagram, etc. Despite their variances in expressiveness and modeling notations, they all share the common concepts of tasks, events, gateways, artifacts and resources, as well as relations between them, such as transition flows [126]. Without loss of

generality, we select and use BPMN in our approach as it is one of the most popular business process modeling language. Therefore, in Section 3.2.1, we present the main elements of BPMN. Configurable BPMN which extends BPMN with configurable elements is then discussed in Section 3.2.2. In Section 3.3, we present some definitions related to an abstract representation of a (configurable) process model, referred to as (configurable) process graph, to which most of the process modeling languages can be mapped. And last, we present Petri nets, a formal process modeling notation used frequently in process mining.

### 3.2.1 Business Process Model and Notation (BPMN)

The Business Process Model and Notation (formerly know as Business Process Modeling Notation) (BPMN) was first released in 2004 by the Business Process Management Initiative (BPMI) [19]. BPMN is a standard for business process modeling that allows to create and document process models. It is considered as the *de facto* process modeling notation that is widely used in industry. In its latest versions, BPMN has been enhanced with executable semantics enabling the execution of the modeled process.

BPMN provides a rich set of elements to capture different perspectives of the business process at different levels of detail. The BPMN elements can be categorized into a *core set* which contains the basic elements to model a business process and an *extended set* which contains more specialized elements to specify more complex business scenarios [148]. Overall, BPMN defines 50 constructs grouped into four categories: *Flow objects*, *Connecting objects*, *Swimlanes* and *Artifacts*. *Flow Objects* allow to model the control flow perspective of a business process in terms of activities, events and gateways. An activity is the main element of a process model and describes the kind of work that must be done. It is graphically represented as a rectangle (see Figure 1.4 for an example of a process model in BPMN notation). An event is something that happens during the execution of a business process. There are three main types of events: Start, Intermediate and End events which may be specialized to *Message* (i.e. an event that can either send a message to a communication partner or react on the arrival of message), *Error* (i.e. an event that reacts on a canceled transaction), etc. An event is graphically represented with a circle. A gateway allows to model the splits and joins in the process model. Three main types are used to represent different behaviors in a business process: *AND* (parallel forking and synchronization), *XOR* (exclusive choice and merging) and *OR* (inclusive choice and merging). Although there exist other more specialized gateways in BPMN such as event-based gateway and complex gateways, they can all be mapped to one of the three main types *OR*, *AND* or *XOR*. The *flow objects* elements are connected through the *Sequence flow* element in *Connecting objects* category. They determine the order in which the activities will be performed in a process.

The *Artifacts*, *Swimlanes* and other elements in *Connecting objects* allow to model the resource and data perspectives in the process. For instance, the *Pools* and *Lanes*

elements in *Swimlanes* allow to group a set of activities that are executed by a specific role. The *Data object* element in *Artifacts* provides information about the data required by an activity. An *Association* element in *Connecting objects* is used to associate *Data objects* with a flow or connect them to activities.

### 3.2.2 Configurable BPMN (C-BPMN)

A configurable process model is a process model with configurable elements. A configurable element allows process analysts to make a *design-time choice* in addition to traditional *run-time choices* [34, 87]. It is graphically modeled with a thick line. For example, in the configurable process model in Figure 1.7, the gateway  $XOR_1^c$  is configurable while  $XOR_3$  is not. The main difference between these two elements is that  $XOR_3$  represents a simple *run-time* choice, i.e. the choice to execute either  $a_3$  or  $a_4$  is based on the run-time execution data. While  $XOR_1^c$  has a *design-time* choice in addition to the run-time choice. The design-time choice, referred to as *configuration choice*, allows to choose one design option from multiple ones. In our example, a choice can be taken to keep or remove one of the outgoing branches of  $XOR_1^c$  (i.e.  $XOR_2^c$  or  $XOR_3$ ) from the model. If one of them is removed,  $XOR_3^c$  is transformed to a simple sequence whose outgoing branch is executed at run-time. If both of them are kept,  $XOR_3^c$  is transformed to a normal *XOR* whose decision is made at run-time.

We define a configurable BPMN notation (C-BPMN for short) in which the control flow elements (i.e. activities, gateways and events) can be configurable. Configurable activities and gateways have been discussed in [34, 149]. In this work, we introduce the new concept of *configurable events*.

A configurable activity can be included (i.e. configured to *ON*), excluded (i.e. configured to *OFF*) or optionally excluded (i.e. configured to *OPT*) from the process model (Figure 3.1). The latter can be seen as a combination of an *ON* configuration (i.e. the activity is included in the model) in an exclusive choice. The exclusive choice allows to make a run-time decision to execute or skip the activity. Thus, in the remainder of this thesis we omit such configuration.

A configurable gateway has a generic behavior which is restricted by configuration. A gateway can be configured by (1) changing its type while preserving its behavior and/or (2) restricting its incoming (respectively outgoing) branches in case of a join (respectively split). Table 3.1 illustrates the configuration constraints of gateways' types [34]. A configurable gateway is denoted by  $[type]^c$ . Each row in the table corresponds to a configurable gateway which can be configured to one or more of the gateways presented in columns. The last column (i.e. *Seq*) corresponds to a *Sequence flow*. For example, the configurable OR ( $OR^c$ ) can be configured to any gateway's type while a configurable AND ( $AND^c$ ) can be only configured to an (*AND*). Please note that a gateway with a join behavior cannot be configured to a gateway with a split behavior (and vice versa). These configuration constraints are formalized through the partial order  $\preceq_g$  that specifies which concrete gateway may be used for

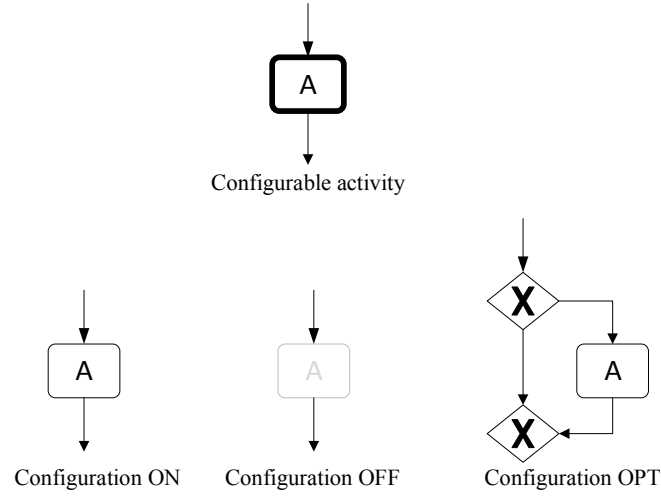


Figure 3.1: A configurable activity and its possible configuration choices

|                        | <i>OR</i> | <i>AND</i> | <i>XOR</i> | <i>Seq</i> |
|------------------------|-----------|------------|------------|------------|
| <i>OR<sup>c</sup></i>  | ✓         | ✓          | ✓          | ✓          |
| <i>AND<sup>c</sup></i> |           | ✓          |            |            |
| <i>XOR<sup>c</sup></i> |           |            | ✓          | ✓          |

Table 3.1: Configuration constraints of configurable gateways

a given configurable gateway [34].

**Definition 3.2.1** (Partial order  $\preceq_g$ ). *Let  $g^c$  be a configurable gateway and  $g$  be a normal gateway or a sequence flow (i.e. “Seq”).  $g \preceq_g g^c$  iff  $(g^c = OR^c) \vee (g^c = XOR^c \wedge g = Seq) \vee (g^c = g)$ .*

For example, in the configurable process model in Figure 1.7, the configurable gateway  $XOR_1^c$  can be configured to an *XOR* with the same outgoing branches or a *Seq* with the restriction of one of its outgoing branches.

A configurable event can be included (i.e. *enable*), excluded (i.e. *disable*) or change its type to one of the BPMN events’ types (e.g. *Message*, *Error*, etc.) (Figure 3.2). The latter configuration is allowed when the corresponding configurable event has an *abstract* type, i.e. a configurable event without a type. We use the *None* event (i.e event without label) from BPMN to denote a configurable abstract event. These configuration constraints are formalized through the partial order  $\preceq_e$  that specifies which concrete event may be used for a given configurable event.

**Definition 3.2.2** (partial order  $\preceq_e$ ). *Let  $e^c$  be a configurable event and  $e$  be a normal event.  $e \preceq_e e^c$  iff  $(e^c = None) \vee (e = e^c)$ .*



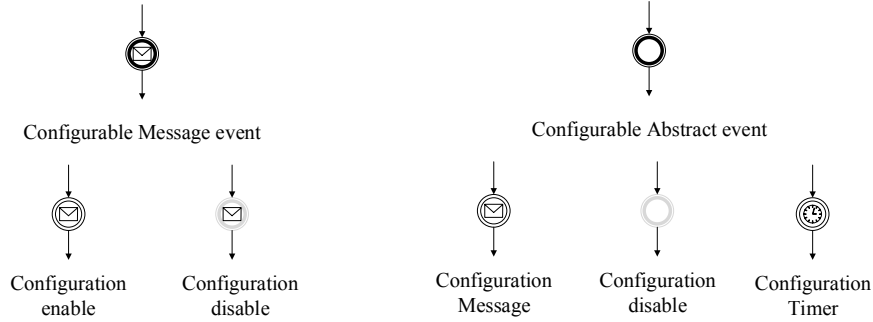


Figure 3.2: Configurable events and their possible configuration choices

For example, in the configurable process in Figure 1.7, the event *cancel request* is configurable. It can be configured to *enable* (i.e. it remains as a *cancel request* event) or to *disable* (i.e. excluded from the model).

### 3.2.3 Petri Nets

Petri nets [17] are formal models for describing concurrent distributed systems. They are widely used for modeling, analyzing and verifying business processes [150, 151] as they have a mathematical foundation of their execution semantics. A simple graphical notation is also available to support the modeling with Petri nets. A Petri net graph consists of places (represented with circles), transitions (represented with rectangles) and arrows connecting them in a bipartite manner. A transition is equivalent to a task in high level process modeling languages (e.g. activity in BPMN). An example of a Petri net for the flight booking process variant in Figure 1.4b is illustrated in Figure 3.3. The formal definition of a Petri net is given in Definition 3.2.3.

**Definition 3.2.3** (Petri net). *A Petri net is a tuple  $PN = (P, T, F)$  where  $P$  is the set of places,  $T$  is the set of transitions such that  $P \cap T = \emptyset$  and  $F = (P \times T) \cup (T \times P)$  is the set of arcs connecting the places and transitions referred to as flow transitions.*

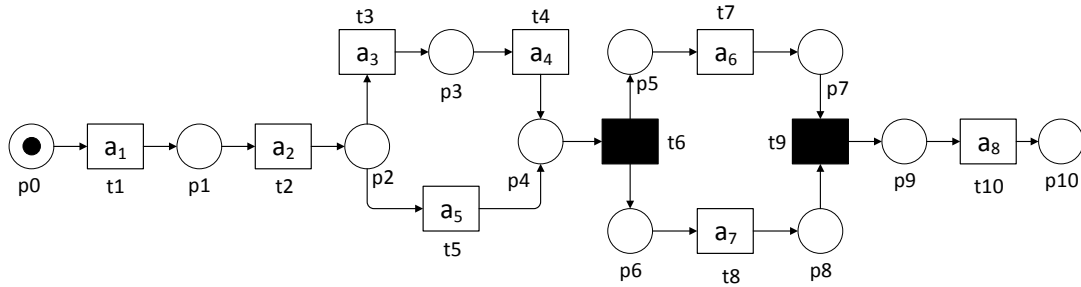


Figure 3.3: An example of a Petri net

A labeled Petri net is a Petri net with a labeling function that assigns labels to transitions. For example, in the Petri net in Figure 3.3,  $a_1$  is the label of the transition  $t_1$ . Some transitions in a Petri net do not have labels and are represented by a filled black rectangle. They refer to *silent transitions* or  $\tau$ -transition and they only distribute the tokens. In the remainder of this thesis we refer to a labeled Petri net by “Petri net”.

The sets of input and output transitions of a place  $p \in P$  are denoted by  $\bullet p$  and  $p\bullet$  respectively. Similarly, the sets of input and output places of a transition  $t \in T$  are denoted by  $\bullet t$  and  $t\bullet$  respectively. We denote by  $p_i$  such that  $\bullet p_i = \phi$  the start place of a Petri net and  $p_f$  such that  $p_f\bullet = \phi$  its final place.

The formal execution semantics of a Petri net are defined in terms of its *markings* and *fired transitions*. A marking represents the execution state of a Petri net in terms of consumed and produced *tokens*. Tokens represent the pre- and post-states of an executed transition. A transition consumes a token, represented by a black dot, from each of its input places to be executed (i.e. fired). Once executed, it produces tokens in each of its output places. As a consequence, a new marking of the Petri net is obtained. The marking of a Petri-net and the enabled and fired transition are formally given in Definition 3.2.4.

**Definition 3.2.4** (Marked Petri net, enabled and fired transition). (*Adapted from [152]*) A marked Petri net is denoted as  $P_M = (P, M)$  where  $P$  is a Petri net and  $M \rightarrow \mathbb{N}$  is a function that assigns tokens to the Petri net places. A transition  $t \in T$  is enabled in the marking  $M$ , denoted as  $M[t\rangle$  iff  $\forall p \in \bullet t : M(p) \geq 1$ . An enabled transition can be fired. The firing of an enabled transition  $t$  changes the marking of a Petri net  $P_M$  to  $P_{M'}$  such that  $P_{M'}$  is defined as:  $\forall p \in \bullet t : M(p) = M(p) - 1 \wedge \forall p \in t\bullet : M(p) = M(p) + 1$ . The firing of  $t$  and the transition to the new marking is denoted as  $M[t\rangle M'$ .

We denote by  $P_{M_i}$  the initial marked Petri net, i.e. the marking where the initial place  $p_i$  is the only place that contains a token. Similarly, the final marked Petri net  $P_{M_f}$  denotes the marking where the final place is the only place that contains tokens. For example, the Petri net in Figure 3.3 is an initial marked Petri net as the start place  $p_0$  is the only one that has a token. In this marking, the transition  $a_1$  is enabled and can be fired. Therefore, it removes the token from  $p_0$  and produces a token in each of its output places, i.e. in  $p_1$ .

A transition sequence represents the firing of a sequence of transitions leading from a marked Petri net  $P_M$  to another marked Petri net  $P_{M'}$ . It is complete if  $P_M$  is the initial marked Petri net and  $P_{M'}$  is the final marked Petri net.

**Definition 3.2.5** ((Complete) Transition sequence). Let  $P_M = (P, M)$  be a Petri net in the marking  $M$ . A transition sequence  $\sigma = \langle t_1, t_2, \dots, t_n \rangle \in T^*$  denotes that there exists a sequence of fired transitions  $t_1, t_2, \dots, t_n$  in  $P$  that leads from  $P_M$  to  $P_{M'}$  such that  $M[t_1\rangle M_1 \wedge M_n[t_n\rangle M' \wedge \forall 1 < i < n : M_i[t_i\rangle M_{i+1}$  where  $T^*$  represents all possible

sequences of transitions in  $T$ .  $\sigma$  is called a complete transition sequence if  $P_M = P_{M_i}$  and  $P_{M'} = P_{M_f}$ .

For example, in the Petri net in Figure 3.3, the sequence  $\sigma = \langle t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_{10} \rangle$  is a complete transition sequence as it leads from the initial marking (where the token is in  $p_0$ ) to the final marking (where the token is in  $p_{10}$ ).

### 3.3 Process Graphs

As the structure of a business process model can be mapped to a graph, we choose graph theory to present a business process. A process model can be represented as a directed graph called process graph that captures the types of nodes and edges as attributes. This representation is derived from the common constructs of graphical process modeling notations and thus can be generalized for most of them (e.g. BPMN and EPC) [153].

**Definition 3.3.1** (Process graph). *A process graph  $P = (id, N, E, T, L, I)$  is a labeled directed graph where:*

- *id is its unique identifier;*
- *$N$  is the set of nodes. In case of BPMN,  $N$  is the set of activities, events and gateways;*
- *$E \subseteq N \times N$  is the set of edges connecting two nodes. We denote by  $source_e$  and  $target_e$  the source and target nodes of an edge  $e \in E$ ;*
- *$T : N \rightarrow \mathcal{T}$  where  $\mathcal{T}$  is the set of the modeling languages metamodel elements' types and  $T$  is a function that assigns for each node  $n \in N$  a type  $t \in \mathcal{T}$ . In case of BPMN,  $\mathcal{T} = \{\text{activity, Start event, End event, Intermediate event, gateway}\}$ ;*
- *$L : N \rightarrow \mathcal{L}$  where  $\mathcal{L}$  is the universe of elements' labels and  $L$  is a function that assigns for each node  $n \in N$  a label  $l \in \mathcal{L}$ . In case of BPMN, if  $T(n) \in \{\text{event, activity}\}$ , then  $L(n)$  is its name, and if  $T(n) = \text{gateway}$  then  $L(n) \in \{\text{OR, XOR, AND}\}$ .*
- *$I : N \rightarrow \mathbb{N}$  is a function that assigns for each node  $n \in N$  a unique identifier  $id \in \mathbb{N}$ .*

An example of a process graph representing the process model in Figure 1.4b is depicted in Figure 3.4. The nodes are attached to a text annotation including their types, labels and identifiers.

Let  $P = (id, N, E, T, L, I)$  be a process graph. A path from a node  $n_x$  to a node  $n_y$  in the process graph  $P$  is a sequence of nodes leading from  $n_x$  to  $n_y$ . Since there may exist multiple path between two nodes, we define the shortest path that has the minimal number of nodes. The definitions of path and shortest path are given in Definition 3.3.2.

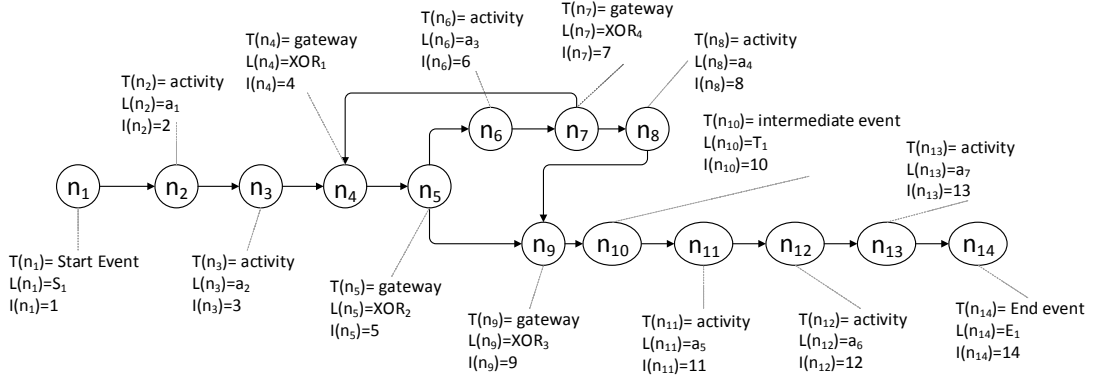


Figure 3.4: An example of a process graph

**Definition 3.3.2** (Path  $P$ , Shortest path  $SP$ ). A path from a node  $n_x \in N$  to a node  $n_y \in N$  is denoted as  $\mathcal{P}_{n_x}^{n_y} = \langle n_1, n_2, \dots, n_n \rangle$  where  $n_1 = n_x$ ,  $n_n = n_y$   $\wedge \forall 1 \leq i \leq n : (n_i, n_{i+1}) \in E$ . The shortest path between  $n_x$  and  $n_y$  denoted as  $SP_{n_x}^{n_y}$  is the path having the minimal number of nodes, i.e.  $SP_{n_x}^{n_y} = \mathcal{P}_{n_x}^{n_y}$  such that  $\nexists \mathcal{P}'_{n_x}^{n_y} : |\mathcal{P}'_{n_x}^{n_y}| < |\mathcal{P}_{n_x}^{n_y}|$ .

For example, in the process graph in Figure 3.4, two paths exist from  $n_3$  to  $n_9$ :  $\mathcal{P}_{n_3}^{n_9} = \langle n_3, n_4, n_5, n_6, n_8, n_9 \rangle$  and  $\mathcal{P}'_{n_3}^{n_9} = \langle n_3, n_4, n_7, n_8, n_9 \rangle$ ;  $|\mathcal{P}'| < |\mathcal{P}|$ , thus  $\mathcal{P}'$  is the shortest path, i.e.  $SP = \mathcal{P}'$ .

The preset and postset of  $n \in N$  is the set of elements in its incoming and outgoing branches respectively (Definition 3.3.3).

**Definition 3.3.3** (preset  $\bullet n$ , postset  $n \bullet$ ). The preset of an element  $n \in N$  denoted as  $\bullet n$  is defined as  $\bullet n = \{n_x \in N : (n_x, n) \in E\}$ . The postset of  $n$  denoted as  $n \bullet$  is defined as  $n \bullet = \{n_x \in N : (n, n_x) \in E\}$ .

A gateway  $g$  is a **split** if  $|g \bullet| > 1$ ; it is a **join** if  $|\bullet g| > 1$ .

A configurable process graph is a process graph in which the nodes can be either configurable or not. A configurable node has a set of configuration choices. In case of BPMN, the configurable nodes can be activities, events and gateways and their configuration choices are as presented in Section 3.2.2. Formally, the configuration of a node is given in Definition 3.3.4.

**Definition 3.3.4** (Configuration  $Conf$ ). A configuration of a configurable node  $n^c$  denoted as  $Conf_{n^c}$  is defined as following:

- if  $T(n^c) = \text{activity}$  then  $Conf_{n^c} \in \{ON, OFF\}$ ;
- if  $T(n^c) = \text{event}$  then  $Conf_{n^c} \in \{\text{enable}, \text{disable}, e\}$  such that  $e \preceq_e n^c$ .
- if  $T(n^c) = \text{gateway}$  then  $Conf_{n^c} \in \{(c', s) : (c', s) \in CT \times \mathcal{P}(S)\}$  where:

- $CT = \{OR, AND, XOR, Seq\}$  and  $c' \preceq_g n^c$ ,
- $S = \bullet n^c$  (respectively  $S = n^c \bullet$ ) in case  $n^c$  is a join (respectively split) gateway.

We denote by  $\mathbb{C}_{n^c}$  the set of all configurations of the configurable element  $n^c$ . For example, in Figure 1.7, the set of configurations of the configurable gateway  $XOR_1^c$  is  $\mathbb{C}_{XOR_1^c} = \{(XOR, \{XOR_2^c, XOR_3^c\}), (Seq, \{XOR_2^c\}), (Seq, \{XOR_3^c\})\}$ .

**Definition 3.3.5** (Configurable process graph). *A configurable process graph is denoted as  $P^c = (id, N, E, T, L, I, B, \mathbb{C}^*)$  where:*

- $id, N, E, T, L, I$  are as specified in Definition 3.3.1;
- $B : N \rightarrow \{true, false\}$  is a boolean function returning true for configurable nodes;
- $\mathbb{C}^* = \{\mathbb{C}_n : n \in N \wedge B(n) = true\}$  is the set of configurations of all configurable nodes.

### 3.4 Event Logs

Event logs are commonly used to represent the processes execution history recorded by information systems. They are used by process mining techniques to discover process models, to check the conformance of a-priori process models, to detect execution errors or to observe social behaviors. An example of an event log for a process that handles fines [1] is illustrated in Table 3.2.

An event log stores the execution history of *one process*. A log *case* corresponds to one *process instance execution*. For example, the log in Table 3.2 records 4 executions (Case ID = 1, 2, 3, 4) of the process. Each row corresponds to one executed *event* in a specific *case* and the events are ordered chronologically. Each event is associated with two mandatory attributes: (i) *Case ID* which refers to the process instance id of the event and (ii) *Task Type* which refers to the activity name executed in the event. Additional attributes can be related to an event such as: (i) *Event Type* which refers to the event state such as *started, paused, resumed, completed*<sup>1</sup>, etc., (ii) *Resource* which refers to the resource name that initiates the event, (iii) *Date* and *Time* which refer to the date and the time at which the event has been executed. For example, in Table 3.2, the first event belongs to the process instance with id=1. The activity “a” (File Fine) is executed by the resource “Anne” and the event has been completed.

A record, called *trace*, is a sequence of a valid execution of *one process instance*. For example, in Table 3.2, for the Case ID = 1, the sequence of events<sup>2</sup>  $\langle a, b, c, d \rangle$  is

<sup>1</sup>In the remainder of this thesis, when we refer to an event, we refer to its completed state

<sup>2</sup>In this thesis, we mainly focus on the control-flow perspective of a process and therefore, we refer to an event by its task name attribute

| Case ID | Task Name           | Event Type | Resource | Date       | Time     | ... |
|---------|---------------------|------------|----------|------------|----------|-----|
| 1       | File Fine (a)       | Completed  | Anne     | 20-07-2004 | 14:00:00 | ... |
| 2       | File Fine (a)       | Completed  | Anne     | 20-07-2004 | 15:00:00 | ... |
| 1       | Send Bill (b)       | Completed  | system   | 20-07-2004 | 15:05:00 | ... |
| 2       | Send Bill (b)       | Completed  | system   | 20-07-2004 | 15:07:00 | ... |
| 3       | File Fine (a)       | Completed  | Anne     | 21-07-2004 | 10:00:00 | ... |
| 3       | Send Bill (b)       | Completed  | system   | 21-07-2004 | 14:00:00 | ... |
| 4       | File Fine (a)       | Completed  | Anne     | 22-07-2004 | 11:00:00 | ... |
| 4       | Send Bill (b)       | Completed  | system   | 22-07-2004 | 11:10:00 | ... |
| 1       | Process Payment (c) | Completed  | system   | 24-07-2004 | 15:05:00 | ... |
| 1       | Close Case (d)      | Completed  | system   | 24-07-2004 | 15:06:00 | ... |
| 2       | Send Reminder (e)   | Completed  | Mary     | 20-08-2004 | 10:00:00 | ... |
| 3       | Send Reminder (e)   | Completed  | John     | 21-08-2004 | 10:00:00 | ... |
| 2       | Process Payment (c) | Completed  | system   | 22-08-2004 | 09:05:00 | ... |
| 2       | Close Case (d)      | Completed  | system   | 22-08-2004 | 09:06:00 | ... |
| 4       | Send Reminder (e)   | Completed  | John     | 22-08-2004 | 15:10:00 | ... |
| 4       | Send Reminder (e)   | Completed  | Mary     | 22-08-2004 | 17:10:00 | ... |
| 4       | Process Payment (c) | Completed  | system   | 29-08-2004 | 14:01:00 | ... |
| 4       | Close Case (d)      | Completed  | system   | 29-08-2004 | 17:30:00 | ... |
| 3       | Send Reminder (e)   | Completed  | John     | 21-09-2004 | 10:00:00 | ... |
| 3       | Send Reminder (e)   | Completed  | John     | 21-10-2004 | 10:00:00 | ... |
| 3       | Process Payment (c) | Completed  | system   | 25-10-2004 | 14:00:00 | ... |
| 3       | Close Case (d)      | Completed  | system   | 25-10-2004 | 14:01:00 | ... |

Table 3.2: Example of event log for a process that handle fines [1]

executed. A trace includes the execution orders of the activities involved in a process instance. For example in the aforementioned trace,  $b$  is executed after  $a$  and before  $c$ . A log is composed by traces that may be repeated in the log. Therefore, a log is a multiset of traces (Definition 3.4.1).

**Definition 3.4.1** (Trace, Event log). *Let  $A \subseteq \mathcal{U}_A$  be a set of activities in some universe of activities  $\mathcal{U}_A$ . A trace  $\sigma \in A^*$  is a sequence of activities. An event log  $L \in \mathbb{M}(A^*)$  is a multiset of traces.*

For example, the event log in Table 3.2 can be represented as: [ $\langle a, b, c, d \rangle$ ,  $\langle a, b, e, c, d \rangle$ ,  $\langle a, b, e, e, e, c, d \rangle$ ,  $\langle a, b, e, e, c, d \rangle$ ]. Based on the activities execution orders in traces, four ordering relations can be derived from an event log:  $>_L$ ,  $\rightarrow_L$ ,  $\parallel_L$  and  $\#_L$  [127].

**Definition 3.4.2** (Log-based ordering relations [127]). *Let  $L \in \mathbb{M}(A^*)$  be an event log over  $A$ . Let  $a, b \in A$ . The four ordering relations: precedence ( $>_L$ ), causality ( $\rightarrow_L$ ), parallelism ( $\parallel_L$ ) and choice ( $\#_L$ ) are defined as follows:*

- *Precedence:  $a >_L b$  iff  $\exists \sigma = \langle a_1, a_2, \dots, a_n \rangle \in L$  and  $1 \leq i \leq n - 1$  such that  $a_i = a \wedge a_{i+1} = b$ .*
- *Causality:  $a \rightarrow_L b$  iff  $a >_L b \wedge b \not\#_L a$ .*

- *Parallelism*:  $a \parallel_L b$  iff  $a >_L \wedge b >_L a$ .
- *Choice*:  $a \#_L b$  iff  $a \not>_L b \wedge b \not>_L a$ .

For example, the ordering relations that can be derived from the event log in Table 3.2 are: precedence ( $a >_L b$ ;  $b >_L c$ ;  $c >_L d$ ;  $b >_L e$ ;  $e >_L c$ ;  $c >_L d$ ;  $e >_L e$ ) and causality ( $a >_L b$ ;  $b >_L c$ ;  $c >_L d$ ;  $b >_L e$ ;  $e >_L c$ ;  $c >_L d$ ).

Based on the four log-based ordering relations, a process model describing the behavior observed in the event log can be discovered using an existing mining algorithm (e.g. [127,154]). For example, the process model in Figure 3.5 is mined from the event log in Table 3.2 and is represented using Petri-nets.

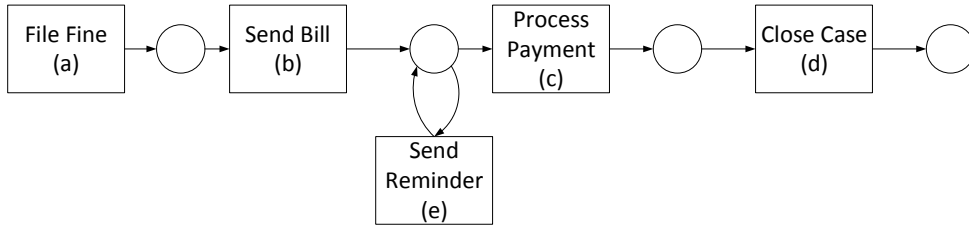


Figure 3.5: The process discovered from the event log in Table 3.2 [1]

Event data can be recorded in event logs using the eXtensible Event Stream (XES) [155] standard which is an XML-based standard for storing event logs on disk and the successor of the MXML standard [156]. In addition to storing event logs, XES provides a standard format for interchanging event log data between tools and application domains.

# Assisting Process Design with Configurable Process Fragments

## Contents

---

|            |   |           |
|------------|---|-----------|
| <b>4.1</b> | <b>Introduction</b>   | <b>69</b> |
| <b>4.2</b> | <b>Aggregated Neighborhood Context Graph</b>                | <b>70</b> |
| <b>4.3</b> | <b>Deriving Aggregated Neighborhood Context Graphs</b>      | <b>75</b> |
| 4.3.1      | Extracting Neighborhood Context Graphs                      | 76        |
| 4.3.2      | Clustering Neighborhood Context Graphs                      | 78        |
| 4.3.3      | Merging Neighborhood Context Graphs                         | 80        |
| 4.3.3.1    | Merging vertices/edges                                      | 81        |
| 4.3.3.2    | Defining edges' labels                                      | 83        |
| 4.3.3.3    | Handling edges sharing their source or target               | 89        |
| 4.3.4      | Behavior Preservation of the Merging Algorithm              | 89        |
| 4.3.5      | Computational Complexity                                    | 90        |
| <b>4.4</b> | <b>From Aggregated Neighborhood Context Graph to C-BPMN</b> | <b>91</b> |
| <b>4.5</b> | <b>Conclusion</b>   | <b>92</b> |

---

## 4.1 Introduction

This chapter presents our approach for assisting the design of configurable process models with *configurable process fragments*. For this purpose, we adapt and reuse the *neighborhood context graph* [2] as a process fragment model which has been developed in a previous work. Basically, the neighborhood context graph is defined as a process fragment around an activity that consists of relations between the associated activity and its neighbors. The neighbor context presents the behavior of the associated activity within the process.

We start the chapter by introducing our notion of *aggregated neighborhood context graph* which consists of an aggregated representation of multiple neighborhood context graphs (Section 4.2).



Next, we present an algorithm that, given an activity selected by the process provider in an ongoing designed process, returns a set of aggregated neighborhood context graphs that include the desired activity and all its possible relations to its activities' neighbors in a repository of process graphs (Section 4.3).

An algorithm is then proposed to transform the aggregated neighborhood context graphs to *configurable process fragments* in the C-BPMN notation (Section 4.4). The resulted configurable fragments are presented to the process provider who can choose to integrate one of them in the ongoing designed process. The process provider can also choose to re-merge two or multiple of them in order to derive richer configurable fragments with more functionalities.

We reuse our motivating example presented in Section 1.3 to illustrate our approach. We assume that a process provider is designing a configurable “travel-booking” process. He sketches out the business process as illustrated in Figure 4.1. At this stage, he needs an assistance to complete the missing parts represented by the symbol “?”. He selects the activity “Select a flight” and asks for configurable fragments that are appropriate to fill-in the first missing part.

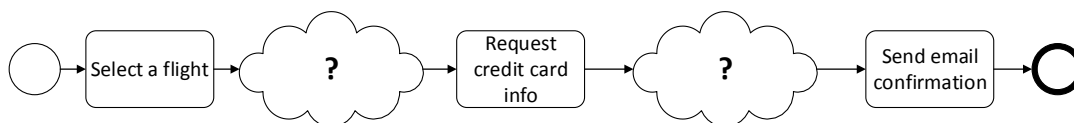


Figure 4.1:  $BP^c$ : An incomplete configurable reference travel booking process

We also assume that there exists a repository of process graphs from which we show the two variants in Figure 4.2 (taken from our motivating example in Section 1.3). In the remainder of this thesis we present the process graphs in the BPMN notation to illustrate our examples. We demonstrate our approach for proposing configurable process fragments for the selected activity “Select a flight”.

The work in this chapter was published in conference proceedings [157] and peer-reviewed journal [158].

## 4.2 Aggregated Neighborhood Context Graph

A process fragment is a process building block containing local knowledge [159]. In other word, a business process graph consists of a set of fragments, which composed together, accomplish a global goal. The *neighborhood context graph* is a process fragment model that represents the behavior of an activity in terms of its relations to its activities' neighbors. It is presented as a graph in which the associated activity is located at the center and its activities' neighbors are located around it in virtual layers. In this work, we define an *aggregated neighborhood context graph* that consists of a concise representation of multiple neighborhood context graphs. We present in the following some definitions that are used to formally define an aggregated neighborhood

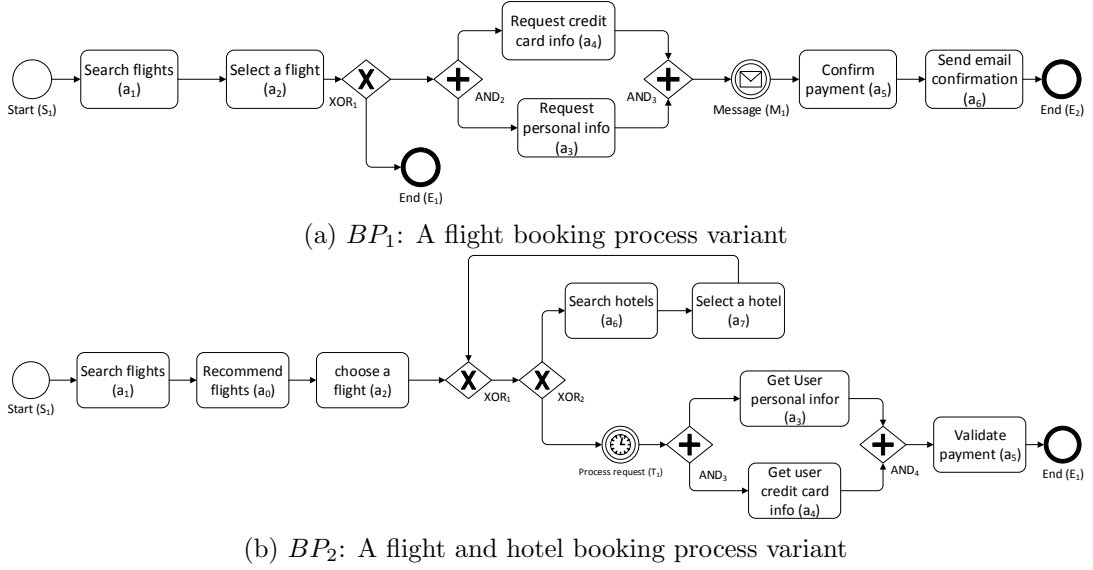


Figure 4.2: Two process variants of a travel booking process

context graph.

Let  $P = (id, N, E, T, L, I)$  be a process graph. We define a *connection flow* between two nodes  $n_x$  and  $n_y$  in  $P$  as the shortest path  $\mathcal{SP}_{n_x}^{n_y}$  (Definition 3.3.2) relating  $n_x$  to  $n_y$  such that the elements in the path are either *sequence flows*, *gateways* or *intermediate events* (we refer to these elements by *connection elements* and to the activities, start and end events by *vertices*). A *connection flow path* exists between two vertices  $v_u$  and  $v_z$  if there exists a sequence of connection flows between  $v_u$  and  $v_z$ . The shortest connection flow path is the connection flow path that has the minimal number of nodes.

**Definition 4.2.1** (Connection element, vertex). A *connection element* denoted as  $\tilde{c}$  can be either a *sequence flow* (sequence for short), a *gateway* or an *intermediate event*. We denote by  $\tilde{\mathcal{C}}$  the set of connection elements in  $P$ . A *vertex* denoted as  $v$  can be either an *activity* or a *start* or an *ened event*. We denote by  $\mathcal{V}$  the set of vertices in  $P$ .

**Definition 4.2.2** (Connection flow). A *connection flow* between two vertices  $v_x, v_y \in \mathcal{V}$  is denoted as  $F_{v_x}^{v_y}$  such that:

- if  $|\mathcal{SP}_{v_x}^{v_y}| = 2$  (i.e.  $(v_x, v_y) \in E$ ), then  $F_{v_x}^{v_y} = \langle \text{sequence} \rangle$ ;
- else  $F_{v_x}^{v_y} = \langle \mathcal{SP}_{v_x}^{v_y}[2], \dots, \mathcal{SP}_{v_x}^{v_y}[n-1] \rangle$  where  $|\mathcal{SP}_{v_x}^{v_y}| = n \wedge \forall 2 \leq i \leq n-1, \mathcal{SP}_{v_x}^{v_y}[i] \in \tilde{\mathcal{C}}$ .

For example, in Figure 4.2a, the connection flow between the activities  $a_1$  and  $a_2$  is  $F_{a_1}^{a_2} = \langle \text{sequence} \rangle$ ; the connection flow between  $a_2$  and  $a_3$  is  $F_{a_2}^{a_3} = \langle XOR_1, AND_2 \rangle$ .

**Definition 4.2.3** (Connection flow path, shortest connection flow path). A connection flow path between two vertices  $v_u$  and  $v_z$  is denoted as  ${}_P F_{v_u}^{v_z} = \langle F_1, F_2, \dots, F_n \rangle$  such that  $\exists v_1, \dots, v_n \in \mathcal{V} : v_1 = v_u, v_n = v_z \wedge \forall 1 \leq i \leq n : F_i \in \{F_{v_i}^{v_{i+1}}, F_{v_{i+1}}^{v_i}\}$ . The shortest connection flow path is denoted as  $\mathcal{S}F_{v_u}^{v_z} = {}_P F_{v_u}^{v_z}$  such that  $\nexists {}_P F_{v_u}^{v_z} : |{}_P F_{v_u}^{v_z}| < |{}_P F_{v_u}^{v_z}|$ .

According to Definition 4.2.3, a connection flow path is undirected which means that connection flows between the vertices can be oriented in different directions. For example, in Figure 4.2b, a connection flow path that exists between  $a_2$  and  $a_7$  is  ${}_P F_{a_2}^{a_7} = \langle F_{a_2}^{a_6}, F_{a_6}^{a_7} \rangle$ ; Two possible connection flow paths that exist between  $a_3$  and  $a_4$  are:  ${}_P F_{a_3}^{a_4} = \langle F_{a_2}^{a_3}, F_{a_2}^{a_4} \rangle$  and  ${}_P F_{a_3}^{a_4} = \langle F_{a_3}^{a_5}, F_{a_4}^{a_5} \rangle$ ;  $\mathcal{S}F_{a_3}^{a_4} = {}_P F_{a_3}^{a_4} = {}_P F_{a_3}^{a_4}$ .

We define the  $k^{\text{th}}$ -layer neighbor of a vertex  $v_x$  as the vertex  $v_y$  having a shortest connection flow path to  $v_x$  of length  $k$  (Definition 4.2.4). We also attribute to the connection flows between the vertices located on the same or adjacent layers a *zone number* based on the associated vertices' layer numbers (Definition 4.2.5).

**Definition 4.2.4** ( $k^{\text{th}}$ -layer neighbor).  $v_y \in \mathcal{V}$  is a  $k^{\text{th}}$ -layer neighbor of  $v_x \in \mathcal{V}$  iff  $\mathcal{S}F_{v_x}^{v_y} = k$ . We denote by  $\mathcal{L}_{v_x}^k$  as the set of  $k^{\text{th}}$ -layer neighbors of  $v_x$ .

**Definition 4.2.5** ( $k^{\text{th}}$ -zone flow).  $F_{v_i}^{v_j}$  is a  $k^{\text{th}}$ -zone flow of  $v_x$  iff  $(v_i, v_j \in \mathcal{L}_{v_x}^{k-1}) \vee (v_i \in \mathcal{L}_{v_x}^{k-1} \wedge v_j \in \mathcal{L}_{v_x}^k) \vee (v_i \in \mathcal{L}_{v_x}^k \wedge v_j \in \mathcal{V}_{v_x}^{k-1})$ . We denote by  $\mathcal{Z}_{v_x}^k$  as the set of  $k^{\text{th}}$ -zone flows of  $v_x$ .

The zone number  $k$  of  $F_{v_i}^{v_j}$  is calculated as follows: if  $v_i$  and  $v_j$  are located on two adjacent layers,  $k$  is equal to the greater layer number; if  $v_i$  and  $v_j$  are located on the same layer,  $k$  is equal to their upper layer number. For example, in Figure 4.2b,  $\mathcal{L}_{a_2}^1 = \{a_0, a_6, a_3, a_4\}$  and  $\mathcal{L}_{a_2}^2 = \{a_1, a_7, a_5\}$ .  $a_6 \in \mathcal{L}_{a_2}^1$  and  $a_7 \in \mathcal{L}_{a_2}^2$ , therefore  $F_{a_6}^{a_7} = \langle \text{sequence} \rangle$  is a 2<sup>nd</sup>-zone flow.  $\mathcal{Z}_{a_2}^2 = \{F_{a_1}^{a_0}, F_{a_6}^{a_7}, F_{a_7}^{a_3}, F_{a_7}^{a_4}, F_{a_7}^{a_5}\}$ .

Having presented the concepts of layer neighbors and zone flows, we define the neighborhood context graph of a vertex  $v \in \mathcal{V}$  of size  $k$  as a graph that consists of  $v$ , the vertices in 1<sup>st</sup>- to  $k^{\text{th}}$ -layer neighbor and the edges in 1<sup>st</sup>- to  $k^{\text{th}}$ -zone flow labeled with connection flows.

**Definition 4.2.6** (Neighborhood Context Graph). The neighborhood context graph of a vertex  $v_x \in \mathcal{V}$  denoted by  $G_{v_x}^k = (id_G, V_{v_x}, E_{v_x}, L_{v_x}, I)$  is a labeled directed graph created from  $P = (id_P, N, E, T, L, I)$  where:

- $k$  is the number of layers in  $G_{v_x}^k$ ;
- $id_G = id_P$ ;
- $V_{v_x} \subseteq \mathcal{V}$  such that  $\forall v_i \in V_{v_x}, v_i \in \mathcal{L}_{v_x}^j, 1 \leq j \leq k$  is the set of vertices;
- $E_{v_x} \subseteq V_{v_x} \times V_{v_x}$  is the set of edges such that  $\forall (v_i, v_j) \in E_{v_x}, \exists F_{v_i}^{v_j} : F_{v_i}^{v_j} \in \mathcal{Z}_{v_x}^l, 1 \leq l \leq k$ ;

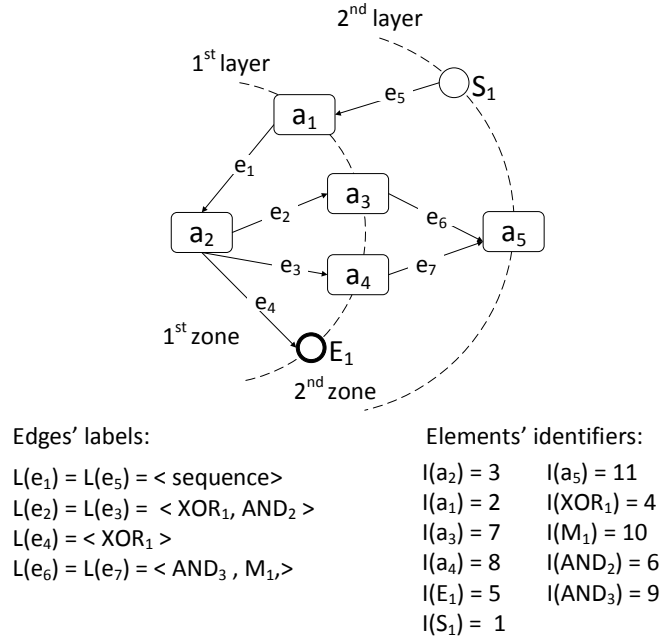
- $L_{v_x} : E_{v_x} \rightarrow F^*$  is a function that assigns labels to the edges such that  $L_{v_x}((v_i, v_j)) = F_{v_i}^{v_j}$ ;
- $I$  : is as defined for  $P$  (Definition 3.3.1).

For example, the neighborhood context graphs of the activities  $a_2$  and  $a_2$  in the process graphs in Figure 4.2a and 4.2b within two layers (i.e.  $k = 2$ ) are depicted in Figure 4.3a and 4.3b respectively. The neighborhood context graphs are presented as circles centered with their associated activities. The activities neighbors are presented on virtual layers around the center and the edges are located in zones between the layers.

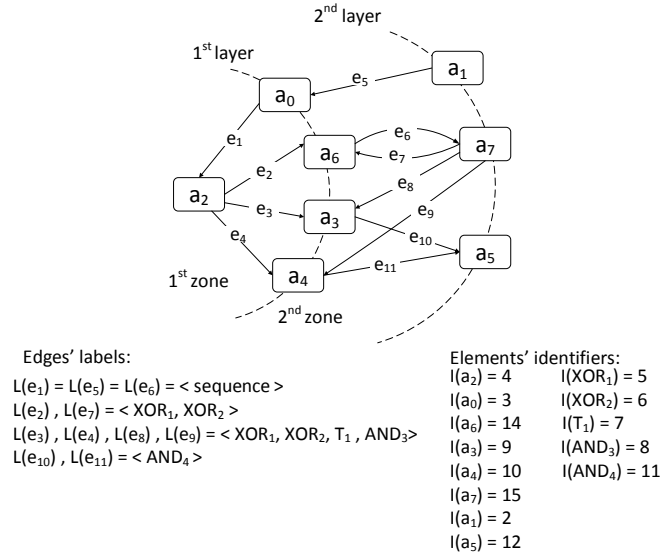
An aggregated neighborhood context graph represents a concise representation of multiple neighborhood context graphs. In the following, we give a generic definition of an aggregated neighborhood context graph and we detail its construction later in this chapter (Section 4.3.3)

**Definition 4.2.7** (Aggregated neighborhood context graph). *The aggregated neighborhood context graph of a vertex  $v_x \in \mathcal{V}$  denoted by  ${}^aG_{v_x}^k = (id_{v_x}^a, V_{v_x}^a, E_{v_x}^a, L_{v_x}^a, \tilde{\mathcal{C}}^a, \tau, \lambda)$  is a labeled directed graph created from a set of neighborhood context graphs  $\mathbb{G} = \{G_{v_{x_i}}^k = (id_i, V_i, E_i, L_i, I_i) : i \geq 1\}$  where:*

- $k$  is the number of layers;
- $id_{v_x}^a$  is a unique identifier;
- $V_{v_x}^a \subseteq \bigcup V_i$  is the set of vertices;
- $E_{v_x}^a \subseteq V_{v_x}^a \times V_{v_x}^a$  is the set of edges;
- $\tau : \mathcal{P}(\text{cod}(L)) \rightarrow (F^a)^*$  where  $\text{cod}(L) = \bigcup_{i \geq 1} \text{cod}(L_i)$  is the set of connection flows in the neighborhood context graphs in  $\mathbb{G}$  and  $\tau$  is a function that generates from a set of connection flows  $\{F_i : 1 \leq i \leq |\mathbb{G}|\} \in \mathcal{P}(\text{cod}(L))$  one connection flow  $F^a \in (F^a)^*$ ;
- $L_{v_x}^a : E_{v_x}^a \rightarrow (F^a)^*$  is a function that assigns connection flows (i.e. labels) to the edges;
- $\tilde{\mathcal{C}}^a$  is the set of connection elements in the connection flows in  $(F^a)^*$ ;
- $\lambda : V_{v_x}^a \cup \tilde{\mathcal{C}}^a \rightarrow \mathcal{P}(ID \times \text{cod}(I))$  where  $ID = \{id_i : i \geq 1\}$  and  $\text{cod}(I) = \bigcup_{i \geq 1} \text{cod}(I_i)$  is a function that assigns for each vertex  $v^a \in V_{v_x}^a$  and each connection element  $\tilde{c}^a \in \tilde{\mathcal{C}}^a$  a set of identifiers  $\{(id_{G_i}, id_{\tilde{c}_i}) : 1 \leq i \leq |\mathbb{G}|\} \in \mathcal{P}(ID \times \text{cod}(I))$  denoting the identifiers of the neighborhood context graphs and their corresponding elements from which  $v^a$  and  $\tilde{c}^a$  originate.



(a)  $G_1^2$ : A neighborhood context graph of the activity  $a_2$  in the process graph in Figure 4.2a



(b)  $G_2^2$ : A neighborhood context graph of the activity  $a_2$  in the process graph in Figure 4.2b

Figure 4.3: Two neighborhood context graphs with 2 layers

Figure 4.4 illustrates an example of an aggregated neighborhood context graph derived from the neighborhood context graphs in Figure 4.3 (its construction is detailed

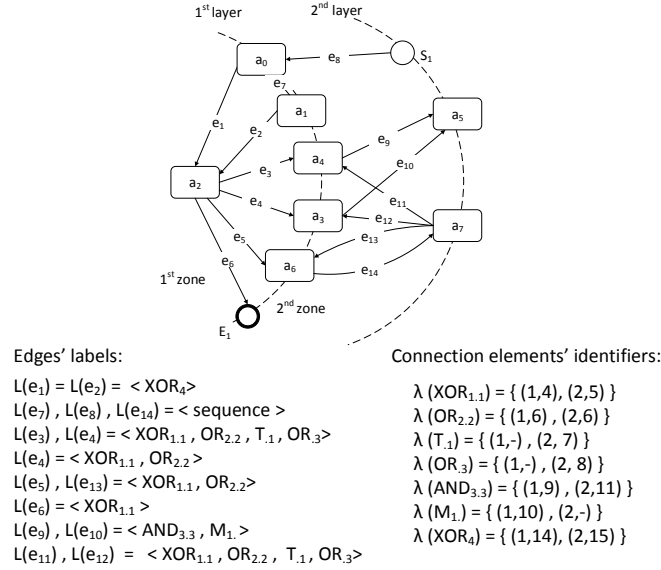


Figure 4.4:  $G_{12}^a$ : The aggregated neighborhood context graph resulted from merging the neighborhood context graphs in Figure 4.3

in Section 4.3).  $V^a = \{a_2, a_0, a_1, a_4, a_3, a_6, E_1, S_1, a_5, a_7\}$ ;  $E^a = \{e_1, e_2, e_3, \dots, e_{14}\}$ ;  $(F^a)^* = \{L(e_1), L(e_2), L(e_3), \dots, L(e_{14})\}$ ;  $\tilde{C} = \{XOR_4, XOR_{1.1}, OR_{2.2}, \dots\}$ .

In the next section, we present an algorithm, that given an activity, it extracts, clusters and merges different neighborhood context graphs into one or multiple aggregated neighborhood context graphs.

### 4.3 Deriving Aggregated Neighborhood Context Graphs

This section presents our approach for deriving aggregated neighborhood context graphs from a repository of process graphs. Let  $\mathbb{P} = \{P_i = (id_i, N_i, E_i, T_i, L_i) : i \geq 1\}$  be a repository of process graphs,  $a_x$  an activity selected by the process provider and  $k$  an integer chosen by the process provider representing the desired size of the returned fragments ( $k$  refers to the desired number of layers in the neighborhood context graphs).

Algorithm 1 illustrates the main steps for deriving aggregated neighborhood context graphs. It takes as input the process repository  $\mathbb{P}$ , the selected activity  $a_x$  and the integer  $k$ . It provides as output the set of aggregated neighborhood context graphs  $\mathbb{G}^a$ . The algorithm proceeds in three main steps. Firstly, the neighborhood context graphs of the activities that are similar to  $a_x$  within  $k$  layers are extracted from  $\mathbb{P}$  (Line 3 detailed in Section 4.3.1). Secondly, the extracted neighborhood context graphs are clustered using a clustering algorithm *Cluster* (Line 4 detailed in Section 4.3.2). *Cluster* takes as input the set of extracted neighborhood context

graphs  $\mathbb{G}$ , a distance matrix  $M_{dist}$  that stores the distance between the neighborhood context graphs in  $\mathbb{G}$  and a minimum similarity threshold  $minSim_G$  as a stopping criteria for the clustering. It provides as output the set of created clusters  $D$ . Lastly, each cluster in  $D$  is merged into one aggregated neighborhood context graph which is stocked in  $\mathbb{G}^a$  (Lines 5 - 7, detailed in Section 4.3.3).

---

**Algorithm 1** Algorithm for deriving aggregated neighborhood context graphs

---

```

1: input:  $\mathbb{P}, k, a_x$ 
2: output:  $\mathbb{G}^a = \{^a G_i^k : \mathcal{M}_A(a_x) = a \wedge 1 \leq i \leq |\mathbb{G}|\}$ 
3:  $\mathbb{G} = extractNCG(\mathbb{P}, k, a_x)$ ;
4:  $D = Cluster(\mathbb{G}, M_{dist}, minSim_G)$ ;
5: for  $C \in D$  do
6:    $^a G^k = Merge(C)$ 
7:    $\mathbb{G}^a = \mathbb{G}^a \cup \{^a G^k\}$ 
8: end for

```

---

### 4.3.1 Extracting Neighborhood Context Graphs

For each process  $P_i \in \mathbb{P}$ , we search for the activity  $a_x$  and extract its neighborhood context graph  $G_i^k = (id_{G_i}, V_{G_i}, E_{G_i}, L_{G_i}, I_{G_i})$  according to Definition 4.2.7. However, in real business processes, it is unrealistic to search for an exact matching of  $a_x$ . Therefore, we target to extract the neighborhood context graphs of the activities that are similar to  $a_x$ . To this end, we use a combination of *syntactic* and *linguistic* similarity metrics that are widely used for computing the similarity between the activities' labels in business process models [91].

Let  $a \in N_i$  such that  $T_i(a) = activity$  be an activity in  $P_i$ . To compute the similarity  $Sim_A$  between  $a$  and  $a_x$ , we integrate the bag-of-words similarity with label pruning technique presented in [97] which has been proved to increase the recall of the activity label matching to 0.44<sup>1</sup> without sacrificing the precision. We use Stanford Part-of-Speech (POS) [108, 109] for stemming string and removing function words from the activities' labels. Then, we prune words from the longer label and measure the similarity of two labels based on the pruned words. The similarity between two labels  $L_i(a)$  and  $L(a_x)$  is computed as follow:

$$Sim_A(L(a_x), L_i(a)) = \frac{\sum_{i=1}^{w^{a_x}} \max_{j=1}^{w^a} (sim(pr_{a_x}^i, pr_a^j)) + \sum_{j=1}^{w^a} \max_{i=1}^{w^{a_x}} (sim(pr_{a_x}^i, pr_a^j))}{2 \times \min(|w^{a_x}|, |w^a|)} \quad (4.1)$$

where  $w^{a_x}$  and  $w^a$  are the list of words contained in the labels  $L(a_x)$  and  $L_i(a)$  after applying the POS technique;  $pr_{a_x} = pr_u(w^{a_x}, w^a)$  and  $pr_a = pr_u(w^a, w^{a_x})$  are

---

<sup>1</sup>Prior research achieved a recall value around 0.26 [97]

the pruned list of words from  $w_{a_x}$  and  $w_a$  respectively;  $pr^i$  is the  $i^{th}$  word in the list of pruned words. The function  $sim$  computes the similarity between the pruned words using existing similarity metrics. In our work, we use a combination of syntactic and linguistic similarity metrics for computing  $sim$  since they are popular for measuring the similarity between activities' labels in business process models [91]. We use a syntactic similarity based on Levenshtein distance [102] which computes the number of edit operations (i.e. insert, delete or substitute a character) needed to transform one string into another. For the linguistic similarity, we use WordNet database [106] which is a lexical database for English words. The WordNet similarity package includes a set of algorithms for returning the synonyms between two words such as Lin metric [110]. The total similarity  $sim$  is the weighted average of the syntactic and the linguistic similarity of two words (Equation (4.2)).

$$sim(pr_{a_x}^i, pr_a^j) = \frac{w_1 \times LD(pr_{a_x}^i, pr_a^j) + w_2 \times Lin(pr_{a_x}^i, pr_a^j)}{w_1 + w_2} \quad (4.2)$$

where  $0 \leq sim \leq 1$ ,  $LD$  and  $Lin$  are functions returning the Levenshtein distance and the WordNet based similarity respectively between  $pr_{a_x}^i$  and  $pr_a^j$ ;  $w_1$  and  $w_2$  are two user's specified weights such that  $0 < w_1, w_2 \leq 1$ .

We say that  $a$  is the **best activity matching for**  $a_x$  if  $a_x$  and  $a$  have a similarity above a user specified threshold and there does not exist another activity in  $N_i$  having a higher similarity with  $a_x$ . Formally, we define the mapping function  $\mathcal{M}_A$  that maps an activity from one process graph  $P_1$  to the best activity matching in another process graph  $P_2$  (Definition 4.3.1).

**Definition 4.3.1** (Activity mapping  $\mathcal{M}_A$ ). *An activity mapping, denoted as  $\mathcal{M}_A : N_2 \rightarrow N_1$  is a partial injective function that maps an activity  $n_2 \in N_2$  to an activity  $n_1 \in N_1$  such that  $Sim_A(L_1(n_1), L_2(n_2)) \geq minSim_A \wedge \nexists n_x \in N_2 : Sim_A(L_2(n_1), L_2(n_x)) > Sim_A(L_1(n_1), L_2(n_2))$ , where  $minSim_A$  is a user specified threshold.*

For example, according to our motivating example,  $a_x = \{select\ a\ flight\}$ ,  $P_1, P_2 \in \mathbb{P}$  where  $P_1$  and  $P_2$  are the process graphs in Figure 4.2a and 4.2b respectively. The activity  $a_2 \in N_1$  is an exact matching for  $a_x$ ; therefore  $\mathcal{M}_A(a_x) = a_2$ . For a  $minSim_A = 0.5$ ,  $a_2 \in N_2$  is the best matching of  $a_x$ , therefore  $\mathcal{M}_A(a_x) = a_2$ . For a  $k = 2$ , the extracted neighborhood context graphs of the activities  $a_2 \in N_1$  and  $a_2 \in N_2$  are depicted in Figure 4.3a and Figure 4.3b respectively.

This step is repeated for all the processes in  $\mathbb{P}$ . At the end of this step, a set of neighborhood context graphs denoted as  $\mathbb{G} = \{G_a^k : \mathcal{M}_A(a_x) = a\}$  is extracted. In the next step, the extracted neighborhood context graphs are clustered according to their similarity before being aggregated.



### 4.3.2 Clustering Neighborhood Context Graphs

In the previous step, we extracted a set of neighborhood context graphs of the activities that are similar to a selected one. In this step, the extracted neighborhood context graphs are clustered in order to create groups of similar neighborhood context graphs that are later merged. This step is required since merging completely different neighborhood context graphs may result in complex and incomprehensible graphs [42].

In order to apply a clustering technique, the *distance* between each pair of the extracted neighborhood context graphs needs to be computed. To do so, we adopt the well know graph-edit-distance [160] as it is widely used for business process model matching [55]. The graph edit distance of two graphs is the minimal set of edit operations (node substitution, node insertion/deletion and edge insertion/deletion) required to transform one graph into the other. In our case, nodes are the vertices in the neighborhood context graph (i.e. activities, start and end events) and edges are the sequence flows between the vertices. A vertex substitution denotes that a vertex in one of the neighborhood context graphs is mapped to a vertex in the other neighborhood context graph. In this respect, we consider that activities are mapped according to the mapping function  $\mathcal{M}_A$  as defined in Definition 4.3.1. The activities that are not mapped are considered as either inserted in one neighborhood context graph or deleted in the other one. Similarly, the events are mapped if they have the same type (i.e. they are either start or end events). We denote by  $\mathcal{M}_{ev}$  the mapping function that maps the start and end events from one neighborhood context graph to another. The events that are not mapped are considered as either inserted in one neighborhood context graph or deleted in the other one. An edge is inserted in one neighborhood context graph (or deleted from the other) if at least one of its source or target vertices are not mapped or if its mapped source and target vertices are not connected through an edge in one of the neighborhood context graphs.

Let  $G_1^k$  and  $G_2^k$  be two neighborhood context graphs in  $\mathbb{G}$ ;  $sub_a$  is the set of substituted activities, i.e.  $\forall a \in sub_a : a \in dom(\mathcal{M}_A) \cup cod(\mathcal{M}_A)$ ;  $skip_a$  is the set of inserted/deleted activities, i.e.  $\forall a \in skip_a : a \notin dom(\mathcal{M}_A) \cup cod(\mathcal{M}_A)$ . Similarly,  $sub_{ev}$  is the set of substituted events, i.e.  $\forall ev \in sub_{ev}, ev \in dom(\mathcal{M}_{Ev}) \cup cod(\mathcal{M}_{Ev})$ ;  $skip_{ev}$  is the set of inserted/deleted events, i.e.  $\forall ev \in skip_{ev} : ev \notin N_1 \cap N_2$ ;  $skip_e$  is the set of inserted/deleted edges, i.e.  $\forall (v_i, v_j) \in skip_e : (v_i \notin dom(\mathcal{M}_A) \cup cod(\mathcal{M}_A)) \vee (v_j \notin dom(\mathcal{M}_A) \cup cod(\mathcal{M}_A)) \vee ((\mathcal{M}(v_i), \mathcal{M}(v_j)) \notin E_2)$ . The graph edit distance between  $G_1^k$  and  $G_2^k$  is computed as follow [55]:

$$GED(G_1^k, G_2^k) = 1 - \frac{wsub_a \times fsub_a + wskip_a \times fskip_a + wsub_{ev} \times fsub_{ev} + wskip_{ev} \times fskip_{ev} + wskip_e \times fskip_e}{wsub_a + wskip_a + wsub_{ev} + wskip_{ev} + wskip_e} \quad (4.3)$$

where  $0 \leq wsub_a, wskip_a, wsub_{ev}, wskip_{ev}, wskip_e \leq 1$  are relative weights of the

edit operations;  $fskip_a$ ,  $fskip_{ev}$  and  $fskip_e$  are the fraction of skipped activities, event and edges respectively;  $fsub_a$  and  $fsub_{ev}$  are the average distances between the substituted activities and events respectively and are defined as follow:

$$fskip_a = \frac{|skip_a|}{|A_1| + |A_2|} \quad fskip_{ev} = \frac{|skip_{ev}|}{|Ev_1| + |Ev_2|} \quad fskip_e = \frac{|skip_e|}{|E_1| + |E_2|} \quad (4.4)$$

where  $A_1 \subseteq N_1$  and  $A_2 \subseteq N_2$  are the activities in  $G_1^k$  and  $G_2^k$  respectively;  $Ev_1 \subseteq N_1$  and  $Ev_2 \subseteq N_2$  are the start and end events in  $G_1^k$  and  $G_2^k$  respectively.

$$fsub_a = \frac{2 \times \sum_{(a,b) \in \mathcal{M}_A} (1.0 - Sim_A(L_1(a), L_2(b)))}{|sub_a|} \quad (4.5)$$

$$fsub_{ev} = \frac{2 \times \sum_{(ev_1, ev_2) \in \mathcal{M}_{Ev}} (1.0 - Sim_{Ev}(L_1(ev_1), L_2(ev_2)))}{|sub_{ev}|}$$

where  $Sim_{Ev}(L_1(ev_1), L_2(ev_2))$  is equal to 1 if  $ev_1$  and  $ev_2$  have equal types (i.e. both are start or end events) and to 0 otherwise;  $\mathcal{M}_{Ev}$  maps the events from one graph to another according to  $Sim_{Ev}$ . A  $GED$  is equal to 1 if the corresponding neighborhood context graphs are exact matching. It is equal to 0 if they are completely dissimilar.

Since the graph matching is known to be NP-complete [160], many heuristics have been proposed such as using a greedy algorithm [55] which is still expensive for computing the graph edit distance. Accordingly, in our approach, we benefit from the neighborhood context graph structured in layers and zones in order to speed up the matching of elements. Instead of searching in the whole graph for a mapping, we search only in common layers. In case a matching satisfying the minimum similarity threshold is found, the elements are mapped. Otherwise, the search is expanded to the next layer and so on until it reaches the last layer (which is limited and equal to  $k$ ).

For example, suppose that we want to match the neighborhood context graphs in Figure 4.3. We refer to the first neighborhood context graph in Figure 4.3a by  $G_1^2$  and to the second one in Figure 4.3b by  $G_2^2$ . In the basic setting (i.e. without considering the layers), in order to find the mapping of  $a_4 \in V_1$  ( $L_1(a_4) = Request\ personal\ info$ ),  $a_4$  needs to be matched with all the vertices in  $V_2$ . Then, the vertex having the best matching, which is  $a_5$  ( $L_2(a_5) = Get\ user\ personal\ info$ ), is taken as a mapping. Considering our proposed approach,  $a_4$  is only matched with the vertices of the first layer in  $V_2$ . As  $a_5 \in V_2$  is the best match of  $a_4$  among the vertices in this layer, it is mapped to  $a_4$ . Take another example which is  $a_3$ .  $a_3$  does not have any matching in the first layer of  $G_2^2$ . Therefore, the search is expanded to the second layer. In this latter,  $a_6$  is the best matching of  $a_3$ , thus it is mapped to  $a_3$ . The GED between  $G_1^2$  and  $G_2^2$  with  $wsub_a = wskip_a = wsub_{ev} = wskip_{ev} = wskip_e = 1$  is computed as follows:  $\mathcal{M}_A(a_2) = a_2$ ;  $\mathcal{M}_A(a_1) = a_1$ ;  $\mathcal{M}_A(a_4) = a_5$ ;  $\mathcal{M}_A(a_3) = a_6$ ;  $\mathcal{M}_A(a_6) = a_7$ ;  $S_1 \in N_2$  is mapped to  $S_1 \in N_1$ ;  $GED = 1 - \frac{\frac{2 \cdot 0.4}{10} + \frac{2}{13} + \frac{0}{2} + \frac{1}{3} + \frac{14}{18}}{5} = 0.71$ .

The GED between every possible pair of neighborhood context graphs in  $\mathbb{G}$  is computed and stored in a  $m \times m$  matrix where  $m = |\mathbb{G}|$ . The column and rows correspond to the neighborhood context graphs. The  $(i, j)^{th}$  entry contains the *GED* of  $G_i^k$  and  $G_j^k$ ,  $1 \leq i, j \leq |\mathbb{G}|$ . A clustering algorithm *Cluster* is then applied. *Cluster* takes as input the similarity matrix  $M_{dist}$  (or a dissimilarity matrix in which the dissimilarity is defined as  $1 - GED$ ), a user specified similarity threshold  $minSim_G$  and the set  $\mathbb{G}$ . It provides as output a set of clusters of neighborhood context graphs.  $minSim_G$  is the minimal similarity between the neighborhood context graphs within the same cluster. Since our aim is to create clusters in which each pair of neighborhood context graphs has a similarity above  $minSim_G$ , we use the adapted Agglomerative Hierarchical Clustering (AHC) presented in [89] for clustering business process fragments. AHC starts with singleton clusters and iteratively combines the pairs of clusters that have the *highest similarity* that is above  $minSim_G$ . The similarity between two clusters is equal to the highest similarity found between the pairs of neighborhood context graphs in the two clusters (known as *complete linkage* strategy). The process of combining the clusters continues until there is only one cluster left or the minimal similarity  $minSim_G$  threshold is no more satisfied.

The result of the clustering can be visualized as a *dendrogram*  $D$ . Figure 4.5 shows an example of a dendrogram resulted from clustering five neighborhood context graphs represented as leaf nodes (in the bottom of the dendrogram). At each level of the dendrogram (in a bottom-up traversal), an intermediate cluster resulting from combining the low level clusters is visualized (e.g.  $C_1$  is an intermediate clusters). The upper level contains the final clusters ( $C_3$  and  $C_2$ ).

In the next section, we present an algorithm that takes the clusters at the top of  $D$  and merges each cluster into one aggregated neighborhood context graph.

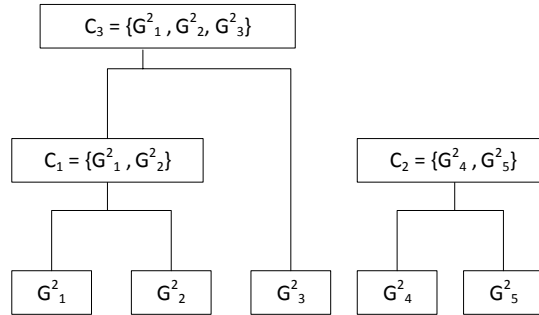


Figure 4.5: A dendrogram illustrating the result of the AHC algorithm

### 4.3.3 Merging Neighborhood Context Graphs

In this section, we present our merging algorithm to build an aggregated neighborhood context graph of a selected activity  $a_x$  **that preserves the behavior of the merged**

**fragments.** Let  $D = \{C_i = \{G_j^k\} : i, j \geq 1\}$  be the set of clusters at the top of the dendrogram obtained from the previous step (Line 4 in Algorithm 1). Our algorithm iterates on each  $C \in D$  (Line 5 in Algorithm 1), merges the neighborhood context graphs in  $C$  into one aggregated neighborhood context graph  ${}^aG^k$  (Line 6 in algorithm 1) and adds the result to the set of aggregated neighborhood context graphs  $\mathbb{G}^a$  (Line 7 in Algorithm 1).

In order to merge the neighborhood context graphs in one cluster, our algorithm starts by merging a pair of neighborhood context graphs and then iteratively merges the result with another neighborhood context graph until all the graphs are merged. The choice of the neighborhood context graphs to be merged is taken from the dendrogram structure which is traversed in a bottom up fashion. For example, in Figure 4.5, suppose that we want to merge the cluster  $C_3$ . The neighborhood context graphs  $G_1^2$  and  $G_2^2$  are first merged. Then, the resulted neighborhood context graph is merged with  $G_3^2$ . Algorithm 2 presents the steps for merging a pair of neighborhood context graphs. It takes as input a pair of neighborhood context graphs  $G_1^k$  and  $G_2^k$ , the set of substituted activities ( $sub_a$ ) and events ( $sub_{ev}$ ) and the set of inserted/deleted activities ( $skip_a$ ), events ( $skip_{ev}$ ) and edges ( $skip_e$ ). It provides as output an aggregated neighborhood context graph  ${}^aG_{12}^k$ . It starts by attributing a random identifier to the aggregated neighborhood context graph  ${}^aG_{12}^k$  (Line 3). Then, it proceeds in three main steps: (i) merge the vertices and edges (Lines 4-21 detailed in Section 4.3.3.1), (ii) merge the edges' labels (Line 25 detailed in Section 4.3.3.2) and (iii) handle the shared vertices (Line 26 detailed in Section 4.3.3.3).

#### 4.3.3.1 Merging vertices/edges

The first step of the algorithm consists of creating an aggregated neighborhood context graph in which the substituted vertices are added once (Lines 4 in algorithm 2). For the merged vertices (i.e. substituted ones), we choose the labels of those originating from the first neighborhood context graph (i.e. the vertices in  $V_1$ ). However, one can define more sophisticated rules as for example picking labels from a domain ontology.

Secondly, the edges in  $E_1$  are added to the set of edges in  $E$ . Regarding the edges in  $E_2$ , only those that belong to  $skip_e$  are added to  $E$  (Line 6). However, since the substituted vertices that originate from  $V_2$  are removed from  $V$ , one has to substitute the edges' sources and targets with their mappings in  $V_1$  before adding them to  $V$  (Lines 7-14).

Lastly, the identifiers of the vertices are identified (Lines 15-24). The identifiers of the merged vertices correspond to the identifiers of the substituted vertices in  $V_1$  and  $V_2$  (Lines 15-16) while the identifiers of the unmerged vertices correspond to their identifiers in the neighborhood context graphs from which they originate (Lines 19-24).

An example of the intermediate representation of the aggregated neighborhood context graph resulting from merging  $G_1^2$  and  $G_2^2$  in Figure 4.3 after this step is

**Algorithm 2** Merging algorithm

---

```

1: input:  $G_1^k = (id_1, V_1, E_1, L_1, I_1)$ ,  $G_2^k = (id_2, V_2, E_2, L_2, I_2)$ ,  $sub_a$ ,  $sub_{ev}$ ,  $skip_a$ ,
    $skip_{ev}$ ,  $skip_e$ 
2: output:  ${}^aG_{12}^k = (id, V, E, L, \tilde{C}, \tau, \lambda)$ 
3:  $id = newID()$ 
   {Step 1: Merge vertices and edges}
4:  $V = (V_1 \cup V_2) \setminus (V_2 \cap (sub_a \cup sub_{ev}))$ 
5:  $E = E_1$ 
6: for  $e_2 \in E_2 \cap skip_e$  do
7:   if  $(source_{e_2} \in sub_a \cup sub_{Ev}) \vee (target_{e_2} \in sub_a \cup sub_{Ev})$  then
8:     if  $source_{e_2} \in sub_a \cup sub_{Ev}$  then
9:        $E = E \cup \{(\mathcal{M}^{-1}(source_{e_2}), target_{e_2})\}$  { $\mathcal{M}$  can be either  $\mathcal{M}_A$  or  $\mathcal{M}_{ev}$  de-
         pending on the vertex type}
10:    else
11:       $E = E \cup \{(source_{e_2}, \mathcal{M}^{-1}(target_{e_2}))\}$ 
12:    end if
13:  end if
14: end for
15: for  $v \in V \cap (sub_a \cup sub_{Ev})$  do
16:    $\lambda(v) = \lambda(v) \cup \{(id_1, I_1(v)), (id_2, I_2(\mathcal{M}(v)))\}$ 
17: end for
18: for  $v \in skip_a \cup skip_{Ev}$  do
19:   if  $v \in V_1$  then
20:      $\lambda(v) = \lambda(v) \cup \{(id_1, I_1(v))\}$ 
21:   else
22:      $\lambda(v) = \lambda(v) \cup \{(id_2, I_2(v))\}$ 
23:   end if
24: end for
   {Step 2: Merge edges' labels}
25:  $MergeLabels({}^aG_{12}^k, G_1^k, G_2^k, skip_e, sub_e)$ 
   {Step 3: Handle shared vertices}
26:  $addXORToSemiSharedEdges(E, L, \tilde{C}, \lambda)$ 

```

---

depicted in Figure 4.6. The merged vertices are  $a_2$ ,  $a_1$ ,  $a_4$ ,  $a_3$  and  $a_5$  while the remaining ones are unmerged. The merged edges are  $e_3$ ,  $e_4$ ,  $e_9$  and  $e_{10}$ . The remaining ones are unmerged and their sources or targets that were initially substituted are updated before being added. For example, the edge  $e_3$  that originates from  $NCG_2^2$  has the source node  $a_2 \in N_2$  which is substituted with  $a_2 \in N_1$ . Since only the copy of  $a_2 \in N_1$  is added to the aggregated graph, the source of  $e_3$  is updated to this copy.

In the next step, we define the edges' labels. We present an approach for merging the labels of the merged edges. The labels of the unmerged edges are simply added.

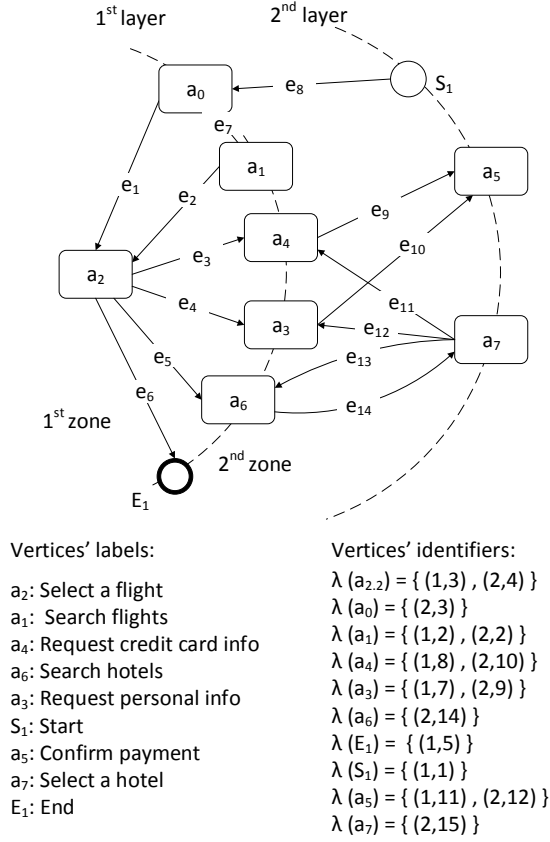


Figure 4.6: The aggregated neighborhood context graph after merging the vertices and edges

#### 4.3.3.2 Defining edges' labels

In the second step of the algorithm, we define the edges' labels (Line 25 in algorithm 2 detailed in Algorithm 3). Firstly, the labels of the unmerged edges that originate either from  $E_1$  or  $E_2$  are simply added. The connection elements in each label are added to the set of the connection elements  $\tilde{C}$  and their identifiers are defined (Lines 2-16 in Algorithm 3).

Secondly, the labels of the merged edges are merged (Lines 17-21). Since edges' labels are sequences of connection elements (i.e. connection flows) which can be mapped to sequences of characters, we define the function  $\tau$  (Line 18) that (1) matches the labels to be merged by aligning the connection elements (**Connection flow alignment**), then (2) merges the aligned connection elements (**Merging aligned connection flows**).

(1) **Connection flow alignment.** Inspired from the approach presented in [161] where log traces are aligned in order to analyze their behavior, we propose to

---

**Algorithm 3** *MergeLabels*( ${}^aG_{12}^k, G_1^k, G_2^k, skip_e, sub_e$ )

---

```

1: {Define labels for unmerged edges}
2: for  $e \in skip_e$  do
3:   if  $e \in E_1$  then
4:      $L(e) = L_1(e)$ 
5:      $\tilde{C} = \tilde{C} \cup \{L_1(e)[i] : 1 \leq i \leq |L_1(e)|\}$ 
6:     for  $c \in \tilde{C} \cap \{L_1(e)[i]\}$  do
7:        $\lambda(c) = \lambda(c) \cup \{(id_1, I_1(L_1(e)[i]))\}$ 
8:     end for
9:   else
10:     $L(e) = L_2(e)$ 
11:     $\tilde{C} = \tilde{C} \cup \{L_2(e)[i] : 1 \leq i \leq |L_2(e)|\}$ 
12:    for  $c \in \tilde{C} \cap \{L_2(e)[i]\}$  do
13:       $\lambda(c) = \lambda(c) \cup \{(id_2, I_1(L_2(e)[i]))\}$ 
14:    end for
15:   end if
16: end for
   {Define labels for merged edges}
17: for  $(x, y) \in E \cap sub_e$  do
18:    $L((x, y)) = \tau(L_1((x, y)), L_2((\mathcal{M}(x), \mathcal{M}(y))))$ 
19:    $\tilde{C} = \tilde{C} \cup \{L((x, y))[i] : 1 \leq i \leq |L((x, y))|\}$ 
20:    $\lambda$  for each  $\tilde{c} \in \tilde{C} \cap \{L((x, y))[i]\}$  is defined in  $\tau$ 
21: end for
22: UpdateLabels()

```

---

align the connection flows (edge's labels) in order to merge them. The alignment of a set of connection flows is the set of edit operations applied to transform the flow connections into each others, with the allowable edit operations being insertion, deletion, or substitution of elements. A gap “–” is added to denote the insertion or deletion of an element.

Let  $\mathbb{F}_x^y = \{F_1, F_2\} : F_1 = L_1((x, y)) \wedge F_2 = L_2((\mathcal{M}(x), \mathcal{M}(y)))$ <sup>2</sup> be the connection flows of the merged edge  $(x, y) \in E$ . For example, in Figure 4.6,  $\mathbb{F}_{a_2}^{a_4} = \{L_1(e_3), L_2(e_3)\}$ . In Definition 4.3.2, we give a formal definition of the connection flow alignment.

**Definition 4.3.2** (Connection flow alignment). *The connection flow alignment of a pair of connection flows  $\mathbb{F} = \{F_1, F_2\}$  is defined as the transformation of  $\mathbb{F}$  into  $\mathbb{F}^A$  where  $\mathbb{F}^A = \{F_1^A, F_2^A\}$  and  $F_i^A = F_i \cup \{-\}^*$  such that:*

- $|F_1^A| = |F_2^A| = m$

---

<sup>2</sup> $\mathcal{M}$  can be either  $\mathcal{M}_A$  or  $\mathcal{M}_{Ev}$  depending on  $x$  and  $y$  types

- $\nexists p, 1 \leq p \leq m$  such that  $F_1^A[p] = F_2^A[p] = \text{“-”}$

An alignment can be considered as a  $2 \times m$  matrix where the lines are the connection flows and gap elements; and the columns are the matched connection elements. The first requirement in Definition 4.3.2 states that the aligned connection flows have the same length after applying the edit operations. The second requirement states that there does not exist a column in the alignment where all the elements are gaps. Two examples of a connection flow alignment for  $\mathbb{F}_{a_2}^{a_1} = \{F_1, F_2\}$  where  $F_1 = \langle XOR_1, AND_2 \rangle$  and  $F_2 = \langle XOR_1, XOR_2, T_1, AND_3 \rangle$  are illustrated in Figure 4.7 and Figure 4.8. The first alignment is  $\mathbb{F}_1^A = \{F_1^{A1}, F_2^{A1}\}$  and the second is  $\mathbb{F}_2^A = \{F_1^{A2}, F_2^{A2}\}$

$$\begin{array}{l} F_1^{A1} : \quad XOR_1 \quad AND_2 \quad - \quad - \\ F_2^{A1} : \quad XOR_1 \quad XOR_2 \quad T_1 \quad AND_3 \end{array}$$

Figure 4.7:  $\mathbb{F}_1^A$ : One possible alignment of  $F_1$  and  $F_2$

$$\begin{array}{l} F_1^{A2} : \quad - \quad - \quad - \quad XOR_1 \quad AND_2 \\ F_2^{A2} : \quad XOR_1 \quad XOR_2 \quad T_1 \quad - \quad AND_3 \end{array}$$

Figure 4.8:  $\mathbb{F}_2^A$ : One possible alignment of  $F_1$  and  $F_2$

In order to derive a valid connection flow, only elements having the same type can be substituted. In other words, a gateway cannot be substituted with an event and vice versa. For example, the two alignments in Figure 4.7 and Figure 4.8 are valid. Since there exist many possible valid alignments of a set of connection flows, we target to find the best alignment between the connection flows. We use the Levenshtein distance (LD) [102] to compute the best alignment. The **minimal LD** value corresponds to **the best alignment**. We add a weight  $W$  for each edit operation in a way that it minimizes the effect of the substitution operation and maximizes that of the insertion/deletion operation. We consider that the weight of substituting two elements having the same type and the same labels (e.g two XOR elements) is less than the weight of substituting two elements having the same type but different labels (e.g. an XOR and AND elements). And this is in its turn less than the inserting or deleting an element. We denote by  $S(c_1, c_2)$  the substitution of the connection elements  $c_1 \in F_1$  and  $c_2 \in F_2$  and  $Indel(c_1)$  the insertion or deletion of  $c_1$  into  $F_1^A$  or



$F_2^A$ . The operations weights are presented in Equation 4.6.

$$\left\{ \begin{array}{l} W_{S(c_1, c_2)} = -1 \quad \text{if } c_1 = c_2 \\ W_{S(c_1, c_2)} = 0 \quad \text{if } T(c_1) = T(c_2) \ \& \\ \quad \quad \quad L(c_1) \neq L(c_2) \\ W_{Indel(c_1)} = 1 \end{array} \right. \quad (4.6)$$

The LD of the alignment of a pair of connection flows denoted as  $\mathbb{F}_{12}^A = \{F_1^A, F_2^A\}$  is defined as:

$$LD(\mathbb{F}_{12}^A) = \sum_1^m w_i \quad (4.7)$$

where

$$w_i = \left\{ \begin{array}{l} W_{S(c_1, c_2)} \quad \text{if } F_1^A[i] = c_1 \ \& \ F_2^A[i] = c_2 \\ W_{Indel(c_1)} \quad \text{if } F_1^A[i] = c_1 \ \& \ F_2^A[i] = \text{“-”} \\ \quad \quad \quad \text{or } F_1^A[i] = \text{“-”} \ \& \ F_2^A[i] = c_1 \end{array} \right. \quad (4.8)$$

For example, in Figure 4.7 and Figure 4.8, the alignments  $\mathbb{F}_1^A$  and  $\mathbb{F}_2^A$  have  $LD = 1$  and  $LD = 3$  respectively.  $LD(\mathbb{F}_1^A) < LD(\mathbb{F}_2^A)$ , therefore  $\mathbb{F}_1^A$  is the best alignment. To compute the best alignment, we refer to [162] where a dynamic programming algorithm has been proposed in order to find the optimal alignment between two amino-acid sequences. The algorithm takes as input the operations weights defined in Equation 4.6 and provides as output the alignment with the minimal  $LD$ .

Once we compute the best alignment for two connection flows of a merged edge, we proceed in the next step to define a set of rules for merging the aligned connection elements into one connection flow.

**(2) Merging aligned connection flows.** Once edge’s connection flows are aligned, we merge each column of the alignment into one connection element. A gap “-” in an alignment represents an empty position in our merged fragment. In order to preserve the behavior of the merged elements, we define a set of merging rules for gateways and events that comply to Definition 3.2.1 and Definition 3.2.2.

1. If the elements are of type  $T = \text{“gateway”}$  (in this case, the gap is treated as a Sequence):
  - If they have the same label, the result is the same gateway. For example a set of AND gateways is merged into an AND.
  - If they are a set of XOR and Sequence, the result is an XOR;
  - if they have different labels, the result is an OR. For example, a set of AND, XOR and Sequence is merged into an OR.
2. If the elements are of type  $T = \text{“event”}$ :

- If they have the same labels, the result is an event with the same label. For example a set of Message events  $M$  is merged into the same Message event  $M$ .
- If they have different labels, the merged event is an abstract event, i.e. *none*. For example, a Message event  $M$  aligned with a timer event  $T$  are merged into a *none*.

For example, the merging result of the alignment presented in Figure 4.7 is illustrated in Figure 4.9.

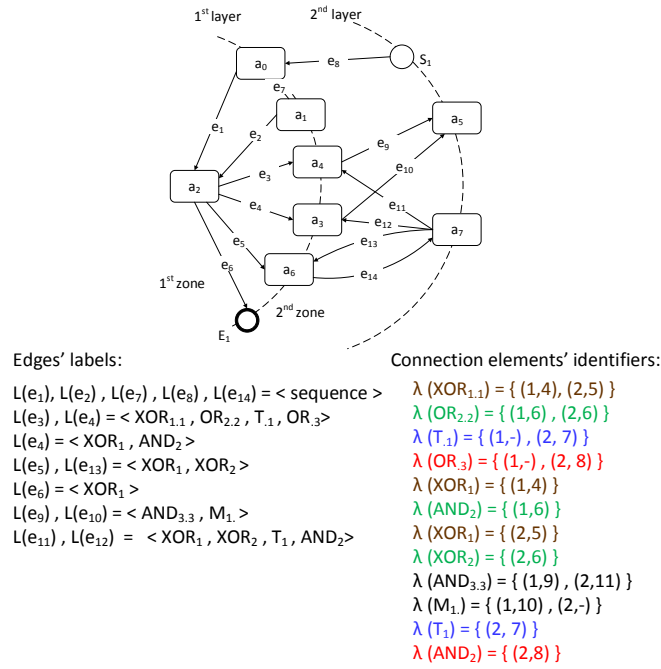
$$\begin{array}{cccc}
 F_1^{A1} : & XOR_1 & AND_2 & - & - \\
 F_2^{A1} : & XOR_1 & - & T_1 & AND_3 \\
 \hline
 & XOR & OR & T & OR
 \end{array}$$

Figure 4.9: The merging result of the alignments in Figure 4.7

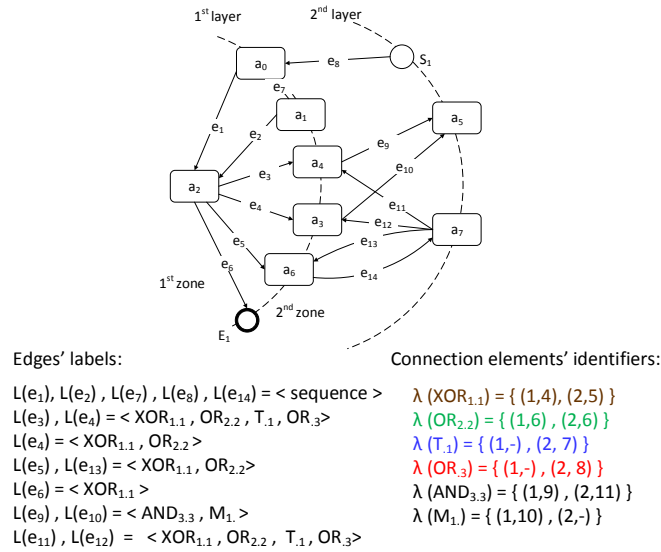
The identifier  $\lambda$  of each merged element is defined as the set of identifiers of the neighborhood context graphs and the elements from which it originates. For example in Figure 4.9, the identifier of the first merged  $XOR$  is  $\lambda(XOR) = \{(1, 4), (2, 4)\}$  where  $(1, 4)$  refers to  $XOR_1$  in  $G_1^2$  (i.e.  $id_{G_1^2} = 1$  and  $I_1(XOR_1) = 4$ ) and  $(2, 4)$  refers to  $XOR_1$  in  $G_2^2$  (i.e.  $id_{G_2^2} = 2$  and  $I_2(XOR_1) = 4$ ).

This aligning/merging procedure is repeated for all the merged edges in  ${}^aG_{12}^k$ . At the end of this step, an aggregated neighborhood context graph with merged labels is obtained. An example of the aggregated neighborhood context graph resulted after this step is illustrated in Figure 4.10a. In this example, the labels of the merged edges  $e_3$ ,  $e_4$ ,  $e_9$  and  $e_{10}$  are merged while the others labels are simply added as they appear in their origin neighborhood context graphs. Please note that the elements are indexed with subscripts to differentiate between them. The identifier of each connection element is depicted in the left of the figure.

One issue encountered after this step is that some of the connection elements that have been merged for some edges remain unchanged for the unmerged edges. Take for example the elements  $XOR_{1,1}$  and  $XOR_1$  in Figure 4.10a.  $XOR_{1,1}$  is the result of merging  $XOR_1 \in G_1^2$  and  $XOR_1 \in G_2^2$ ;  $XOR_1 \in G_1^2$  in the unmerged edges (e.g. in  $e_4$ ,  $e_6$ , etc.) has not been updated to  $XOR_{1,1}$ . Therefore, in the last step of the algorithm (Line 22 in Algorithm 3) the unmerged connection elements are tested and updated if require. This is done through testing the identifiers of the connection elements for membership. In case the identifier of an unmerged connection element belongs to the identifier of a merged one, the former is updated in the labels where it appears. For example, in Figure 4.10a, the different colors in the connection elements' identifiers part refer to the unmerged connection elements that should be updated by the merged ones having the same color. For instance,  $AND_2$  should be updated to



(a) The aggregated neighborhood context graph after defining the edges' labels



(b) The aggregated neighborhood context graph after updating the unmerged connection elements

Figure 4.10: The aggregated neighborhood context graphs during the definition of the edges' labels

$OR_{.3}$ ,  $T_1$  to  $T_{.1}$ ,  $XOR_1$  and  $AND_2$  to  $OR_{2.2}$  and so on. The result of this step is illustrated in Figure 4.10b.

The last step in the merging algorithm is to handle the edges sharing exclusively their sources or targets.

#### 4.3.3.3 Handling edges sharing their source or target

So far, our merging algorithm focuses on merging the labels of merged edges and keeping others unchanged. However, we can face the case where some edges share exclusively their source or target. In fact, in our merged fragment, a set of edges having the same source share it if the source is a merged vertex and the edges are unmerged ones (the same holds for the shared target). In this case, we add a split  $XOR$  (respectively join  $XOR$ ) to the beginning (respectively end) of the edges' labels sharing their source (respectively target) (Line 26 in Algorithm 2). For example, in the aggregated neighborhood context graphs in Figure 4.10b, the edges  $e_1$  and  $e_2$  are unmerged ones and have the same merged target  $a_2$ . Therefore,  $e_1$  and  $e_2$  share their target and a join  $XOR$  is added at the end of their corresponding labels ( $XOR_4$  in Figure 4.4). A new random identifier is defined for the added  $XOR$  that takes the form of  $\{(id_{G_1^k}, nb_1), (id_{G_2^k}, nb_2)\}$  where  $nb_1$  and  $nb_2$  are two random numbers. And last,  $XOR$  is added to the set of connection elements  $\tilde{\mathcal{C}}$ .

#### 4.3.4 Behavior Preservation of the Merging Algorithm

In this section, we show that, by construction, our configurable process fragment resulted from our merging algorithm preserves the behavior of the input merged fragments (Proposition 4.3.1).

Let  ${}^aG_{a_x}^k = (id^a, V^a, E^a, L^a, \mathcal{C}, \tau, \lambda)$  be an aggregated neighborhood context graph resulted from merging the neighborhood context graphs:  $G_{a_x} = \{G_1, G_2\}$  such that  $\forall i : 1 \leq i \leq n, G_i = (id_i, V_i, E_i, L_i, I_i)$ .

**Proposition 4.3.1.**  ${}^cG_{a_x}^k$  subsumes the behavior of  $G_{a_x}$  if:

1.  $\forall v \in V_i, 1 \leq i \leq 2 \implies \exists v \in V^a$
2.  $\forall (a, b) \in E_i, 1 \leq i \leq 2 \implies \exists (a, b) \in E^a$
3.  $\forall e \in E_i, L_i(e) = F_i \wedge 1 \leq i \leq 2 \implies \exists e^a \in E^a, L^a(e^a) = F^a$  such that  $\forall k, 1 \leq k \leq |F_i| : F_i[k] = F^a[k] \vee F_i[k] \preceq F^a[k], \preceq \in \{\preceq_g, \preceq_e\}$

*Proof.* The first step of our merging algorithm (Section 4.3.3.1) consists of merging the vertices and the edges that are substituted. Therefore, for each substituted vertex  $a \in \cup_i V_i$  one copy is added to  $V^a$  and for each edge substituted  $(a, b) \in \cup_i E_i$  one copy is added to  $E^a$ . The unsubstituted vertices and edges are simply added. Consequently, the first two requirements are met.

The third requirement states that, for each edge's label  $F_i$  in  $G_i$ , we can find either the same edge associated to the same label (i.e.  $F_i$ ) in  ${}^aG_{a_x}^k$ , or the same edge associated to a label  $F^a$  that includes its behavior. Returning back to the second step of our merging algorithm (Section 4.3.3.2), the edges' labels of  ${}^aG_{a_x}^k$  are defined in two steps. First, if the corresponding edge has not been merged, then the same label as it appears in  $G_i$  is added. Therefore,  $\forall k, 1 \leq k \leq |F_i|, F^a[k] = F_i[k]$ . Hence, the third requirement is validated. Second, if the edge has been merged, then their corresponding labels in each of the  $G_i$  sources are also merged. The labels' merging is defined in the second step of our merging algorithm using two operators: the connection flow alignment and the connection elements merger. Let  $F_1, F_2$  be two labels of the merged edge  $(a, b) \in E^a$ ;  $\mathbb{F}^A = \{F_1^A, F_2^A\}$  is the resulted alignment and  $F^a$  is the resulted merged label. From the connection flow alignment definition (Definition 4.3.2) and from our merging rules we have:

- $|F_1^A| = |F_2^A| = |F^a| = m$ : the alignment and the merging result have the same sizes;
- each column  $p$  in the alignment  $\mathbb{F}^A$  is merged into one connection element based on the partial order for gateways (Definition 3.2.1) and events (Definition 3.2.2). Therefore,  $\forall 1 \leq k \leq m, 1 \leq p \leq 2 : F_p^A[k] = c$  then  $\exists c^a : F^a[k] = c^a \wedge c \preceq c^a$ . Hence, the third requirement is validated for the alignment  $F_i^A$  of  $F_i$ .

From the connection flow alignment definition (Definition 4.3.2), we also have  $|F_i| \leq |F_i^A|$ , i.e. the sizes of  $F_i$  and  $F_i^A$  are equals or  $F_i^A$  contains gap elements. However, the inserted gap elements into  $F_i^A$  are silent elements (they represent a sequence in the case of a gateway and a disabled event in the case of an event) and do not alter the order or the behavior of the connection elements. Thus  $F_i$  and  $F_i^A$  have the same behavior and consequently the third requirement validated on  $F_i^A$  is also valid for  $F_i$ , i.e.  $\forall k : F_i[k] = c, \exists c^a : F^a[k] = c^a \wedge c \preceq c^a, 1 \leq k \leq m$ .

In the fourth step of our merging algorithm, we add an *XOR* for the edges that share their source or target. this *XOR* is transformed to a configurable one in the C-BPMN notation. Therefore, it can be configured to a sequence flow (i.e. *sequence*  $\preceq_g$  *XOR*). Thus, the inserted *XOR* elements do not alter the behavior of  ${}^aG_{a_x}^k$  (i.e. silent elements).  $\square$

### 4.3.5 Computational Complexity

Our merging algorithm (Algorithm 2) consists of two main steps. The first one consists of merging the activities and edges within the same layer and zone respectively. The worst case of this computation step is  $O(k \times n_a^2 \times n_e^2 \times n_G) \simeq O(n^5)$  where  $k$  is the number of layers in the extracted neighborhood context graphs,  $n_a$  is the maximum number of vertices on a layer,  $n_e$  is the maximum number of edges in a zone and  $n_G$  is the number of neighborhood context graphs to be merged. On one hand, we match only the activities and edges within the same layer and zone respectively. On the

other hand, the number of activities on the same layer is not great, so is the number of edges within the same zone. Therefore, we don't face the problem of the whole graph matching that has been demonstrated to be NP-complete [115].

The second steps consists of merging the connection flows of the merged edges. It calls for two operators: (1) the connection flow alignment and (2) the connection elements merger. The complexity of the first operator is  $O(n'_e \times n^2 \times l^2) \simeq O(n^5)$  where  $n'_e$  is the number of merged edges,  $n$  is the number of connection flows to be aligned and  $l$  is the size of a connection flow. The aligned connection flows are then merged using the connection elements merger operator. The alignment can be seen as matrix where each column is merged into one element. Thus, the complexity of this operator is  $O(n'_e \times n \times l) \simeq O(n^3)$ . The overall complexity of this algorithm is  $O(n^5)$ . Accordingly, our merging algorithm is performed in a polynomial time and the worst case computation is  $O(n^5)$ .

#### 4.4 From Aggregated Neighborhood Context Graph to C-BPMN

The aggregated neighborhood context graphs resulted from our merging algorithm are transformed to a configurable standard modeling notation. In our work, we use the C-BPMN notation. Let  ${}^aG^k = (id^a, V^a, E^a, L^a, \mathcal{C}, \tau, \lambda)$  be an aggregated neighborhood context graph and  $P^c = (id, N, E, T, L, I, B, \mathbb{C}^*)$  be its corresponding C-BPMN notation.  $P^c$  is derived from  ${}^aG^k$  as follows:

- the set of vertices in  ${}^aG^k$  is added to the set of nodes in  $P^c$  and their types are defined (i.e. activity or start or end event);
- The set of connection elements  $\tilde{C}$  in  ${}^aG^k$  are added to the set of nodes in  $P^c$ , their types in  $P^c$  are set to *gateway* and their labels (*OR*, *AND* or *XOR*) are defined.
- for each edge  $e^a = (v_1^a, v_2^a) \in E^a$  and its corresponding label  $L^a(e) = \langle \tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_n \rangle$  in  ${}^aG^k$ , a sequence of edges  $(n_i, n_{i+1}), 0 \leq i \leq n + 1$  such that  $n_0 = v_1^a$  and  $n_{n+1} = v_2^a$  and  $1 \leq i \leq n, n_i = \tilde{c}_i$  is added to the set of edges in  $P^c$ ;
- The configurable elements in  $P^c$  are the connection elements in  $\tilde{C}$  whose identifier set size is greater than 1. This means that they are the result of a merging step. For example, in Figure 4.4, the size of the identifier sets of the connection elements is 2. Therefore, they are configurable.

Formally, the configurable process graph  $P^c$  derived from the aggregated neighborhood context graphs  ${}^aG^k$  is given in Definition 4.4.1.

**Definition 4.4.1** (From  ${}^aG^k$  to  $P^c$ ). *A configurable process graph  $P^c$  is derived from an aggregated neighborhood context graph  ${}^aG^k$  as follows:*

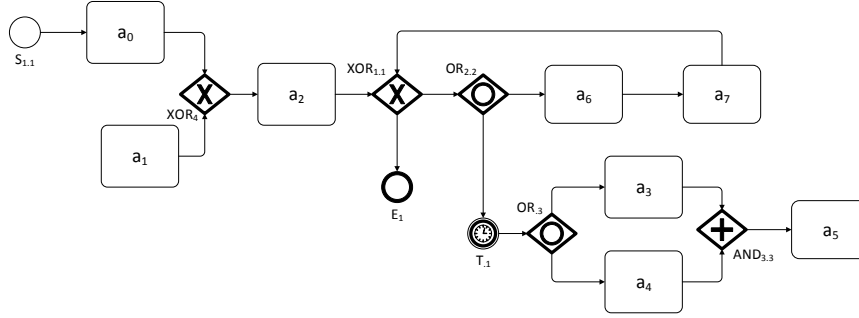


Figure 4.11: A C-BPMN derived from the aggregated neighborhood context graph in Figure 4.4

- $id = id^a$
- $N = V^a \cup \tilde{C}$ ;
- $E = \bigcup_{(x,y) \in E^a} (\bigcup_{i=0}^{i=n+1} (n_i, n_{i+1}) : n_0 = x \wedge n_{n+1} = y \wedge \forall 1 \leq i \leq n, n_i = L^a(x, y)[i])$ ;
- $T$  and  $L$  are as defined in Definition 3.3.5. They assign types and labels to the elements.
- $B : N \rightarrow \{true, false\}$  is a boolean function such that  $B(c) = true, c \in N \cap \tilde{C}$  iff  $(|\lambda(c)| > 1)$ ;
- $\mathbb{C}^*$  is as defined in Definition 3.3.5. It defines the valid configurations of configurable elements according to Definition 3.3.4.

An example of the configurable process graph derived from the aggregated neighborhood context graph in Figure 4.4 is illustrated in Figure 4.11.

## 4.5 Conclusion

In this chapter, we answered the two questions raised in the thesis problematic (section 1.2.1), which are: *How to identify process fragments that are close to process provider interests?* and *How to derive configurable process fragments?*.

To identify process fragments that are close to process provider interests, we used the *neighborhood context graph* which is defined as a process fragment around a selected activity. It contains the associated activity and the relations to its closest neighbors.

To derive configurable process fragments, we extracted the neighborhood context graphs of the activities that are similar to the selected one in different business process models. The extracted graphs are matched and clustered based on their similarity

before being merged. We developed a merging algorithm that takes as input a cluster of neighborhood context graphs and iteratively merge them into one *aggregated neighborhood context graph*. This latter is a concise representation of the merged fragments. The aggregated neighborhood context graph is then transformed to a C-BPMN fragment.

In contrast to creating configurable process models at once, recommending configurable fragments allows process providers to flexibly adjust and improve their designed processes. It gives them the hand to define specific configurable parts in their processes. Our approach can be used when a process provider is looking for small fragments to complete missing parts in an ongoing designed process, to replace some parts or to extend an existing configurable process.

Our principles presented in Section 1.4.1 are respected:

- **Automation:** We propose an automated approach to derive configurable fragments. We do not ask the process provider for any additional manual effort except selecting the position (represented by an activity) in the process graph for which he needs to discover a configurable fragment.
- **Implicit knowledge exploitation:** We exploit activity neighborhood context which is implicit knowledge hidden in process models to derive our process fragments.
- **Focused results:** We merge configurable process fragments instead of entire process models. So, our approach do not make process providers confused with large and complex merged models.
- **Balanced computation:** We match specific parts represented as small fragments instead of entire process models to derive our configurable fragments. The neighborhood context graph structured in layers allows us to employ a simple heuristic. We match only activities located on the same layer to find the best mapping. If a matching is not found, the search is expanded to the next layer. Since the number of layers in a neighborhood context graph is limited and in general is not too big (for a process fragment use, it does not exceed 4 layers), our approach we do not face the NP-complete problem. The computational complexity of our approach is polynomial.





# Supporting Process Configuration with Configuration Guidance Models

## Contents

---

|            |   |            |
|------------|---|------------|
| <b>5.1</b> | <b>Introduction</b>   | <b>95</b>  |
| <b>5.2</b> | <b>Motivating Example</b>   | <b>96</b>  |
| <b>5.3</b> | <b>Configuration Guidance Model</b>                                     | <b>99</b>  |
| <b>5.4</b> | <b>Approach Overview</b>  | <b>102</b> |
| <b>5.5</b> | <b>Extracting configuration guidelines from existing process models</b> | <b>103</b> |
| 5.5.1      | Retrieving process elements' configurations                             | 103        |
| 5.5.2      | Apriori-based approach for deriving configuration guidelines            | 106        |
| <b>5.6</b> | <b>Inferring Configuration Steps order</b>                              | <b>108</b> |
| <b>5.7</b> | <b>Formalizing Configuration Guidelines Dependencies Relations</b>      | <b>110</b> |
| 5.7.1      | Deriving a transition system from configuration guidelines              | 110        |
| 5.7.2      | Deriving a Petri-Net using Theory of Regions                            | 113        |
| <b>5.8</b> | <b>Conclusion</b>   | <b>114</b> |

---

## 5.1 Introduction

This chapter presents our approach for supporting business process configuration. As mentioned in Chapter 1, configurable process models tend to be very complex which makes their configuration an increasingly difficult task. To this end, many approaches have been proposed to build configuration support systems that assist process users selecting desirable configuration choices according to specific requirements (e.g. [6, 7, 50]). Nevertheless, these systems are actually *manually created* by domain experts, which is undoubtedly a time-consuming, and tedious task. In addition, relying solely on the expert knowledge is not only error-prone, but also challengeable, especially in today's dynamic and fast changing business requirements.

In this chapter, we present an automated approach for supporting the creation of configuration support systems. Inspired by the need to integrate the users' experience in process configuration [3, 34], we propose to learn from the experience gained through past process configurations in order to extract useful and implicit knowledge for configuration decision making.

We start the chapter by presenting a motivating example and the challenges that arise when it comes to configure a process (Section 5.2). In light of the identified challenges, we present in Section 5.3 our *configuration guidance model* that aims to fulfill the requirements for a configuration support system. Basically, a configuration guidance model targets to answer the following questions:

1. *How* an element is configured given the previously selected choices?
2. *How often* a specific decision has been made?
3. *When* a configuration decision can be taken for a configurable element?

In Section 5.4, we present an overview of the proposed approach for the automatic extraction of configuration guidance models which consists of three steps: (i) extracting configuration guidelines, (ii) inferring the configuration steps order and (iii) formalizing the guidelines dependencies' relations. Sections 5.5, 5.7 and 5.6 detail the different steps of our approach. Finally, we conclude the chapter in Section 5.8

The work in this chapter was published in conference proceedings [163, 164].

## 5.2 Motivating Example

We reuse our motivating example presented in Section 1.3 to illustrate our approach. We assume that a process provider has designed a configurable travel booking process (Figure 5.1) with the assistance of our previously presented approach (Chapter 4).

The configurable process in Figure 5.1 includes four main functionalities: (1) flight booking with alternatives, (2) recommendation, (3) discount offer and (4) payment (see Figure 1.7 for the explicit visualization of the different functionalities' parts). It contains 17 configurable elements (9 gateways, 7 activities and 1 event) which are graphically modeled with a thick line <sup>1</sup>. These elements are configured according to the process users' specific needs. For example, one process user may not need a recommendation functionality neither a discount offer. This corresponds to configuring  $XOR_1^c$  to a sequence starting with  $XOR_2$  (i.e. the outgoing branch of  $XOR_1^c$  starting with  $XOR_3$  is removed) and  $a_{16}$  to  $OFF$ . However, in order to be inline with the best practices in a given domain, process users need guidelines on how to configure the elements and derive process variants. Examples of such guidelines are:

<sup>1</sup>Please note that the end events in the BPMN notation are modeled with a thick line but are not configurable

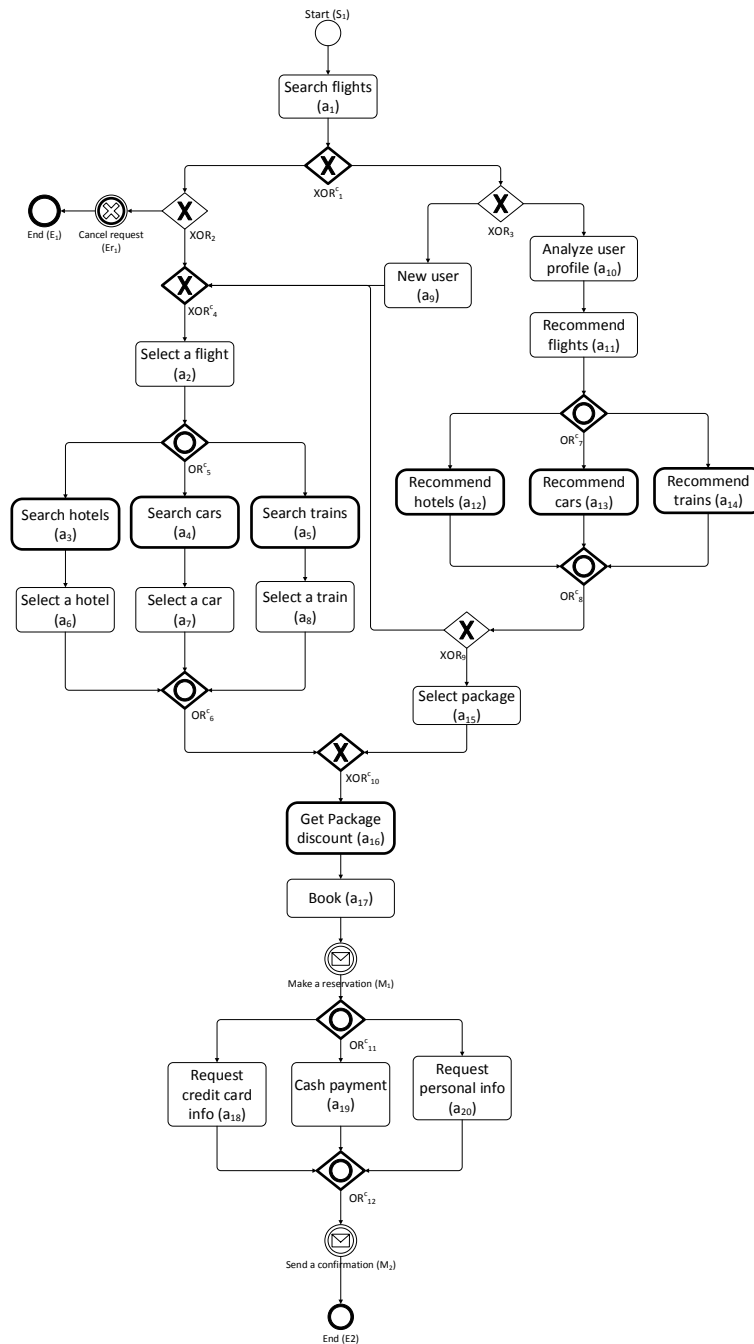


Figure 5.1:  $P^c$ : A configurable travel booking process

1. if the recommendation functionality is included in the derived variant, then the discount offer is also included;

2. if the trains and cars booking are included in the derived variant, then the cash payment is also included;
3. if the hotel booking functionality is included in the derived variant, then the hotel recommendation functionality is also included.

These guidelines are related to the best practices in the travel booking domain and can be formally modeled as *if*→*then* rules where the *if* and *then* parts contain configurations of different configurable elements. Table 5.1 illustrates an excerpt of the configuration guidelines identified for the process in Figure 5.1. The first configuration guideline  $G_1$  states that if the configurable gateway  $XOR_1^c$  is configured to an  $XOR$  with the outgoing branches starting with  $XOR_2$  and  $XOR_3$ , then the configurable activity  $a_{16}$  is configured to  $ON$ . This rule matches the first guideline in the aforementioned example since the recommendation functionality (i.e. branch starting with  $XOR_3$ ) and the discount offer (i.e. activity  $a_{16}$ ) are included. Similarly,  $G_2$  states that if  $a_4$  and  $a_7$  are configured to  $ON$ , then  $OR_{11}^c$  is configured to an  $OR$  with the outgoing branches starting with  $a_{18}$ ,  $a_{19}$  and  $a_{20}$ .  $G_2$  and  $G_3$  match the second guideline in the aforementioned example.

|          |   |
|----------|---|
| $G_1$    | $XOR_1^c = (XOR, \{XOR_2, XOR_3\}) \rightarrow a_{16} = ON$   |
| $G_2$    | $a_4 = ON \wedge a_5 = ON \rightarrow OR_{11}^c = (OR, \{a_{18}, a_{19}, a_{20}\})$                                       |
| $G_3$    | $OR_5^c = (OR, \{a_4, a_5\}) \rightarrow OR_{11}^c = \{AND, \{a_{19}, a_{20}\}\}$   |
| $G_4$    | $XOR_1^c = (Seq, \{XOR_2\}) \rightarrow XOR_{10}^c = (Seq, \{OR_6^c\})$   |
| $G_5$    | $a_4 = ON \rightarrow OR_5^c = (OR, \{a_4, a_5\})$  |
| $G_6$    | $a_5 = ON \rightarrow a_{14} = ON$  |
| $G_7$    | $OR_5^c = \{Seq, \{a_3\}\} \rightarrow a_{12} = ON$   |
| $G_8$    | $OR_5^c = \{Seq, \{a_3\}\} \rightarrow OR_7^c = (Seq, \{a_{12}\})$  |
| $G_9$    | $XOR_1^c = (Seq, \{XOR_3\}) \rightarrow XOR_4^c = (Seq, \{XOR_9\})$   |
| $G_{10}$ | $a_{16} = OFF \rightarrow a_{12} = OFF \wedge a_{13} = OFF \wedge a_{14} = OFF$   |
| $G_{11}$ | $OR_{11}^c = (AND, \{a_{19}, a_{20}\}) \wedge OR_{12}^c = (AND, \{a_{19}, a_{20}\}) \rightarrow a_4 = ON \wedge a_3 = ON$ |
| $G_{12}$ | $OR_{11}^c = (AND, \{a_{19}, a_{20}\}) \rightarrow OR_{12}^c = (AND, \{a_{19}, a_{20}\})$                                 |
| $G_{13}$ | $a_{16} = OFF \rightarrow OR_5^c = (Seq, \{a_3\})$  |

Table 5.1: An excerpt of the configuration guidelines for the configurable process model in Figure 5.1

This simple example raises three main issues. First, since there may exist a large number of interdependencies between different configuration choices in a configurable process model [7, 34], manually identifying the configuration guidelines is a tedious, if not impossible task. Second, this large number of interdependencies results in a high number of configuration guidelines which may confuse the end user and lead to an inconsistent and incorrect application of the guidelines. For example, in Table 5.1, we can observe that  $G_1$  and  $G_4$  cannot be applied concurrently as they result in different configurations for the configurable element  $XOR_1^c$ ; Similarly,  $G_2$  and  $G_3$  result in different configurations for the configurable element  $OR_{11}^c$  and cannot be applied

concurrently;  $G_5$  and  $G_6$  should be applied before  $G_2$ ; and so on. These interdependencies between the configuration guidelines are not intuitive and may become more complex [34]. Thus, they should be explicitly and formally represented to the end users in order to assist them correctly applying them during the configuration process. Third, as there may exist a large number of configurable elements that may have all kind of interdependencies between their configuration choices, end users need to know the order in which the elements are configured.

In order to overcome these issues, we introduce in the next section our configuration guidance model that implements three main functionalities to overcome the above presented limitations: (i) recommending the configuration guidelines, (ii) explicitly represent their dependencies relations and (iii) recommending the order in which the configuration steps are performed.

### 5.3 Configuration Guidance Model

Inspired from variability modeling approaches in software product line engineering [84, 165] and taking into account the requirements identified in section 5.1, we define a configuration guidance model as a tree-like structure with cross-tree relations. An excerpt of the configuration guidance model for the configurable process in Figure 5.1 is illustrated in Figure 5.2.

The tree structure allows a “hierarchical” ordering of the configurable elements of a process model in a parent-child fashion, that is the parent element is configured before the child element. Roughly speaking, the parent-child relation encodes a configuration dependency, i.e. the configuration of the child element is probably unknown until, at least, the parent element is configured. The tree elements are graphically modeled with circles. For example, in Figure 5.2, the configuration of the connectors  $XOR_4^c$  and  $XOR_{10}^c$  depends on that of  $XOR_1^c$ , thus they are child of the parent  $XOR_1^c$ .

Each tree element has multiple configuration choices (Definition 3.3.4) labeled with their probability of selection. Graphically, the configuration choices are modeled with dotted rectangles attached to their configurable elements. For example, in Figure 5.2, the element  $XOR_1^c$  has the configuration choices  $(Seq, \{XOR_2\})$ ,  $(Seq, \{XOR_3\})$  and  $(XOR, \{XOR_2, XOR_3\})$  with probabilities 0.5, 0.3 and 0.2 respectively. This means that, in 50%, 30% and 20% of the previously derived variants, the configurations  $(Seq, \{XOR_2\})$ ,  $(Seq, \{XOR_3\})$  and  $(XOR, \{XOR_2, XOR_3\})$  are respectively selected.

The configuration guidelines represented as cross-tree relations are graphically modeled with dotted lines between the different elements and have also their probability of certainty. The probability of certainty expresses to which extent a configuration guideline is valid. For example, in Figure 5.2, the configuration guideline  $XOR_1^c = (XOR, \{Seq, \{XOR_2\}\}) \rightarrow XOR_4^c = (Seq, \{XOR_2^c\})$  has a probability equal to 0.82. This means that in 82% of the cases, whenever the configuration  $(Seq, \{XOR_2\})$  of  $XOR_1^c$  is selected, the configuration  $(Seq, \{XOR_2^c\})$  of  $XOR_4^c$  is

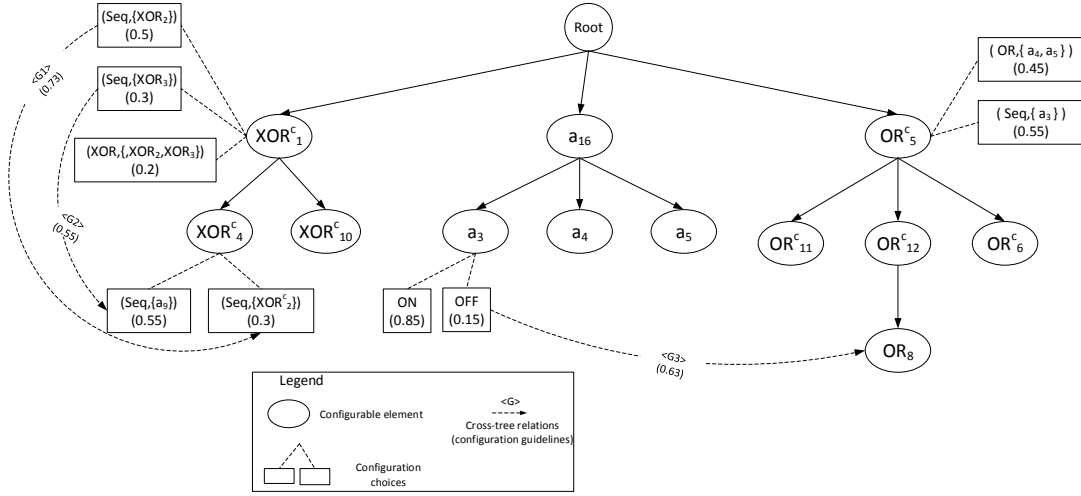


Figure 5.2: An excerpt of the configuration guidance model extracted for the configurable process model in Figure 5.1

also selected.

Besides, a configuration guidance model provides also an explicit representation of the dependencies relations that may exist between the configuration guidelines. We have identified three main relations: causality (i.e. which guidelines can be applied given a set of already applied guidelines), concurrency (i.e. which guidelines can be applied in parallel) and exclusiveness (i.e. which guidelines exclude each others). The dependencies relations are formally represented using Petri-nets in which each transition corresponds to a configuration guideline and the flow relation between them correspond to their dependencies' relations. We refer to the resulted net as *configuration system*. An excerpt of the resulted configuration system for the configuration guidelines in Table 5.1 is illustrated in Figure 5.3. Each trace in the resulted petri-net corresponds to one possible application of the configuration guidelines. For example, one can apply the guidelines (i)  $G_7$  and  $G_8$  then  $G_2$  or (ii)  $G_3, G_{12}, G_{11}$  then  $G_5$ .

Let  $P^c = (N, E, T, L, B, \mathbb{C}^*)$  be a configurable process model. A configuration guidance model  $\mathcal{G}_M^c$  is formally given in Definition 5.3.1.

**Definition 5.3.1** (Configuration guidance Model). *A configuration guidance model is defined as  $\mathcal{G}_M^c = (\mathcal{T}^c, \mathbb{C}^* \mathcal{F}, \mathbb{G}, P_{conf}, P, CS)$  where:*

- $\mathcal{T}^c = (N^c, E)$  is a directed acyclic graph whose underlying undirected graph is a tree where  $N^c \subseteq N$  is the set of configurable elements in  $P^c$  and  $E \subseteq N^c \times N^c$  is the set of edges.
- $\mathbb{C}^*$  is the set of valid configuration choices for the configurable elements;
- $\mathcal{F} : N^c \rightarrow \mathcal{P}(\mathbb{C}^*)$  is a function that maps a tree element  $n \in N^c$  to its valid configurations  $\mathbb{C}_n \subseteq \mathbb{C}^*$ ;

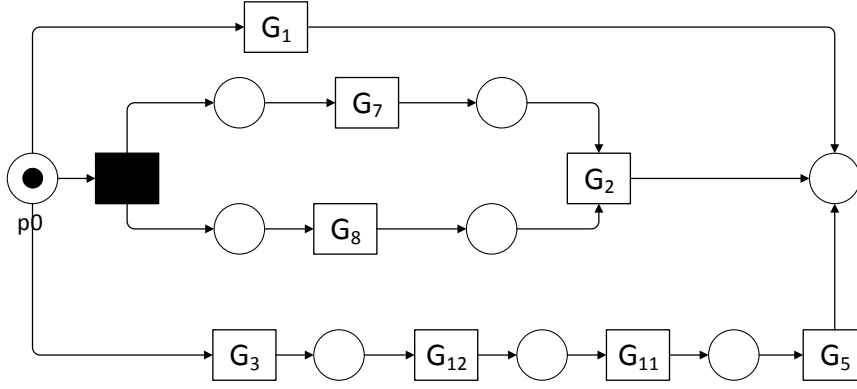


Figure 5.3: An excerpt of the configuration system derived from the configuration guidelines in Table 5.1

- $\mathbb{G}$  is the set of configuration guidelines;
- $P_{conf} : \mathbb{C}^* \rightarrow \mathbb{D}$  is a function that assigns for each configuration choice  $conf \in \mathbb{C}^*$  a probability of occurrence;
- $P : \mathbb{G} \rightarrow \mathbb{D}$  is a function that assigns for each inclusion and exclusion relation a probability of certainty, i.e. to which extent we are certain that a configuration guideline holds;
- $CS = (P, \mathbb{G}, F)$  is the configuration system that formalizes the configuration guidelines dependencies' relations where:
  - $P$  is the set of places;
  - $\mathbb{G}$  is the set of configuration guidelines such that  $P \cap \mathbb{G} = \emptyset$ ;
  - $F \subseteq (P \times \mathbb{G}) \cup (\mathbb{G} \times P)$  is the set of guidelines' relations.

For example, the configuration guidance model in Figure 5.2 is defined as follows:

- $\mathcal{T}^c = (N^c, E)$  where  $N^c = \{XOR_1^c, a_6, OR_5^c, XOR_4^c, \dots\}$ ,  $E = \{(XOR_1^c, XOR_4^c), (XOR_1^c, XOR_{10}^c), (a_{16}, a_3), (a_{16}, a_4), \dots\}$ ;
- $\mathbb{C}^*$  is as defined for  $P^c$ ;
- $\mathcal{F}(XOR_1^c) = \{(Seq, \{XOR_2\}), (Seq, \{XOR_3\}), (XOR, \{XOR_2, XOR_3\})\}$ ;  $\mathcal{F}(a_3) = \{ON, OFF\}$ , etc.;
- $\mathbb{G} = \{G_1, G_2, G_3, \dots\}$  are the configuration guidelines which are cross-tree relations; examples of configuration guidelines are shown in Table 5.1;
- $P_{conf}((Seq, \{XOR_2\})) = 0.5$ ;  $P_{conf}((Seq, \{XOR_3\})) = 0.3$ ; etc.;



- $P(G_1) = 0.73$ ;  $P(G_2) = 0.55$ ; etc.
- $CS$  is the configuration system depicted in Figure 5.3.

In the next section, we present an overview of our approach for extracting configuration guidance models from business process repositories.

## 5.4 Approach Overview

In this section, we present an overview of our automated approach for extracting configuration guidance models from business process repositories. Let  $P^c = (N^c, E^c, T^c, L^c, B, \mathbb{C}^*)$  be a configurable process model and  $\mathbb{P} = \{P_i = (N_i, E_i, T_i, L_i) : i \geq 1\}$  a set of existing process models that are previously derived from  $P^c$ . The processes in  $\mathbb{P}$  can be collected by computing a similarity value (e.g. using behavioral profiles [166]).  $P^c$  and  $\mathbb{P}$  are used as inputs by our algorithm (Algorithm 4) to generate a configuration guidance model  $\mathcal{G}_M^c$ .

---

### Algorithm 4 Building a configuration guidance model

---

- 1: **input:**  $P^c, \mathbb{P}$
  - 2: **output:**  $\mathcal{G}_M^c = (\mathcal{T}^c, \mathbb{C}^*, \mathcal{F}, \mathbb{G}, P_{conf}, P)$   
 {extract configuration guidelines  $\mathbb{G}$  and probability information  $P_{conf}$  and  $P$ }
  - 3:  $E_{conf} = geElementConf(P^c, \mathbb{P})$
  - 4:  $\mathbb{G} = Apriori(E_{conf}, minS, minC)$
  - 5:  $P_{conf}(C_i) = Sup(C_i) : C_i \in \mathbb{C}^*$
  - 6:  $P(G_i) = Sup(G_i) : G_i \in \mathbb{G}$   
 {derive tree hierarchy  $\mathcal{T}^c$ }
  - 7: derive probabilistic dependency matrix  $M_P = getProbabilisticMatrix(\mathbb{G}, 2)$
  - 8: derive implication graph  $G_{\rightarrow} = getImplicationGraph(M_P)$
  - 9: generate tree hierarchy  $\mathcal{T}^c = getTreeHierarchy(G_{\rightarrow})$   
 {Formalize configuration guidelines dependencies}
  - 10:  $TS = generateTS(\mathbb{G})$
  - 11:  $PN = synthesizePN(TS)$
- 

The algorithm proceeds in three main steps. In the first step, the set of configuration guidelines and the probability information are extracted from existing process models in  $\mathbb{P}$  (Lines 3-6 detailed in Section 5.5). Then, in the second step, the tree hierarchy which represents the configuration steps order is derived from the extracted guidelines (Lines 7-9 detailed in Section 5.6). The third step consists of formalizing the dependencies relations between the configuration guidelines to ensure that they are are correctly and consistently applied (Lines 10-11, detailed in Section 5.7).

Since our approach requires as input a configurable process model and an existing business process repository, we assume that a large number of configured process

variants are collected in a business repository. These variants may be similar or derived from the configurable process model and may have undergone further changes according to the companies specific requirements. For example, the process variant in Figure 5.4a is derived from the configurable process model in Figure 5.1 and has undergone some additional changes. Table 5.4b show the configuration choices that have been taken for each configurable element to derive the process variant and the additional changes that have been done after the configuration.

In the following sections, we show how our approach can extract the correct configurations from existing business processes even with the presence of additional changes based on a similarity notion.

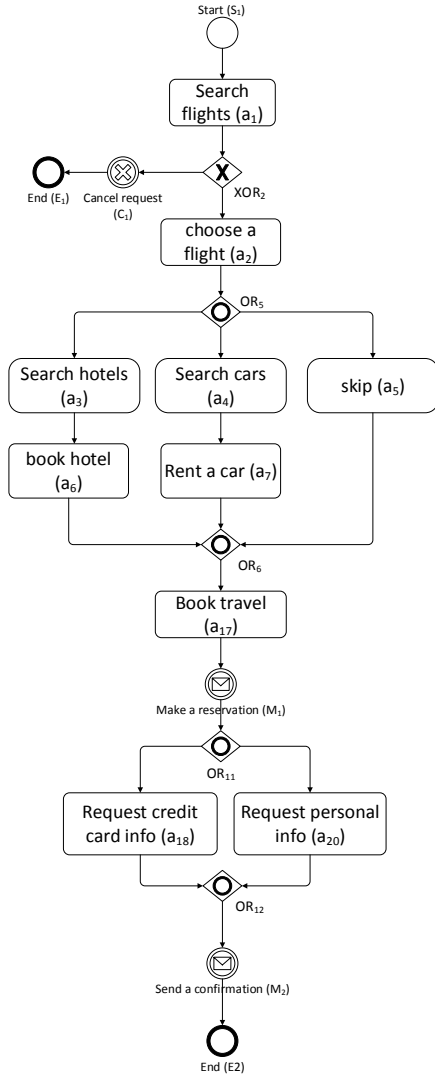
## 5.5 Extracting configuration guidelines from existing process models

The first step of our proposed approach consists of extracting the set of configuration guidelines  $\mathbb{G}$  from existing business process repositories. As mentioned in Section 5.4, our aim is to extract for a configurable process model  $P^c$  its configuration guidelines  $\mathbb{G}$  from a set of process models  $\mathbb{P}$ . The extraction of the configuration guidelines consists of two main steps. First, the set of the elements' configurations in  $P^c$  are extracted from the processes in  $\mathbb{P}$  (see Section 5.5.1). Then, the configuration guidelines are derived from the extracted configurations using association rule mining (see Section 5.5.2).

### 5.5.1 Retrieving process elements' configurations

In this step, we extract from each process variant  $P_i \in \mathbb{P}$  the configurations corresponding to the configurable elements in  $P^c$  and store them in a *configuration matrix*, a data structure suitable for applying association rule mining techniques. For example, let  $P^c$  be the process in Figure 5.1 and  $P_1 \in \mathbb{P}$  be the process in Figure 5.4a. The gateway  $XOR_1^c$  in  $P^c$  has a configuration  $Conf_{XOR_1^c} = (Seq, \{XOR_2\})$  as  $XOR_2$  in  $P_1$  is similar to  $XOR_2$  in  $P^c$ ; Similarly the gateway  $OR_6^c$  has a configuration  $Conf_{OR_6^c} = (OR_6, \{a_6, a_7\})$  in  $P_1$  as the activities  $a_6, a_7 \in P^c$  are similar to the activities  $a_6, a_7 \in P_1$ . This example shows that existing process variants may not contain the same elements as those in the configurable process and therefore retrieving exact configurations is not realistic. Thus, in order to retrieve the elements configurations, we use a similarity metric to match the configurable elements in  $P^c$  and their candidate configurations in the existing processes  $P_i$ . We compute three similarities: the similarity  $Sim_A$  between activities,  $Sim_G$  between gateways and  $Sim_E$  between events.

**Retrieving activities' configuration.** A configurable activity  $a^c \in N^c$  can be configured to *ON* or *OFF*. A configuration  $Conf_{a^c} = ON$  is retrieved from a process variant  $P_i$ , if there exists an activity  $a' \in N_i$  such that  $a'$  is the best activity mapping



(a)  $P_1$ : A process variant derived from the configurable process model in Figure 5.1

| Configurable element           | Element configuration           |
|--------------------------------|---------------------------------|
| $XOR_1^c$                      | $(Seq, \{XOR_2\})$              |
| Cancel request                 | ON                              |
| $XOR_4^c$                      | $(Seq, \{XOR_2\})$              |
| $OR_5^c$                       | $(OR, \{a_3, a_4, a_5\})$       |
| $OR_6^c$                       | $(OR, \{a_6, a_7\})$            |
| Search hotels                  | ON                              |
| Search cars                    | ON                              |
| $XOR_{10}^c$                   | $(Seq, \{OR_6^c\})$             |
| $a_{16}$                       | OFF                             |
| $OR_{11}^c$                    | $(OR_{11}, \{a_{18}, a_{19}\})$ |
| $OR_{12}^c$                    | $(OR_{12}, \{a_{18}, a_{19}\})$ |
| add $a_5$                      |                                 |
| rename $a_2$                   |                                 |
| rename $a_2, a_6, a_7, a_{17}$ |                                 |

(b) The selected elements' configurations and the additional changes

Figure 5.4: A configured process variant derived from the process model in Figure 5.1 and the selected configurations

to  $a$  according to Definition 4.3.1. Otherwise, the configuration  $Conf_{a^c} = OFF$  holds. For example, in our running example in Figures 5.1 and 5.4a, for a  $minSim_A = 0.5$ , the configurable activity  $a_{16}$  in  $P^c$  does not have any best mapping in  $P_1$  thus the configuration  $Conf_{a_{16}} = OFF$  is retrieved from  $P_1$ .

**Retrieving gateways' configuration.** Let  $g^c \in N^c$  such that  $T^c(g^c) = gateway$  and  $g_1 \in N_1$  such that  $T_1(g_1) = gateway$  be two gateways in  $P^c$  and  $P_1$  respectively. The similarity between gateways cannot be computed in the same way as activities since gateways' labels do not have linguistic semantics. Hence, in order to compute the similarity  $Sim_G$  between  $g_1$  and  $g_2$ , we compute a similarity based on (1) the gateways' behavior which can be inferred from the partial order  $\preceq_g$  (Definition 3.2.1) and (2) the similarity between the nodes in their preset and postset (adapted from [43]). The similarity  $Sim_G$  between gateways is computed as:

$$Sim_G(g^c, g_1) = \begin{cases} \frac{|\mathcal{M}(g^c \bullet, g_1 \bullet)| + |\mathcal{M}(\bullet g^c, \bullet g_1)|}{|g^c \bullet| + |\bullet g^c|} & \text{if } g_1 \preceq_g g^c \\ 0 & \text{otherwise} \end{cases} \quad (5.1)$$

where  $|\mathcal{M}(g^c \bullet, g_1 \bullet)|$  (respectively  $|\mathcal{M}(\bullet g^c, \bullet g_1)|$ ) returns the number of best elements' matching (activities, events or gateways) in  $g_1 \bullet$  (respectively  $\bullet g_1$ ) that correspond to those in  $g^c \bullet$  (respectively  $\bullet g^c$ ). We say that  $g_1$  **is the best gateway matching for  $g^c$**  iff:  $Sim_G(g^c, g_1) \geq minSim_G \wedge \nexists g_x \in N_1 : Sim_G(g^c, g_x) > Sim_G(g^c, g_1)$  where  $minSim_G$  is a user specified threshold. For example, in Figure 5.1 and 5.4a, the similarity between  $OR_6^c$  in  $P^c$  and  $OR_6$  in  $P_1$  is  $Sim_G(OR_6^c, OR_6) = \frac{2+0}{1+3} = 0.5$ . For a  $minSim_G = 0.5$ ,  $OR_6$  is the best gateway matching for  $OR_6^c$  as it has the highest similarity value.

A configurable split (respectively join) gateway  $g^c \in N^c$  can be configured with respect to its type and postset (respectively preset) (Definition 3.3.4). A configuration  $Conf_{g^c} = (g_i, g \bullet)^2$  is retrieved from a process variant  $P_i$ :

- if there exists a gateway  $g_i \in N_i$  such that:
  1.  $g_i$  is the best gateway matching for  $g^c$ ,
  2.  $g \bullet$  is the set of elements in  $g^c \bullet$  that have best elements' matching in  $g_i \bullet$  according to Equation 5.1.
- else if there exists an edge  $(e', e'') \in N$  such that  $e''$  is the best element matching for  $e_{post} \in g^c \bullet$  and  $e'$  is the best element matching for  $e_{pre} \in \bullet g^c$ . In this case,  $g^c$  is configured to a *sequence*, i.e.  $g' = Seq$  and  $g \bullet = \{e_{post}\}$ .

For example, for a  $minSim_G = 0.5$ , the configuration  $Conf_{OR_6^c} = (OR_6, \{a_6, a_7\})$  of the configurable connector  $OR_6^c$  in  $P^c$  is retrieved from  $P_1$  since (1)  $OR_6$  in  $P_1$  is the best gateway matching for  $OR_6^c$  in  $P^c$ , and (2)  $a_6, a_7 \in N^c$  have  $a_6, a_7 \in N_1$  as the best activities' matching respectively.

<sup>2</sup>we show the case for a split gateway

**Retrieving events' configurations.** Let  $e^c \in N^c$  such that  $T^c(e^c) = event$  and  $e_1 \in N_1$  such that  $T_1(e_1) = event$  be two events in  $P^c$  and  $P_1$  respectively. The similarity between events is computed based on (i) their behavior which can be inferred from the partial order  $\preceq_e$  (Definition 3.2.2) and (2) the similarity between the nodes in their preset and postset. The similarity  $Sim_E$  between events is computed as:

$$Sim_E(e^c, e_1) = \begin{cases} \frac{|\mathcal{M}(e^c \bullet, e_1 \bullet)| + |\mathcal{M}(\bullet e^c, \bullet e_1)|}{|e^c \bullet| + |\bullet e^c|} & \text{if } e_1 \preceq_e e^c \\ 0 & \text{otherwise} \end{cases} \quad (5.2)$$

where  $|\mathcal{M}(e^c \bullet, e_1 \bullet)|$  (respectively  $|\mathcal{M}(\bullet e^c, \bullet e_1)|$ ) returns the number of best elements' matching (activities, events or gateways) in  $e_1 \bullet$  (respectively  $\bullet e_1$ ) that correspond to those in  $e^c \bullet$  (respectively  $e^c \bullet$ ). We say that  $e_1$  is the **best event matching for  $e^c$**  iff:  $Sim_E(e^c, e_1) \geq minSim_E \wedge \nexists e_x \in N_1 : Sim_E(e^c, e_x) > Sim_E(e^c, e_1)$  where  $minSim_E$  is a user specified threshold. For example, in Figure 5.1 and 5.4a, the similarity between the events  $Er_1$  in  $P^c$  and  $C_1$  in  $P_1$  is  $Sim_E(Er_1, C_1) = \frac{1+1}{1+1} = 1$ . For a  $minSim_E = 0.5$ ,  $C_1$  is the best gateway matching for  $Er_1$  as it has the highest similarity value.

A configurable event  $e^c \in N^c$  can be configured to *enable*, *disable* or to an event  $e$  such that  $e \preceq_e e^c$ . A configuration  $Conf_{e^c} = enable$  is retrieved from a process variant  $P_i$ , if there exists an event  $e' \in N_i$  such that  $e'$  is the best matching to  $e$  and  $L_i(e') = L^c(e^c)$  (i.e.  $e^c$  and  $e'$  have equal labels). In case they have different labels, the configuration  $Conf_{e^c} = e'$  is retrieved. Otherwise, the configuration  $Conf_{e^c} = disable$  holds. For example, in our running example in Figures 5.1 and 5.4a, for a  $minSim_A = 0.5$ , the configurable event  $Er_1$  in  $P^c$  has the event  $C_1$  in  $P_1$  as a best event matching;  $L(Er_1) = L(C_1) = Error$  thus the configuration  $Conf_{Er_1} = enable$  is retrieved from  $P_1$ .

### 5.5.2 Apriori-based approach for deriving configuration guidelines

So far, we extracted for each configurable element in  $P^c$ , the set of its configurations from each process variant  $P_i \in \mathbb{P}$ . The extracted configurations are used as input in order to derive the configuration guidelines using association rule mining techniques. Association rule mining [62] is one of the most important techniques of data mining. It aims to find rules for predicting the occurrence of an item based on the occurrences of other items in a transactional database or other repositories. It has been first applied to the marketing domain for predicting the items that are frequently purchased together. Thereafter, it manifested its power and usefulness in other areas such as web mining [167] and recommender systems [168]. The Apriori algorithm [169] is one of the earliest and relevant proposed algorithms for extracting association rules.

In our work, we also use the Apriori algorithm for deriving our configuration guidelines. In order to be able to apply Apriori, we store our retrieved configurations in a suitable data structure called **configuration matrix** and denoted as  $\mathcal{M}^c$ .  $\mathcal{M}^c$  is a  $p \times m$  matrix where  $p$  is the number of process variants in  $\mathbb{P}$  and  $m$  is the number

| $P_{id}$ | $XOR_1^c$                 | $Er_1$         | $XOR_4^c$               | $OR_5^c$              | $a_3$      | $a_{12}$   |
|----------|---------------------------|----------------|-------------------------|-----------------------|------------|------------|
| $P_1$    | $(Seq, \{XOR_2\})$        | <i>enable</i>  | $(Seq, \{XOR_2\})$      | $(OR, \{a_3, a_4\})$  | <i>ON</i>  | <i>OFF</i> |
| $P_2$    | $(XOR, \{XOR_2, XOR_3\})$ | <i>disable</i> | $(XOR, \{XOR_2, a_9\})$ | $(AND, \{a_3, a_4\})$ | <i>ON</i>  | <i>ON</i>  |
| $P_3$    | $(Seq, \{XOR_3\})$        | <i>disable</i> | $(Seq, \{e_9\})$        | $(Seq, \{a_5\})$      | <i>OFF</i> | <i>OFF</i> |
| ..       | ..                        | ..             | ..                      | ..                    | ..         | ..         |

Table 5.2: An excerpt of a configuration matrix

of configurable elements in  $P^c$ . A row in the configuration matrix corresponds to one process variant in  $\mathbb{P}$ . A column corresponds to one configurable element in  $P^c$ . The  $(i, j)^{th}$  entry contains the configuration of the  $j^{th}$  element in  $P^c$  retrieved from the  $i^{th}$  process  $P_i \in \mathbb{P}$ . An example of the configuration matrix for the configurable process in Figure 5.1 is depicted in Table 5.2. For example, the second row corresponds to the configurations retrieved from the process variant  $P_1$  in Figure 5.4a. A dash “-” denotes that there does not exist a configuration in the corresponding process.

Taking the configuration matrix as input, the Apriori algorithm proceeds in two phases. In the first phase, the set of frequent *correlated configurations* (i.e. configurations that often appear together in the same row in the configuration matrix) are discovered according to a frequency threshold. The Apriori algorithm uses the monotonicity property that all subsets of a frequent correlated set are also frequent. Therefore, Apriori starts by selecting the frequent single configurations, then generates the candidate pairs of configurations (i.e. correlated sets of two configurations) from the frequent singles and so on, until it finds all possible correlated configurations according to the frequency metric. It uses *Support*, a well known metric to compute the frequency of a set of correlated configurations. The support is defined as the fraction of process variants in which the correlated configurations appear together. Let  $\mathcal{C} = \{Conf_i : 1 \leq i \leq k\}$  be a set of  $k$ -correlated configurations. The support is computed as:

$$Sup = \frac{|P_{\mathcal{C}}|}{|\mathbb{P}|} \quad (5.3)$$

where  $|P_{\mathcal{C}}|$  is the number of process variants in  $\mathbb{P}$  that contain the configurations in  $\mathcal{C}$  and  $|\mathbb{P}|$  is the number of process variants in  $\mathbb{P}$ . A support is equal to 1 if all the process variants in the process repository contain the correlated configurations. A support is equal to 0 if none of the process variants contain the corresponding configurations together. A set of correlated configurations is frequent if its support is above a given threshold *minSupp*.

In the second step of the algorithm, the set of relevant configuration guidelines in the form of  $LHS \rightarrow RHS$  are derived from the frequent correlated configurations. For example, for a frequent correlated configuration set  $\mathcal{C} = \{(Seq, \{XOR_2\}), (Seq, \{a_3\}), ON\}$ , multiple possible configuration guidelines exist such as  $(Seq, \{XOR_2\}) \wedge (Seq, \{a_3\}) \rightarrow ON$ ;  $(Seq, \{XOR_2\}) \rightarrow (Seq, \{a_3\}) \wedge ON$ ;  $(Seq, \{a_3\}) \rightarrow ON \wedge (Seq, \{XOR_2\})$ , etc. In order to keep only relevant guidelines, the *confidence* metric is computed to evaluate the probability of occurrence of a guideline. The confidence of a config-

uration guideline  $G : LHS \rightarrow RHS$  is defined as the probability of occurrence of the configurations in the right-hand side  $RHS$  given that the configurations in the left-hand side  $LHS$  are selected. It is computed as:

$$C = \frac{Sup_{(RHS \cup LHS)}}{Sup_{RHS}} \quad (5.4)$$

where  $Sup_{(RHS \cup LHS)}$  is the support of the configurations in the right-hand and left-hand sides of  $G$  and  $Sup_{RHS}$  is the support of the configurations in the right-hand side. A confidence is equal to 1 if whenever the configurations in the right-hand side are selected, then the configurations in the left-hand side are also selected. A configuration guideline is relevant if its confidence is above a given confidence threshold  $minConf$ . An example of a subset of the configuration guidelines returned by Apriori for a support  $S = 0.2$  and a confidence  $C = 0.5$  is given in Table 5.1.

## 5.6 Inferring Configuration Steps order

In this section, we present our approach for deriving the tree hierarchy  $\mathcal{T}^c$  of the configuration guidance model  $\mathcal{G}_{\mathcal{M}}^c$ . The tree hierarchy  $\mathcal{T}^c$  consists of parent-child relations between the configurable elements. An element  $n_1^c$  is a candidate parent of a child element  $n_2^c$  if the configuration of  $n_2^c$  highly depends on that of  $n_1^c$ . The dependencies relations between the configurable elements can be derived from their configuration choices. In fact, the more are their configuration choices dependent, the more are the configurable elements dependent. The dependency of a configuration choice  $conf_2$  on another configuration choice  $conf_1$  corresponds to their conditional probability  $P(conf_2|conf_1)$  which can be derived from the confidence of their configuration guideline  $conf_1 \rightarrow conf_2 \in \mathbb{G}$ . It is computed as:

$$P(conf_2|conf_1) = \frac{P(conf_1 \cap conf_2)}{P(conf_2)} = \frac{Sup(conf_1 \cup conf_2)}{Sup(conf_2)} = C(conf_1 \rightarrow conf_2) \quad (5.5)$$

where  $P(conf_1 \cap conf_2)$  is the probability of co-occurrence of  $conf_1$  and  $conf_2$ ;  $P(conf_2)$  is the probability of occurrence of  $conf_2$ . The probabilities are derived from the support metric computed by Apriori. Having the dependencies probabilities between the configuration choices, the conditional probability between two configurable elements  $n_1^c$  and  $n_2^c$  is computed as:

$$P(n_2^c|n_1^c) = \frac{\sum_j P(conf_{n_2^c} | n_1^c)}{\#conf_{n_2^c}} = \frac{\sum_j \frac{\sum_i P(conf_{n_2^c} | conf_{n_1^c_i})}{\#conf_{n_1^c_i}}}{\#conf_{n_2^c}} \quad (5.6)$$

where  $P(n_2^c|n_1^c)$  is the average of the conditional probabilities between the configuration choices of  $n_1^c$  and  $n_2^c$ .  $\sum_j P(conf_{n_2^c} | n_1^c)$  is the sum of the conditional probabilities between each configuration choice  $conf_{n_2^c_j}$  of  $n_2^c$  and the configurable element  $n_1^c$ . The probability  $P(conf_{n_2^c_j} | n_1^c)$  is in turn defined as the average of the

conditional probabilities between the configuration choice  $conf_{n_{2_j}^c}$  and each configuration choice  $conf_{n_{1_i}^c}$  of  $n_1^c$ . It can be computed by dividing the sum of the conditional probabilities between  $conf_{n_{2_j}^c}$  and each  $conf_{n_{1_i}^c}$  of  $n_1^c$  by the number of  $conf_{n_{1_i}^c}$  such that  $P(conf_{n_{2_j}^c}|conf_{n_{1_i}^c}) \neq 0$ ;  $\#conf_{n_{2_j}^c}$  is the number of the configuration choices of  $n_2^c$  such that  $P(conf_{n_{2_j}^c}|n_1^c) \neq 0$ . For example, the probability  $P(OR_8|a_{10})$  can be computed from the configuration guidelines in Table 5.4b as:  $P(XOR_1^c|a_6) = ((P(a|ON) + P(a|OFF))/2 + P(a'|ON) + P(a'|OFF))/3$  where  $a = (XOR, \{XOR_2, XOR_3\})$ ,  $a' = (Seq, \{XOR_2\})$  and  $a'' = (Seq, \{XOR_3\})$  are the configuration choices of  $XOR_{10}^c$ .

The conditional probabilities between each pair of configurable elements are computed and stored in a *dependency probabilistic matrix* denoted as  $M_P$ .  $M_P$  is a  $m \times m$  matrix where  $m$  is the number of configurable elements. An entry  $(i, j)$  in  $M_P$  corresponds to the conditional probability  $P(n_j^c|n_i^c)$  where  $n_j^c$  is the element in the  $j^{th}$  column and  $n_i^c$  is the element in the  $i^{th}$  row. We say that a configurable element  $n_2^c$  depends on another element  $n_1^c$  denoted as  $n_1^c \rightarrow n_2^c$  iff  $P(n_2^c|n_1^c) \geq minP$  where  $minP$  is a given threshold.

The derived dependencies' relations with their probabilities are modeled in a graph, called *implication graph*  $G_{\rightarrow}$  [170]. The nodes in  $G_{\rightarrow}$  correspond to the configurable elements. A weighted edge exists from a node  $n_1^c$  to  $n_2^c$  iff  $n_1^c \rightarrow n_2^c$ ; the edge's weight is the probability  $P(n_2^c|n_1^c)$ . An excerpt of  $G_{\rightarrow}$  derived from a set of dependencies relations is illustrated in Fig. 5.5a. Having  $G_{\rightarrow}$ , the tree hierarchy corresponds

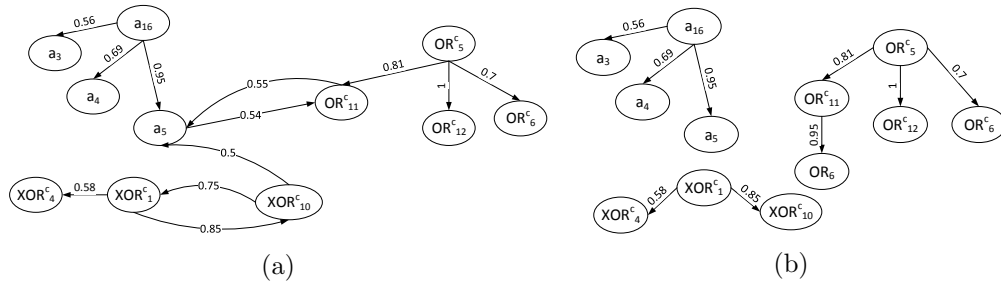


Figure 5.5: (a) An implication graph and (b) its derived optimal spanning tree

to extracting a spanning tree (called arborescence for directed graphs) [170]. Since, there exist multiple possible spanning trees, we aim at deriving the optimal hierarchy that maximizes the dependencies' relations weights. The problem can be mapped to finding the *minimal spanning tree* which can be solved using existing algorithms such as Edmonds' algorithm [171] and efficient implementations such as [65]. Figure 5.5b illustrates an excerpt of the optimal spanning tree extracted from the implication graph in Fig. 5.5a which contains multiple trees. In this case, an artificial root node is added and connected to them in order to obtain the tree hierarchy in Figure 5.2.



## 5.7 Formalizing Configuration Guidelines Dependencies Relations

In this section, we complete our approach for deriving a configuration system from the extracted configuration guidelines. Our aim is to formalize the dependencies' relations between the configuration guidelines in order to assist users incrementally applying them in a consistent and valid way. Let  $\mathbb{G} = \{G_i : i \geq 1\}$  be the set of the extracted configuration guidelines for a configurable process model  $P^c = (N^c, E^c, T^c, L^c, B, \mathbb{C}^*)$ . In the following sections, an approach based on the Theory of Regions is proposed to derive the configuration system. To do so, we first generate a transition system from the configuration guidelines in  $\mathbb{G}$  (Section 5.7.1). Then, we use the Theory of Regions to synthesize a Petri-net (Section 5.7.2).

### 5.7.1 Deriving a transition system from configuration guidelines

In order to build a transition system that explicitly shows the process configuration states resulting from the application of the configuration guidelines and all possible transitions between them, we need to identify (1) the process configuration states, i.e. the states in which a process can be as a result of applying the configuration guidelines and (2) the transitions between the states labeled with the configuration guidelines, i.e. which configuration guideline leads from one state to another.

A *process configuration state* represents the set of selected elements' configuration at an instant  $t$ . It is formally defined as a  $m$ -dimensional vector, where  $m$  is the number of configurable elements in  $P^c$  and each entry in the vector represents a selected configuration of one configurable element. An entry is set to “-” if no configuration is selected for the corresponding configurable element.

**Definition 5.7.1** (process configuration state  $\mathbb{V}^c$ ). *A process configuration state of a configurable process  $P^c$  is a  $m$ -dimensional vector denoted as  $\mathbb{V}^c$  where:*

1.  $m = |n \in N^c : B(n) = \text{true}|$ , i.e.  $m$  is the number of configurable elements in  $P^c$ ;
2.  $\forall i : 1 \leq i \leq m, \mathbb{V}^c[i] \in \{\text{Conf}_{a_i}, \text{“-”}\}$  where  $a_i \in N^c \wedge B(a_i) = \text{true} \wedge \text{Conf}_{a_i} \in \mathbb{C}_{a_i}$ ; each entry in the vector represents a configuration of one configurable element in  $P^c$ . A “-” denotes that the corresponding element is not configured yet.

For example, in our running example in Figure 5.1, if at some instant of the configuration process, the configurable elements  $XOR_1^c$  and  $XOR_4^c$  are configured to  $\text{Conf}_{XOR_1^c} = (\text{Seq}, \{XOR_3\})$  and  $\text{Conf}_{XOR_4^c} = (\text{Seq}, \{a_9\})$  respectively, then the resulted process configuration state is  $\mathbb{V}_1^c = [(\text{Seq}, \{XOR_3\}), (\text{Seq}, \{e_9\}), -, -, \dots, -]$  where each entry in the vector corresponds to the configuration of one configurable

element in  $P^c$  (i.e. the size of the vector is equal to the number of configurable elements in  $P^c$ ).

A configuration guideline  $G : LHS \rightarrow RHS$  can be triggered if there exists a state, called pre-configuration state, in which the configurations in the  $LHS$  part are selected. If triggered, a new configuration state, called post-configuration state, in which the configurations in the  $RHS$  are added to the already selected configurations is resulted. The pre-and post-configuration states of a configuration guideline  $G \in \mathbb{G}$  are formally given in Definition 5.7.2.

**Definition 5.7.2** (pre- and post- configuration states). *A pre-configuration state of a configuration guideline  $G$  denoted as  $\mathbb{V}_{G.pre}^c$  is defined as:*

1.  $\forall conf \in LHS, \exists 1 \leq i \leq |\mathbb{V}_{G.pre}^c| : \mathbb{V}_{G.pre}^c[i] = conf;$
2.  $\exists conf \in RHS : \forall 1 \leq i \leq |\mathbb{V}_{G.pre}^c|, \mathbb{V}_{G.pre}^c[i] \neq conf.$

*A post-configuration state of  $G$  denoted as  $\mathbb{V}_{G.post}^c$  is defined as:*

1.  $\forall \mathbb{V}_{G.pre}^c[i] \neq \text{“-”}, 1 \leq i \leq |\mathbb{V}_{G.pre}^c| : \mathbb{V}_{G.post}^c[i] = \mathbb{V}_{G.pre}^c[i];$
2.  $\forall conf \in RHS, \exists 1 \leq i \leq |\mathbb{V}_{G.post}^c| : \mathbb{V}_{G.post}^c[i] = conf.$

The first requirement in Definition 5.7.2 for  $\mathbb{V}_{G.pre}^c$  states that all the configurations in the left-hand side  $LHS$  of  $G$  should be selected in  $\mathbb{V}_{G.pre}^c$ . The second requirement prevents the case in which the pre- and post-configuration states of  $G$  are equal. The first requirement for  $\mathbb{V}_{G.post}^c$  states that  $\mathbb{V}_{G.post}^c$  has the same selected configurations as  $\mathbb{V}_{G.pre}^c$ . The second requirement, states that the configurations in the  $RHS$  part of  $G$  are set in  $\mathbb{V}_{G.post}^c$ . For example, let  $\mathbb{V}^c = [-, \dots, a_4 = ON, \dots, a_5 = ON, -]$  be a process configuration state for the process in Figure 5.1.  $\mathbb{V}^c$  is a pre-configuration state of the configuration guidelines  $G_2, G_5$  and  $G_6$  in Table 5.1. The post-configuration state of  $G_2$  is  $\mathbb{V}_{G_2.post}^c = [-, \dots, a_4 = ON, \dots, a_5 = ON, \dots, (OR, \{a_{18}, a_{19}, a_{20}\}), ..]$ . A configuration guideline may have multiple possible pre- and post-configurations states. For example, all configuration states having the configurations  $a_4 = ON$  and  $a_5 = ON$  are possible pre-configuration states of  $G_2, G_5$  and  $G_{12}$ . Thus, we denote by  $Pre_G$  and  $Post_G$  the sets of the pre- and post-configuration states of  $G$  respectively.

A transition labeled with a configuration guideline  $G \in \mathbb{G}$  exists from a process configuration state  $\mathbb{V}_1^c$  to another state  $\mathbb{V}_2^c$  iff  $\mathbb{V}_1^c \in Pre_G$  and  $\mathbb{V}_2^c \in Post_G$ . Therefore, a transition system derived from the set of configuration guidelines  $\mathbb{G}$  represents the set of pre- and post-configuration states of each guideline  $G \in \mathbb{G}$  and the transitions between them labeled with the corresponding guidelines. Formally, it is defined as:

**Definition 5.7.3** (Transition system  $TS_{\mathbb{G}}$ ). *A labeled transition system  $TS_{\mathbb{G}} = (S, T, \mathbb{G}, S_i, S_f)$  is derived from the guidelines in  $\mathbb{G}$  such that:*

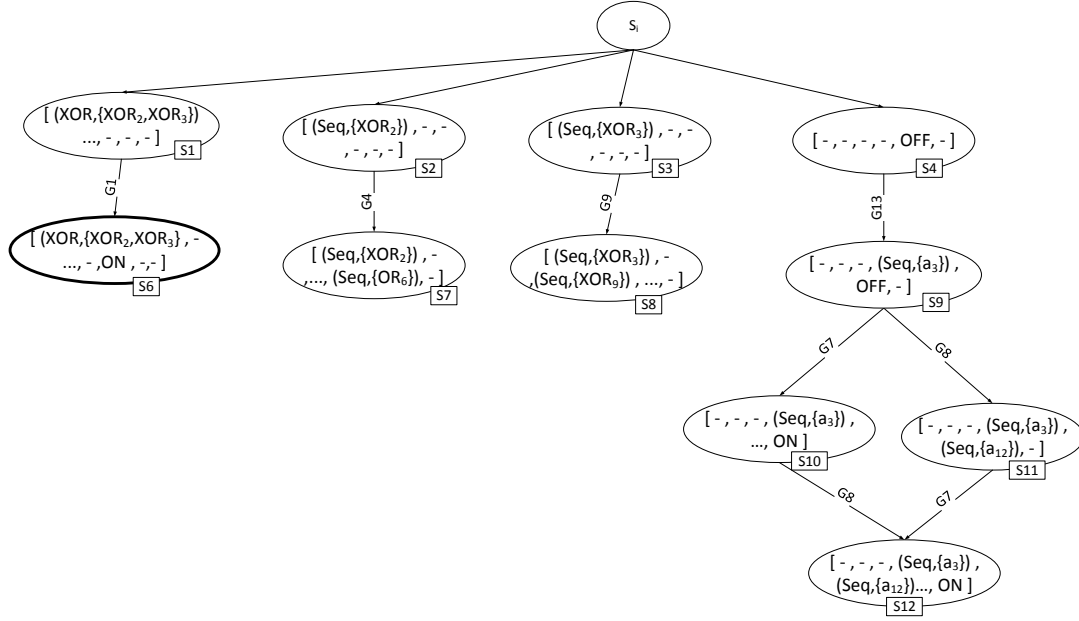


Figure 5.6: An excerpt of a transition system  $TS_{\mathbb{G}}$  derived from the configuration guidelines in Table 5.1

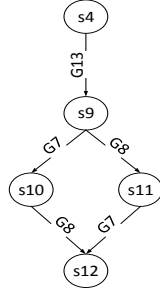


Figure 5.7: Transition system

|       |                      |
|-------|----------------------|
| $R_1$ | $\{s_4\}$            |
| $R_2$ | $\{s_9, s_{11}\}$    |
| $R_3$ | $\{s_9, s_{10}\}$    |
| $R_4$ | $\{s_{10}, s_{12}\}$ |
| $R_5$ | $\{s_{11}, s_{12}\}$ |

Figure 5.8: Regions

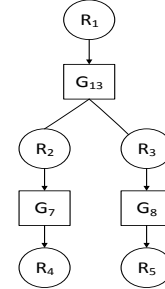


Figure 5.9: Synthesized petri net

- $S = Pre_G \cup Post_G : G \in \mathbb{G}$  is the set of process configuration states;
- $\mathbb{G}$  is the set of transitions' labels;
- $T = \{(s, G, s') \in S \times \mathbb{G} \times S\}$  is the set of relations connecting two states such that  $s \in Pre_G \wedge s' \in Post_G$ ;
- $S_i \in S$  is the initial state such that  $in(S_i) = \phi$  where  $in(s)$  returns the set of states in the incoming branches of  $s \in S$ ;
- $S_f \subseteq S$  is the set of final states such that  $\forall s_f \in S_f : out(s_f) = \phi$  where  $out(s)$

returns the set of states in the outgoing branches of  $s \in S$ .

An excerpt of the transition system derived from the configuration guidelines in Table 5.1 is illustrated in Figure 5.6. The state  $S_i$  is an artificial added start state that is connected to the states initially with no incoming branches (According to Definition 5.7.3, a transition system  $TS_{\mathbb{G}}$  has one start state). The final states are modeled with thick lines. For instance, there exists a transition from the process configuration state  $S_1$  to  $S_6$  labeled with the guideline  $G_1$  as  $S_1$  and  $S_6$  belong to its pre- and post-configuration states respectively. Since  $TS_{\mathbb{G}}$  is a directed acyclic graph (see A for a proof), it is always possible to define the initial and final states.

### 5.7.2 Deriving a Petri-Net using Theory of Regions

Having the transition system  $TS_{\mathbb{G}}$  as input, we use the Theory of Regions [64] in order to derive a configuration system  $CS$  represented as a Petri net. The theory of regions is well known for net synthesis and has been first applied to the synthesis of Petri nets. First, we give the classical definition of a region adapted from [172]. More details on the region extraction algorithms can be found in [64, 173, 174].

**Definition 5.7.4** (Region). *Let  $TS_{\mathbb{G}} = (S, T, \mathbb{G}, S_i, S_f)$  be a transition system derived from the guidelines in  $\mathbb{G}$  and  $S' \subseteq S$  be a subset of states.  $S'$  is a region, if for each configuration guideline  $G \in \mathbb{G}$  one of the following conditions hold:*

1. *all the transitions  $(s_1, G, s_2)$  enters  $S'$ , i.e.  $s_1 \in S' \wedge s_2 \notin S'$ . We say that  $S'$  is a post-region of  $G$ ;*
2. *all the transitions  $(s_1, G, s_2)$  exist  $S'$ , i.e.  $s_1 \notin S' \wedge s_2 \in S'$ . We say that  $S'$  is a pre-region of  $G$ ;*
3. *all the transitions  $(s_1, G, s_2)$  do not cross  $S'$ , i.e.  $s_1, s_2 \in S' \vee s_1, s_2 \notin S'$ .*

Two trivial regions,  $\phi$  (the empty region) and  $S$  (the region consisting of all states) exist in any transition system  $TS_{\mathbb{G}}$ . In the remainder we only consider non-trivial regions. A fragment of a transition system extracted from the one in Figure 5.6 and its set of regions are illustrated in Figures 5.7 and 5.8 respectively. For example,  $R_1$  satisfies the second requirement in Definition 5.7.4 since all transitions labeled with  $G_{13}$  (i.e.  $(s_4, G_{13}, s_9)$ ) exit it;  $R_1$  is a pre-region of  $G_{13}$ ;  $R_5$  satisfies the first requirement since all transitions labeled with  $G_8$  (i.e.  $(s_9, G_8, s_{11})$  and  $(s_{10}, G_8, s_{12})$ ) enter it;  $R_5$  is a post-region of  $G_8$ .

A region  $r'$  is a *sub-region* of a region  $r$  if  $r' \subset r$ . For instance, in our example in Figures 5.7 and 5.8,  $R_1$  and  $R_2$  are sub-regions of the region  $R = \{\{\}, \{s_4\}, \{s_9, s_{11}\}\}$ . A region  $r$  is *minimal* if there is no other region  $r'$  which is a sub-region of it. For example, the regions in our example in Figure 5.8 are minimal regions.

A configuration system  $CS$ , represented as a Petri net, is derived from  $TS_{\mathbb{G}}$  using the Theory of Regions through the following steps:

1. each minimal region corresponds to a place in  $CS$ ;
2. each configuration guideline (i.e. transition label in  $TS_{\mathbb{G}}$ ) corresponds to a guideline  $G \in \mathbb{G}$  in  $CS$ ;
3. The flow relations of  $CS$  are built in the following way: for each place  $p_i$ ,  $G \in p_i \bullet$  if there exists a region  $r_i$  such that  $r_i$  is a pre-region of  $G$ ; and  $G \in \bullet p_i$  if  $r_i$  is a post-region of  $G$ .

An example of a Petri net derived from the transition system in Figure 5.7 is given in Figure 5.9. The above presented steps for Petri net synthesis work well for a special class of transition systems, called *elementary transition systems*. In [175, 176], the algorithm is generalized for handling any transition system.

## 5.8 Conclusion

In this chapter, we answered the question raised in the thesis problematic ( section 1.2.2), which is: *How to assist the creation of configuration support systems?* and its two sub-questions which are: *How to assist the identification of domain constraints?* and *How to assist the identification of configuration steps order?*

To assist the creation of configuration support systems, we introduced our *configuration guidance model* which aims at providing relevant recommendations for the creation of configuration support systems. These recommendations are mainly related to (i) the configuration guidelines that need to be integrated in a configuration support system and (ii) the order in which the configuration steps are performed. They are provided to answer the two sub-questions on how to assist the identification of domains constraints and how to assist the identification of configuration steps order.

To construct our model, we proposed an automated approach that learns from the previous experience in process modeling and configuration. We used Data Mining techniques in particular Association Rule Mining to extract configuration guidelines from existing business process models. These guidelines reveal how the configuration choices are interrelated in a configurable process model. In order to ensure a correct and consistent application of the guidelines, we proposed to formalize their dependencies' relations using Petri-nets. Finally, we proposed to infer the order in which the configuration steps are performed using Graph theory in particular the derivation of optimal spanning trees.

Our principles presented in Section 1.4.1 are respected:

- **Automation:** We propose an automated approach to extract configuration guidance models from business process repositories. We do not ask the process provider for any additional manual effort.
- **Implicit knowledge exploitation:** We exploit the information hidden in process models to infer the functionalities of a configuration guidance model.

We extract configuration guidelines and infer the configuration steps order from the extracted guidelines.

- **Focused results:** Our configuration guidance model assists step by step the process provider and recommends elements' configurations instead of entire process configurations.
- **Balanced computation:** To derive the configuration guidelines, we match and extract only the elements' configurations instead of entire process models from existing business process repositories. Our computation time is bounded by the used techniques such as the derivation of association rules and of optimal spanning trees for which efficient algorithms have been developed in the literature [169, 171, 177].

Our approach is complementary to the existing domain-based ones. Along with the domain expert knowledge, it improves the quality of created configuration support systems by (i) integrating the users' experience in process configuration and (ii) reducing time and manual effort.



# Using event Logs for Configurable Process Design and Configuration

## Contents

---

|            |  |            |
|------------|--|------------|
| <b>6.1</b> | <b>Introduction</b>  | <b>117</b> |
| <b>6.2</b> | <b>Deriving Configurable Process Fragments from Event Logs</b> | <b>120</b> |
| 6.2.1      | Extracting Log-based Neighborhood Contexts                     | 121        |
| 6.2.2      | Mining Configurable Process Fragments                          | 123        |
| <b>6.3</b> | <b>Mining Ranked Configuration Guidelines</b>                  | <b>125</b> |
| 6.3.1      | A frequency suffix tree for configuration executions           | 125        |
| 6.3.2      | Deriving ranked configuration guidelines                       | 128        |
| <b>6.4</b> | <b>Conclusion</b>  | <b>131</b> |

---

## 6.1 Introduction

In this chapter, we present an approach that builds upon event logs to assist the design and configuration of configurable process models using *event logs*. We exploit event logs because of four reasons:

1. They exist in all transactional information systems such as ERP, CRM, or workflow management systems [127]. So, they are large resources that are commonly available.
2. Business process models do not always exist. For example, in case of the flow of patients in a hospital, all activities are logged but information about the underlying process is typically missing [127]. In this case, our previous techniques presented in the previous chapters cannot be applied. Therefore, we propose to use logs as an alternative input.



3. They present the reality of the business process execution. They include the activity execution frequency, which is useful information to identify the importance of a particular configuration and is not presented by an a-priori process model.
4. They contain useful information that can be discovered to assist the business process design and diagnosis. For example, they can be mined to check the conformance of a-priori business process models [93, 128], to detect execution errors [129, 130] or to observe social behaviors between users or services [131].

Given a collection of event logs for different variants of the same process, we propose to discover, for a selected position in a process graph, a configurable process fragment. Then, we propose a frequency-based approach that guides the configuration of the discovered fragment with *ranked configuration guidelines*. Thanks to these guidelines, a process user is interactively assisted with recommendations on the suitable elements' configurations that are frequently executed together.

Concretely, we define and capture the *log-based neighborhood context* from a collection of event logs. We merge the extracted log-based neighborhood contexts into one log and propose to discover a single process fragment that describes the behavior of all merged logs. Configurable elements are then identified using the notion of log-based *shared* and *unshared* activities. To derive ranked configuration guidelines, we propose to explore and model the activities' execution importance reflected by their frequency of appearance in the merged logs using *suffix trees* [66, 178]. Suffix trees are efficient data structure that provide linear-time solutions for pattern matching problems. Then, using concepts from Set theory, we derive the elements' configurations and rank their execution frequency from the suffix tree. The ranked configuration executions are presented as guidelines to the process user and are updated after each step to take into account the already chosen ones.

We reuse our motivating example presented in Section 1.3 to illustrate our approach. We assume that a process provider is designing a travel booking process (Figure 6.1). He is using our approach for having recommendations on the configurable fragments to fill-in the missing parts in the process. He selects the activity  $a_4$  (Request credit card info), set the number of considered layers to  $k = 2$  and asks the system for recommendation.

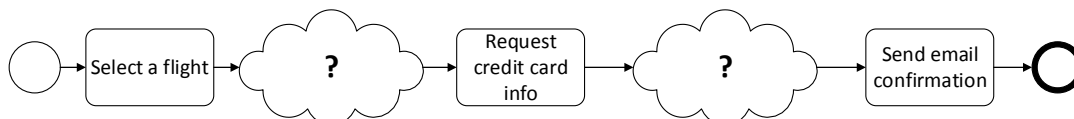


Figure 6.1: An ongoing design of a travel booking process

We also assume that there exists a collection of event logs that record the exe-

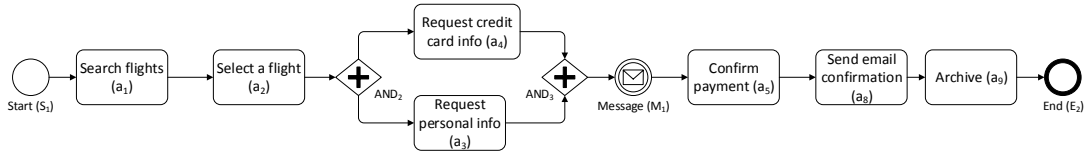
cution of different processes from which we show two excerpts of event logs <sup>1</sup>  $L_1$  in Table 6.1 (which corresponds to the process variant in Figure 6.2a) and  $L_2$  in Table 6.2 (which corresponds to the process variant in Figure 6.2b). Each activity may be executed in different logs and each trace may be executed multiple times. For example, the execution of the activities  $a_1$  and  $a_2$  are recorded in  $L_1$  and  $L_2$ ; the trace  $\langle a_1, a_2, a_4, a_3, a_5, a_8, a_9 \rangle$  is executed 4 times (Trace ID = 1, 3, 4, 5, 6).

| Trace ID | Log traces  |
|----------|---|
| 1        | $\langle a_1, a_2, a_4, a_3, a_5, a_8, a_9 \rangle$ |
| 2        | $\langle a_1, a_2, a_3, a_4, a_5, a_8, a_9 \rangle$ |
| 3        | $\langle a_1, a_2, a_4, a_3, a_5, a_8, a_9 \rangle$ |
| 4        | $\langle a_1, a_2, a_4, a_3, a_5, a_8, a_9 \rangle$ |
| 5        | $\langle a_1, a_2, a_4, a_3, a_5, a_8, a_9 \rangle$ |
| 6        | $\langle a_1, a_2, a_4, a_3, a_5, a_8, a_9 \rangle$ |
| 7        | $\langle a_1, a_2, a_3, a_4, a_5, a_8, a_9 \rangle$ |

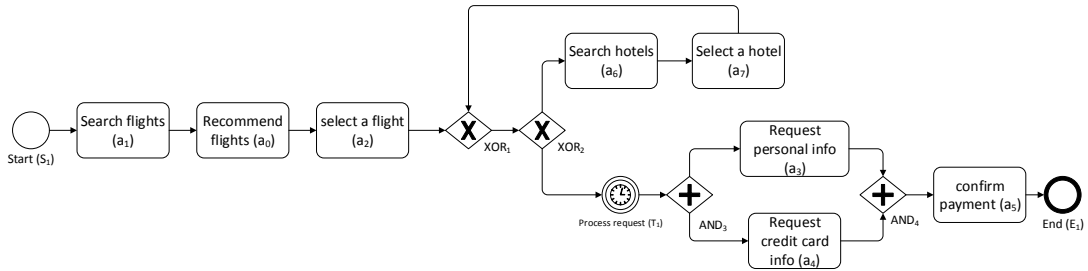
Table 6.1:  $L_1$ : Event log of the process variant in Figure 6.2a

| Trace ID | Log traces   |
|----------|--|
| 1        | $\langle a_1, a_0, a_2, a_6, a_7, a_3, a_4, a_5 \rangle$ |
| 2        | $\langle a_1, a_0, a_2, a_6, a_7, a_4, a_3, a_5 \rangle$ |
| 3        | $\langle a_1, a_0, a_2, a_4, a_3, a_5 \rangle$           |
| 4        | $\langle a_1, a_0, a_2, a_3, a_4, a_5 \rangle$           |
| 5        | $\langle a_1, a_0, a_2, a_6, a_7 \rangle$                |
| 6        | $\langle a_1, a_0, a_2, a_6, a_7 \rangle$                |
| 7        | $\langle a_1, a_0, a_2, a_6, a_7 \rangle$                |
| 8        | $\langle a_1, a_0, a_2, a_3, a_4, a_5 \rangle$           |
| 9        | $\langle a_1, a_0, a_2, a_4, a_3, a_5 \rangle$           |

Table 6.2:  $L_2$ : Event log of the process variant in Figure 6.2b



(a)  $BP_1$ : A flight booking process variant



(b)  $BP_2$ : A flight and hotel booking process variant

Using our approach, we recommend a configurable fragment discovered from the collection of event logs. We also discover a set of ranked configuration guidelines for assisting the configuration of the configurable elements. An example of the discovered fragment and its configuration guidelines are illustrated in Figure 6.3. The fragment

<sup>1</sup>The event logs are shown in the form of multisets of traces and we assume that no error occurs during the business process execution

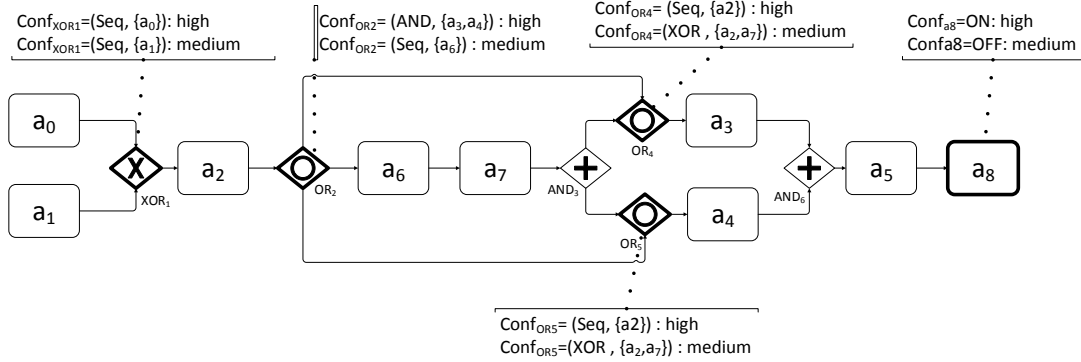


Figure 6.3: The discovered configurable fragment of the activity  $a_4$  within 2-layers and the ranked configuration guidelines attached to the configurable elements as text annotations

includes the selected activity  $a_4$ , its activities' neighbors and their relations through configurable elements. The discovered configuration guidelines are attached to the configurable elements in text annotations. They depict the rank of the elements' configurations which reflect their importance, in terms of frequency of execution in the logs. These guidelines are updated after each configuration step in order to take into account the already selected ones.

The work in this chapter was published in conference proceedings [179].

This chapter is organized as follows: In Section 6.2, we elaborate our technique for deriving configurable process fragments from a collection of event logs. Section 6.3 presents our approach to mine ranked configuration guidelines for the discovered configurable fragment. Finally, we conclude the chapter in Section 6.4.

## 6.2 Deriving Configurable Process Fragments from Event Logs

This section elaborates our approach to mine a configurable process fragment out of collections of event logs. Let  $\mathbb{L} = \{L_i \in \mathbb{M}(A_i^*) : i \geq 1\}$  be a collection of event logs where  $A_i$  is a set of activities in some universe of activities  $\mathcal{U}_A$ ;  $a_x$  and  $k$  are the activity and the desired number of layers selected by the process designer.

Algorithm 5 illustrates the different steps to discover a configurable fragment  $P^c = (N, E, T, L, B, \mathbb{C}^*)$ . First, we extract from each event log  $L_i \in \mathbb{L}$ , the log-based neighborhood context of  $a_x$  (Lines 3-8, detailed in Section 6.2.1). Then, a configurable process fragment is discovered from the extracted log-based neighborhood contexts in two steps (Lines 10-11, detailed in Section 6.2.2).

---

**Algorithm 5** Algorithm for deriving configurable fragments from a collection of event logs

---

```

1: input:  $\mathbb{L}, a_x, k$ 
2: output:  $P^c = (N, E, T, L, B, \mathbb{C}^*)$ 
3: for  $L_i \in \mathbb{L}$  do
4:    $C_{a_x}^k = \{a_x \rightarrow_j b\} \cup \{b \rightarrow_j a_x\}, 1 \leq j \leq k \wedge b \in A_i$ 
5:    $P_{a_x}^k = \{a_x \parallel_j b\}, 1 \leq j \leq k \wedge b \in A_i$ 
6:    $C_{a_x}^k = \{a_x \#_j b\}, 1 \leq j \leq k \wedge b \in A_i$ 
7:    $L_{a_x}^k = \cup_{\sigma \in L_i} (\sigma \downarrow_{\{a_x\}} \cup C_{a_x}^k \cup P_{a_x}^k \cup E_{a_x}^k)$ 
8:    $\mathbb{L}_{a_x}^k = \mathbb{L}_{a_x}^k \cup L_{a_x}^k$ 
9: end for
10:  $L_{merged} = Merge(\mathbb{L}_{a_x}^k)$ 
11:  $P = MineProcess(L_{merged})$ 
12:  $P^c = setConfigurableElements(P, \cup_i A_{a_{x_i}}^k)$ 

```

---

### 6.2.1 Extracting Log-based Neighborhood Contexts

The log-based neighborhood context of an activity  $a_x \in A$  derived from an event log  $L$  within  $k$ -layers represents the portions of traces in  $L$  that contain  $a_x$  and its neighbor activities within a distance of length  $k$ . A *neighbor activity* can be connected to  $a_x$  via one of the log-based ordering relations (Definition 3.4.2): causality ( $\rightarrow$ ), parallelism ( $\parallel$ ) or choice ( $\#$ ). The causality relation  $a_x \rightarrow b$  assumes that  $b$  is a direct successor of  $a_x$ . Therefore, the distance from  $a_x$  to  $b$  is equal to 1 and  $b$  is a 1<sup>st</sup>-layer neighbor of  $a$ . In order to define a  $k^{th}$ -layer neighbor, we define the distance  $k$  of a causal relation (Definition 6.2.1). This definition is inspired from an earlier work on Petri-nets semantics [180].

**Definition 6.2.1** ( $k^{th}$ -causality). *Two activities  $a, b \in A$  are in a  $k^{th}$ -causal relation in the event log  $L$  denoted by  $a \rightarrow_k b$  iff:  $\exists \sigma \in L$  and  $1 \leq i \leq |\sigma| - k$  such that  $\sigma[i] = a, \sigma[i + k] = b \wedge \forall i \leq j \leq i + k : \sigma[j] \rightarrow \sigma[j + 1]$ .*

For example, in the event log in Table 6.1,  $a_1$  is in a 2<sup>nd</sup>-causal relation with  $a_4$  (i.e.  $a_1 \rightarrow_2 a_4$ ) since there exists the trace  $\langle a_1, a_2, a_4, a_3, a_5, a_8 \rangle$  and we have  $a_1 \rightarrow a_2$  and  $a_2 \rightarrow a_4$ .

In contrast to the causal relation, the parallel and choice relations do not reference a distance notion. This issue is illustrated in Figure 6.7. The process fragment in Figure 6.5 is mined from the log traces in Table 6.4. This fragment contains the activity  $d$  which is in a parallel relation with  $b$  and  $c$  (i.e. from the log traces, we have  $d \parallel b$  and  $d \parallel c$ ). The neighborhood context graph of the activity  $d$  within 3-layers is depicted in Figure 6.6. The activity  $a$  is on the 1<sup>st</sup>-layer of  $d$  since  $|\mathcal{SF}_d^a| = |\langle F_a^d \rangle| = 1$ ;  $b$  is on the 2<sup>nd</sup>-layer since  $|\mathcal{SF}_d^b| = |\langle F_a^d, F_a^b \rangle| = 2$  and  $c$  is on the 3<sup>rd</sup>-layer since  $|\mathcal{SF}_d^c| = |\langle F_a^d, F_a^b, F_a^c \rangle| = 3$  (Definition 4.2.4). Therefore, if we

| Trace ID | Log traces                   |
|----------|------------------------------|
| 1        | $\langle a, b, c, d \rangle$ |
| 2        | $\langle a, b, d, c \rangle$ |
| 3        | $\langle a, d, b, c \rangle$ |

Figure 6.4: An example of an event log with parallel relations

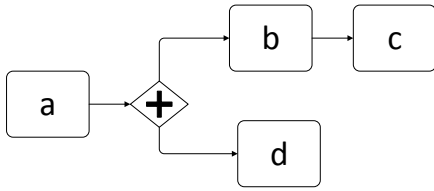


Figure 6.5: The fragment corresponding to the event log in Figure 6.4

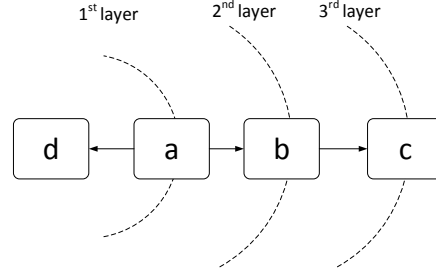


Figure 6.6: The neighborhood context graph of the activity  $g$  in the process fragment in Figure 6.7

Figure 6.7: The parallel relations in neighborhood context graphs

want to extract the log-based neighborhood context of the activity  $d$  from the event log in Table 6.4 and within 2-layers, only the activities  $a$  and  $b$  should be considered. This issue comes from the fact that the activities in a neighborhood context graph are distributed over the layers based on a causal relation. For example,  $b$  is on the 2<sup>nd</sup>-layer of  $d$  because there exists a path from  $a$  to  $b$  then from  $a$  to  $b$ ;  $c$  is on the 3<sup>rd</sup>-layer because there exists a path from  $a$  to  $d$ , then from  $a$  to  $b$  and last from  $b$  to  $c$ . In order to overcome this issue, we introduce the notion of distance in parallel and choice relations in an event log (Definition 6.2.2 and Definition 6.2.3).

**Definition 6.2.2** ( $k^{\text{th}}$ -parallelism). *Two activities  $a, b \in A$  are in a  $k^{\text{th}}$ -parallel relation in the event log  $L$  denoted by  $a ||_k b$  iff:  $a || b$  and  $\exists a_l \in A$  such that  $a_l \rightarrow_{k-1} a \wedge a_l \rightarrow_1 b$ .*

**Definition 6.2.3** ( $k^{\text{th}}$ -choice). *Two activities  $a, b \in A$  are in a  $k^{\text{th}}$ -choice relation in the event log  $L$  denoted by  $a \#_k b$  iff:  $a \# b$  and  $\exists a_l \in A$  such that  $a_l \rightarrow_{k-1} a \wedge a_l \rightarrow_1 b$ .*

Referring to the log traces in Table 6.4, we have  $d ||_3 c$  since  $d || c$  and there exists  $a, b \in A$  where  $a \rightarrow_2 c$  and  $a \rightarrow_1 d$ .

Having introduced the  $k^{\text{th}}$ -causal, -parallel and -choice relations with an activity  $a_x$  in an event log  $L$ , the first step of our approach consists of extracting the activities that are within 1<sup>st</sup>- to  $k^{\text{th}}$  relations with  $a_x$  in each event log  $L_i \in \mathbb{L}$  (Lines 3-6 in Algorithm 5). The log-based neighborhood context graph of  $a_x$  within  $k$ -layers is then

defined as the projection of the log traces on the extracted activities (Line 7). For example, the log-based neighborhood context graphs of the activity  $a_4$  within 2-layers extracted from the event logs in Table 6.1 and Table 6.2 are depicted in Table 6.3 and Table 6.4 respectively.  $L_{a_4}^2$  is the projection of  $L_1$  on  $\{a_1, a_2, a_3, a_4, a_5, a_8\}$  as we have the following relations  $a_2 \rightarrow_1 a_4$ ;  $a_4 \rightarrow_1 a_5$ ;  $a_1 \rightarrow_2 a_4$ ;  $a_3 \parallel_2 a_4$  and  $a_4 \rightarrow_2 a_8$ .  $L_{a_4}^2$  is the projection of  $L_2$  on  $\{a_0, a_2, a_3, a_4, a_5, a_6, a_7\}$  as we have the following relations  $a_2 \rightarrow_1 a_4$ ;  $a_4 \rightarrow_1 a_5$ ;  $a_7 \rightarrow_1 a_4$ ;  $a_0 \rightarrow_2 a_4$ ;  $a_6 \rightarrow_2 a_4$  and  $a_4 \parallel_2 a_3$ .

| Trace ID | Log traces                                     |
|----------|--|
| 1        | $\langle a_1, a_2, a_4, a_3, a_5, a_8 \rangle$ |
| 2        | $\langle a_1, a_2, a_3, a_4, a_5, a_8 \rangle$ |
| 3        | $\langle a_1, a_2, a_4, a_3, a_5, a_8 \rangle$ |
| 4        | $\langle a_1, a_2, a_4, a_3, a_5, a_8 \rangle$ |
| 5        | $\langle a_1, a_2, a_4, a_3, a_5, a_8 \rangle$ |
| 6        | $\langle a_1, a_2, a_4, a_3, a_5, a_8 \rangle$ |
| 7        | $\langle a_1, a_2, a_3, a_4, a_5, a_8 \rangle$ |

Table 6.3:  $L_{a_4}^2$ : Log-based event log extracted from  $L_1$  in Table 6.1

| Trace ID | Log traces  |
|----------|---|
| 1        | $\langle a_0, a_2, a_6, a_7, a_3, a_4, a_5 \rangle$ |
| 2        | $\langle a_0, a_2, a_6, a_7, a_4, a_3, a_5 \rangle$ |
| 3        | $\langle a_0, a_2, a_4, a_3, a_5 \rangle$           |
| 4        | $\langle a_0, a_2, a_3, a_4, a_5 \rangle$           |
| 5        | $\langle a_0, a_2, a_6, a_7 \rangle$                |
| 6        | $\langle a_0, a_2, a_6, a_7 \rangle$                |
| 7        | $\langle a_0, a_2, a_6, a_7 \rangle$                |
| 8        | $\langle a_0, a_2, a_3, a_4, a_5 \rangle$           |
| 9        | $\langle a_0, a_2, a_4, a_3, a_5 \rangle$           |

Table 6.4:  $L_{a_4}^2$ : Log-based event log extracted from  $L_2$  in Table 6.2

At the end of this step, a set of extracted logs stocked in  $\mathbb{L}_{a_x}^k$  is obtained (Line 8). This log set is used in the next step in order to mine a configurable process fragment.

## 6.2.2 Mining Configurable Process Fragments

The logs in  $\mathbb{L}_{a_x}^k$  obtained from the previous step are merged into one log  $L_{merged}$  (Line 10 in Algorithm 5). The merged log is then used as input for an existing mining algorithm (such as alpha algorithm [127, 154]) in order to discover a process fragment that describes the recorded behavior (Line 11). An example of the discovered process fragment from the merged log  $L_{merged} = Merge(L_{a_4}^2, L_{a_4}^2)$  is illustrated in Figure 6.8.

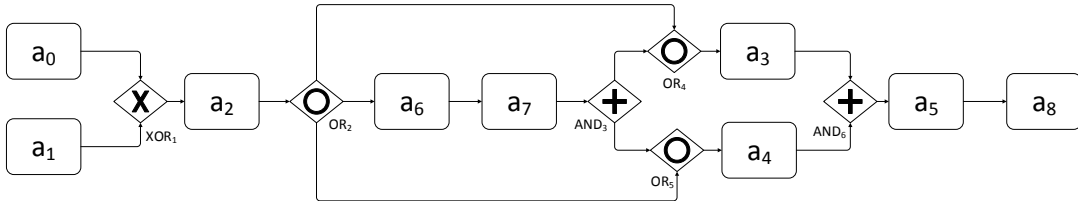


Figure 6.8: The process fragment discovered from the logs  $L_{a_4}^2$  and  $L_{a_4}^2$  in Table 6.3 and Table 6.4 respectively

However, as existing mining algorithms are not able to explicitly specify the con-

figurable elements in the discovered fragment, we provide in the following a simple approach for detecting configurable activities and gateways (Line 12).

Let  $P = (N, E, T, L)$  be the discovered process fragment from the merged log  $L_{merged}$  and  $\mathbb{A}_{a_x}^k = \bigcup_i A_{a_{x_i}}^k$  be the set of alphabets of the extracted logs  $\mathbb{L}_{a_x}^k = \bigcup_i L_{a_x}^k$ . We say that an activity  $a \in P$  is **shared** if it belongs to multiple alphabets, i.e.  $a \in \bigcap_{2 \leq i \leq |\mathbb{L}_{a_x}^k|} L_{a_{x_i}}^k$ ; otherwise it is **unshared**. For example, in the log-based neighborhood contexts in Table 6.3 and Table 6.4, the activities  $a_2, a_3, a_4$  and  $a_5$  are shared as they are common between the two logs while  $a_1$  and  $a_8$  in  $L_{a_{41}}^1$  and  $a_0, a_6$  and  $a_7$  in  $L_{a_{42}}^2$  are unshared as they belong only to  $L_{a_{41}}^1$  and  $L_{a_{42}}^2$  respectively.

A **shared activity** means that it appears in all the origin event logs and therefore it is always included in the derived variants and **cannot be configurable**, e.g. it cannot be configured to *OFF* to exclude it. While an **unshared activity** means that in some process variants, this activity does not appear and therefore is **configurable**, e.g. it can be configured to *OFF* to exclude it from the derived process variant or to *ON* to keep it. For example, in the process fragment in Figure 6.8, the activities  $a_0, a_1, a_6, a_7$  and  $a_8$  should be configurable.

A split (respectively join) **gateway is configurable if it has unshared activities** in its transitive postset<sup>2</sup> (respectively transitive preset) and those activities originate from different alphabets. For example, in Figure 6.8, the join  $XOR_1$  has two unshared activities  $a_0$  and  $a_1$  in its transitive preset that originate from  $L_{a_{42}}^2$  and  $L_{a_{41}}^1$  respectively, therefore it is configurable.  $OR_2$  has the unshared activity  $a_6$  in its transitive postset, thus it is configurable. The same holds for  $OR_4$  and  $OR_5$ .  $AND_3$  and  $AND_4$  have the shared activities  $a_3$  and  $a_4$  in their transitive postset and preset respectively, thus they are not configurable. An example of the resulted configurable process fragment after this step is illustrated in Figure 6.9.

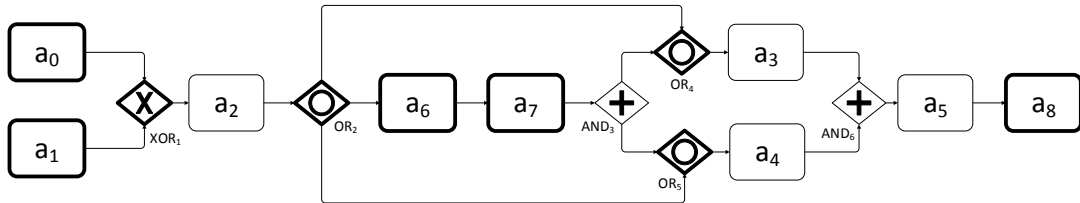


Figure 6.9: The resulted configurable process fragment after applying the shared/unshared activity strategy

In this example, the activities  $a_0$  and  $a_1$  are unnecessary configurable since  $XOR_2$  is configurable. Through the configuration of  $XOR_2$ ,  $a_0$  and  $a_1$  can be included or excluded. The same holds for  $a_6$  and  $a_7$  which can be included or excluded through the configuration of the configurable gateways  $OR_2, OR_4$  and  $OR_5$ . Therefore, unnecessary configurable activities are those that are in the transitive postset or preset of a

<sup>2</sup>The transitive postset of a gateway  $g$  denoted as  $g \bullet^c$  is the set of activities in its outgoing branches that are reachable from  $g$  via a chain of gateways [43]

configurable gateway. One could apply this reduction rule to remove the unnecessary configurable elements in the discovered fragment. The resulted fragment after this step is illustrated in Figure 6.3.

In the next section, we present our frequency-based approach to derive ranked configuration guidelines for the configurable elements in  $P^c$  that are attached as text annotations to the configurable elements in  $P^c$  (e.g. in Figure 6.3). These guidelines assist the process user selecting desirable configuration choices.

### 6.3 Mining Ranked Configuration Guidelines

This section elaborates our approach to derive *ranked configuration guidelines* that assist the configuration of the discovered process fragment  $P^c$ . Basically, we propose to explore the recorded executions in the merged sublog  $L_{merged}$  in order to recommend the *frequently executed* paths as *ranked guidelines*. An example of these guidelines is depicted in Figure 6.3. The guidelines are attached as text annotations to the configurable elements. For example, the guidelines of the configurable  $XOR_1$  state that the configuration  $(Seq, \{a_0\})$  is highly executed while  $(Seq, \{a_1\})$  is moderately executed in  $L_{merged}$ . The guidelines of the configurable  $OR_2$  state that the configuration  $(AND, \{a_3, a_4\})$ <sup>3</sup> is highly executed while  $(Seq, \{a_6\})$  is moderately executed in  $L_{merged}$ . These guidelines are dynamic in the sense that they are updated after each configuration step in order to take the already chosen configurations into consideration.

To extract the guidelines, first, we model the traces recorded in  $L_{merged}$  in a frequency-based suffix tree (Section 6.3.1). Then, for each configurable element in  $P^c$ , we derive its configurations and rank them with *high*, *medium* or *low* depending on their frequencies recorded by the suffix tree and taking into account the previously chosen configurations (Section 6.3.2).

#### 6.3.1 A frequency suffix tree for configuration executions

Since the log traces are sequences of executed activities, finding a specific configuration in an event log can be mapped to a *string pattern matching problem*. For example, finding a configuration  $Conf_{AND^c} = \langle AND, \{a, b\} \rangle$  refers to searching for a parallel relation between  $a$  and  $b$ ; that is, searching for the traces including  $a$  and  $b$  in different orders. We say that a configuration  $Conf_{AND^c}$  has been executed  $x$  times if, in  $L_{merged}$ , the activities  $a$  and  $b$  appears  $x$  times in the parallel relation.

In light of the above, we propose to use *suffix trees* [178] to model all possible executed configurations and their frequencies. A suffix tree is an efficient data structure for storing all possible substrings of a given string in a linear time [181]. It provides linear-time solutions for string pattern matching problems and has been widely used

<sup>3</sup>Please note that in this chapter we represent the outgoing branches of a split gateway configuration by the transitive postset as the logs record only the activities' executions



in biological domain for DNA sequence analysis. A suffix tree for a string  $S$  having  $l$  distinct characters is a rooted directed tree where each edge is labeled with a nonempty substring of  $S$  and each internal node has at least two children. Two edges issued from the same node cannot have the same labels. Furthermore, the concatenation of the edges' label of every path from a root to an internal node represents a suffix of  $S$ . A suffix tree can be efficiently built using Ukkonen's algorithm [181] which is a linear-time algorithm for building incrementally the suffix tree for a set of strings. Some variants of the suffix tree have been proposed such as the probabilistic suffix tree [182] and the weighted suffix tree [183]. In our work, we propose to use a frequency-based suffix tree where each node is labeled by the frequency of occurrence of the suffix starting from the root to that node in an event log. An excerpt of the frequency suffix tree for  $L_{merged} = Merge(L_{a_{41}}^2, L_{a_{41}}^2)$  is illustrated in Figure 6.10. For example, in this suffix tree,  $\langle a_1, a_2 \rangle$  is executed 7 times in  $L_{merged}$ ,  $\langle a_1, a_2, a_4, a_3, a_5, a_8 \rangle$  is executed 5 times, and son on.

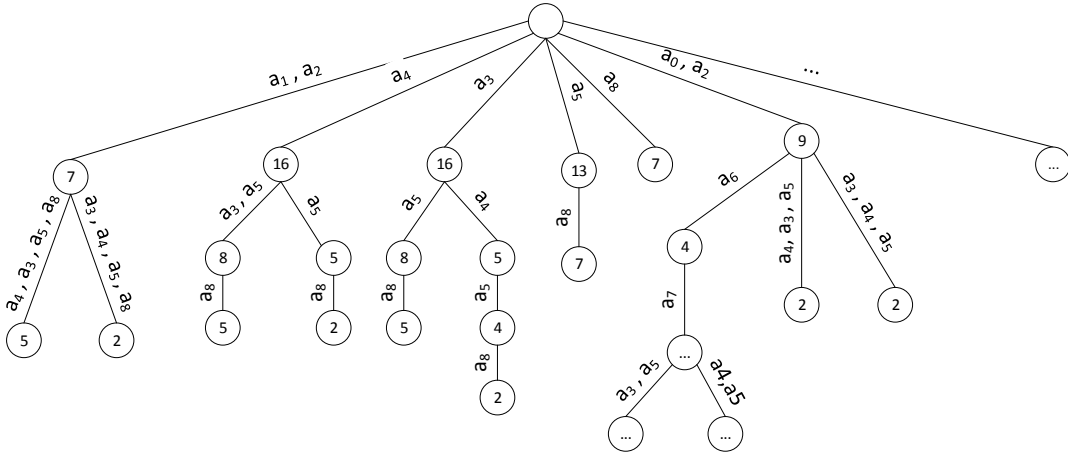


Figure 6.10: An excerpt of a frequency suffix tree derived from  $L_{merged}$

In order to build the frequency suffix tree for the elements' configurations in  $P^c$  referred to as  $S_{conf}$ , we propose to project the traces in  $L_{merged}$  on (i) the configurable activities and (ii) the activities in the transitive preset and postset of the configurable gateways. We refer to the projected log as *Configuration log* (Definition 6.3.1).

**Definition 6.3.1** (Configuration log). A configuration log  $L_{conf}$  derived from  $L_{merged}$  and  $P^c$  is the projection of  $L_{merged}$  on (i) the configurable activities and (ii) the activities in the transitive preset and postset of the join and split configurable gateways respectively, i.e.  $L_{conf} = \bigcup_{\sigma \in L_{merged}} \sigma_{\downarrow C}$  where  $C = A \cup T$  such that  $A = \{a \in N : B(a) = true \wedge T(a) = activity\}$  and  $T = \{a \in N : a \in \bullet^c g \wedge B(g) = true \wedge |\bullet g| > 1\} \cup \{a \in N : a \in g \bullet^c \wedge B(g) = true \wedge |g \bullet| > 1\}$ .

For example, the configuration log  $L_{conf}$  derived from  $L_{merged} = Merge(L_{a_{41}}^2, L_{a_{42}}^2)$  and the configurable fragment in Figure 6.3 is illustrated in Table 6.5 and is defined

as the projection of  $L_{merged}$  on the activities  $a_1, a_0, a_2, a_3, a_4, a_6, a_7$  and  $a_8$  ( $a_1$  and  $a_2$  are in the transitive preset of the configurable gateway  $XOR_1$ ;  $a_2$  and  $a_7$  are in the transitive preset of  $OR_4$  and  $OR_5$ ;  $a_6, a_3$  and  $a_4$  are in the transitive postset of  $OR_2$ ;  $a_8$  is a configurable activity). The corresponding suffix tree  $S_{conf}$  is illustrated in Figure 6.11.

| Trace ID | Log traces                                     |
|----------|--|
| 1        | $\langle a_1, a_2, a_4, a_3, a_8 \rangle$      |
| 2        | $\langle a_1, a_2, a_3, a_4, a_8 \rangle$      |
| 3        | $\langle a_1, a_2, a_4, a_3, a_8 \rangle$      |
| 4        | $\langle a_1, a_2, a_4, a_3, a_8 \rangle$      |
| 5        | $\langle a_1, a_2, a_4, a_3, a_8 \rangle$      |
| 6        | $\langle a_1, a_2, a_4, a_3, a_8 \rangle$      |
| 7        | $\langle a_1, a_2, a_3, a_4, a_8 \rangle$      |
| 8        | $\langle a_0, a_2, a_6, a_7, a_3, a_4 \rangle$ |
| 9        | $\langle a_0, a_2, a_6, a_7, a_4, a_3 \rangle$ |
| 10       | $\langle a_0, a_2, a_4, a_3 \rangle$           |
| 11       | $\langle a_0, a_2, a_3, a_4 \rangle$           |
| 12       | $\langle a_0, a_2, a_6, a_7 \rangle$           |
| 13       | $\langle a_0, a_2, a_6, a_7 \rangle$           |
| 14       | $\langle a_0, a_2, a_6, a_7 \rangle$           |
| 15       | $\langle a_0, a_2, a_3, a_4 \rangle$           |
| 15       | $\langle a_0, a_2, a_4, a_3 \rangle$           |

Table 6.5: the configuration log  $L_{conf}$  derived from  $L_{merged}$

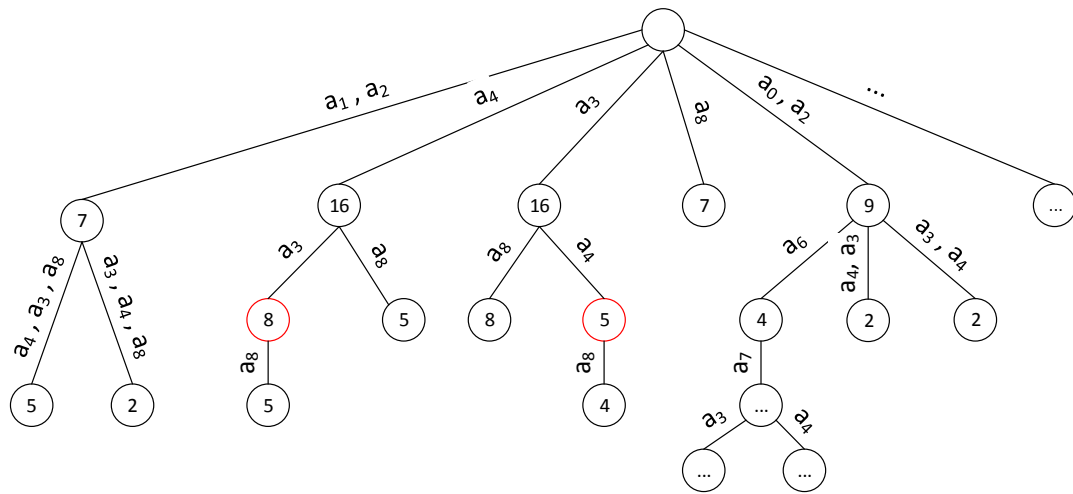


Figure 6.11: The suffix tree  $S_{conf}$  of the configuration log  $L_{conf}$

In the next section, we propose an approach to derive from  $S_{conf}$ , the elements' configurations and their execution frequencies as *ranked configuration guidelines*.

### 6.3.2 Deriving ranked configuration guidelines

Using the frequency-based suffix tree  $S_{conf}$ , we compute for each configurable element the frequency of execution of all its possible configurations. Then, based on a predefined execution frequency threshold, we attribute to each configuration a rank  $r \in \{high, medium, low\}$ . This process is repeated after each selected configuration in order to recompute the configurations' frequencies of the remaining configurable elements based on the previously chosen ones. Algorithm 6 illustrates the different steps. It takes as input the suffix tree  $S_{conf}$ , the configurable fragment  $P^c$ , the set of configurable elements  $E^c$  in  $P^c$  and the list of selected elements' configuration  $L$  which is initially empty and provides as output the list of ranked configuration guidelines  $L_{guid}$ .

The algorithm consists of a loop that is repeated until there are no more configurable elements in  $E^c$  (Line 3), i.e. all configurable elements have been configured by the process user. After each selected configuration, the list of ranked configuration guidelines  $L_{guid}$  is reset in order to recompute the new guidelines that take into account the already chosen configurations (Line 4). For each configurable element  $e^c \in E^c$ , we get its possible configurations according to Definition 3.3.4 (Lines 5-6). Then, we compute for each configuration, its frequency of execution derived from  $S_{conf}$  (Lines 7-19). Two cases can be presented: (i) we are at the initial stage where no elements are configured yet (Lines 8-10) or (ii) there exists a set of already selected configurations in  $L$  (Lines 10-15).

For the first case, the elements' configurations and their frequencies of execution are derived from  $S_{conf}$  as follows. If  $e^c$  is an activity (Line 9), then the frequency of the configuration  $ON$ , denoted  $F_{e^c=ON}$  is equal to the frequency of the suffix  $e^c$  in  $S_{conf}$ . For example, for the configurable activity  $a_8$  in the configurable process in Figure 6.3, the frequency of the configuration  $C_{a_8} = ON$  in  $S_{conf}$  in Figure 6.11 is  $F_{a_8=ON} = 7$ . The frequency  $F_{a_8=OFF}$  of the configuration  $C_{a_8} = OFF$  is equal to the maximal frequency in  $S_{conf}$  minus  $F_{C=ON}$ . For example, for  $a_8$ ,  $F_{a_8=OFF} = 16 - 7 = 9$ .

Regarding the configuration of the gateways ( $OR$ ,  $AND$  and  $XOR$ ), we show that using the frequency suffix tree, we compute the frequency of an  $AND$  configuration denoted as  $F_{\wedge C}$ , from which we can derive the frequencies of an  $XOR$  configuration denoted as  $F_{\times C}$  and an  $OR$  configuration denoted as  $F_{\vee C}$ . In the following, we show the split gateway case; the same holds for a join gateway. Let  $C = (AND, P)$  such that  $P \in \mathcal{P}(e^c \bullet^c)$  be a configuration of the gateway element  $e^c$ . To find the frequency of  $C$ , we search in  $S_{conf}$  for all the suffixes containing the activities in  $P$  in different positions. For example, Let  $C = (AND, \{a_3, a_4\})$  be a configuration of  $OR_2$  in the configurable process in Figure 6.3. Searching in the suffix tree in Figure 6.11 for the configuration  $C$  returns the suffixes  $a_4a_3$  and  $a_3a_4$ . For each matched suffix  $m$ , we

**Algorithm 6** Algorithm for deriving ranked guidelines

---

```

1: input: suffix tree  $S_{conf}$ ,  $P^c$  configurable elements  $E^c = \{e \in N : B(e) = true\}$ ,
   list of selected configurations  $L = \{C_{conf_i} : 1 \leq i \leq |E^c|\}$ 
2: output: list of ranked guidelines  $L_{guid}$ 
3: while  $E^c \neq \phi$  do
4:    $L_{guid} = new List()$ 
5:   for  $e^c \in E^c$  do
6:     Get the set of all valid configurations based on Definition 3.3.4
7:     for each possible configuration  $C$  do
8:       if  $L = \phi$  {Initially no configured elements} then
9:         if  $e^c$  is an activity: Compute the frequency  $F_C$  of the configuration  $C_{e^c}$ 
10:        if  $e^c$  is a gateway: Compute the frequency  $F_{\times C}$  and/or  $F_{\wedge C}$  according
            to its type
11:       else
12:         get the logical formula  $LF = \bigwedge_{i=1}^{|L|} C_i \wedge C$  where  $C_i$  is a previously
            selected configuration and  $C$  is the current one
13:         Get the disjunctive normal form  $DN$  of  $LF$ 
14:         Compute the frequency  $F_{\times DN}$  of  $DN$ 
15:       end if
16:       Compute the corresponding execution ratio  $R_C$ 
17:       add the guideline  $G = \langle C, r \rangle$  to the list  $L_{guid}$  where  $r$  is the rank
            corresponding to  $R_C$ 
18:     end for
19:   end for
20:   add the selected configuration to  $L$ 
21:   remove the corresponding configurable gateway from  $E^c$ 
22: end while

```

---

denote by  $f_m$  its frequency of occurrence. For example the frequency of the matches  $a_3a_4$  and  $a_4a_3$  are  $f_{a_3a_4} = 16$  and  $f_{a_4a_3} = 8$  respectively (the frequencies in the nodes in red color in the suffix tree in Figure 6.11). In order, to find the frequency  $F_{\wedge C}$  of  $C$ , we sum the frequencies of all found matches:

$$F_{\wedge C} = \sum_i f_{m_i} \quad (6.1)$$

The execution of an *XOR* configuration  $C = \langle XOR, P \rangle$  represents the cases in which the corresponding activities in  $P$  are not executed together. While the execution of an *OR* configuration  $C = \langle OR, P \rangle$  includes those of the *AND* and the *XOR* executions. To derive the frequency of execution of an *XOR*, we use the Set theory to represent the activities in  $P$  according to the configuration type (*AND*, *OR* or *XOR*). For example, suppose that  $P = \{A, B\}$ , then  $C = (AND, P)$ ,

$C = (XOR, P)$  and  $C = (OR, P)$  are represented as in Figure 6.12. Consequently, the configurations  $F_{\times C}$  is computed using the statistical theory as follows:

$$F_{\times C} = \sum_{i=1}^n f_{a_i} - \sum_{k\%2}^n k \times \sum_k F_{\wedge P_k} + \sum_{l=k\%2+1}^n l \times \sum_l F_{\wedge P_l} \quad (6.2)$$

where  $n = |P|$ ;  $P_k, P_l \in \mathcal{P}(P)$ ;  $|P_k| = k$  and  $|P_l| = l$ . For example, the frequency of execution of the  $XOR$  configuration  $C = \langle XOR, \{a_3, a_4, a_6\} \rangle$  is  $F_{\times(a_3, a_4, a_6)} = f_{a_3} + f_{a_4} + f_{a_6} - 2 \times (F_{\wedge(a_3, a_4)} + F_{\wedge(a_4, a_6)} + F_{\wedge(a_3, a_6)}) + 3 \times F_{\wedge(a_3, a_4, a_6)}$ .

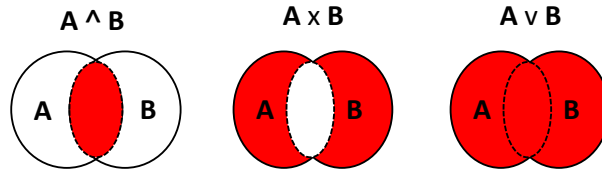


Figure 6.12: The representation of the three relations  $AND$ ,  $XOR$  and  $OR$  using the set theory

The frequency of execution of an  $OR$  configuration denoted as  $F_{\vee C}$  is the sum of  $F_{\wedge C}$  and  $F_{\times C}$ . Since  $F_{\vee C}$  would be the highest in all cases, **we omit ranking the  $OR$  configurations** in this work, and we let its choice to the process user.

Having derived and computed the frequencies of the elements' configurations, we rank each configuration  $C$  of the element  $e^c$  by computing its frequency ration  $R_C$  (Line 16) as:

$$R_C = \frac{F_C}{\max(\cup_i F_{C_i})} \quad (6.3)$$

where  $F_C$  is the frequency of the corresponding configuration and  $\max(\cup_i F_{C_i})$  is the maximal frequency among all possible configurations of  $e^c$ . A ratio equal to 1 denotes that the corresponding configuration appears in all possible executions, while a ratio equal to 0 denotes that it has been never executed. Based on  $R_C$ , we propose to assign a rank  $r \in \{high, medium, low\}$  such that:

$$r = \begin{cases} high & \text{if } \min H \leq R_C \leq 1 \\ medium & \text{if } \min M \leq R_C < \min H \\ low & \text{if } 0 < R_C < \min M \end{cases} \quad (6.4)$$

Where  $\min H$  and  $\min M$  are two predefined thresholds. The derived configurations and their computed rank are added to the list of guidelines  $L_{guid}$  (Line 17) which is presented to the process user as a set of text annotations attached to the configurable elements (Figure 6.3).

For the second case where the set of selected configurations  $L$  is not empty (Lines 12-14), we compute the configurations frequencies and rank them by taking into

account the frequency of the already selected configurations in  $L$ . To do so, we create the logical formula  $LF$  by intersecting the previous configurations with the current one (Line 12). For example, in Figure 6.3, suppose that the configuration  $C_{XOR_1} = (XOR, \{a_1, a_0\})$  is selected by the process user and added to  $L$ . To compute the frequency of the configuration  $C_{OR_2} = (AND, \{a_3, a_4\})$ , we derive the logical formula  $LF = (a_1 \times a_0) \wedge (a_3 \wedge a_4)$ . Since, the frequency of an  $AND$  configuration is the basis for deriving the others frequencies, we transform the logical formula  $LF$  into its disjunctive normal form  $DF$  (i.e. the disjunction of conjunction) (Line 13). Having  $DF$ , we can now compute the final frequency  $F_{\times DF}$  (Line 14). For example, the disjunctive normal form of  $LF$  is  $DF = (a_1 \wedge a_3) \times (a_1 \wedge a_4) \times (a_2 \wedge a_3) \times (a_2 \wedge a_4) = A \times B \times C \times D$  where  $A = a_1 \wedge a_3$ ,  $B = a_1 \wedge a_4$ ,  $C = a_2 \wedge a_3$  and  $D = a_2 \wedge a_4$ . The corresponding frequency  $F_{\times\{A,B,C,D\}} = F_A + F_B + F_C + F_D - 2 \times (F_{\wedge\{A,B\}} + F_{\wedge\{A,C\}} + F_{\wedge\{A,D\}} + F_{\wedge\{B,C\}} + F_{\wedge\{B,D\}} + F_{\wedge\{C,D\}}) - 4 \times F_{\wedge\{A,B,C,D\}} + 3 \times (F_{\wedge\{A,B,C\}} + F_{\wedge\{A,B,D\}} + F_{\wedge\{B,C,D\}})$ .  $F_{\times\{A,B,C,D\}}$  corresponds to the frequency of occurrence of the configuration  $C_{OR_2} = (AND, \{a_3, a_4\})$  along with  $C_{XOR_1} = (XOR, \{a_0, a_1\})$ . The corresponding execution ratio is  $R_C = \frac{F_{\times DF}}{\max(\cup F_{C_{OR_2}})}$ . In this way, the ranked configuration guidelines are repeatedly updated after each configuration step to take into account the newly added configurations to  $L$ .

## 6.4 Conclusion

In this chapter, we answered the question raised in the thesis problematic (Section 1.2), which is: *Can execution logs be useful? and How?*. We showed that event logs can be useful in both cases: (i) to assist the design of configurable process models and (ii) to support their configuration. We proposed to discover a configurable process fragment from a collection of event logs and a set of configuration guidelines for assisting its configuration. We captured and extracted the *log-based neighborhood contexts* of an activity from different event logs. We merged the extracted logs and used an existing mining algorithm to discover a fragment that describes the behavior of all merged logs. To identify configurable elements, we defined a set of rules based on the notion of log-based shared and unshared activities.

We then explored the recorded executions in the merged logs in order to recommend the frequently executed configurations as *ranked guidelines*. We modeled the activities' executions and their frequencies in a suffix tree which provide linear-time solutions for pattern matching problems. We used concepts from Set theory to derive the elements' configurations and ranked their frequency of execution from the suffix tree. The ranked configuration executions are updated after each configuration step to take into account the already chosen ones.

Compared to the approaches that propose to mine process models and identify their elements' configurations by *replaying* the logs on the discovered model (e.g. [46, 47]), our approach is simpler and faster. It does not only identify the config-

urable elements and their configurations but also their *importance* reflected by their frequency of execution in the event logs. It also provides an interactive configuration assistance by recommending the elements' configurations that are suitable taking into account the previously selected ones.

Our principles presented in Section 1.4.1 are also respected in this chapter:

- **Automation:** We use process mining techniques to automatically discover configurable process fragments and mine configuration guidelines. We do not ask process providers for any additional manual effort or input data except the activity in the process graph for which he needs to discover a configurable fragment.
- **Implicit knowledge exploitation:** We exploit the log-based activity neighborhood context and the elements' configuration execution importance which are implicit knowledge hidden in event logs.
- **Focused results:** We recommend configurable fragments instead of entire configurable models. We also recommend ranked configuration guidelines for each element configurations instead of entire process configurations.
- **Balanced computation:** We discover configurable fragments instead of entire process models from event logs. We project the logs on a small subset of activities that represent the closet neighborhood context of a selected one. We use suffix trees to mine the configuration guidelines which provide linear-time solutions. Our approach does not face the NP-complete problem.

# Evaluation and Validation

## Contents

---

|            |   |            |
|------------|---|------------|
| <b>7.1</b> | <b>Introduction</b>                       | <b>134</b> |
| <b>7.2</b> | <b>Proof of Concept</b>                   | <b>135</b> |
| 7.2.1      | Signavio Extension                        | 135        |
| 7.2.2      | ProM Plug-in                              | 137        |
| <b>7.3</b> | <b>Experimentation</b>                    | <b>139</b> |
| 7.3.1      | Configurable Process Design Experiments   | 140        |
| 7.3.1.1    | Approach feasibility and parameter impact | 141        |
| 7.3.1.2    | Results quality                           | 143        |
| 7.3.1.3    | Algorithm performance                     | 145        |
| 7.3.1.4    | Synthesis                                 | 146        |
| 7.3.2      | Process Configuration Experiments         | 147        |
| 7.3.2.1    | Results quality and parameter impact      | 148        |
| 7.3.2.2    | Approach accuracy and parameter impact    | 150        |
| 7.3.2.3    | Synthesis                                 | 151        |
| 7.3.3      | Log-based Experiments                     | 152        |
| 7.3.3.1    | Configurable fragments quality            | 153        |
| 7.3.3.2    | Configuration guidelines efficiency       | 155        |
| 7.3.3.3    | Synthesis                                 | 156        |
| <b>7.4</b> | <b>Case Study</b>                         | <b>156</b> |
| 7.4.1      | Case Study Objective                      | 157        |
| 7.4.2      | Design, Data Collection and Execution     | 157        |
| 7.4.3      | Results Analysis and Findings             | 158        |
| 7.4.4      | Threats to Validity                       | 160        |
| <b>7.5</b> | <b>Conclusion</b>                         | <b>160</b> |

---



## 7.1 Introduction

In this chapter, we present (i) the implementation we have done to realize our automated support techniques, (ii) the experiments we have made to evaluate the efficiency of our solutions and (iii) the case study we have conducted to show the effectiveness of a frequency-based approach for process configuration. Our goal is (i) to prove that our approach is feasible and accurate in real use-cases and (ii) to analyze the parameters that impact our results' quality.

We implemented three proof of concepts as extensions of two open-source business process management tools, namely *Signavio* which is a web based process modeling tool and *ProM* which is an extensible framework for process mining.

- The first proof of concept, named *FragMerg*, is an extension of Signavio process modeling editor. It assists the process provider while designing a configurable process by recommending *configurable fragments*. Its implementation is based on the algorithms presented in Chapter 4.
- The second proof of concept, named *ConfRule*, is also an extension of Signavio process modeling editor. It extracts a configuration guidance model for assisting the process provider building a configuration support system. It implements the approach for *extracting configuration guidance models* presented in Chapter 5.
- The third proof of concept, named *MineFrag*, is an extension of ProM. It implements the log-based approach for assisting the design and configuration of process models presented in Chapter 6. It recommends *configurable fragments* for selected positions in an ongoing designed process. It also recommends ranked guidelines for assisting the configuration of the fragment.

We performed experiments on two large public datasets. The first one is shared by the Cognitive Computing and Industry Solutions (formerly known as Business Integration Technologies) research group at the IBM Zurich Research Laboratory. The second one is from the SAP reference model.

- For the automated support of the configurable process design, we used the large public dataset of real business processes shared by an IBM research group. We showed that our approach is *feasible* by providing statistics on the recommended results and analyzing the parameters that impact them. We evaluated the *quality* of the resulted configurable fragments in terms of *model complexity and understandability*. We also evaluated the *performance* of our algorithms in terms of *execution time* and compared our approach with the existing ones.
- For the automated support of the process configuration, we used the dataset from the SAP reference model. We provided statistics on the extracted configuration guidelines to assess their *complexity* and *completeness*. We also evaluated

the *accuracy* of the extracted configuration guidance models by computing their *Precision* and *Recall* metrics. We analyzed the parameters that impact each of the computed metrics.

- For the log-based approach, we used synthetic event logs generated from the IBM dataset. We evaluated the quality of the discovered configurable fragments and observed the parameter impact on it. We also evaluated the efficiency of the ranked configuration guidelines.

We carried-out a case study From the Telecommunication domain with domain and IT experts in order to show the practical usefulness of a frequency-based approach for process configuration. Through this case study, we aimed to assess the usefulness of our configuration guidance models when process providers build their configuration support systems using existing manual approaches. We followed the guidelines presented in [184] for conducting the case study. We defined the case study objective, collected and executed the data and analyzed the results. The results show that our approach saves time and recommends configuration guidelines that would have not be possible to be identified based only on the domain experts knowledge.

We present in Section 7.2 our three implemented proof of concepts. The experiments and related discussion are presented in Section 7.3. In Section 7.4, we describe and present the results of our case study. Finally, we conclude the chapter in Section 7.5.

## 7.2 Proof of Concept

In this section, we present the proof of concepts that we have developed to realize our approach. We firstly present the Signavio extensions for process design and configuration (section 7.2.1). These applications use an existing process model repository to derive configurable fragments and extract configuration guidance models. Second, we present a ProM plugin for the log-based process design and configuration (section 7.2.2). This application uses an existing repository of event logs to mine configurable process fragments and configuration guidelines.

### 7.2.1 Signavio Extension

We have implemented and deployed our approaches for process design and configuration presented in Chapter 4 and Chapter 5 as extensions of Signavio <sup>1</sup>, an open source web-based application for modeling business processes using BPMN. Signavio has two versions: commercial and open source. The open source version with limited features is published <sup>2</sup> for free downloading and testing.

---

<sup>1</sup><http://www.signavio.com/>

<sup>2</sup><http://code.google.com/p/signavio-core-components/>

By developing our approaches based on Signavio, we achieve two targets: (1) we make our approach more user-friendly through the graphical suite and (2) we widen the user community and make our approach more visible as Signavio is widely known in the community. Our tool for merging process fragments named *FragMerg* is published at: <http://www-inf.it-sudparis.eu/SIMBAD/tools/fragmerg/>. The tool for extracting configuration guidance models named *ConfRule* is published at: <http://www-inf.it-sudparis.eu/SIMBAD/tools/confRule/>.

Since Signavio does not allow for a configurable business process modeling, we extended BPMN 2.0 with configurable elements<sup>3</sup> and integrated it within Signavio. Configurable elements are graphically modeled with thick lines. A screen-shot of the graphical interface is shown in Figure 7.1.

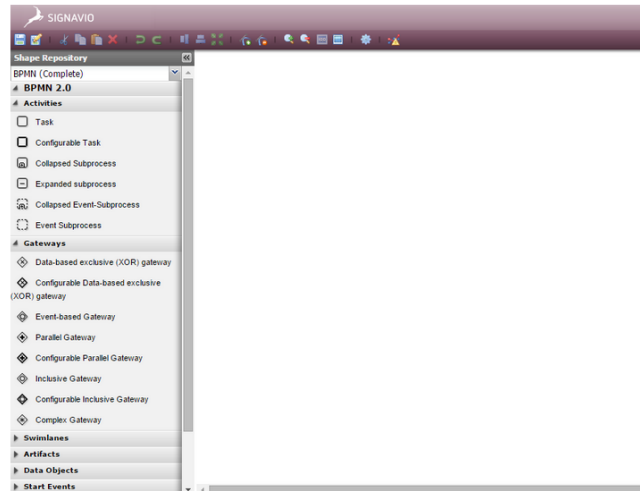


Figure 7.1: A screen-shot of the graphical interface for configurable process modeling in Signavio

Regarding *FragMerg*, the process provider can select the activity to which he needs some assistance, and the number of layers to be considered. *FragMerg* runs the algorithms presented in Chapter 4 for collecting process fragments from existing designed business processes and merging them into one configurable fragment. We used business processes from three domains (movie purchase, hotel reservation and book purchase) and stored them in a MySQL database.

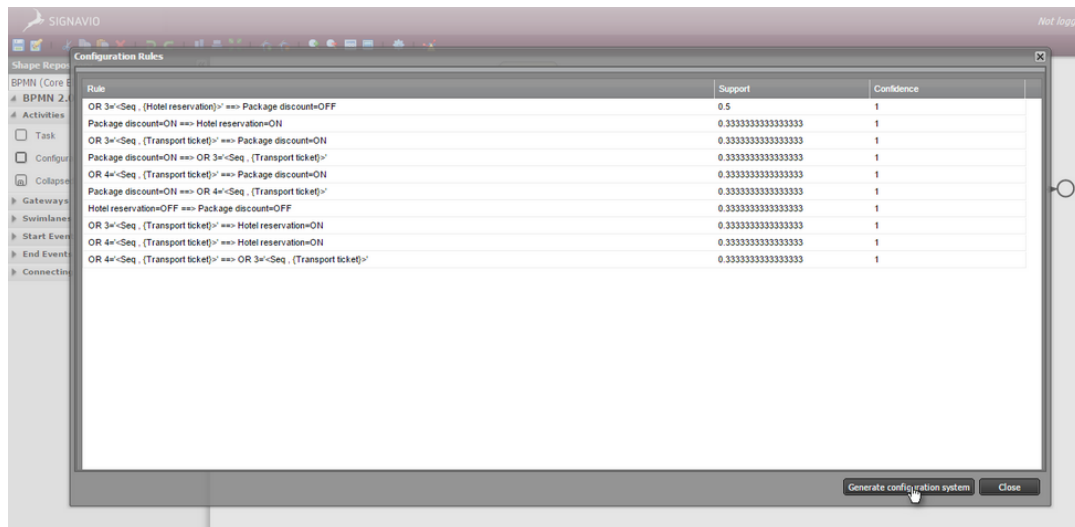
Regarding *ConfRule*, we have extended Signavio with three main functionalities to allow the extraction of configuration guidance models as presented in Chapter 5:

1. *Configuration guidelines' extraction and visualization*: This functionality allows the extraction of configuration guidelines from our existing process repository

<sup>3</sup>the C-BPMN schema definition can be found here: <http://www-inf.it-sudparis.eu/SIMBAD/tools/confRule/cbpmn.xsd>

as described in Section 5.5. The process provider can choose (1) a minimum support and confidence values for the Apriori algorithm and (2) a minimum similarity value for computing the similarity between the configurable process elements and the candidate configurations in the existing process variants. The set of extracted guidelines are returned and visualized in a pop-up window (Figure 7.2).

2. *Petri-net derivation and visualization*: This functionality allows the derivation of a configuration system from the extracted guidelines as described in Section 5.7. A graphical representation of the derived model is visualized in petri nets (Figure 7.3).



| Rule  | Support            | Confidence |
|---|--------------------|------------|
| OR 3<-Seq, (Hotel reservation)> ==> Package discount=OFF          | 0.5                | 1          |
| Package discount=ON ==> Hotel reservation=ON                      | 0.3333333333333333 | 1          |
| OR 3<-Seq, (Transport ticket)> ==> Package discount=ON            | 0.3333333333333333 | 1          |
| Package discount=ON ==> OR 3<-Seq, (Transport ticket)>            | 0.3333333333333333 | 1          |
| OR 4<-Seq, (Transport ticket)> ==> Package discount=ON            | 0.3333333333333333 | 1          |
| Package discount=ON ==> OR 4<-Seq, (Transport ticket)>            | 0.3333333333333333 | 1          |
| Hotel reservation=OFF ==> Package discount=OFF                    | 0.3333333333333333 | 1          |
| OR 3<-Seq, (Transport ticket)> ==> Hotel reservation=ON           | 0.3333333333333333 | 1          |
| OR 4<-Seq, (Transport ticket)> ==> Hotel reservation=ON           | 0.3333333333333333 | 1          |
| OR 4<-Seq, (Transport ticket)> ==> OR 3<-Seq, (Transport ticket)> | 0.3333333333333333 | 1          |

Figure 7.2: A screen-shot of the Signavio graphical interface for visualizing the configuration guidelines

## 7.2.2 ProM Plug-in

We have implemented our log-based approach presented in Chapter 6 as a plugin in ProM framework<sup>4</sup> named *mineFrag*. ProM is a well-known open-source framework for implementing process mining tools. Our objective is twofold: (1) we validate our approach using a proof of concept to show the feasibility of our approach and (2) we implement a tool, which is a plug-in within ProM, to make our approach more visible and widely used by the community. Source codes and tutorial of our *mineFrag* application are published at: <http://www-inf.it-sudparis.eu/SIMBAD/tools/mineFrag/>. A screen-shot of the application is shown in Figure 7.4.

<sup>4</sup><http://www.promtools.org/prom6/http://www.promtools.org/prom6/>

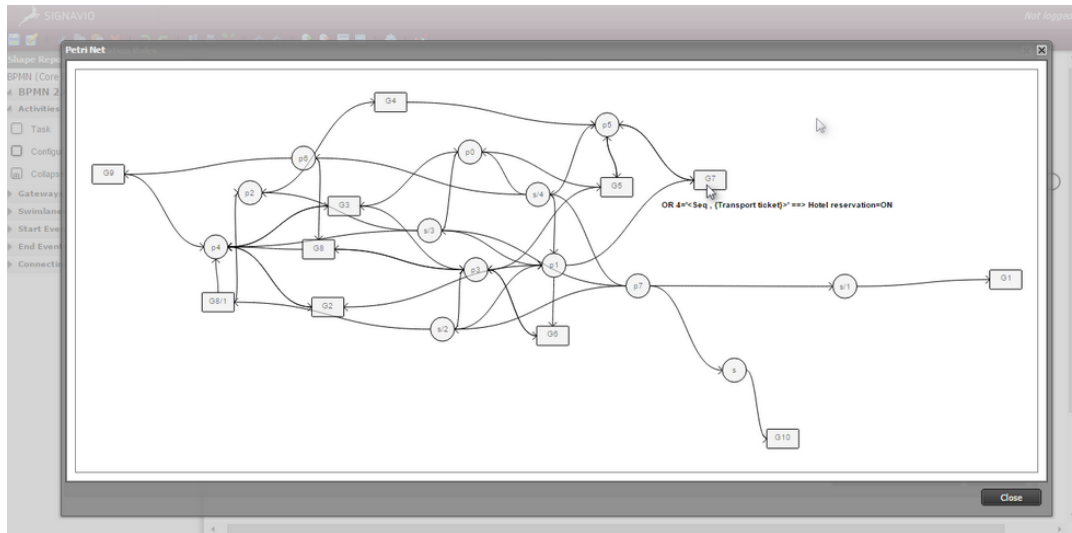


Figure 7.3: A screen-shot of the Signavio graphical interface for visualizing the configuration guidelines dependencies' in Petri-nets

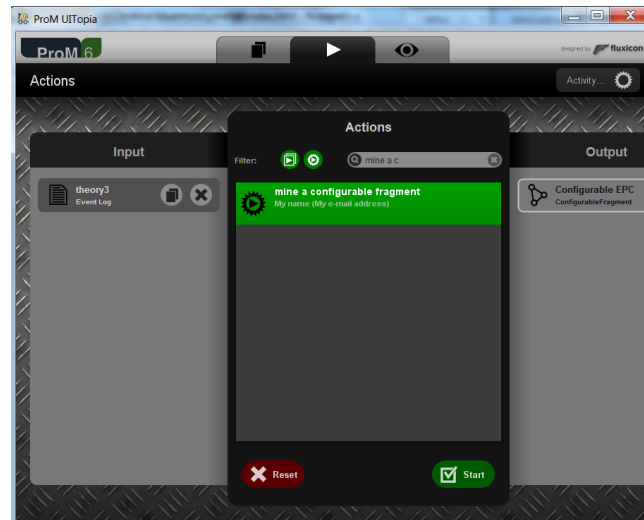


Figure 7.4: A screen-shot of the *mineFrag* plugin in ProM

The plugin takes as input an existing event log for which it discovers a process fragment in the EPC notation. It uses the alpha algorithm [127] and the *EPCConversion* plugins available in ProM in order to discover a petri-net based process and convert it into the EPC notation. The process user selects an activity from the discovered process for which he needs to discover a configurable fragment and its configuration guidelines. A screen-shot of this step is shown in Figure 7.5.

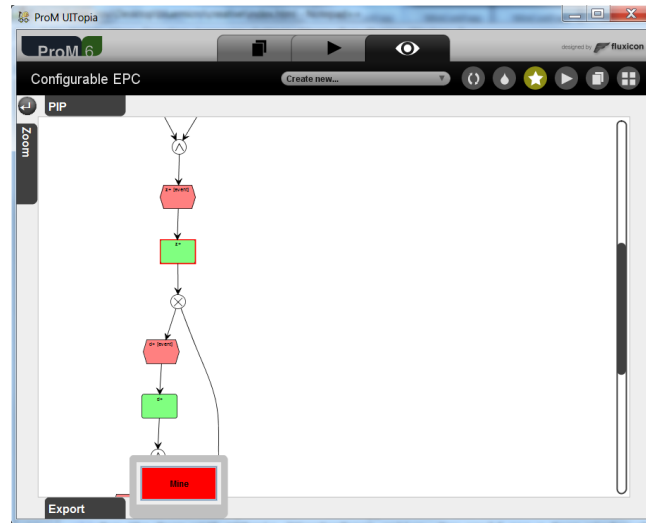


Figure 7.5: An activity selected in an existing business process for which a configurable fragment and its configuration guidelines will be discovered

We extended the alpha algorithm to mine a configurable process fragment. The output of the alpha algorithm is a Petri net. Therefore, we reused the *EPCConversion* plugin to convert the petri net into the EPC notation and then identify the configurable elements according to our approach (Section 6.2). The derivation of ranked configuration guidelines requires the computation of disjunctive normal forms (Section 6.3) which is NP-complete. Therefore, we used a Shared Binary Decision Diagram (SBDD) solver<sup>5</sup> that provides efficient algorithms to derive disjunctive normal forms. A screen-shot of the fragment configuration assisted with the derived ranked guidelines is shown in Figure 7.6.

### 7.3 Experimentation

In this section, we present the experiments that we performed to evaluate our approach. We firstly present the experiments for assisting the design of configurable process models (Section 7.3.1). We run the algorithms presented in chapter 4 on the public dataset from IBM research group. This dataset consists of real business processes designed for financial services, telecommunications and other domains. We evaluate the quality of the recommended configurable fragments. We also present the computation time of our algorithms. We analyze the parameters that impact the quality of our results and make a comparison with existing approaches mainly the works presented in [43, 44] as they are directly related to ours.

<sup>5</sup>We used the BDDC calculator available at: <http://www-verimag.imag.fr/~raymond/tools/bddc-manual/bddc-manual-pages.html>

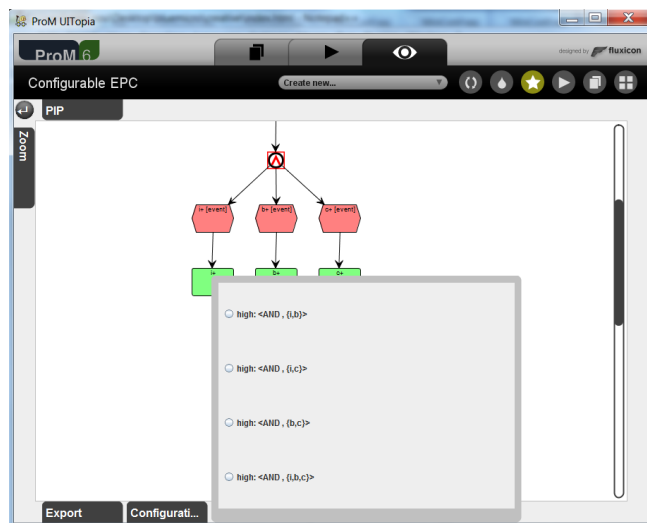


Figure 7.6: The configuration of the process fragment assisted with the ranked configuration guidelines

Second, we present the experiments for supporting the process configuration with configuration guidance models (Section 7.3.2). We run the algorithms presented in Chapter 5 on a dataset from the SAP reference model. We evaluate the accuracy of our extracted configuration guidance models by computing their precision and recall. We also evaluate the quality of the extracted configuration guidelines. We analyze the parameters that impact the quality of our results. We do not make a quantitative comparison with existing approaches as they are manual.

Thirdly, we present the experiments for the log-based process design and configuration (Section 7.3.3). We run the algorithms presented in Chapter 6 on a set of synthetic logs generated from the IBM dataset. We evaluate the quality of the discovered configurable fragments and analyze the parameters that impact it. We also evaluate the efficiency of our guideline-driven approach by computing the amount of reduction in the space of possible configurations.

### 7.3.1 Configurable Process Design Experiments

We performed our experiments on a large collection of real business process models. This dataset is shared by the Cognitive Computing and Industry Solutions research group<sup>6</sup> at the IBM Zurich Research Laboratory. It was presented in [69]. It contains business process models designed for financial services, telecommunications, and other domains. It is presented in XML format following BPMN 2.0 standard. Each XML file stores the data of a business process, including elements' IDs, activity names, and

<sup>6</sup><http://www.research.ibm.com/labs/zurich/ics/>

the sequence flows between elements. The dataset consists of 560 business processes with 6363 activities. There are 3781 different activities in which 1196 activities appear in more than one process. In average, there are 11.36 activities, 2.59 start events, 3.42 end events, 18.96 gateways (including OR, AND and XOR) and 46.81 sequence flows in a process (Table 7.1). In order to have focused results and show the performance of our approach with existing ones, we pre-processed the data to get only the activities that appear in more than 5 processes. We were left with 386 processes in which 162 activities appear in more than one process. Each activity is repeated in at least 5 processes and at most 65 processes.

|                                    | Min. | Max. | Avg.  |
|------------------------------------|------|------|-------|
| Nb. of activities in a process     | 1    | 195  | 11.36 |
| Nb. of start events in a process   | 1    | 32   | 2.59  |
| Nb. of end events in a process     | 1    | 32   | 3.42  |
| Nb. of gateways in a process       | 1    | 139  | 18.96 |
| Nb. of sequence flows in a process | 2    | 326  | 46.81 |

Table 7.1: Statistics of the dataset [2]

We performed three experiments to show that our approach is *feasible*, of *good results' quality* and of *good performance*. In the first experiment, we evaluate the feasibility of our approach by measuring the number of extracted fragments for a selected activity, the number of merged fragments and their reduction in size. We observe the impact of (i) the number of considered layers and (ii) the clustering step on the results' statistics (Section 7.3.1.1). In the second experiment, we evaluate the quality of the recommended configurable fragments in terms of *complexity and comprehensibility metrics* (Section 7.3.1.2). In the third experiment, we measure the performance of our algorithm based on the computation time. We compare the results with the works of La Rosa et al. [43] and Derguech et al. [44] (Section 7.3.1.3).

### 7.3.1.1 Approach feasibility and parameter impact

In this experiment, we run the proposed algorithm in chapter 4 (i) with different  $k^{th}$ -layer values and (ii) with and without the clustering step. We compute the similarity between each activity and the other activities in the other processes. Due to the limitation of the dataset, which provides only the elements' identifiers, the similarity between activities is based on a perfect syntactical matching of the activities' names. As presented in Chapter 4, we use the Agglomerative Hierarchical Clustering (AHC) algorithm with a complete link between clusters. We set the minimum similarity threshold to 0.5. We observe the overall minimum, maximum and average number of returned configurable fragments and their sizes with different parameter values. To assess the size of a configurable fragment and compare it with its merged fragments,



we compute the compression factor [185] which is defined as:

$$C = 1 - (\text{size}(^aG^k) / \sum_i (\text{size}(G_i^k))) \quad (7.1)$$

where  $^aG^k$  is the configurable process fragment of the activity  $a_x$  within  $k$ -layers, resulted from merging the set of the process fragments  $G_i^k$ . The size of the process fragment is the number of nodes (activities, events and gateways) in the graph (after transformation to the C-BPMN notation). The compression factor ranges from 0 to 1 and reflects the reduction in size of the merged fragment with respect to the sizes of the input process fragments. A compression factor close to 0.5 means that the merged fragment size is roughly half the sum of the input process fragments sizes and that the input process fragments are similar. A compression factor close to 0 means that the merged fragment size is equal to the sum of the input process fragments size and that the input process fragments are totally different.

|                               | Min. |      |       | Max. |      |       | Avg. |      |      |
|-------------------------------|------|------|-------|------|------|-------|------|------|------|
|                               | k=1  | k=2  | k=3   | k=1  | k=2  | k=3   | k=1  | k=2  | k=3  |
| Nb. of extracted fragments    | 6    |      |       | 65   |      |       | 8    |      |      |
| Results without clustering    |      |      |       |      |      |       |      |      |      |
| Nb. of configurable fragments | 1    |      |       | 1    |      |       | 1    |      |      |
| Compression factor            | 0.38 | 0.14 | 0.132 | 0.81 | 0.72 | 0.7   | 0.44 | 0.33 | 0.26 |
| Results with clustering       |      |      |       |      |      |       |      |      |      |
| Nb. of configurable fragments | 1.2  | 1.89 | 2.05  | 4.67 | 7.12 | 10.55 | 1.55 | 2.58 | 3.45 |
| Compression factor            | 0.56 | 0.5  | 0.42  | 0.92 | 0.85 | 0.83  | 0.61 | 0.52 | 0.45 |

Table 7.2: The overall minimum, maximum and average number of recommended fragments and their compression with different  $k^{\text{th}}$ -layer values/ with and without clustering

Table 7.2 summarizes the results for  $k \in \{1, 2, 3\}$ . It shows that our algorithm can find, for a selected activity, at least 6, at most 65 and in average 8 activities that are similar to the selected one.

Without the clustering step, the extracted fragments are merged at once and therefore one configurable fragment is recommended. With the clustering step, the extracted fragments are clustered based on their similarity and therefore the number of recommended fragments depends on the number of created clusters. We notice that *the number of created clusters increases when  $k$  increases* meaning that *the similarity between the extracted fragments decreases when  $k$  increases*. This can be explained by the fact that our algorithm merges similar activities and the connection flows connecting them. Since the behavior of an activity is strongly reflected by its closest neighbors, when  $k$  increases, the number of unmatched activities increases faster than the matched activities. This yields the reduction of similar fragments and therefore the increasing in the number of created clusters with larger values of  $k$ .

On the other side, the compression factor records larger values with the clustering step than without it. This is because with the clustering step, only highly similar

fragments are merged which leads to the increase of the compression factor. Without the clustering step, all the extracted fragments that may contain similar and dissimilar fragments are merged. This leads to the decrease of the compression factor. In both cases (with and without the clustering step), we notice that the compression factor decreases when  $k$  increases. Again this can be explained by the fact that the similarity between fragments decreases when  $k$  increases yielding to the reduction of the compression factor with larger values of  $k$ .

The compression factor has been also studied in [43,44]. In Table 7.3 we compare the results of these works with ours taking into account the clustering step. It is worth noting that the results of La Rosa et al. [43] are for pairs of merged fragments from the SAP reference model having a similarity above 0.5. The results of Derguech et al. [44] are for 5 merged registration processes taken from the civil affairs department of Dutch municipalities [186]. The similarity between the merged processes is not referenced in this work. In our approach, we merge at least 4, at most 26 and in average 3 fragments that have a similarity above 0.5. The results show that our approach records higher compression factor values in most cases. Especially, with  $k = 1$ , our approach performs better. The results can be explained by the fact that we focus on small parts instead of the entire processes. These parts reflects the behavior of similar activities in different processes which show a high similarity between each others.

| Compression factor | La Rosa et al. [43]<br>(pairs) | Derguech et al. [44]<br>(5 processes) | Our approach |             |             |               |
|--------------------|--------------------------------|---------------------------------------|--------------|-------------|-------------|---------------|
|                    |                                |                                       | $k = 1$      | $k = 2$     | $k = 3$     | Nb. fragments |
| Min.               | 0.5                            | 0.37                                  | <u>0.56</u>  | <u>0.5</u>  | <u>0.42</u> | 4             |
| Max.               | 1.06                           | 0.52                                  | <u>0.92</u>  | <u>0.85</u> | <u>0.83</u> | 26            |
| Avg.               | 0.69                           | 0.44                                  | <u>0.61</u>  | 0.52        | 0.45        | 3             |

Table 7.3: Comparison of the compression factor values with existing works

### 7.3.1.2 Results quality

In this experiment, we assess the quality of the recommended configurable fragments in terms of their structural complexity. We compute the well known complexity metrics proposed in the literature: **CFC** (Control Flow Complexity), **ACD** (Average Connector Degree), **CNC** (Coefficient of Network Connectivity) and **density**. The CFC [187] metric evaluates the complexity of the fragment by the presence of XOR-split, OR-split and AND-split and is defined as:

$$CFC = \sum_{c \in AND} 1 + \sum_{c \in XOR} |c \bullet| + \sum_{c \in OR} 2^{|c \bullet|} + 1 \quad (7.2)$$

The ACD [188] metric gives the number of nodes a connector is in average connected to and is defined as:

$$ACD = \frac{1}{|C|} \sum_{c \in C} |c \bullet| + |\bullet c| \quad (7.3)$$

The CNC [189] gives the ratio of edges to nodes and is defined as:

$$CNC = \frac{|A|}{|N|} \quad (7.4)$$

where  $|A|$  is the number of edges and  $|N|$  is the number of nodes. An increase in the CNC means that there exists a high number of connections between a relatively smaller number of nodes. The density metric [190] relates the number of available connections (i.e. edges) to the number of maximum connections that may exist between nodes and is defined as:

$$density = \frac{|A|}{|N| \times (|N| - 1)} \quad (7.5)$$

The density ranges from 0 to 1. A density close to 1 means that the process graph is highly dense, i.e. all possible connections between the nodes are present.

We set the size of fragments to  $k = 1$  and conduct a comparative analysis of the structural complexity between the proposed configurable fragments on the one side and the input process fragments on the other side. Our objective is to demonstrate that, although we aggregate multiple fragments into one model, our approach can recommend a configurable fragment of a reasonable complexity when compared to the average complexity of its input merged fragments. Table 7.4 summarizes the obtained results.

| Complexity metric | Merged fragment | input fragments |
|-------------------|-----------------|-----------------|
| CFC (avg)         | 7.41            | 3.71            |
| ACD (avg)         | 2.92            | 2.57            |
| CNC (avg)         | 0.95            | 1.13            |
| density (avg)     | 0.05            | 0.22            |

Table 7.4: Structural complexity metrics for the configurable fragments and the average of their corresponding merged fragments

We observe that the recommended configurable fragments have roughly the same structural complexity as the average complexity of the input merged fragments. However, a noticeable difference in the CFC metric can be detected. The CFC of our configurable fragments is in average twice the average of the CFC of the input fragments. This can be explained by the fact that in our merging algorithm, the matched connectors with different types are merged into an OR. From equation 7.2, we can see that the increase in the number of OR-split, increases the CFC metric. Nonetheless, as the authors in [191] state, models with the same structure but with different connector labels may have a huge difference in CFC while they are equally easy to understand. Yet, our approach can be optimized in order to minimize the CFC metric. For instance, we can impose a penalty weight when matching connectors with different types which would reduce the injection of OR-splits in the configurable fragment.

We also notice that the density records a very low value for the configurable fragments (0.05) which is less than the average of that of their input merged fragments (0.22). This can be explained by the fact that, when the input fragments are less similar or similar activities are connected with less similar connection flows, many configurable connectors are injected in the configurable fragment. This leads to the increase of the number of connections that may exist in the configurable fragment while the number of available connections remain slightly the same as in the input merged fragments. Consequently, the density of the configurable fragment decreases significantly compared to the average density of the input merged fragments.

The complexity of merged process models has been studied in the work of La Rosa et al. [43] in terms of *density*, *structuredness*, and *sequentiality* [192]. The structuredness refers to the degree to which a process graph is composed of single-entry single-exit (SESE) fragments. This metric is not valuable in our study as we merge small fragments representing the relations of an activity to its closest neighbors. Therefore, we cannot ensure the detection of SESE fragments. Sequentiality measures the degree to which a graph is constructed of sequences of nodes. In our approach, a high sequentiality (which is positively correlated with the understandability of a process graph) can be obtained only if in the input process fragments, there exist similar activities that are directly connected through a sequence flow. Therefore, the sequentiality does not depend on our merging algorithm but rather on the input fragments structure and therefore has not been considered in our study. The average result of the density metric in [43] for merged processes from the SAP reference model is 0.127 versus a density of 0.158 for the average of the input process models (modeled as “juxtaposed” models). The low difference between the density of configurable models and that of the input merged models in these results can be explained by the fact that the authors reduced the number of configurable elements in the configurable process by analyzing the “entanglement” causes in the merged models. This approach can be also directly applied on our configurable fragments to reduce the number of configurable elements.

### 7.3.1.3 Algorithm performance

We performed the experiments on a laptop with i5-3360M CPU, 2.80Hz, 8 GB memory, running Windows 7 64 bits and Java Virtual Machine version 1.5 with 512 MB of allocated memory. We set the number of considered layers to  $k = 3$ . We took three cases: first, we merged 6 fragments having a size between 3 and 19. Second, we merged 65 fragments having a size between 11 and 475. And last, we computed the average number of processes in which an activity is repeated. We found that, in average, an activity is repeated in 8 processes, and the corresponding fragments have a size between 3 and 475. The execution time results include the matching and merging time and are illustrated in Table 7.5. These results show that our algorithm can deal with a high number of fragments (65 fragments) in a small fraction of time

(2093 msec from which the merging takes 94 msec). Compared to the works in [43,44] that deal with only few numbers of complete process models, our approach can merge a larger number of fragments in a smaller time. For example, in [43], 2 process models having the sizes 339 and 357 respectively are merged in 7409 ms. And in [44], 5 processes having a minimal size of 24 and a maximal size of 56 are merged in 157 ms (this corresponds to the merging algorithm time, the matching time is not considered in this work).

| Nb. input fragments      | Min (size) | Max (size) | execution time (in msec) |
|--------------------------|------------|------------|--------------------------|
| 6                        | 3          | 19         | 108(16)                  |
| 65                       | 11         | 475        | 2093(94)                 |
| Average                  |            |            |                          |
| 8                        | 3          | 475        | (1904)16.48              |
| La Rosa et al. [43]      |            |            |                          |
| 2                        | 339        | 357        | 7409(79)                 |
| 2                        | 22         | 78         | 78(0)                    |
| Derguech and Bhiri. [44] |            |            |                          |
| 5                        | 24         | 56         | 157 (157)                |

Table 7.5: The execution time of our proposed algorithm compared with the existing works

#### 7.3.1.4 Synthesis

We performed experiments to evaluate the feasibility of our approach. We also observed the impact of parameters (including the  $k^{th}$ -layer and the clustering) on the number and compression of the recommended configurable fragments. Experimental results showed that our approach is feasible. They also showed that when  $k$  increases the similarity between the process fragments decreases and therefore the number of created clusters (which corresponds to the number of derived configurable fragments) increases. On the other side, we noticed that the clustering step is in favor of the reduction in size of the configurable fragments (which is computed in terms of the compression factor). The results showed that the configurable fragments obtained with a clustering step record higher compression factor values since they are resulted from merging highly similar fragments. We compared the obtained compression factor values of our results with the works of La Rosa et al. [43] and Derguech et al. [44] to show that merging small and focused fragments instead of entire process models improve the concision of the results and the computation time performance.

To evaluate the quality of our results, we computed the structural complexity of the configurable process fragments using well known complexity metrics proposed in the literature. We compared the obtained values with the average complexity of the input merged fragments. The results showed that, although we aggregate multiple

fragments into one model, our approach can recommend configurable fragments of a reasonable complexity. We analyzed the complexity metrics used in the approach by La Rosa et al. [43] and showed that the *density* metric was the only relevant one in our study. Although we obtained lower density values, our approach does not address the complexity caused by possible “entanglements” in the configurable fragments as the authors did in [43]. However, the “entanglements” in our fragments can be avoided using the same approach as in [43] since we use the C-BPMN notation which share common elements with the C-EPC notation used in [43].

To evaluate the performance of our algorithm, we measured the computation time. Experimental results showed that our algorithm has acceptable computation time as it computes the configurable fragments from 65 input fragments within a short time (in approximately 2 seconds) which is not very long for a business process matching and merging approach. Moreover, the comparative analysis with existing works [43,44] showed that our approach performs much more better, especially for the matching step. Our employed heuristics based on the neighborhood context graph structured in layers reduces greatly the matching time.

### 7.3.2 Process Configuration Experiments

In order to evaluate the configuration-based approach presented in Chapter 5, we performed experiments on a dataset from the SAP reference model which contains 604 models in EPC notation [70]. Our approach requires as input a configurable process model for which we extract the configuration guidance model from an existing business process repository. We take the SAP dataset as a business process repository. To create configurable process models, we propose to merge the similar processes in the SAP dataset into configurable process models in C-EPC notation. We do so by clustering similar processes using Agglomerative Hierarchical Clustering (AHC) method [193] and merging the resulted clusters using the merging algorithm presented in [43]. We chose in particular AHC with a *complete link* between clusters to compute their similarity, as it generates clusters with a high benefit-to-cost ratio for the SAP dataset [89]. To compute the similarity between process models, we use the graph edit distance [194] as it performs well for business process model matching [99]. The similarity between activities’ labels is computed as described in Section 4.3.1. We tune AHC to a minimum similarity threshold equal to 0.8 with a maximal number of possible clusters (i.e. AHC algorithm stops when the number of clusters is equal to 1 or there are no more clusters having a similarity value above 0.8). The rationale behind choosing a minimal similarity threshold equal to 0.8 is that we want to generate configurable process models from the *highly similar* processes in the SAP dataset and leave those that are less similar (e.g those that have a similarity above 0.5 and less than 0.8) to be used, with the highly similar ones, for extracting the configuration guidance models. By doing so, we simulate a real setting, in which we have a business process repository containing previously configured processes (in our case, the processes that

have been merged to create configurable ones) and other existing ones (in our case, the processes that have not be merged but that may be similar to a configurable process).

As a result, we obtain 20 clusters, each of them having at least two processes (trivial clusters with one process model are removed from the set of clusters). Each cluster is then merged into one configurable process model. The characteristics of each obtained cluster and the resulted configurable process models are reported in Table 7.6.

|                    | Nb. processes/cluster |     |      | size (Nb. nodes) |     |        | Nb. configurable nodes |     |       |
|--------------------|-----------------------|-----|------|------------------|-----|--------|------------------------|-----|-------|
|                    | min                   | max | avg. | min              | max | avg.   | min                    | max | avg.  |
| cluster            | 3                     | 8   | 4.35 | 3                | 94  | 23.15  | -                      | -   | -     |
| configurable model | -                     | -   | -    | 3                | 95  | 34.175 | 1                      | 36  | 5.575 |

Table 7.6: Statistics of the clusters and the configurable process models

The obtained configurable process models along with the SAP dataset have been used as inputs for the evaluation of the *quality* and *accuracy* of our results. In the first experiment, we evaluate the quality of the recommended configuration guidelines in terms of *completeness* and *complexity* (Section 7.3.2.1). In the second experiment, we evaluate the accuracy of the extracted configuration guidance models by computing the *Precision* and *Recall* values (Section 7.3.2.2). We also observe and analyze the impact of the Apriori support and confidence thresholds values on the results' quality and accuracy.

### 7.3.2.1 Results quality and parameter impact

In this experiment, we target to *study the relation between the support threshold values, the complexity and the completeness of the extracted configuration guidelines*. To do so, we compute (1) the number of extracted guidelines, (2) the number of configurations per guideline and (3) the percentage of extracted configurations per configurable element with different support threshold values and a confidence threshold  $C = 0.8$ . The complexity is expressed in term of the number of extracted guidelines. The higher the number of extracted guidelines, the more the complexity increases. The completeness is expressed in terms of the number of configurations per guideline and the percentage of extracted configurations per configurable element. On the one hand, a high number of configurations per guideline means that we are able to cover the association between the configurations of all configurable elements in a process model. On the other hand, a high percentage of retrieved configurations per element means that our guidelines cover all possible elements' configuration choices and that we are able to assist process users in almost all their configuration decisions.

The minimum, maximum and average number of extracted configuration guidelines with different support values are shown in Table 7.7. In Figure 7.7, the minimum,

maximum and average number of configurations per guideline and the percentage of extracted configurations per configurable element are depicted.

| # guidelines | Min. | Max.              | Avg.  |
|--------------|------|-------------------|-------|
| $S = 0.01$   | 2    | $7.5 \times 10^2$ | 101.7 |
| $S = 0.05$   | 2    | $6 \times 10^2$   | 92.32 |
| $S = 0.1$    | 2    | 123               | 40.56 |
| $S = 0.3$    | 0    | 30                | 10.7  |
| $S = 0.5$    | 0    | 10                | 5.2   |

Table 7.7: Number of guidelines for different minimum support threshold values and a minimum confidence threshold  $C = 0.8$

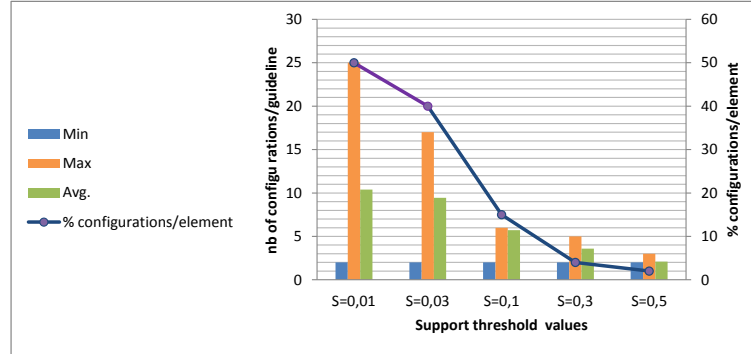


Figure 7.7: Number of configurations per guideline and per element for different minimum support thresholds and a minimum confidence threshold  $C = 0.8$

The results show three interesting findings. First, low support values ( $S = 0.01$  and  $S = 0.05$ ) record (1) a high number of extracted guidelines (101.7 and 92.32 in average in Table 7.7), (2) a high number of configurations per guidelines (20 and 17 in average in Figure 7.7 for a maximal number of 36 configurable elements), and (2) a high percentage of retrieved configurations per configurable element (50% and 40% in Figure 7.7). This leads to the conclusion that the complexity of the configuration guidelines is positively correlated with their completeness (i.e. when the complexity increases the completeness increases) while both the completeness and complexity are negatively correlated with the support threshold value (i.e. when the support decreases, the completeness and complexity increase). Therefore, one has to choose or to find a compromise between the complexity and the completeness of the results.

Second, for relatively low support values ( $S = 0.3$  and  $S = 0.5$ ), the number of guidelines (10.7 and 5.2 in average), the number of configuration per guidelines (3 in average) and the percentage of configurations per element (4% and 2% in average) are relatively too small. Thus, for higher support values, one can expect an empty



set of configuration guidelines. This can be explained by the fact that, even in one specific domain, the specific requirements of process users are largely diverse. Hence, the configuration decisions are closely related to the specific needs and therefore those that are selected in almost all process configurations are scarce.

Third, as shown in Figure 7.7, the percentage of retrieved configurations per element is the half of the valid element's configurations even with very low support threshold value ( $S = 0.01$ ). Again, this result shows that many configuration choices may be valid from the technical perspective of the configurable process language but invalid from a business perspective.

### 7.3.2.2 Approach accuracy and parameter impact

In the second experiment, we compute the precision and recall metrics of our configuration guidance models for different support and confidence threshold values. For each obtained configurable process model  $P^c$ , we extract its configuration guidance model  $\mathcal{G}_{\mathcal{M}}^c$  from  $\mathbb{P}$ .  $\mathbb{P}$  is constructed from the process models in the SAP dataset and contains the merged models of  $P^c$  and the process models that have a similarity above 0.5.

The configurations generated by our configuration guidance model  $\mathcal{G}_{\mathcal{M}}^c$  and the configurations retrieved from the process models in  $\mathbb{P}$  are stored in the matrices  $M_{generated}$  and  $M_{relevant}$  respectively.  $M_{generated}$  is a  $n \times m$  matrix where  $n$  is the number of generated configurations by  $\mathcal{G}_{\mathcal{M}}^c$  and  $m$  is the number of configurable elements in  $\mathcal{G}_{\mathcal{M}}^c$ . The  $i^{th}$  row corresponds to one process configuration generated by  $\mathcal{G}_{\mathcal{M}}^c$ . It is computed by generating one trace from the petri-net based representation (Section 5.7). The  $(i, j)^{th}$  entry corresponds to a configuration of the  $j^{th}$  element generated by the  $i^{th}$  process configuration.  $M_{relevant}$  is constructed from the processes in  $\mathbb{P}$  in the same way. Each row in  $M_{relevant}$  corresponds to a process configuration retrieved from  $P_i \in \mathbb{P}$  (see Section 5.5 for more details on the construction of  $M_{relevant}$ ). Having  $M_{generated}$  and  $M_{relevant}$ , the precision and recall are computed as in Equation (7.6).

$$precision = \frac{|M_{generated}| \cap_G |M_{relevant}|}{|M_{generated}|} \quad recall = \frac{|M_{generated}| \cap_R |M_{relevant}|}{|M_{relevant}|} \quad (7.6)$$

where  $|M_{generated}|$  and  $|M_{relevant}|$  return the number of rows in  $M_{generated}$  and  $M_{relevant}$  respectively;  $|M_{generated}| \cap_G |M_{relevant}|$  and  $|M_{generated}| \cap_R |M_{relevant}|$  are defined as in Equation (7.7).

$$\begin{aligned} |M_{generated}| \cap_G |M_{relevant}| &= \sum_i \max_j \left( \frac{|M_{generated}[i] \cap M_{relevant}[j]|}{|M_{generated}[i]|} \right) \\ |M_{generated}| \cap_R |M_{relevant}| &= \sum_j \max_i \left( \frac{|M_{generated}[i] \cap M_{relevant}[j]|}{|M_{relevant}[j]|} \right) \end{aligned} \quad (7.7)$$

where  $i = |M_{generated}|$  and  $j = |M_{relevant}|$ ;  $|M_{generated}[i]|$  returns the number of elements' configurations in the  $i^{th}$  row.

The results for the average values of the precision and recall with different Apriori support and confidence threshold values are reported in Figure 7.8.

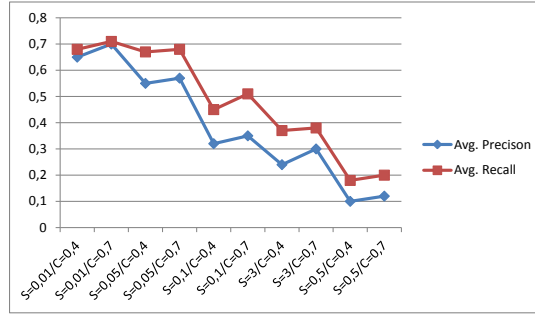


Figure 7.8: The average precision and recall values with different support and confidence thresholds

First, we notice that, both the precision and recall depend on the minimal support threshold rather than the minimal confidence threshold. Indeed, for the same support with different confidence values, we can see that the precision and recall curves do not record significant variations. For example, for a support  $S = 0.1$ , the precision is equal to 0.65 and 0.68 with confidence values equal to 0.4 and 0.7 respectively. While, for the same confidence with different support values, we clearly see the precision and recall curves' variations. For example, for a confidence  $C = 0.4$ , the precision is equal to 0.67 and 0.1 with support values equal to 0.01 and 0.5 respectively. This can be explained by the fact that the support threshold value determines the number of retrieved elements' configurations and therefore the generated process variants.

Second, we notice that the precision and recall values are inversely proportional to the support values. A low support records high precision and recall values while a high support records low precision and recall values. Thus, we can conclude that a high number of retrieved configurations (i.e. low support) is in favor of the configuration guidance model reproducibility (i.e. high recall) but, may be in our specific setting, to the detriment of the system generalization (i.e. high precision, the configuration guidance model is too specific, it does not generate new variants that may be inspired from the existing ones).

### 7.3.2.3 Synthesis

We performed experiments to evaluate the quality and accuracy of our approach. We observed the impact of the Apriori support and confidence thresholds values on the quality and accuracy of our results. We evaluated the quality of the recommended configuration guidelines in terms of *completeness* and *complexity*. Evaluation results

showed that the completeness of the configuration guidelines is positively correlated with their complexity. This means that when the completeness increases, the complexity also increases. The results also showed that the completeness and complexity are negatively correlated with the support threshold value. In other words, higher completeness and complexity values are achieved with lower support threshold values. Therefore, the process provider has to carefully select the support threshold value in order to balance between the completeness and the complexity of the returned results.

To evaluate the accuracy of our approach, we computed the *Precision* and *Recall* values of the extracted configuration guidance models. We examined 5 cases in which we computed the precision and recall values with different support and confidence threshold values. The results showed that the support threshold has a direct impact on the variations in the precision and recall while the confidence threshold does not record significant modifications. These results are explained by the fact that the number of retrieved configurations is determined by the support threshold rather than the confidence threshold. Experimental results showed also that higher precision and recall values are achieved with lower support values. Although, this is a desirable result, one may agree that moderate precision values may be preferred for a process provider who does not want to be restricted with the available information and therefore searches for more generalized configuration guidance models. Again, a compromise between the model reproducibility (i.e. high recall) and generalization (i.e. moderate precision) has to be found by carefully choosing the support threshold.

### 7.3.3 Log-based Experiments

In order to evaluate the log-based approach presented in Chapter 6, we conducted experiments on a set of synthetic event logs generated from the IBM dataset. Indeed, getting real logs from big size workflow examples that are enough various turns out to be a difficult task. The advantage in using simulated logs is that it is easier to fix and vary external factors ensuring a better diversity of the examples and a better and more accurate validation.

To generate logs, we use a log simulating tool [195] which creates random XML logs by simulating already designed workflow processes based on CPN tools. These tools support the modeling, the execution and the analysis of colored Petri nets [196]. They enable to create simulated logs conforming with the XML structure proposed in [197]. Modifications were brought to these tools to call predefined functions that create logs for each executed workflow instance. This stage implies modifications in the modeling level of CPN workflow declarations, particularly in the actions and the transition input/output levels. These functions indicate in particular the place, the prefix and the extension of the XML files that CPN tools create for each executed workflow instance.

Our set consists of 719 event logs in XES format for 560 different processes. The average number of traces in each log is between 50 and 299 traces. There exist in total

3003 distinct activities, from which 996 appear in multiple event logs. In average, an activity appears in 2 to 115 event log. In order to have focused results, we reduced the number of logs by taking those that contain the activities appeared in 10 to 20 logs. We were left with 223 event logs.

We evaluated our approach with respect to (i) the quality of the discovered configurable fragments (Section 7.3.3.1) and (ii) the efficiency of the discovered configuration guidelines (Section 7.3.3.2).

### 7.3.3.1 Configurable fragments quality

In this experiment, we assess the quality of the discovered configurable fragments in terms of their similarity with the origin process fragments from which they are discovered. This metric has been identified as a quality dimension in process discovery [138] and allows to quantify the difference between the discovered configurable process fragments and their input fragments with the aim of improving the process discovery. To compute the similarity, we use the structural and behavioral precision and recall metrics that were proposed in [93] to quantify the amount of equivalence between two process models based on observed behavior. Structural precision and recall compare two models  $M_1$  and  $M_2$  based on their graphical structure and are defined as:

$$Precision^S(M_1, M_2) = \frac{|E_1 \cap E_2|}{|E_2|}; \quad Recall^S(M_1, M_2) = \frac{|E_1 \cap E_2|}{|E_1|} \quad (7.8)$$

where  $E_1$  and  $E_2$  are the set of edges in  $M_1$  and  $M_2$  respectively;  $E_1$  returns the number of edges in  $M_1$ ;  $|E_1 \cap E_2|$  returns the set of common edges.  $Precision^S(M_1, M_2)$  returns the fraction of edges in  $M_2$  that are also present in  $M_1$ .  $Recall^S(M_1, M_2)$  returns the fraction of edges in  $M_1$  that are also present in  $M_2$ .

Behavioral precision and recall are inspired by the *fitness* notion [198, 199]. They compare two models  $M_1$  and  $M_2$  on the basis of an existing event log  $L$  that records a typical behavior and are defined as:

$$Precision(M_1, M_2, L) = \frac{\sum_{t \in L} \frac{\#t}{|t|} \sum_{i=0}^{|t|-1} \frac{|enabled(M_1, t[i], L) \cap enabled(M_2, t[i], L)|}{|enabled(M_2, t[i], L)|}}{|L|} \quad (7.9)$$

$$Recall(M_1, M_2, L) = \frac{\sum_{t \in L} \frac{\#t}{|t|} \sum_{i=0}^{|t|-1} \frac{|enabled(M_1, t[i], L) \cap enabled(M_2, t[i], L)|}{|enabled(M_1, t[i], L)|}}{|L|}$$

where:

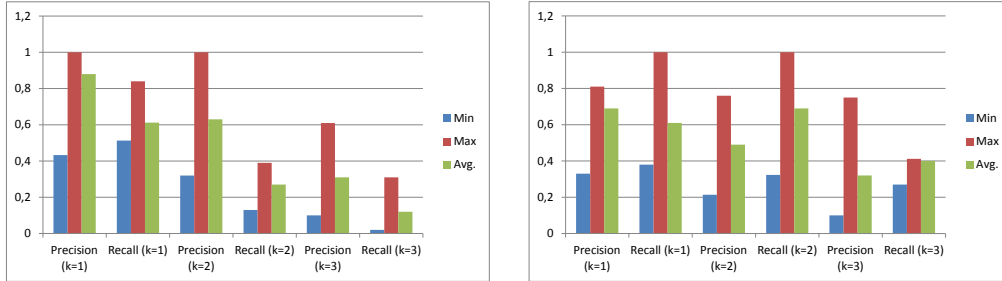
- $t$  is a trace in  $L$ ;  $\#t$  is the number of traces in  $L$ ;  $|t|$  is the number of events in  $t$ ;  $t[i]$  is the  $i^{th}$  event in  $t$ ;
- $enabled(M_1, t[i], L)$  returns, if it exists, the activity in  $M_1$  enabled by the event  $t[i]$  in  $L$ ;

- $|enabled(M_1, t[i], L) \cap enabled(M_2, t[i], L)|$  returns the number of activities enabled by the event  $t[i]$  in both  $M_1$  and  $M_2$ ;
- $|enabled(M_2, t[i], L)|$  returns the number of activities enabled by the event  $t[i]$  in  $M_2$ .

To compute the structural and behavioral precision and recall values of our discovered configurable fragments, we followed the following steps:

1. For each activity repeated in multiple event logs, we discover its corresponding configurable fragment  $P^c$ . Let  $\mathbb{L} = \{L_i\}$  be the set of event logs from which the fragment is discovered;
2. For each  $L_i \in \mathbb{L}$ , we discover a single process fragment. Let  $\mathbb{P} = \{P_i\}$  be the set of process fragments discovered from the set  $\mathbb{L}$  such that  $\forall 1 \leq i \leq |\mathbb{L}|$ ,  $P_i$  is discovered from  $L_i$ . Each discovered fragment represents one possible configuration of the configurable process fragment  $P^c$ ;
3. We compute the structural precision and recall of  $P^c$  with each of the processes  $P_i \in \mathbb{P}$ ;
4. We compute the behavioral precision and recall of  $P^c$  with each of the processes  $P_i \in \mathbb{P}$  based on the merged event log  $\mathbb{L}$ .

The results for the minimal, maximal and average of the structural and behavioral precision and recall values with varied  $k^{th}$ -layer values are reported in Figure 7.9a and Figure 7.9b respectively.



(a) Minimum, maximum and average values of the structural precision and recall values with different  $k^{th}$ -layer values

(b) Minimum, maximum and average values of the behavioral precision and recall values with different  $k^{th}$ -layer values

Figure 7.9: Precision and recall metrics values with different  $k^{th}$ -layer values

As a first inspection of the results, we can see that in overall the structural and behavioral recall are negatively correlated with  $k$ . They record high values for low  $k$  values. For instance, the structural precision with  $k = 1$  records a minimum of 0.433,

a maximum of 1 and an average of 0.88. However, low values can be observed when  $k$  increases. For example, the structural recall with  $k = 2$  records a minimum of 0.13, a maximum of 0.39 and an average of 0.27; similarly, the behavioral recall with  $k = 3$  records a minimum of 0.27, a maximum of 0.412 and an average of 0.4. These results validate the fact that, when  $k$  increases, the dissimilarity between process fragments increases leading to a configurable fragment with a relatively large size and many differences with respect to its input fragments. We also notice that the behavioral recall values outperform those of the structural recall even for low values. This can be explained by the fact that although when input fragments show many differences in their structure, they still share behavioral commonalities.

The results also show that in overall the differences between the minimum, maximum and average values are noticeable. For instance, the structural precision with  $k = 2$  records a minimum of 0.32, a maximum of 1 and an average of 0.63; the same holds for the behavioral recall with  $k = 1$  which records a minimum of 0.38, a maximum of 1 and an average of 0.61. This result means that, for the same  $k^{\text{th}}$ -layer value, some of the recommended configurable fragments are highly similar to their input fragments while others are highly dissimilar. To overcome this issue, log traces need to be clustered based on their similarity before a configurable fragment is discovered using for example the approaches presented in [139, 200, 201]. In this way, we can ensure that configurable fragments are discovered only from highly similar input fragments.

### 7.3.3.2 Configuration guidelines efficiency

In our approach, we do not replay the event logs on the discovered configurable fragment to identify elements' configurations as in [46, 47]. Instead, we mine a configurable process fragment that can be freely configured by the process user. Then, we mine ranked guidelines to assist the process users' configuring the fragments.

In this experiment, we evaluate the space of valid configurations with and without our configuration guidelines. We compute the average number of all possible configurations for each configurable process fragment. Then, we show that using our guideline driven approach, we restrict this number by assisting the process user in deriving a suitable fragment. We take the scenario for deriving the configurations having a high ranking (i.e.  $r = high$ ).

|          | configurable elements | Total possible configurations | configurations $r = high$ |
|----------|-----------------------|-------------------------------|---------------------------|
| Avg. Nb. | 25                    | $68 \times 10^{19}$           | 1565                      |

Table 7.8: The average number of possible configurations

The results reported in Table 7.8 show that, using our approach, the space of possible configurations is strongly reduced. The results also show that, among the

configurations observed in the event logs, only a small portion has a high frequency of execution. Therefore, depicting the frequency in configuration guidelines allows filtering the irrelevant configurations.

### 7.3.3.3 Synthesis

We performed experiments to evaluate the quality of our discovered configurable fragments. We computed the similarity between the configurable process fragments and each of their input fragments from which they are discovered. We used the structural and behavioral precision and recall values proposed in [93]. We observed the impact of the  $k^{\text{th}}$ -layer values on the quality of the configurable fragments.

The results showed that in overall, we achieve good quality values. These values decrease when the values of  $k$  increase. These results, also inline with the results of the compression factor computed in Section 7.3.1.1, validate our hypothesis that the dissimilarity between fragments increases when  $k$  increase. The results also showed that the behavioral recall values outperform those of the structural recall in almost all the cases meaning that even when a structural dissimilarity exists, behavioral similarity can be always observed. Finally, we observed that there exists a noticeable difference between the minimum, maximum and average values of the quality results for the same  $k$  value. This means that some of the results have relatively high quality values (i.e. the configurable fragment and its origin fragments are relatively similar) while others have relatively low values (i.e. the configurable fragment and its origin fragments are relatively dissimilar). To overcome this issue, we propose to cluster the log traces based on their similarity before discovering our configurable fragments. In this way, we can expect that configurable fragments are discovered from relatively similar process fragments and therefore share many similarities with them. We leave this for a future work.

We evaluated also the efficiency of our ranked configuration guidelines. We computed the space of valid configurations that can be generated by our configurable fragments with and without the assistance of the guidelines. We took the case for the guidelines having a *high* rank (i.e. guidelines for the configurations that are frequently executed in the logs). The results showed that the space of allowed configurations is dramatically reduced by our guidelines. They also showed that, among the configurations observed in the event logs, only a small portion has a high frequency of execution.

## 7.4 Case Study

To evaluate the practical usefulness of our frequency-based approach for process configuration (Chapter 5), we conducted a case study with a group of professionals (engineers in Service Oriented Architecture domain) and academics (from Business Process Management and Service Oriented Architectures fields). Our case study methodology

follows the guidelines presented in [184] and is structured in three main steps:

1. Define the case study objective in terms of research questions related to hypothesis [202] (Section 7.4.1);
2. Design the case, collect the data and execute them on the studied case (Section 7.4.2);
3. Analyze the results and present the findings (Section 7.4.3).

### 7.4.1 Case Study Objective

Through this case study, we aim to assess the usefulness of a configuration guidance model when process providers create their configuration support systems using existing manual approaches. Therefore, our research questions are defined as follows:

- *Q1: How configuration guidance models can assist process providers in the creation of their configuration support systems?*
- *Q2: Do configuration guidance models provide sufficient and necessary information for the configuration support system construction?*

In order to answer these questions, we formulate two hypotheses:

- *h1: Configuration guidance models save time and facilitate the identification of the configuration steps order and the configuration guidelines;*
- *h2: When configuration guidance models are used, additional configuration guidelines that were unknown according to the process provider knowledge can be identified.*

Given the above questions and hypotheses, we present in the next section the case design, data collection and execution.

### 7.4.2 Design, Data Collection and Execution

Our case study is from the Telecommunication domain and corresponds to a configurable service supervision process adopted by Orange, a french telecommunication company. This process is used by different affiliates of Orange in different cities and countries and is configured according to their specific needs. After different meetings with Orange agents, we collected different variants used by Orange affiliates. The collected variants along with the configurable process are used as inputs in order to extract a configuration guidance model.

With a population of around 20 participants, we targeted to build a configuration support system that assists Orange affiliates during the configuration of the process. The participants are professionals (engineers in Service Oriented Architecture



domain) and academics (from Business Process Management and Service Oriented Architectures fields) that are familiar with process configuration. With the purpose of collecting different results, we partitioned the population into two ways. First, the persons are divided into binomials groups of (1) domain experts who do not understand the technical details of the configurable process and (2) business process experts who are aware of the configurable process technical details. Then, half of the groups are asked to manually create a configuration support system (we refer to them as group  $G_M$ ) while the other half are provided a configuration guidance model and are asked to create the configuration support system (we refer to them as group  $G_A$ ).

The configuration support systems are modeled according the questionnaire approach [7]. Briefly, this approach consists of a framework that allows to capture the process model variability based on a set of questions defined by domain experts. This questionnaire model includes order dependencies and domain constraints represented as logic expressions over facts (i.e. answers to questions). We choose in particular this approach because (1) it provides an abstraction from the notation used to model the configurable process and (2) it is suitable to the users who are not competent in modeling notations [203].

After a workshop organized to explain the basics needed in this study, we asked the groups in  $G_A$  and  $G_M$  to build a configuration support system for our configurable process model. Then, we asked them to map the created configuration support systems to the variation points in the configurable process. This step allows to link the questions and answers in the configuration support system (i.e. questionnaire model) to the configurable elements and configuration choices in the configurable process.

The resulted configuration support systems with the established mappings are then collected for comparison. In order to answer the two identified research questions and confirm their hypotheses, we evaluated the results according to two parameters: (1) the time needed to build the configuration support system and to define the mapping with and without the assistance of a configuration guidance model and (2) the amount of information provided by the configuration support systems constructed with and without the assistance of a configuration guidance model. The amount of information is computed based on the number and the granularity of the identified domain constraints.

### 7.4.3 Results Analysis and Findings

Firstly, regarding the time needed to build the configuration support system, the groups in  $G_M$  took in average 12 man-hours while the groups in  $G_A$  took 6 man-hours. Table 7.9 shows the distribution of the time on the different parts of the work. From this table, we notice that the two groups took approximately the same time to define the set of questions. This is mainly because the domain experts, who are naturally responsible of defining the questions, will not take advantage of the technical guidance provided by our configuration guidance model. Therefore, they were based

on their own knowledge to define the questions.

|       | Identify questions | Identify order dependencies | Identify domain constraints | Define mapping |
|-------|--------------------|-----------------------------|-----------------------------|----------------|
| $G_M$ | 2 man-hours        | 4 man-hours                 | 5 man-hours                 | 1 man-hour     |
| $G_A$ | 2.5 man-hour       | 2 man-hours                 | 1 man-hour                  | 0.5 man-hour   |

Table 7.9: The average time in man-hour unit spent to build a configuration support system with and without the assistance of configuration guidance models

The remaining parts of the work show a significant difference in the time. In order to understand these results, we asked the groups in  $G_A$  on the methodology they followed to do the job. First, most of the groups affirmed that, in order to benefit from the information provided by the configuration guidance model, they unconsciously defined the mapping between the questions and the configurable elements in the configuration guidance model before starting the other steps. Once this mapping is established, they easily followed (1) the order dependencies between the configurable elements in the configuration guidance model and mapped them to the questions order and (2) the inclusion and exclusion relations in the configuration guidance model and mapped them to the domain constraints. These results support the hypothesis that configuration guidance model can save time and assist the creation of configuration support systems, mainly in the identification of the questions' order (i.e. configuration steps order) and the configuration constraints (i.e. configuration guidelines).

Secondly, the created configuration support systems were compared against the number and the granularity of the identified domain constraints. We found that the groups in  $G_A$  identified approximately triple the number of the constraints identified by the groups in  $G_M$ . A closer analysis allowed us to group many of the constraints identified by  $G_A$  and map them to individual constraints identified by  $G_M$ . This result can be explained by the fact that the constraints derived from the configuration guidance model are related to the process structure and are defined at a fine granular level (i.e. at the level of the process elements). While in reality, the domain experts define more generic constraints that may encompass many configuration decisions. Although the high number of configuration constraints may increase the complexity of the model, a lower number with more generic constraints can make the mapping between the configuration support system and the configurable process a tedious task. Therefore, a compromise between the granularity of a technical-based model and the generality of a business-based model should be found.

On the other side, we noticed that approximately 30% of the constraints identified by  $G_A$  were missed by  $G_M$ . The missed constraints were mainly those that are not directly related to the domain but are rather reflected by the users' behavior and therefore included in the configuration guidance model. Yet, the domain experts confirmed that these constraints are not contradictory with the domain. These results confirm our second hypothesis that configuration guidance models provide additional

information on the configuration constraints that should be taken into account when creating a configuration support system.

#### 7.4.4 Threats to Validity

There are several threats to validity in our study. The first one is related to our assumption that we collected a representative set of process variants, from which we learned the configuration guidance model. However, in order to ensure the extraction of a representative and generalized configuration guidance model, a large number of process variants that represent the entire configuration space is required. In the current study, we have not fully explored this assumption apart from ensuring that the identified process variants are relevant and depict various business needs.

The second one is related to the validity and generalization of our case study results. Our method has been applied on one case study from an industrial partner and has been conducted by group of professionals and academics. The results depend on the specific domain chosen for the study and the participants background. While insights can be drawn from our study results, we do not claim that they can be generalized. These results serve as a basis for evaluating a frequency-based approach for process configuration. We believe that a broader set of domains and a larger group of participants with varied backgrounds need to be studied in order to ensure the external validity and reliability of the case study results [184]. We leave this to a future work.

The third one is that we did not considered in this study the approach for assisting the design of configurable process models. We believe that the potential economic benefits (e.g. time-savings, errors reduction, increased model quality, etc.) of reusing previously modeled processes to create new ones have been well studied and proved in the literature [24, 204, 205].

## 7.5 Conclusion

In this chapter, we answered the question raised in the thesis problematic (Section 1.2), which is: *How efficient our approach is?* We presented the implementations, experiments and case study to validate our approach. Three applications, developed as extensions of Signavio and ProM, were implemented to prove that our approach is feasible.

We performed experiments with two large datasets from IBM and the SAP reference model. Experimental results showed that our approach recommends comprehensible configurable fragments of low complexity and has a good performance. They also showed that the extracted configuration guidance models are of good quality and accurate.

We conducted a case study with professionals and academics to evaluate the practical usefulness of a frequency-based approach for process configuration. The study

results showed that our approach saves time and recommends configuration guidelines that were not be possible to be identified based only on the expert knowledge.



# Conclusion and Future Works

---

The research problem of this thesis is expressed by this interrogation: *How to propose an automated support for configurable process models?* Previous chapters presented in details our solutions to answer the raised question. In this chapter, we summary our work 8.1 and present the future work 8.2.

## 8.1 Contributions

Configurable process models allow a systematic reuse of business processes in a flexible way. They are gaining momentum due to their capability of explicitly representing the common and variable parts of similar processes into one customizable process model. A configurable process model needs to be configured to derive individual process variants that suit the specific requirements of an organization. Supporting the variability in configurable process models has been a hot topic over the last years. It became a big challenge that involves many researches in both academics and industry.

Contemporary approaches address this problem in different ways. They develop configurable process modeling languages to allow process variability modeling. They propose automated approaches to build configurable process models by merging or mining similar process variants. To derive individual variants, they create configuration support systems that assist process users selecting desirable configuration choices. Despite the considerable advances achieved by exiting works, serious challenges still exist regarding the (i) the complexity of the created configurable process models and (ii) the level of automation in the creation of configuration support systems.

On the one hand, existing approaches target to build entire configurable processes which result in large and complex models that are difficult to reuse and manage. They also face the computational complexity and scalability problems when merging or mining a high number of large processes with hundreds of process elements. To address this issue, we proposed in this thesis an approach for assisting the design of configurable process models in a fine-grained way using *configurable process fragments*. Our approach gives the hand to process providers to specify the configurable parts in their processes and can be used in two typical cases:

- when a process provider is looking for process fragments that are suitable to fill-in a missing part in an ongoing designed process;
- and when he wants to extend or improve specific parts in an existing configurable process.

To identify process fragments that are close to process provider interests, we used the *neighborhood context graph* which is defined as a process fragment around a selected activity. A neighborhood context graphs contains the associated activity and the relations to its closest neighbors. To derive configurable process fragments, we proposed to extract the neighborhood context graphs of the activities that are similar to a a selected one from different business process models. The extracted graphs are matched and clustered based on their similarity before being merged. We also proposed to capture and extract the *log-based neighborhood contexts* of an activity from a collection of event logs and then mine a configurable process fragment.

On the other hand, existing approaches for creating configuration support systems rely heavily on the domain expert knowledge. The configuration support systems are to a large extent manually constructed by domain experts, which is time-consuming and costly. In addition, relying solely on the expert knowledge is not only error-prone, but also challengeable in today's dynamic and fast changing business requirements. To address this issue, we proposed in this thesis an automated approach for extracting *configuration guidance models* from existing business process repositories. Our configuration guidance models can be used in three typical cases:

- when a process provider needs an assistance to create a configuration support system for a configurable process model. Instead of starting from scratch, a configuration guidance model recommends him the order in which the configuration steps are performed and the configuration guidelines. These information are necessary for creating a configuration support system;
- when a process provider wants to understand and analyze the variability in his process. Such analysis can yield useful information in order to improve the quality of the designed configurable process. In this regard, learning from previous process configurations gives him insights on how the configurable process is actually used by process users;
- when a process user is aware of the technical details of the configurable process model. Instead of using a business-driven model, he can directly use our configuration guidance model as a configuration support system.

We proposed a two-step approach for extracting a configuration guidance model from a repository of process models. The first step consists of extracting *configuration guidelines* using data mining techniques in particular Association Rule Mining. We also proposed to formalize the guidelines dependencies' relations represented in

Petri-nets using the Theory of Regions. This step is required to ensure a correct application of the guidelines. The second step consists of inferring the order in which the configuration steps are performed using Graph Theory techniques in particular the derivation of optimal spanning trees.

To validate our approach, we developed three proof of concepts *FragMerg*, *ConfRule* and *MineFrag* as extensions of Signavio and ProM. *FragMerg* and *ConfRule* are developed as extensions of Signavio to validate our approach for recommending configurable process fragments and extracting configuration guidance models. *MineFrag* is developed as a plugin in ProM to validate our log-based approach for mining configurable process fragments and configuration guidelines. We performed experiments with two large datasets from IBM and the SAP reference model. Experimental results showed that our approach (i) recommends comprehensible configurable fragments of low complexity, (ii) extract accurate configuration guidance models and (iii) has a good performance. We also conducted a case study with professionals and academics to evaluate the practical usefulness of a frequency-based approach for process configuration. The study results showed that our approach saves time and recommends configuration guidelines that were not be possible to be identified based only on the expert knowledge.

The principles presented in Section 1.4.1 have been respected:

- *Automation:* We propose an automated approach for deriving configurable process fragments and supporting the creation of configuration support systems. Our solutions use automated techniques such as similarity search, clustering, process merging, process mining and data mining. We do not ask process providers for additional information. We learn from existing data, which are business process models and event logs to generate our results.
- *Implicit knowledge exploitation:* We exploit implicit knowledge hidden in existing data. Concretely, we extract (i) neighborhood contexts of activities for deriving our configurable fragments and (ii) configuration guidelines for creating our configuration guidance models from process models and event logs.
- *Focused results:* For process design, our approach recommends configurable process fragments instead of entire business processes. For process configuration, our approach recommends the configuration steps order and the element' configurations instead of entire process configurations. So, our recommendations are focused and easy to apprehend.
- *Balanced computation:* The complexity of our algorithms is polynomial and we do not face the NP-complete problem. The computation time is acceptable for users.

Last but not least, our approach is complementary and can be associated to other approaches to better provide an automated support. For example, in the process configuration context, our approach can be associated to the domain-based approaches



to build configuration support systems that take into account specific business considerations but also best practices in a specific domain.

## 8.2 Future work

In the future work, we intend to improve the automated support quality of our current work (Section 8.2.1) and extend the applicability of process configuration to the cloud computing environment (Section 8.2.2).

### 8.2.1 Improving automated support quality

Currently, our work takes into account only the control-flow perspective in business processes. In our future work, we intend to extend our approach by taking into account other equally important perspectives such as the resource and data perspectives. We intend to automatically derive configurable process fragments that capture the resources and data involved in the execution of the activities. We could represent our fragments using the configurable multi-perspective process modeling language called C-iEPC that has been recently proposed as an extension of the C-EPC notation [73].

We also want to enrich our configuration guidance models with multi-perspective configuration guidelines. Currently our configuration guidelines depict how the configuration choices of the control flow elements are interrelated in a process model. By taking into account other perspectives, one can derive four types of configuration guidelines: (i) control-flow based guidelines, (ii) resource-based guidelines, (iii) data-based guidelines and (iv) hybrid guidelines. The latter allows to depict how the configuration choices of different perspectives are interrelated.

Moreover, we intend to improve our configuration guidance model in two ways. First, we aim at providing a zooming-in/-out facility that allows to navigate at different levels of abstractions. We could use natural language processing [206], clustering and business process abstraction techniques [207] to identify and group the closely related configurable elements in the configuration guidance model. Second, we plan to map our model to a domain-based representation by using domain ontologies for example. This allows us to enrich the model with business-driven information and facilitate its use by domain experts.

Our approach can be also extended to take into account the structural (absence of disconnected nodes) and behavioral (absence of deadlocks and livelocks) correctness of the variants that can be derived by our configuration guidance model. Currently, our approach ensures a consistent and correct application of the configuration guidelines. For instance, we guarantee that some of the guidelines cannot be applied concurrently as they result in different configurations for the the same configurable element. However, we do not guarantee that the application of the guidelines does not result in structural or behavioral issues in the derived variants. Although some approaches have been proposed to verify the correctness of the configured variants [58, 59], it

would be interesting to allow an a-priori detection of the configuration guidelines that result in structural and behavioral anomalies so that they can be removed from the configuration guidance model.

In our approach, we consider the best configurations as those that are frequently selected in a process model or recorded in an event log. Additional parameters can be considered for a best configuration selection such as the configurations that maximize some performance metrics (e.g. configuration with a minimal execution time, a minimal cost, or a combination of these two metrics, etc.). This issue, identified in [3] as a serious limitation has not been addressed before. For example, we could use existing works in process mining for performance analysis [208–210] in order to mine performance metrics for the variation points. We can leverage the process configuration to a Constraint Satisfaction Problem (CSP) [211] in order to derive a configuration that satisfies a user defined performance constraints. We use a CSP solver along with a user specified objective function to find all the optimal configurations.

### 8.2.2 Business process configuration in the cloud

Cloud Computing has emerged as an advanced paradigm for developing and providing services over the internet. Its innovation lies in its economic model for enabling ubiquitous, convenient and on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with a minimal effort [212]. In this paradigm, there are basically three layers of services known as “IaaS” for Infrastructure as a Service, “PaaS” for Platform as a Service and “SaaS” for Software as a Service.

Provisioning business processes in the cloud allows enterprises to adopt agile, flexible and cost-effective business solutions and also to reduce process development and maintenance costs. Specifically, using *configurable business processes* allows a cloud provider to share a common process among different tenants that is customized and executed according to specific needs [213].

The provisioning of cloud-based configurable business processes consists of two main steps (i) allocating the required cloud resources (from IaaS, PaaS or SaaS) to host and execute the process activities and (ii) deploying the process before starting configuring and executing it by different tenants. The resource allocation consists of specifying the needed resources for each activity, their providers, the dependencies that may exist between them (e.g. substitution, backup, peering, etc.), the allowed actions (e.g. create, edit, duplicate), etc. following one of the Cloud standards such as TOSCA (Topology and Orchestration Specification for Cloud Applications) [214] or OCCI (Open Cloud Computing Interface defined by the Open Grid Forum) [215]. The deployment consists of instantiating the process and its specified resources. The instantiation of the resources consists of creating the manifest file in OCCI or the topology instance in TOSCA.

However, in such a multi-tenant environment, following a traditional resource al-

location approach in which the needed resources, their dependencies and actions are pre-defined in a *rigid* and *static* way is unrealistic. Since different tenants have different requirements and needs, the resource allocation should also account for variability. In fact, the configurable process model should be extended with *resource configuration facilities* that allow the tenants to customize the needed resources, their dependencies and the actions allowed on them. Although there exist some works on resource configuration in business processes [5, 49, 73], they all addressed the problem in a generic way and did not take into account the specificity of cloud resources' properties (such as elastic or not, shareable or not) nor the configuration of the dependencies and actions between them. The identification of the configuration operations for cloud resource allocation remains an open question.

In addition, the deployment of a cloud-based configurable process model should take into account the variable, dynamic and economic nature of the cloud ecosystem (e.g. pay-as-you go model, dynamic resource assignment and release, variable and dynamic cloud offers, etc.). It would be inefficient if all the allocated process resources are instantiated along with their dependencies and actions while some of them will not be used if they are not selected by the tenant during process configuration. Therefore, a *configurable deployment* approach that allows to explicitly state that some of the allocated resources need to be configured and should not be physically instantiated is required. Once the tenant completes the configuration, a *deployment configuration* in which the selected resources can be physically instantiated is derived. To enable such a configurable deployment, cloud resources' definition and management should be also extended with configuration facilities.

# Appendices



# Proof for $TS_{\mathbb{G}}$ is a directed acyclic graph

First, we give two definitions, a configuration path and a state length in  $TS_{\mathbb{G}}$ .

**Definition A.0.1** (configuration path  $P_{s_n}^{s_m}$ ). *A configuration path from a state  $s_n \in S$  to a state  $s_m \in S$  denoted as  $P_{s_n}^{s_m} = \langle s_n, \dots, s_m \rangle$  is a sequence of states  $s_i \in S : 1 \leq i \leq k$  such that  $s_1 = s_n, s_k = s_m \wedge \forall 1 \leq i \leq k-1, (s_i, G_i, s_{i+1}) \in T$ .*

**Definition A.0.2** (State length  $|s_{\neq-}|$ ). *The length of a process configuration state  $s \in S$ , denoted as  $|s_{\neq-}|$  is the number of entries in its corresponding vector that are different from “-”.*

**Proposition A.0.1.** *The transition system  $TS_{\mathbb{G}}$  is a directed acyclic graph, i.e.  $\nexists P_{s_1}^{s_m} = \langle s_1, \dots, s_m \rangle : s_1 = s_m$ .*

*Proof.* (By contradiction)  $TS_{\mathbb{G}}$  is cyclic iff it contains a loop, i.e.  $\exists P_{s_1}^{s_m} : s_1 = s_m$ . According to Definition 5.7.2, for a configuration guideline  $G$ ,  $|\mathbb{V}_{G.pre_{\neq-}}^c| < |\mathbb{V}_{G.post_{\neq-}}^c|$ , i.e. the number of configured entries in the pre-configuration state of  $G$  is **strictly less** than the number of configured entries in the post-configuration state. According to Definition 5.7.3, a transition  $(s', G, s'') \in T$ , iff  $s' \in Pre_G \wedge s'' \in Post_G$ , therefore  $\forall (s', G, s'') \in T$  we have  $|s'_{\neq-}| < |s''_{\neq-}|$ .

Three cases can be presented:

1.  $P_{s_1}^{s_m}$  is a self-loop, i.e.  $s_1 = s_m \wedge (s_1, G, s_1) \in P$ . In this case we have  $|s_{1_{\neq-}}| < |s_{1_{\neq-}}|$  which is contradictory.
2.  $P_{s_1}^{s_m}$  is a direct loop, i.e.  $\exists G, G' \in \mathbb{G} : (s_1, G, s_m) \in T \wedge (s_m, G', s_1) \in T$ . In this case we have  $|s_{1_{\neq-}}| < |s_{m_{\neq-}}|$  and  $|s_{m_{\neq-}}| < |s_{1_{\neq-}}|$  which is contradictory.
3.  $P_{s_1}^{s_m}$  is an indirect loop, i.e.  $\exists G_1, G', G_m \in \mathbb{G} \wedge \exists s' \in P_{s_1}^{s_m} : (s_1, G_1, s') \in T \wedge (s', G', s_m) \in T \wedge (s_m, G_m, s_1) \in T$ . In this case we have  $|s_{1_{\neq-}}| < |s'_{\neq-}|$  and  $|s'_{\neq-}| < |s_{m_{\neq-}}|$  and  $|s_{m_{\neq-}}| < |s_{1_{\neq-}}|$ . Thus  $|s_{1_{\neq-}}| < |s_{m_{\neq-}}|$  and  $|s_{m_{\neq-}}| < |s_{1_{\neq-}}|$  which is contradictory.

□



# List of Publications

## Journal Article

1. Nour Assy, Nguyen Ngoc Chan, Walid Gaaloul and Bruno Defude, *An Automated Approach for Assisting the Design of Configurable Process Models*, IEEE Transactions on Services Computing, 2015 (To appear).
2. Karn Yongsiriwit, Nour Assy and Walid Gaaloul, *A Semantic Framework for Multi-tenant Business Process Design*, Journal of Networking and Computer Applications published by Elsevier, 2015 (To appear).
3. Nour Assy, Nguyen Ngoc Chan, Walid Gaaloul and Bruno Defude, *Deriving Configurable Fragments for Process Design*, International Journal of Business Process Integration and Management, IJBPIIM 2014, InderScience, ISSN:1741-8771, Volume 7, pages 2-21, 2014.

## Conference Proceeding

1. Nour Assy and Walid Gaaloul, *Extracting Configuration Guidance Models from Business Process Repositories*, Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015.
2. Nour Assy and Walid Gaaloul, *Configuration Rule Mining for Variability Analysis in Configurable Process Models*, 12<sup>th</sup> International Conference on Service Oriented Computing, ICSOC 2014, France, Paris, November 03-06, 2014.
3. Nour Assy, Walid Gaaloul and Bruno Defude, *Mining Configurable Process Fragments for Business Process Design*, 9th International Conference, DESRIST 2014, Miami, FL, USA, May 22-24, 2014.
4. Nour Assy, Karn Yongsiriwit, Walid Gaaloul and Imen Grida Ben Yahia, *A Framework for Semantic Telco Process Management-An Industrial Case Study*, 14<sup>th</sup> International Conference on Intelligent Systems Design and Applications, IEEE ISDA 2014, Okinawa, Japan, November 27-29, 2014.



5. Nour Assy, Nguyen Ngoc Chan and Walid Gaaloul, *Assisting Business Process Design with Configurable Process Fragments*, 10th International Conference on Services Computing, IEEE SCC 2013, Santa Clara, CA, USA, June 27-July 2, 2013.

# Proof of Concepts

1. “Merging configurable process fragments”, extension of Signavio process editor at <http://www-inf.it-sudparis.eu/SIMBAD/tools/fragmerg/>
2. “Mining configurable process fragments”, a ProM plugin at <http://www-inf.it-sudparis.eu/SIMBAD/tools/mineFrag/>
3. “Assisting the configuration of business process models”, extension of Signavio at <http://www-inf.it-sudparis.eu/SIMBAD/tools/confRule/>



## APPENDIX D

# Résumé

Les systèmes d'information centrés processus sont de plus en plus adoptés par les entreprises d'aujourd'hui afin d'assurer une gestion et une exécution optimales de leurs processus explicitement conçus, dénommés processus métiers [10]. Cependant, avec l'évolution rapide et continue dans les exigences métiers, il n'y a aucun doute que la mise en place de nouveaux paradigmes pour la gestion des processus des entreprises se transforme en un besoin pressant. Dans un tel environnement très dynamique, la réutilisation [24] et l'adaptabilité [25] des processus devient une exigence pour une conception de processus prospère. À cette fin, *les modèles de processus configurables* [34] fournissent un moyen de modélisation de la variabilité dans les modèles de processus de référence. Un modèle de processus configurable est un modèle générique qui intègre multiples variantes d'un même processus dans un seul modèle personnalisable grâce à des points de variation. Ces points sont appelés *éléments configurables* et permettent de multiples options de conception dans le modèle de processus. Un modèle de processus configurable doit être configuré selon une exigence spécifique en sélectionnant une option de conception pour chaque élément configurable. De cette manière, une variante individuelle du processus est dérivée avec un effort de conception minimal.

La conception et la configuration des modèles de processus configurables impliquent de plus en plus de nombreuses activités de recherche dans les secteurs académiques et industriels. D'une part, la conception manuelle des modèles de processus configurables est sans aucun doute une tâche coûteuse et source d'erreurs. Bien que des approches automatiques ont été proposées dans la littérature, elles ont tous ciblées la construction d'un modèle de processus configurable en entier [43–47]. Cela a conduit à des processus très larges et complexes qui sont difficiles à réutiliser. D'autre part, avec un nombre croissant d'éléments configurables dans le modèle de processus et de nombreuses interdépendances entre leurs choix de configuration, les utilisateurs ont besoin de moyens de soutien pour configurer le processus. Pour combler cette lacune, certains travaux proposent d'utiliser des questionnaires [7], des modèles de fonction connu sous *feature models* [6, 50] ou des ontologies [48] afin d'assister les utilisateurs pour choisir les bonnes configurations selon leurs besoins. Cependant, les approches existantes nécessitent un *travail manuel coûteux* de la part des experts de domaine pour (1) analyser et identifier les contraintes de domaine et (2) créer le *système de configuration* (à savoir les questionnaires, les feature models, les ontologies) pour guider le processus de configuration. D'une part, la capture et l'identification

manuelle de toutes les contraintes de domaine possibles nécessitent un travail intensif. D'autre part, elles sont non seulement sujettes aux erreurs, mais aussi représentent un défi.

Dans cette thèse, nous abordons les inconvénients mentionnés ci-dessus en proposant une approche automatisée pour soutenir la conception et la configuration des modèles de processus configurables. Nous cibons assister les analystes métiers (i) à concevoir leurs modèles de processus configurables d'une manière fine pour éviter des résultats larges et complexes et (ii) à créer leurs systèmes d'aide à la configuration avec un effort manuel minimal. Pour ce faire, nous proposons d'apprendre de l'expérience passée dans la modélisation et la configuration des processus afin (i) de dériver des fragments de processus configurables qui répondent aux besoins des analystes et (ii) de générer des modèles d'aide à la configuration en utilisant des techniques de fouilles de données et de la théorie des graphes.

Afin de recommander des fragments configurables, nous définissons un fragment de processus comme un graphe qui se compose d'une activité et de ses relations à ses activités de voisinage [2], dénommé *Neighborhood Context Graph (NCG)*. Cette définition est ciblée et granulaire de sorte qu'elle permet aux analystes métiers de voir les interactions possibles d'une activité à ses plus proches voisins. Pour une activité choisie par l'analyste, nous proposons de découvrir à partir des processus dans un référentiel existant, les NCGs autour des activités ayant une fonctionnalité similaire à celle sélectionnée. Ces fragments sont *extraits, regroupés et fusionnés* en fragments de processus configurables. Les fragments configurables générés contiennent l'activité choisie et ses relations avec ses plus proches voisins dans les différents processus à travers des éléments configurables.

Pour soutenir la création de systèmes d'aide à la configuration, nous introduisons le nouveau concept *configuration guidance model (CGM)* qui fournit des recommandations sur (i) les directives de configuration pour la sélection des choix de configuration souhaitables dans un processus configurable et (ii) l'ordre des étapes de configuration. Nous proposons une approche automatisée pour extraire les CGMs à partir des référentiels de processus existants.

La première étape consiste à *extraire des directives de configuration* à partir de modèles de processus métier existants. Ces directives montrent comment les décisions de configuration sont interdépendantes dans un modèle de processus configurable. Pour ce faire, nous proposons d'utiliser des techniques de fouille de données [61], en particulier la fouille des règles d'association [62]. Nous remarquons que les directives de configuration dérivées *doivent être soigneusement et correctement appliquées pour éviter des résultats de configuration incohérents*. Par conséquent, nous poussons plus loin notre travail et nous proposons de formaliser les dépendances entre les directives de configuration à l'aide des réseaux de Petri [63]. Nous identifions trois principales relations de dépendances qui peuvent exister entre les différentes directives de configuration: *causalité, concurrence* et *exclusivité*. Ces relations sont automatiquement dérivées en utilisant la théorie des régions [64].

La deuxième étape consiste à *déduire l'ordre dans lequel les étapes de configuration sont effectuées*. Pour ce faire, nous proposons de déduire un ordre partiel entre les éléments configurables du modèle de processus. Nous remarquons que la structure du processus impose un ordre partiel entre les éléments configurables, mais ce dernier ne reflète pas leur dépendance d'un point de vue de la configuration. Par conséquent, nous proposons une autre approche qui tient compte de la *dépendance entre les choix de configuration des éléments* et construit une structure en forme d'arbre constitué d'éléments configurables dans des relations parents-enfants (à savoir l'élément parent est configuré avant l'élément enfant). Pour ce faire, nous utilisons des techniques de la Théorie des graphes, en particulier la dérivation de arbres couvrant [65].

Enfin, nous proposons une approche basée sur les historiques d'exécution des processus pour aider à la conception et à la configuration des modèles de processus configurables. Nous proposons d'utiliser les techniques de fouille de processus (connu sous process mining) afin de *découvrir des fragments de processus configurables à partir des traces d'exécution*. Nous proposons également de *découvrir des directives de configuration* pour supporter la configuration du fragment découvert. Ces directives tiennent compte de l'importance de l'exécution des activités qui se traduit par leur présence dans les traces d'exécution. Pour ce faire, nous utilisons les arbres de suffixes [?] et la théorie des ensembles.

Nous avons validé notre approche en trois étapes. Tout d'abord, nous avons développé trois proof-of-concepts *FragMerg*, *ConfRule* et *MineFrag* comme des extensions de Signavio [?], un outil de modélisation de processus basé sur le Web et ProM [68], un framework extensible de fouille de processus. *FragMerg* est une extension de Signavio et permet de modéliser et de recommander des fragments de processus configurables pour une activité choisie dans un processus métier. *ConfRule* est aussi une extension de Signavio et permet d'extraire un CGM pour un processus configurable. *MineFrag* est un plugin de ProM et permet de recommander des fragments de processus configurables avec des directives de configuration.

Deuxièmement, nous avons effectué des expérimentations avec deux grands ensembles de modèles de processus d'IBM [69] et de SAP [70]. Nous avons évalué la *faisabilité, l'efficacité et la précision* de nos solutions proposées. Nous avons calculé des statistiques sur les résultats pour estimer leur qualité. En plus, nous avons calculé les valeurs précision et de rappel et nous avons mesuré les performances de nos algorithmes en terme de temps de calcul. Nous avons également analysé les paramètres qui influent sur la qualité de nos résultats.

Troisièmement, nous avons réalisé une étude de cas avec des professionnels et des universitaires afin de montrer l'utilité pratique d'une approche basée sur l'apprentissage du passé pour la configuration des processus. Grâce à cette étude de cas, nous avons cherché à évaluer l'utilité de nos CGMs lorsque les analystes métiers construisent leurs systèmes d'aide à la configuration en utilisant les approches manuelles existantes.



# Bibliography

- [1] H.M.W. Eric Verbeek and R. P. Jagadeesh Chandra Bose. ProM 6 Tutorial. Technical report, 2010.
- [2] Ngoc Chan Nguyen. *Service recommendation for individual and process use*. PhD thesis, Th. doct. : Informatique, Université d'Evry Val d'Essonne, Institut Mines-Télécom-Télécom SudParis, Directeur de thèse: TATA Samir, Encadrant: GAALOUL Walid, december 2012, 2012.
- [3] F. Gottschalk. *Configurable Process Models*. Phd thesis, Eindhoven University of Technology, December 2009.
- [4] Marcello La Rosa, Wil M.P. van der Aalst, Marlon Dumas, and Fredrik P. Milani. Business process variability modeling : A survey, 2013. ACM Computing Surveys.
- [5] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Capturing variability in business process models: The provop approach. *J. Softw. Maint. Evol.*, 22(6-7):519–546, 2010.
- [6] Gerd Gröner, Marko Bošković, Fernando Silva Parreiras, and Dragan Gašević. Modeling and validation of business process families. *Information Systems*, 38(5):709 – 726, 2013.
- [7] Marcello La Rosa, Wil M.P. van der Aalst, Marlon Dumas, and Arthur H.M. ter Hofstede. Questionnaire-based variability modeling for system configuration. *Software & Systems Modeling*, 8(2):251–274, 2009.
- [8] Antonucci Y.L. Using workflow technologies to improve organizational competitiveness. *International journal of management*, 14(1):117–126, 1997.
- [9] Richard Lenz and Manfred Reichert. It support for healthcare processes - premises, challenges, perspectives. *Data Knowl. Eng.*, 61(1):39–58, April 2007.
- [10] Marlon Dumas, Wil M. van der Aalst, and Arthur H. ter Hofstede. *Process-aware Information Systems: Bridging People and Software Through Process Technology*. John Wiley & Sons, Inc., New York, NY, USA, 2005.
- [11] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske. Business process management: A survey. In *Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003, Proceedings*, pages 1–12, 2003.
- [12] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition*. Springer, 2012.



- 
- [13] Howard Smith and Peter Fingar. *Business process management: the third wave*. Springer, 2003.
- [14] Wil M. P. van der Aalst and Kees M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.
- [15] Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distrib. Parallel Databases*, 3(2):119–153, April 1995.
- [16] Becker J., Kugeler M., and Rosemann M. *Process Management: A guide for the design of business processes*. Springer, 2003.
- [17] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
- [18] W. M. P. van der Aalst and A. H. M. ter Hofstede. Yawl: Yet another workflow language. *Inf. Syst.*, 30(4):245–275, June 2005.
- [19] OMG. Business process model and notation (bpmn) 2.0. <http://www.omg.org/spec/BPMN/2.0/>, 2011.
- [20] v2.3 OMG. Unified modelling language. <http://www.omg.org/spec/uml/2.3/>.
- [21] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 2011.
- [22] Jörg Becker, Martin Kugeler, and Michael Rosemann. *Process management : a guide for the design of business processes / Jörg Becker, Martin Kugeler, Michael Rosemann, editors*. Springer Berlin ; Heidelberg ; New York, 2003.
- [23] Wil M. P. van der Aalst. Workflow verification: Finding control-flow errors using petri-net-based techniques. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 161–183, London, UK, UK, 2000. Springer-Verlag.
- [24] Peter Fettke and Peter Loos. Classification of reference models: a methodology and its application. *Information Systems and eBusiness Management*, 2003.
- [25] Helen Schonenberg and et al. Towards a taxonomy of process flexibility. In *CAiSE Forum*, pages 81–84, 2008.
- [26] Rania Khalaf. From rosettanelt pips to bpm processes: a three level approach for business protocols. In *BPM '05*, 2005.

- 
- [27] Matthias Kunze and Mathias Weske. Metric trees for efficient similarity search in large process model repositories. In *Business Process Management Workshops '10*. 2010.
- [28] Zhiqiang Yan, Remco M. Dijkman, and Paul W. P. J. Grefen. Fast business process similarity search. *Distributed and Parallel Databases*, 2012.
- [29] Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. Context-based service recommendation for assisting business process design. In *E-Commerce and Web Technologies - 12th International Conference, EC-Web 2011, Toulouse, France, August 30 - September 1, 2011. Proceedings*, pages 39–51, 2011.
- [30] Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. Assisting business process design by activity neighborhood context matching. In *Service-Oriented Computing - 10th International Conference, ICSOC 2012, Shanghai, China, November 12-15, 2012. Proceedings*, pages 541–549, 2012.
- [31] Gabriel Hermosillo, Lionel Seinturier, and Laurence Duchien. Using Complex Event Processing for Dynamic Business Process Adaptation. In *IEEE International Conference on Services Computing*, pages 466–473, 2010.
- [32] Manfred Reichert and Barbara Weber. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer Publishing Company, Incorporated, 2012.
- [33] Irina Rychkova and Selmin Nurcan. Towards adaptability and control for knowledge-intensive business processes: Declarative configurable process specifications. pages 1–10. IEEE Computer Society, 2011.
- [34] M. Rosemann and W. M. P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 32(1):1–23, 2007.
- [35] Michiel Koning, Chang-ai Sun, Marco Sinnema, and Paris Avgeriou. Vxbpel: Supporting variability for web services in bpel. *Inf. Softw. Technol.*, 2009.
- [36] Walid Fdhila, Aymen Baouab, Karim Dahman, Claude Godart, Olivier Perrin, and François Charoy. Change propagation in decentralized composite web services. In Dimitrios Georgakopoulos and James B. D. Joshi, editors, *CollaborateCom*, pages 508–511. ICST / IEEE, 2011.
- [37] Chang-ai Sun, Rowan Rossing, Marco Sinnema, Pavel Bulanov, and Marco Aiello. Modeling and managing the variability of web service-based systems. *J. Syst. Softw.*, 2010.
- [38] Milan Milanovic, Dragan Gasevic, and Luis Rocha. Modeling flexible business processes with business rule patterns. EDOC '11.

- [39] Gabriel Hermosillo, Lionel Seinturier, and Laurence Duchien. Creating context-adaptive business processes. In Paul P. Maglio, Mathias Weske, Jian Yang, and Marcelo Fantinato, editors, *Service-Oriented Computing*, volume 6470 of *Lecture Notes in Computer Science*, pages 228–242. Springer Berlin Heidelberg, 2010.
- [40] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni, and Jean-Bernard Stefani. A component-based middleware platform for reconfigurable service-oriented architectures. *Softw. Pract. Exper.*, 42(5):559–583, May 2012.
- [41] Florian Gottschalk, Wil M. P. van der Aalst, Monique H. Jansen-Vullers, and Marcello La Rosa. Configurable workflow models. *Int. J. Cooperative Inf. Syst.*, 17(2):177–221, 2008.
- [42] Remco M. Dijkman, Marcello La Rosa, and Hajo A. Reijers. Managing large collections of business process models - current techniques and challenges. *Computers in Industry*, 2012.
- [43] Marcello La Rosa, Marlon Dumas, Reina Uba, and Remco Dijkman. Business process model merging: An approach to business process consolidation. *ACM Trans. Softw. Eng. Methodol.*, 22(2):11:1–11:42, 2013.
- [44] Wassim Derguech and Sami Bhiri. An automation support for creating configurable process models. In *Web Information System Engineering - WISE 2011 - 12th International Conference, Sydney, Australia, October 13-14, 2011. Proceedings*, pages 199–212, 2011.
- [45] Florian Gottschalk, Wil M. Aalst, and Monique H. Jansen-Vullers. Merging event-driven process chains. In *OTM '08*. 2008.
- [46] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Mining configurable process models from collections of event logs. In *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, pages 33–48, 2013.
- [47] Florian Gottschalk, Wil M. P. van der Aalst, and Monique H. Jansen-Vullers. Mining reference process models and their configurations. In *On the Move to Meaningful Internet Systems: OTM 2008 Workshops, OTM Confederated International Workshops and Posters, ADI, AWeSoMe, COMBEK, EI2N, IWSSA, MONET, OnToContent + QSI, ORM, PerSys, RDDS, SEMELS, and SWWS 2008, Monterrey, Mexico, November 9-14, 2008. Proceedings*, pages 263–272, 2008.
- [48] Ying Huang, Zaiwen Feng, Keqing He, and Yiwang Huang. Ontology-based configuration for service-based business process model. In *IEEE SCC*, pages 296–303, 2013.

- 
- [49] Akhil Kumar and Wen Yao. Design and management of flexible process variants using templates and rules. *Comput. Ind.*, 63(2):112–130, 2012.
- [50] Mohsen Asadi, Bardia Mohabbati, Gerd Gröner, and Dragan Gasevic. Development and validation of customized process models. *Journal of Systems and Software*, 96:73–92, 2014.
- [51] Jan Mendling and Carlo Simon. Business process design by view integration. In Johann Eder and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 55–64. Springer Berlin Heidelberg, 2006.
- [52] Mehrdad Sabetzadeh and Steve Easterbrook. An algebraic framework for merging incomplete and inconsistent views. Technical report, 2004.
- [53] Hajo A. Reijers, R. S. Mans, and Robert A. van der Toorn. Improved model management with aggregated business process models. *Data Knowl. Eng.*, 2009.
- [54] Monique H. Jansen-Vullers, Wil M. P. van der Aalst, and Michael Rosemann. Mining configurable enterprise information systems. *Data Knowl. Eng.*, 2006.
- [55] Remco Dijkman, Marlon Dumas, and Luciano Garcia-Banuelos. Graph matching algorithms for business process model similarity search. In *BPM '09*.
- [56] Matthias Weidlich, Remco Dijkman, and Jan Mendling. The icop framework: identification of correspondences between process models. In *CAiSE '10*.
- [57] Marcello La Rosa and Marlon Dumas. Configurable process models: How to adopt standard practices in your own way?, 2008.
- [58] Wil M. P. van der Aalst, Niels Lohmann, and Marcello La Rosa. Ensuring correctness during process configuration via partner synthesis. *Inf. Syst.*, 37(6):574–592, 2012.
- [59] Wil M. P. van der Aalst, Marlon Dumas, Florian Gottschalk, Arthur H. M. ter Hofstede, Marcello La Rosa, and Jan Mendling. Preserving correctness during business process model configuration. *Formal Asp. Comput.*, 22(3-4):459–482, 2010.
- [60] WilM.P. van der Aalst. Challenges in business process analysis. In Joaquim Filipe, José Cordeiro, and Jorge Cardoso, editors, *Enterprise Information Systems*, volume 12 of *Lecture Notes in Business Information Processing*, pages 27–42. Springer Berlin Heidelberg, 2008.
- [61] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., 2005.

- [62] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD'93*, pages 207–216.
- [63] Jörg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press, New York, NY, USA, 1995.
- [64] Jörg Desel and Wolfgang Reisig. The synthesis problem of petri nets. *Acta Informatica*, 33(4):297–315, 1996.
- [65] H N Gabow, Z Galil, T Spencer, and R E Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, January 1986.
- [66] Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 1976.
- [67] Signavio. <http://www.signavio.com/>.
- [68] van Dongen and et al. The prom framework: A new era in process mining tool support. ICATPN '05.
- [69] Dirk Fahland, Cédric Favre, Jana Koehler, Niels Lohmann, Hagen Völzer, and Karsten Wolf. Analysis on demand: Instantaneous soundness checking of industrial business process models. *Data Knowl. Eng.*, 2011.
- [70] Gerhard Keller and Thomas Teufel. *Sap R/3 Process Oriented Implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1998.
- [71] GeorgeM. Giaglis. A taxonomy of business process modeling and information systems modeling techniques. *International Journal of Flexible Manufacturing Systems*, 13(2):209–228, 2001.
- [72] Jan Recker, Michael Rosemann, Wil M. P. van der Aalst, and Jan Mendling. On the syntax of reference model configuration - transforming the C-EPC into lawful EPC models. In *Business Process Management Workshops, BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, Nancy, France, September 5, 2005, Revised Selected Papers*, pages 497–511, 2005.
- [73] Marcello La Rosa, Marlon Dumas, Arthur H. M. ter Hofstede, and Jan Mendling. Configurable multi-perspective business process models. *Inf. Syst.*, 36(2):313–340, April 2011.
- [74] Mark Vervuurt. Modeling business process variability : a search for innovative solutions to business process variability modeling problems, October 2007.

- 
- [75] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Managing process variants in the process life cycle. Technical report, University of Twente, 2007.
- [76] Frank Puhlmann, Arnd Schnieders, Jens Weiland, and Mathias Weske. Variability Mechanisms for Process Models. Technical report.
- [77] Arnd Schnieders and Frank Puhlmann. Variability mechanisms in e-business process families. In *Proc. International Conference on Business Information Systems (BIS 2006)*, pages 583–601. 2006.
- [78] Monika Weidmann, Falko Kötter, Maximilien Kintz, Daniel Schleicher, Ralph Mietzner, and Frank Leymann. Adaptive Business Process Modeling in the Internet of Services (ABIS). In Proceedings of the Sixth International Conference on Internet, Web Applications, and Services (ICIW) 2011, editors, *Adaptive Business Process Modeling in the Internet of Services (ABIS)*, pages 29–34. Xpert Publishing Services, March 2011.
- [79] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, and J. Mylopoulos. On Goal-based Variability Acquisition and Analysis. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, 2006.
- [80] Florian Gottschalk, Wil M. P. van der Aalst, Monique H. Jansen-Vullers, and H. M. W. Verbeek. Protos2cpn: using colored petri nets for configuring and testing business processes. *STTT*, 10(1):95–110, 2008.
- [81] Maryam Razavian and Ramtin Khosravi. Modeling variability in business process models using uml. In *Proceedings of the Fifth International Conference on Information Technology: New Generations*, ITNG '08, pages 82–87, 2008.
- [82] D. M. M. Schunselaar and H. M. W. Verbeek. Creating sound and reversible configurable processes models using cosenets. Technical report, 2011.
- [83] August-Wilhelm Scheer. *ARIS - vom Geschäftsprozess zum Anwendungssystem*. Springer, 4., durchges. aufl. edition, 2002.
- [84] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., 2000.
- [85] A. Lie, R. Conradi, T. M. Didriksen, and E.-A. Karlsson. Change oriented versioning in a software engineering database. *SIGSOFT Softw. Eng. Notes*, 14(7):56–65, 1989.
- [86] Eirik Tryggeseth, Bjorn Gulla, and Reidar Conradi. Modelling systems with variability using the proteus configuration language. In *Selected Papers from the ICSE SCM-4 and SCM-5 Workshops, on Software Configuration Management*, pages 216–240, London, UK, UK, 1995. Springer-Verlag.

- [87] Florian Gottschalk, WilM.P. van der Aalst, and MoniqueH. Jansen-Vullers. Configurable process models — a foundational approach. In Jörg Becker and Patrick Delfmann, editors, *Reference Modeling*, pages 59–77. Physica-Verlag HD, 2007.
- [88] C. Li, M. Reichert, and A. Wombacher. The MINADEPT Clustering Approach for Discovering Reference Process Models Out of Process Variants. *International Journal of Cooperative Information Systems*, 19(3-4):159–203, 2010.
- [89] Chathura C. Ekanayake, Marlon Dumas, Luciano García-Bañuelos, Marcello La Rosa, and Arthur H. M. ter Hofstede. Approximate clone detection in repositories of business process models. In *BPM*, volume 7481, pages 302–318. 2012.
- [90] Roger Tregear. Business process standardization. In *Handbook on Business Process Management 2, Strategic Alignment, Governance, People and Culture, 2nd Ed.*, pages 421–441. 2015.
- [91] Remco Dijkman, Marlon Dumas, Boudewijn van Dongen, Reina Käärrik, and Jan Mendling. Similarity of business process models: Metrics and evaluation. *Inf. Syst.*, 36(2):498–516, April 2011.
- [92] Matthias Kunze, Matthias Weidlich, and Mathias Weske. Behavioral similarity - a proper metric. In *BPM*, pages 166–181. 2011.
- [93] A. K. Alves de Medeiros, W. M. P. van der Aalst, and A. J. M. M. Weijters. Quantifying process equivalence based on observed behavior. *Data Knowl. Eng.*, 2008.
- [94] Remco Dijkman. A classification of differences between similar business processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, EDOC '07*, pages 37–, Washington, DC, USA, 2007. IEEE Computer Society.
- [95] Chen Li, Manfred Reichert, and Andreas Wombacher. On measuring process model similarity based on high-level change operations. In Qing Li, Stefano Spaccapietra, Eric Yu, and Antoni Olivé, editors, *Conceptual Modeling - ER 2008*, volume 5231 of *Lecture Notes in Computer Science*, pages 248–264. Springer Berlin Heidelberg, 2008.
- [96] Marc Ehrig, Agnes Koschmider, and Andreas Oberweis. Measuring similarity between semantic business process models. In *Proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling - Volume 67, APCCM '07*, pages 71–80, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

- 
- [97] Christopher Klinkmueller, Ingo Weber, Jan Mendling, Henrik Leopold, and Andre Ludwig. Increasing recall of process model matching by improved activity label matching. In *International Conference on Business Process Management*, pages 211–218. China, 2013.
- [98] Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas, and Luciano García-Bañuelos. Behavioral comparison of process models based on canonically reduced event structures. In Shazia Sadiq, Pnina Soffer, and Hagen Völzer, editors, *Business Process Management*, volume 8659 of *Lecture Notes in Computer Science*, pages 267–282. Springer International Publishing, 2014.
- [99] Boudewijn F. van Dongen, Remco M. Dijkman, and Jan Mendling. Measuring similarity between business process models. In *CAiSE*, volume 5074, pages 450–464, 2008.
- [100] B. Mahleko and A. Wombacher. Indexing business processes based on annotated finite state automata. In *IEEE International Conference on Web Services, ICWS 2006*, pages 303–311, Los Alamitos, September 2006. IEEE Computer Society Press.
- [101] Marlon Dumas, Luciano García-Bañuelos, Marcello La Rosa, and Reina Uba. Fast detection of exact clones in business process model repositories. *Inf. Syst.*, 38(4):619–633, 2013.
- [102] VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 1966.
- [103] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recogn. Lett.*, 18(9):689–694, 1997.
- [104] Deborah L. McGuinness and Frank van Harmelen. Owl web ontology language overview. Technical report, W3C, 2004.
- [105] Alexander Maedche and Steffen Staab. Measuring similarity between ontologies. In *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 251–263, London, UK, 2002. Springer-Verlag.
- [106] Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. Wordnet: : Similarity - measuring the relatedness of concepts. In *AAAI*, pages 1024–1025, 2004.
- [107] Saartje Brockmans, Marc Ehrig, Agnes Koschmider, Andreas Oberweis, and Rudi Studer. Semantic alignment of business processes. In *IN: PROCEEDINGS OF THE EIGHTH INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS (ICEIS 2006)*, pages 191–196. INSTICC Press, 2006.



- [108] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.
- [109] Kristina Toutanova and Christopher D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora: Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13, EMNLP '00*, pages 63–70. 2000.
- [110] Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning, ICML '98*, pages 296–304. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.
- [111] Matthias Weidlich, Jan Mendling, and Mathias Weske. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Trans. Software Eng.*, 37(3):410–429, 2011.
- [112] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*. Springer Publishing Company, Incorporated, 1st edition, 2010.
- [113] Contextual petri nets, asymmetric event structures, and processes. *Information and Computation*, 171(1):1 – 49, 2001.
- [114] Rob van Glabbeek and Ursula Goltz. Equivalence notions for concurrent systems and refinement of actions. In Antoni Kreczmar and Grazyna Mirkowska, editors, *Mathematical Foundations of Computer Science 1989*, volume 379 of *Lecture Notes in Computer Science*, pages 237–248. Springer Berlin Heidelberg, 1989.
- [115] Mohammad Abdulkader Abdulrahim. *Parallel algorithms for labeled graph matching*. PhD thesis, Golden, CO, USA, 1998.
- [116] Reina Uba, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Clone detection in repositories of business process models. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 248–264. Springer Berlin Heidelberg, 2011.
- [117] Jochen Malte Küster, Christian Gerth, Alexander Förster, and Gregor Engels. A tool for process merging in business-driven development. In *CAiSE '08*.
- [118] Shuang Sun, Akhil Kumar, and John Yen. Merging workflows: A new perspective on connecting business processes. *Decision Support Systems*, 42(2):844 – 858, 2006.

- 
- [119] MohamedAnis Zemni, NejibBen Hadj-Alouane, and Amel Mammar. Business process fragments behavioral merge. In Robert Meersman, Hervé Panetto, Tharam Dillon, Michele Missikoff, Lin Liu, Oscar Pastor, Alfredo Cuzzocrea, and Timos Sellis, editors, *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, volume 8841 of *Lecture Notes in Computer Science*, pages 112–129. Springer Berlin Heidelberg, 2014.
- [120] J. Vanhatalo, H. Völzer, and F. Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In B. Krämer, K. Lin, and P. Narasimhan, editors, *Proceedings of Service-Oriented Computing (ICSOC 2007)*, volume 4749 of *Lecture Notes in Computer Science*, pages 43–55. Springer-Verlag, Berlin, 2007.
- [121] Luciano García-Bañuelos, Marlon Dumas, Marcello La Rosa, Jochen De Weerd, and Chathura C. Ekanayake. Controlled automated discovery of collections of business process models. *Information Systems*, 46:85 – 101, 2014.
- [122] W. M. P. van der Aalst and H. M. W. Verbeek. Process discovery and conformance checking using passages. *Fundam. Inf.*, 131(1):103–138, January 2014.
- [123] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Discovering and navigating a collection of process models using multiple quality dimensions. In *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, pages 3–14, 2013.
- [124] Chathura C. Ekanayake, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Slice, mine and dice: Complexity-aware automated discovery of business process models. In *Proceedings of the 11th International Conference on Business Process Management, BPM'13*, pages 49–64, Berlin, Heidelberg, 2013. Springer-Verlag.
- [125] SanderJ.J. Leemans, Dirk Fahland, and WilM.P. van der Aalst. Scalable process discovery with guarantees. In Khaled Gaaloul, Rainer Schmidt, Selmin Nurcan, Sérgio Guerreiro, and Qin Ma, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 214 of *Lecture Notes in Business Information Processing*, pages 85–101. Springer International Publishing, 2015.
- [126] A. J. M. M. Weijters and W. M. P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integr. Comput.-Aided Eng.*, 2003.
- [127] Wil van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, 2004.

- [128] Jorge Munoz-Gama, Josep Carmona, and WilM.P. van der Aalst. Conformance checking in the large: Partitioning and topology. In Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *Business Process Management*, volume 8094 of *Lecture Notes in Computer Science*, pages 130–145. Springer Berlin Heidelberg, 2013.
- [129] Jan Mendling, Gustaf Neumann, and Wil van der Aalst. Understanding the occurrence of errors in process models based on metrics. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803 of *Lecture Notes in Computer Science*, pages 113–130. Springer Berlin Heidelberg, 2007.
- [130] J. Mendling, H.M.W. Verbeek, B.F. van Dongen, W.M.P. van der Aalst, and G. Neumann. Detection and prediction of errors in {EPCs} of the {SAP} reference model. *Data & Knowledge Engineering*, 64(1):312 – 329, 2008. Fourth International Conference on Business Process Management (BPM 2006)8th International Conference on Enterprise Information Systems (ICEIS' 2006)Four selected and extended papersThree selected and extended papers.
- [131] WilM.P. van der Aalst and Minseok Song. Mining social networks: Uncovering interaction patterns in business processes. In Jörg Desel, Barbara Pernici, and Mathias Weske, editors, *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer Berlin Heidelberg, 2004.
- [132] Process discovery: Capturing the invisible. *IEEE Comp. Int. Mag.*, 5(1):28–41, 2010.
- [133] Jonathan E. Cook and Alexander L. Wolf. Automating process discovery through event-data analysis. In *Proceedings of the 17th International Conference on Software Engineering*, ICSE '95, pages 73–82, New York, NY, USA, 1995. ACM.
- [134] W. M. P. van der Aalst, H. A. Reijers, A. J. M. M. Weijters, B. F. van Dongen, A. K. Alves de Medeiros, M. Song, and H. M. W. Verbeek. Business process mining: An industrial application. *Inf. Syst.*, 32(5):713–732, July 2007.
- [135] The process mining toolkit 5.0. <http://www.promtools.org/doku.php?id=prom65>.
- [136] A. Rozinat and W. M. P. van der Aalst. Decision mining in prom. In *Proceedings of the 4th International Conference on Business Process Management*, BPM'06, pages 420–425, Berlin, Heidelberg, 2006. Springer-Verlag.
- [137] Jan Claes and Geert Poels. Process mining and the prom framework: An exploratory survey. In Marcello La Rosa and Pnina Soffer, editors, *Business*

- Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 187–198. Springer Berlin Heidelberg, 2013.
- [138] JCAM Buijs. *Flexible Evolutionary Algorithms for Mining Structured Process Models*. PhD thesis, PhD thesis. Eindhoven, The Netherlands: Technische Universiteit Eindhoven, 2014 (cit. on p. 179), 2014.
- [139] Y.P.J.M. van Oirschot. Using Trace Clustering for Configurable Process Discovery Explained by Event Log Data. Master’s thesis, 2014.
- [140] Marcello La Rosa, Johannes Lux, Stefan Seidel, Marlon Dumas, and Arthur H.M. ter Hofstede. Questionnaire-driven configuration of reference process models. In John Krogstie, Andreas Opdahl, and Guttorm Sindre, editors, *Advanced Information Systems Engineering*, volume 4495 of *Lecture Notes in Computer Science*, pages 424–438. Springer Berlin Heidelberg, 2007.
- [141] Colin Atkinson, Philipp Bostan, Daniel Brenner, Giovanni Falcone, Matthias Gutheil, Oliver Hummel, Monika Juhasz, and Dietmar Stoll. Modeling components and component-based systems in kobra. In Andreas Rausch, Ralf Reussner, Raffaella Mirandola, and František Plášil, editors, *The Common Component Modeling Example*, volume 5153 of *Lecture Notes in Computer Science*, pages 54–84. Springer Berlin Heidelberg, 2008.
- [142] Shamim Hasnat Ripon, Kamrul Hasan Talukder, and M. Khademul Islam Molla. Modelling variability for system families. *CoRR*, abs/1009.5088, 2010.
- [143] H.M.W. Verbeek W.M.P. van der Aalst D. Schunselaar, H. Leopold and H.A. Reijers. Configuring configurable process models made easier: An automated approach. In *International Workshop on Process Model Collections: Management and Reuse (PMC-MR)*. 2015.
- [144] Alexei Lapouchnian, Yijun Yu, and John Mylopoulos. Requirements-driven design and configuration management of business processes. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 246–261. Springer Berlin Heidelberg, 2007.
- [145] Paul C. Clements. Managing variability for software product lines: Working with variability mechanisms. In *SPLC*, pages 207–208, 2006.
- [146] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.
- [147] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, July 2003.

- [148] Michael Zur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *Proceedings of the 20th International Conference on Advanced Information Systems Engineering, CAiSE '08*, pages 465–479, Berlin, Heidelberg, 2008. Springer-Verlag.
- [149] W.M.P. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann, and M.H. Jansen-Vullers. Configurable process models as a basis for reference modeling. *Business Process Management Workshops*, 2006.
- [150] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.
- [151] Wil van der Aalst and Christian Stahl. *Modeling Business Processes: A Petri Net-Oriented Approach*. The MIT Press, 2011.
- [152] Mourad Amziani, Tarek Melliti, and Samir Tata. A generic framework for service-based business process elasticity in the cloud. In *Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012. Proceedings*, pages 194–199. 2012.
- [153] Marcello La Rosa, Hajo A. Reijers, Wil M.P. van der Aalst, Remco M. Dijkman, Jan Mendling, Marlon Dumas, and Luciano García-Bañuelos. Apro-more: An advanced process model repository. *Expert Systems with Applications*, 38(6):7029 – 7040, 2011.
- [154] A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. Process mining: Extending the alpha-algorithm to mine short loops. In *Eindhoven University of Technology, Eindhoven*, 2004.
- [155] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. XES, XESame, and ProM 6. In P. Soffer and E. Proper, editors, *Information Systems Evolution*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer-Verlag, Berlin, 2010.
- [156] B. F. Van Dongen. A meta model for process mining data. In *In Proceedings of the CAiSE WORKSHOPS*, pages 309–320, 2005.
- [157] Nour Assy, Nguyen Ngoc Chan, and Walid Gaaloul. Assisting business process design with configurable process fragments. In *SCC '13*.
- [158] Nour Assy, Nguyen Ngoc Chan, Walid Gaaloul, and Bruno Defude. Deriving configurable fragments for process design. *International Journal of Business Process Integration and Management* 10, 7(1):2–21, 2014.

- [159] Hanna Eberle, Tobias Unger, and Frank Leymann. Process fragments. In *On the Move to Meaningful Internet Systems: OTM 2009, Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009, Vilamoura, Portugal, November 1-6, 2009, Proceedings, Part I*, pages 398–405.
- [160] Bruno T. Messmer. *Efficient Graph Matching Algorithms*. PhD thesis, Switzerland, 1995.
- [161] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Trace alignment in process mining: Opportunities for process diagnostics. In *BPM*, 2010.
- [162] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48, 1970.
- [163] Nour Assy and Walid Gaaloul. Configuration rule mining for variability analysis in configurable process models. In *ICSOC*, volume 8831, pages 1–15. 2014.
- [164] Nour Assy and Walid Gaaloul. Extracting configuration guidance models from business process repositories. In *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, pages 198–206, 2015.
- [165] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., 2005.
- [166] Matthias Weidlich, Jan Mendling, and Mathias Weske. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Trans. Softw. Eng.*, 37(3):410–429, May 2011.
- [167] Xiaobin Fu, Jay Budzik, and Kristian J. Hammond. Mining Navigation History for Recommendation. In *IUI '00*, pages 106–112. 2000.
- [168] Weiyang Lin, Sergio A. Alvarez, and Carolina Ruiz. Collaborative recommendation via adaptive association rule mining. In *Data Mining and Knowledge Discovery*. 2000.
- [169] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94*, pages 487–499. 1994.
- [170] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. Reverse engineering feature models. In *ICSE*. 2011.
- [171] Jack Edmonds. Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

- [172] Wil M.P van der Aalst, V. Rubin, B. F. Van Dongen, E. Kindler, and C. W. Günther. Process mining: A two-step approach using transition systems and regions. Technical report, BPM Center Report BPM-06-30, BPM Center, 2006.
- [173] Eric Badouel and Philippe Darondeau. Theory of regions. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer, 1998.
- [174] The synthesis problem for elementary net systems is np-complete. *Theoretical Computer Science*, 186(1–2):107 – 134, 1997.
- [175] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving petri nets from finite transition systems. *Computers, IEEE Transactions on*, 47(8):859–882, Aug 1998.
- [176] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing petri nets from state-based models. In *Computer-Aided Design, 1995. ICCAD-95. Digest of Technical Papers., 1995 IEEE/ACM International Conference on*, pages 164–171. Nov 1995.
- [177] Ferenc Bodon. A fast apriori implementation. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03), volume 90 of Workshop Proceedings*, 2003.
- [178] Alfred V. Aho and John E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., 1974.
- [179] Nour Assy, Walid Gaaloul, and Bruno Defude. Mining configurable process fragments for business process design. In *DESRIST*, volume 8463, pages 209–224. 2014.
- [180] Matthias Weidlich and Jan Martijn E. M. van der Werf. On profiles and footprints - relational semantics for petri nets. In *Petri Nets*, volume 7347, pages 148–167, 2012.
- [181] Esko Ukkonen. On-Line Construction of Suffix Trees. *Algorithmica*, 1995.
- [182] Gill Bejerano and Golan Yona. Variations on probabilistic suffix trees: statistical modeling and prediction of protein families. *Bioinformatics*, 2001.
- [183] Iliopoulos and et al. The weighted suffix tree: An efficient data structure for handling molecular weighted sequences and its applications. *Fundam. Inform.*, 2006.

- 
- [184] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164, April 2009.
- [185] David Salomon. *Data compression - The Complete Reference, 4th Edition*. Springer, 2007.
- [186] Configurable process models : experiences from a municipality case study. In *21th International Conference on Advanced Information Systems Engineering, 10-12 June 2009, Amsterdam, the Netherlands, 2009*.
- [187] Jorge Cardoso. Evaluating the process control-flow complexity measure. In *IEEE ICWS '05*.
- [188] Jorge Cardoso, Jan Mendling, Gustaf Neumann, and Hajo A. Reijers. A discourse on complexity of process models. In *Business Process Management Workshops '06*.
- [189] Beate List and Birgit Korherr. An evaluation of conceptual business process modelling languages. In *Proceedings of the ACM symposium on Applied computing '06, 2006*.
- [190] Hajo A. Reijers and Irene T. P. Vanderfeesten. Cohesion and coupling metrics for workflow process design. In *BPM '04*.
- [191] Jan Mendling. *Detection and prediction of errors in EPC business process models*. PhD thesis, 2007.
- [192] Jan Mendling. *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*. Springer Publishing Company, Incorporated, 1 edition, 2008.
- [193] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.
- [194] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968.
- [195] A. K. Alves De Medeiros and C. W. Günther. Process mining: Using cpn tools to create test logs for mining algorithms. In *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 177–190, 2005.
- [196] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 2*. Springer-Verlag, London, UK, UK, 1995.



- [197] Wil M. P. van der Aalst, Ana Karla A. de Medeiros, and A. J. M. M. Weijters. Process equivalence: Comparing two process models based on observed behavior. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2006.
- [198] A.K.Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic process mining: A basic approach and its challenges. In ChristophJ. Bussler and Armin Haller, editors, *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 203–215. Springer Berlin Heidelberg, 2006.
- [199] A. Rozinat. Conformance testing: Measuring the fit and appropriateness of event logs and process models. In *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812 of *Lecture Notes in Computer Science*, pages 163–176. Springer-Verlag, 2006.
- [200] Minseok Song, ChristianW. Günther, and WilM.P. van der Aalst. Trace clustering in process mining. In Danilo Ardagna, Massimo Mecella, and Jian Yang, editors, *Business Process Management Workshops*, volume 17 of *Lecture Notes in Business Information Processing*, pages 109–120. Springer Berlin Heidelberg, 2009.
- [201] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Context aware trace clustering: Towards improving process mining results. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2009, April 30 - May 2, 2009, Sparks, Nevada, USA*, pages 401–412, 2009.
- [202] Robert K. Yin. *Case Study Research: Design and Methods, 3rd Edition (Applied Social Research Methods, Vol. 5)*. SAGE Publications, Inc, 3rd edition, December.
- [203] Marcello La Rosa. *Managing variability in process-aware information systems*. PhD thesis, Queensland University of Technology, 2009.
- [204] W. C. Lim. Effects of reuse on quality, productivity, and economics. *Software, IEEE*, 1994.
- [205] Oliver Holschke, Jannis Rake, and Olga Levina. Granularity as a cognitive factor in the effectiveness of business process model reuse. In *Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings*, pages 245–260, 2009.
- [206] Vijayan Sugumaran and Veda C. Storey. A semantic-based approach to component retrieval. *SIGMIS Database*, 34(3):8–24, August 2003.

- 
- [207] Sergey Smirnov, Hajo A. Reijers, Mathias Weske, and Thijs Nugteren. Business process model abstraction: A definition, catalog, and survey. *Distrib. Parallel Databases*, 30(1):63–99, February 2012.
- [208] C.W. Günther. *Process Mining in Flexible Environments*. Phd thesis, Eindhoven University of Technology, September 2009.
- [209] W.M.P. van der Aalst and B.F. van Dongen. Discovering workflow performance models from timed logs. In Yanbo Han, Stefan Tai, and Dietmar Wikarski, editors, *Engineering and Deployment of Cooperative Information Systems*, volume 2480 of *Lecture Notes in Computer Science*, pages 45–63. 2002.
- [210] P.T.G. Hornix. *Performance Analysis of Business Processes using Process Mining*. Phd thesis, Eindhoven University of Technology, January 2007.
- [211] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1995.
- [212] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, 2011.
- [213] W.M.P. van der Aalst. Business process configuration in the cloud: How to support and analyze multi-tenant processes? *Web Services, European Conference on*, 0:3–10, 2011.
- [214] Topology and orchestration specification for cloud applications version 1.0. <http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>, January 2013.
- [215] Occi open cloud computing interface specification. <http://occi-wg.org/about/specification/>.